Distributed Artificial Intelligence and Intelligent Agents

Marco Peressutti

April 20, 2022

Contents

Chapter 1

Introduction

1.1 Definition of an Agent

- an agent has **independent behaviour**: when there is no possibility for direct supervision
- agents **collaborate** and **communicate** with each other
- agents have **proactive behaviour**: they have the reasoning possibility to proactively propose some solutions
- **privacy ensurance**: privacy should be ensured via **personalization**. A person's interest should not be disclose to some external entity other than the agent itself.

independent behaviour

collaborate

communicate

proactive behaviour

privacy ensurance

personalization

1.1.1 Formal definitions

1. American Heritage Dictionary:

   "One that acts or has the power or authority to **act** ... or **represent** another"

   Agents are not independent entities that operate by their own view but they always represent some interest of those who create them.

2. Negroponte

   "Digital sister in law"

   Agents should have specialized expertise and knowledge of preferences of those who create them

3. Russel and Norvig

   “An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.”

4. Pattie Maes

   “Autonomous Agents are computational systems that **inhabit** some complex dynamic environment, **sense** and **act** autonomously in this environment, and by doing so **realize** a set of goals or tasks for which they are designed”

   Agents leave in an environment.

5. IBM

   “Intelligent agents are software entities that carry out some set of **operations on behalf of** a user or another program with some degree of **independence** or **autonomy**, and in doing so, employ some **knowledge** or **representations** of the user’s **goals** or **desires**”

6. Coen

   “Software agents are programs that engage in dialog [and] **negotiate** and **coordinate** transfer of information”

   Agents must have the possibility to communicate and consequently coordinate and negotiate with one another.

1.1.2 Logicale behind autonomous systems

- More and more everyday tasks are computer based
- The world is in a midst of an information revolution
- Increasingly more users are untrained
- **Lack of programming paradigm** for decentralized program/system construction in dynamic environment.
  Existing paradigms were constructed for closed world (i.e. completely specified environment). However in AI the environment is not specified completely (hence there is the need for dynamic exploration of the environment).

Lack of programming paradigm

The solution to this lack of paradigm is to **emulate human behaviour**, therefore agents must be able to:

emulate human behaviour

- perceive the environment
- affect the environment or act in the environment
- have a **model of behaviour**
- have intentions/motivations to be fulfilled by implementing corresponding goals.
  In fact, goals are not enough, the system needs to generate new goals from its intentions and beliefs.
- communication

model of behaviour

1.1.3 Agent definition and properties

Wooldridge, Jennings (**weak notion**):
“Agent is a hardware or (more usually) software-based computer system that enjoys the following properties:”

weak notion

- **autonomy**.
  Agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.
  Hence in order for an agent to be autonomous it must:

  – Act independently

autonomy

  - Have control over its internal state.
    Contrary to objects, agents need to encapsulate behaviour other than their state in the environment.
    They, in fact, differs from objects since they are required to have a:
    1. **degree of autonomy**: agents embody a stronger notion of autonomy than objects, in particular, agents decide for themselves whether or not to perform an action.
       degree of autonomy
    2. **degree of smartness**: capable of flexible (reactive, pro-active, social) behaviour; standard object models do not have such behaviour
       degree of smartness
    3. **degree of activeness**: a multi-agent system is inherently multi-threaded in that each agent is assumed to have at least one thread of active control.
       degree of activeness

- **pro-activeness**.
  pro-activeness
  Agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.
  In short, agents must be able to create goals on their own initiative.
- **reactivity**.
  reactivity
  Agents perceive their environment and respond in a timely fashion to changes that occur in it.
  In short, communication with the environment.
- **social ability**.
  social ability
  Agents interact with other agents (and possibly humans) via some kind of agent-communication language.
  In short, communication with other agents

Wooldridge, Jennings (**strong notion**):

strong notion

- **Mentalistic notions**, such as beliefs and intentions are often referred to as properties of strong agents
  Mentalistic notions
- **Veracity**, agents will not knowingly communicate false information
  Veracity
- **Benevolence**: agents do not have conflicting goals and always try to do what is asked of it
  Benevolence
- **Rationality**: an agent will act in order to achieve its goals and will not act in such a way as to prevent its goals being achieved
  Rationality
- **Mobility**: the ability of an agent to move around a network
  Mobility

In addition to these basic properties several other alternatives have been proposed over the years:

1. Isaac Asimov's laws of robotics:

   Law One
   A robot may not injure a human being or, through inaction, allow a human being to come to harm

   Law Two
   A robot must obey orders given to it by human beings except where such orders would conflict with the First Law

   Law Three
   A robot must protect its own existence, as long as such protection does not conflict with the First or Second Law

   Law Zeroth
   A robot may not harm humanity, or, by inaction, allow humanity to come to harm

2. A robot must establish its identity as a robot in all cases

3. A robot must know it is a robot

4. A robot must reproduce. As long as such reproduction does not contradict with Laws 1,2 and 3

5. All robots endowed with comparable human reason and conscience should act towards one another in a spirit of brotherhood

Summary:

- Agents acts on behalf of other entities
- Agents must have weak agent characteristics
- Agents may have strong agent characteristics

1.2 Individual and Group Perspective

There are two fundamental agents dimensions:

- **Individual agents perspective**. That deals with how to build agents that are capable of independent autonomous actions in order to successfully carry out the tasks that we delegate to them (**Micro aspects**).
- **Group agents perspective**. That deals with how to build agents that are capable of interacting (cooperating, coordinating, negotiating) with other agents in order to successfully carry out the tasks we delegate to them (**Macro aspects**).

Individual agents perspective

Micro aspects

Group agents perspective

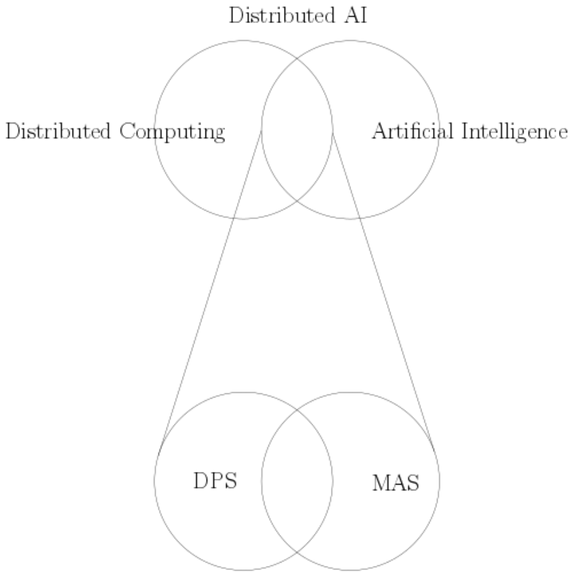
Macro aspects

1.3 Distributed AI and MAS

- Distributed AI is a topic/subject rather than the creation of a brain that is distributed.
- It became part of AI when it was possible to execute on more than one CPU.
- For this reason, it is often considered as the intersection of **Distributed Computing** and **Artificial Intelligence**.

Distributed Computing

Artificial Intelligence



- Distributed AI includes two different sub field: **Distributed Problem Solving (DPS)** and **Multi-Agent Systems (MAS)**.
- DAI deals with several aspects and dimensions such as:
  - Agent granularity
  - Heterogeneity of agents
  - Communication possibilities
  - Methods of distributing control among agents

Distributed Problem Solving (DPS)

Multi-Agent Systems (MAS)

1.3.1 Distributed Problem Solving (DPS)

DPS considers how the task of solving a particular problem can be divided among a number of modules that cooperate in dividing and sharing knowledge about the problem and its evolving solution(s):

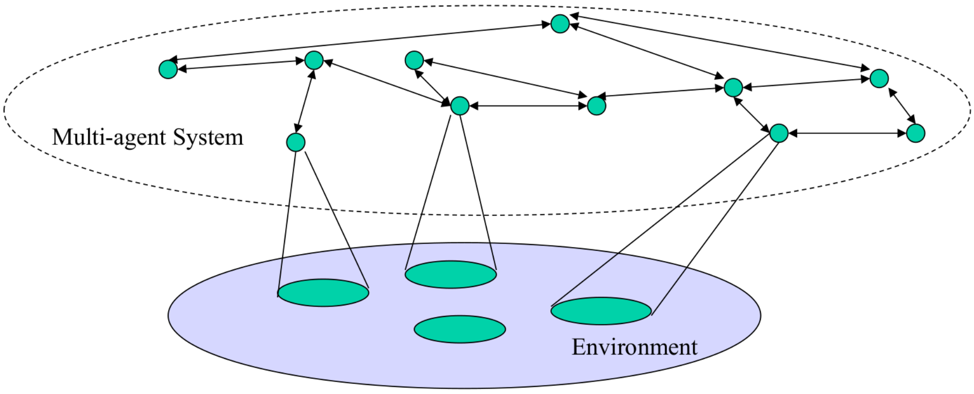
- In pure DPS systems, all interaction strategies are incorporated as an integral part of the system
- DPS has focused on achieving goals under varying environmental conditions, having agents with established properties

In other terms in DPS:

1. a problem is divided into modules
2. each module is allocated to a different agent
3. the agents will solve the problem by means of communicating the solution to their allocated module

1.3.2 Multi-Agent Systems (MAS)

- MAS are designed in a decentralized way with great part of independency and autonomy
- Agents with individual preferences will interact in particular environments such that each will consent to act in a way that leads to desired global goal
- MAS asks how, for a particular environment, can certain collective goal be realized if the properties of agents can vary uncontrollably:
  - loosely-coupled networks of problem solvers (agents) that work together to solve problems that are beyond their capabilities
  - no necessary guarantees about other agent
- MAS contain a number of agents which interact with one another through communication. The agents are able to act in an environment; where each agent will act upon or influence different parts of the environment. If two field of influence overlap, the conflict is (hopefully) resolved via negotiation and communication



- An important concept in MAS is that there is no central control (because the control is distributed to the various agents) and knowledge or information sources may also be distributed

In other terms, contrary to DPS, MAS do not deal with task to be solved but rather rules to be followed, and these rules are for example, their beliefs, desire and intentions or protocols for negotiation and communication.

The motives behind the use of MAS are:

- To solve problems that are too large for a centralized agent
- To allow interconnection and inter-operation of multiple legacy systems
- To provide a solution to inherently distributed problems
- To provide solutions which draw from distributed information sources
- To provide solutions where expertise is distributed
- To offer conceptual clarity and simplicity of design

The are two sub-classes of MAS:

- **Cooperative**. Cooperative
  Agents are designed by interdependent designers.
  Agents act for increased good of the system.
  Agents are concerned with increasing the performance of the system.

  Hence if we imagine that the goodness of the system is represented by a global function. Agents in a cooperative MAS will try to maximize such function.
- **Self-interested** Self-interested
  Agents designed by independent designers.
  Agents have their own agenda and motivation.
  Agents are concerned with the benefit and performance of the individual agent.

  Hence, if we imagine that the goodness of each agent is represented by a local function (one for each agent). They will try to maximize their own function.

Among the benefits of using MAS over DPS we find that, MAS:

- Faster problem solving
- Decrease in communication
- Flexibility
- Increased reliability

However, its main drawback is the **lack of predictability**. lack of predictability

Summary of definitions:

- Distributed Computing: focus on low level parallelization and synchronization
- Distributed AI: Intelligent control as well as data may be distributed. Focus on problem solving, communication and coordination
- DPS: Task decomposition (task sharing) and/or solution synthesis (result sharing): information management
- MAS: Behavior coordination and management

1.4 Emergence, Swarm Intelligence and other terms

1.4.1 Emergence

With the term **Emergence**, we refer to those global (macro level) behaviour, patterns and properties that arise from the interactions between local parts of the system (micro level).
Systems that exhibit emergence can be characterized as simple, robust and adaptive.

Emergence

1.4.2 Swarm Intelligence

The term **Swarm Intelligence** refers to any attempt to design algorithms or distributed problem-solving devices inspired by the collective behaviour of social insect colonies and other animal societies.
Multi-robot systems, which implement or adapt the concept of emergent behaviour, are commonly referred to as **swarm robotic systems**.

Swarm Intelligence

swarm robotic systems

1.4.3 Self-Organisation

Self-Organization is a dynamical and adaptive process where systems acquire and maintain structure themselves, without external control.
The term self-organization is also used as the process that leads to the state of emergence.
Self-organization and emergent systems are distinct concepts, they still have one thing in common, that is: there is no explicitly external control whatsoever.
The main difference between self-organization and emergence is that in the case of self-organization, individual entities can be aware of the system's intended global behaviour. In consequence, self-organization can be seen as a weak form of emergence.
The intuitive and regularly used approach to realize self-organization is applying the concept of feedback loops.

Self-Organization

1.4.4 Self-Adaptation

When the approach with feedback loop is applicable to single entity system it is usually referred as **self-adaptation**.
Self-adaptive software modifies its own behaviour in response to changes in its operating environment.
If a decentralized system containing several entities exhibits adaptive behaviour to external changes this is as well considered self-adaptation.

self-adaptation

1.4.5 Characteristics of Emergence, Swarm Intelligence, Self-Adaptation and Self-Organization

Having ensembles of robust (multi-)agent systems that maintain their structure and feature a high level of adaptation.
It would no longer be necessary to exactly specify the low lever system behaviour in all possible situations that might occur, but rather leaving the system with a certain degree of freedom to allow for autonomous reaction and adaptation to new situations in an intelligent way.

1.4.6 Application

Self-organization algorithms have been applied in many multi-agent domains, like combinatorial optimization, communication networks and robotics.
The algorithms, respectively mechanism, are used for various purposes, like motion control, information sharing and decision making.

Chapter 2

Agent Negotiation

- Davis and Smith
  "Negotiation is a process of improving agreement (reducing inconsistency and uncertainty) on common viewpoints or plans through the exchange of relevant information".

  Negotiation is a two way exchange of information (more than one agent must be involved for negotiation to happen).
  Each party evaluates the information from its own perspective.
  final agreement is achieved by **mutual selection**.

  mutual selection
- Pruitt
  "Negotiation is a process by which a joint decision is made by two or more parties. The parties first verbalize contradictory demands and then move towards agreement by a process of concession or search for new alternatives"

  Mutual conflict is the necessary condition to start negotiation.
  Negotiation involves three main elements: **communication**, **decision making** (decision about next concession to make in order to continue negotiation) and a **procedural model** (protocol inside which we communicate and make decision).
  Implicit negotiation where one or more parties does not know that there is conflict can happen but it will not be treated in the course.

  Mutual conflict
  communication
  decision making
  procedural model
  Implicit negotiation

From the definitions provided we can conclude that there are three basic negotiation categories:

- **Negotiation language category** (Communication)
  We will consider the language category more in depth in chapter 3: Agent Communication.
  The language category deals with the concepts of:

  Negotiation language category

  – **Language primitives**
    Low level communications (message sending) and conversational aspects (relative to speech acts and performatives)

    Language primitives
  – **Object structure**
    What is the object of negotiation (price, schedule, tasks, ...) and what is the context of the message/information.

    Object structure
  – **Internal protocol**
    Specify the possibilities of initiating a negotiation cycle and responding to a message

    Internal protocol
  – **Semantics**
    Strictly related to language primitives (pre-conditions, post-conditions, modal logic).

    Semantics

- **Negotiation decision category** (Decision making)
  The decision category deals with which strategy and consequently which language primitive to choose inside of a protocol.
  It deals with the concepts of:

  Negotiation decision category

  – preferences
    what are preferable outcomes
  – utility functions
    numerical/functional representation of preferences
  – comparing and matching functions
  – negotiation strategies

- **Negotiation process category** (Protocols/procedural model)
  It deals with the concepts of:

  Negotiation process category

  – **Procedural negotiation model**
    what is the protocol of negotiation and their properties

    Procedural negotiation model
  – **System behaviour and analysis**
    analysis of the interaction between different parts of the system in order to recognize what are protocols and preferences (it will not be considered in the course)

    System behaviour and analysis

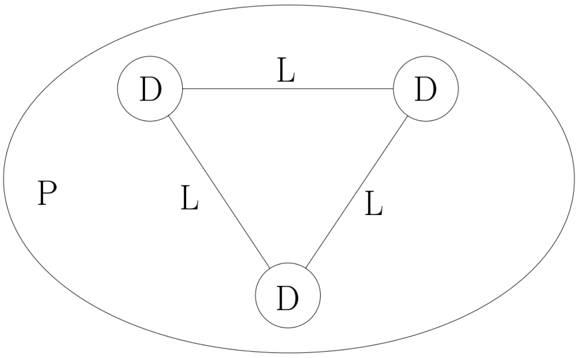


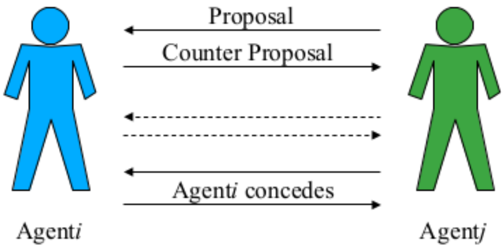
Figure 2.1: trivial form of how the categories are interrelated in the negotiation context

Furthermore, since negotiation happens when mutual conflict occurs, we can say that negotiation is a **conflict resolution** approach/method/area.

conflict resolution

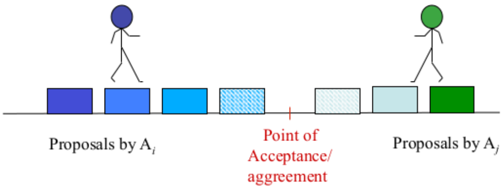
In general terms, negotiation usually proceeds in **series of rounds** with every agent making a proposal at every round. For this reason, the negotiation process can be seen as a mutual exchange of information.

series of rounds



Another way of looking at the negotiation process is as an iterative concession process: each agent starts at its most preferred outcome and makes a proposal, it will then proceed to conceed to a less preferred outcome until a **Point of acceptance or agreement** is reached. (Be aware of the fact that not all negotiation end with agreement).

Point of acceptance or agreement



Hence the Negotiation process can be seen as moving towards a point of agreement.

2.1 Multiagent Interactions

In the typical structure of a multiagent systems, one can notice the presence of some common elements:

- the system contains a number of agents
- each agent has the ability to communicate with another agent
- each agent has the ability to act in an environment
- different agents have different **spheres of influence**, i.e. they are able to influence or have control over the environment, or a part of it.

spheres of influence

- these spheres of interest may coincide or overlap giving rise to dependencies between agents.

Thus:
“When faced with what appears to be a multiagent domain, it is critically important to understand the type of interaction that takes place between agents in order to be able to make the best decision possible about what action to perform.”

2.1.1 Utilities and Preferences

In order to simplify the analysis of multiagent interactions, throughout this chapter we will consider the following assumption, unless stated differently:

- Two agents act and interact in an environment. We will refer to these two agents as agent i and agent j
- The two agents are **self-interest**, i.e. each agent has its own preferences and desires about how the world should be

self-interest

- There exists a **set of outcomes** that the two agents have preferences over. We will refer to this set with the notation:

$$\Omega = \{\omega_1, \omega_2, ...\}$$

set of outcomes

The preferences of the two agents are formally captured by means of **utility functions**, which maps each outcome in the set Ω to a real number describing how “good” an outcome is.
Since the agents are self-interest each agent will have its own utility function, hence:

utility functions

$$u_i : \Omega \to \mathbb{R} \qquad ; \qquad u_j : \Omega \to \mathbb{R}$$

The introduction of an utility function eventually leads to a **preference ordering** over the outcomes of the set for each agent. This means that, if ω and ω' are both possible outcomes contained in Ω and $u_i(\omega) \geq u_i(\omega')$, then agent i prefers outcome ω at least as much as outcome ω'.

preference ordering

Since we will make extensive use of the concept of preference ordering throughout the next sections, it is worth introduce a much more compact notation. We will write:

$$\begin{array}{lll} \omega \succeq_i \omega' & \text{as an abbreviation for} & u_i(\omega) \geq u_i(\omega') \\ \omega \succ_i \omega' & \text{as an abbreviation for} & u_i(\omega) > u_i(\omega') \end{array}$$

where the second expression represents the case in which outcome ω is **stricly preferred** by agent i over ω'.

stricly preferred

In other words, the notation can be summarized as follows:

$$\boxed{\omega \succ_i \omega' \iff u_i(\omega) \geq u_i(\omega') \text{ and not } u_i(\omega) = u_i(\omega')}$$

Moreover we can notice that the ordering relation expressed by the operator $\succeq$, has the following properties:

1. **Reflexivity**

Reflexivity

$$\forall \omega \in \Omega \to \omega \succeq_i \omega$$

2. **Transitivity**

Transitivity

$$\text{If } \omega \succeq_i \omega' \ \&\& \ \omega' \succeq_i \omega\text{“} \to \omega \succeq_i \omega\text{“}$$

3. **Comparability**

Comparability

$$\forall \omega \in \Omega, \forall \omega' \in \Omega \to \omega \succeq_i \omega' \ || \ \omega' \succeq_i \omega$$

The strict preference relationship still has the second and third property, however it is not reflexive

2.1.2 Setting the scene

So far, we have introduced a model to represent the agents preferences, but we still need to formalize a model of the environment in which agents act and interact. In particular, we will assume that:

- Two agents will simultaneously choose an action to perform in the environment
- as a result of the actions they select, an outcome in Ω will result
- the **actual outcome** that will result will depend on the particular **combination of actions** performed.
  In other terms: both agents can influence the actual outcome
- The two agents must perform an action
- The two agents cannot see the action performed by the other agent

actual outcome

combination of actions

We will restrict our analysis to two possible actions that the agent can choose from: cooperate C and defect D. Hence, we will refer to the **set of actions** with the notation

set of actions

$$Ac = \{C, D\}$$

Given all the above, the environment is formally described by a **state transformer function**:

state transformer function

$$\tau : \underbrace{Ac}_{\text{agent } i\text{'s action}} \times \underbrace{Ac}_{\text{agent } j\text{'s action}} \to \Omega$$

Hence, several scenarios can happen:

1. The environment maps each combination of actions to a different outcome, and thus is sensitive to the actions of both agents

$$\tau(D,D) = \omega_1 \qquad \tau(D,C) = \omega_2 \qquad \tau(C,D) = \omega_3 \qquad \tau(C,C) = \omega_4$$

2. The environment maps each combination of actions to the same outcome and thus neither of the agents have influence in the environment

$$\tau(D,D) = \omega_1 \qquad \tau(D,C) = \omega_1 \qquad \tau(C,D) = \omega_1 \qquad \tau(C,C) = \omega_1$$

3. The environment maps each combination of actions to an outcome that is correlated to the choice of only one agent and thus the outcome depends solely on the action performed by one agent

$$\tau(D,D) = \omega_1 \qquad \tau(D,C) = \omega_2 \qquad \tau(C,D) = \omega_1 \qquad \tau(C,C) = \omega_2$$

Since the latter two cases are of no interest for our analysis we will focus on the scenario in which both agents exert some kind of influence on the actual outcome. We, therefore, will consider the agent preferences (in the form of utility function) as follows

$$\left.\begin{array}{llll} u_i(\omega_1) = 1 & u_i(\omega_2) = 1 & u_i(\omega_3) = 4 & u_i(\omega_4) = 4 \\ u_j(\omega_1) = 1 & u_j(\omega_2) = 4 & u_j(\omega_3) = 1 & u_j(\omega_4) = 4 \end{array}\right\}$$

Given the state transformer function, we can abuse notation and write

$$\left.\begin{array}{llll} u_i(D,D) = 1 & u_i(D,C) = 1 & u_i(C,D) = 4 & u_i(C,C) = 4 \\ u_j(D,D) = 1 & u_j(D,C) = 4 & u_j(C,D) = 1 & u_j(C,C) = 4 \end{array}\right\}$$

Hence, with respect to agent i's preferences (first row) over the possible outcomes, we can characterize the preference ordering as follows:

$$(C,C) \succeq_i (C,D) \succeq_i (D,C) \succeq_i (D,D)$$

Similarly for agent j, the resulting preference ordering can be characterize as follows:

$$(C,C) \succeq_i (D,C) \succeq_i (C,D) \succeq_i (D,D)$$

	i defects	i cooperates
j defects	1 1	4 1
j cooperates	1 4	4 4

It is straightforward to see that if both agents **act rationally**, i.e. "they both choose to perform the action that will lead to their preferred outcomes"[1], they will both choose to cooperate. This is because, both agents prefer all the outcomes in which they cooperate, regardless of what the other agent action is.

act rationally

A common way to represent the previously described interaction scenario is via **pay-off matrix** (standard game-theoretic notation): The pay-off matrix uniquely defines a **game in strategic form**. The way to interpret it is as follows:

pay-off matrix

game in strategic form

- each cell in the matrix correspond to one of the possible outcomes
- The top-right value in each cell corresponds to the payoff received by the column player
- The bottom-left value in each cell corresponds to the payoff received by the row player

2.1.3 Solution Concepts and Solution Properties

A question still remains unanswered: What should an agent do? What action should it choose?

We briefly touched on the topic on the previous section, but we have provided neither the concepts that make up the solution nor the desirable properties that a solution should have.

In this section, we will tackle the problem to its core.

Dominant strategies - Dominance

One of the main concept that we will introduce is that of **dominance**.

dominance

"A strategy s_i is [said to be] **dominant** for player i if, no matter what strategy s_j agent j chooses, i will do at least as well playing s_i as it would doing anything else"[1]. The notion of **dominant strategy** is strictly related to the concept of **best response**, in the sense that "a strategy s_i for agent i is dominant if it is the best response to **all** of agent j strategies"[1].

dominant strategy

best response

Thus, a dominant strategy, if it exists, simplify the decision about what action to perform: "the agent guarantees its best outcome by performing the dominant strategy"[1].

Nash Equilibria

The notion of **equilibrium**, or more precisely **Nash equilibrium**, is hard to formalize in the context of strategic decision making.

equilibrium

We can, however, provide a basic definition by saying that two strategies s_1 and s_2 are in Nash equilibrium if:

Nash equilibrium

1. under the assumption that agent i plays s_1, agent j can do no better than play s_2, and
2. under the assumption that agent j plays s_2, agent i can do no better than play s_1

From this conditions, one can clearly notice that **two strategies are in Nash equilibrium if they are the best response to each other**.

The mutual nature of the concept makes it so that *neither agent has any incentive to deviate from a Nash equilibrium*: even if an agent chooses a different strategy, the other agent can do no better than to choose the strategy in Nash equilibrium.

Technically, this type of Nash equilibrium is known as **pure strategy Nash equilibrium**, and as much as it is appealing from its theoretical stand point, it is extremely expensive from a computational stand point.
In fact, finding one or more Nash equilibria requires to consider each combination of strategy and check if they are in Nash equilibrium.

pure strategy Nash equilibrium

Thus, if there are n agents, each with m possible strategies to choose from, the number of possible combinations of actions (and hence possible outcomes) will be m^n.

Nonetheless, the presence of Nash equilibrium provides a definite answer to what action an agent should choose. However, two common issues might emerge:

1. Not every interaction scenario has a pure strategy Nash equilibrium
2. Some interaction scenarios have more than one pure strategy Nash equilibrium

With regard to the first issue, we need to modify our notion of what a strategy is. So far, we have implicitly considered a strategy as a deterministic choice of an action. This is inline with the fact that the subroutines that an agent can execute should not be subject to uncertainties or randomness (in general, we would like a subroutine to yield the same correct result on different execution).

However, "it can be useful to introduce randomness or uncertainty into our actions"[1]. The reason why that is can be best explained considering the game or rock-paper-scissors, of which the pay-off matrix is provided below: From the provided payoff matrix, one can clearly notice that the game has no pure strategy Nash equilibrium as well as no dominant strategy. Yet, this is not completely true: a **mixed strategy** allows the agent to choose between possible choices by introducing randomness into the selection.

	i plays rock	i plays paper	i plays scissors
j plays rock	0 0	1 -1	-1 1
j plays paper	-1 1	0 0	1 -1
j plays scissors	1 -1	-1 1	0 0

mixed strategy

If the agent chooses one of the actions at random, with each choice having equal probability of being selected, the strategy turns out to be in **Nash equilibrium with itself**. This is because if an agent decides to choose an action at random, the other agent can do no better than adopting the same strategy and vice versa.

In general, if a player has k possible choices, $s_1, s_2, ..., s_k$, then a mixed strategy over these choices takes the form:

- play s_1 with probability p_1
- play s_2 with probability p_2
- ...
- play s_k with probability p_k

In other terms, a mixed strategy over $s_1, s_2, ..., s_k$ is a probability distribution over $s_1, s_2, ..., s_k$.

The result formalized by John Forbes Nash, Jr. best summarize the discussion that we have made so far:
"Every game in which every player has a finite set of possible strategies has a Nash equilibrium in mixed strategies"

Pareto efficiency

The notion of **Pareto efficiency** or **Pareto optimality**, contrary to the notion of Nash equilibrium and dominant strategy, is more of a (desirable) property of solutions rather than a solution concept.

Pareto efficiency

Pareto optimality

Formally:
"an outcome is Pareto efficient if there is no other outcome that improves one player's utility without making somebody else worse off".[1]
On the other hand: "An outcome is said to be Pareto inefficient if there is another outcome that makes at least one player better off without making anybody else worse off"[1].

To put it in simpler terms we can consider an example: a brother and a sister have to divide a cake. Among the solutions of this problem, those who are Pareto efficient are:

- The brother eats the whole cake
- The sister eats the whole cake
- Any other distribution of the cake, which leaves no cake left

Hence, a solution is Pareto efficient if it consumes/uses the total utility.

Maximizing social welfare

Similarly to Pareto efficiency, **social welfare** is an important property of outcomes, but is not generally a way of directly selecting them.

social welfare

The core principle of **Maximizing social welfare** involves the measurement of how much utility is created by an outcome in total.
Formally, let $sw(\omega)$ denote the sum of utilities of each agent for outcome ω

Maximizing social welfare

$$sw(\omega) = \sum_{i \in Ag} u_i(\omega)$$

the outcome that maximizes social welfare is the one that maximizes this value.

From an individual agent's point of view, the problem with maximizing social welfare is that it does not look at the pay-offs of individual agents, only to the total welfare created. (maximizing social welfare does not care about how the utility of an outcome is divided among the players).

Other minor properties

- **Convergence/ guaranteed success**
  If it ensures that eventually agreement is certain to be reached.

  Convergence/ guaranteed success
- **Computational efficiency**
  As little computations is needed as possible. Trade off between the cost of the process and the solution quality

  Computational efficiency
- **Distribution**
  All else being equal, distributed protocols should be preferred to avoid a single point of failure and a performance bottleneck. This may conflict with minimizing the amount of communication that is required.

  Distribution
- **Stability**
  Among self interested agents, mechanism should be designed to be stable (non-manipulable), it should motivate each agent to behave in the desired manner.

  Stability
- **Individual rationality**
  Participation in a negotiation is individually rational to an agent if the agent's payoff in the negotiated solution is no less than payoff that the agent would get by not participating in the negotiation.
  A mechanism is individually rational if participation is individually rational for all agents.
  If the negotiated solution is not individually rational for some agent then self-interested agent would not participate in that negotiation.

  Individual rationality

2.1.4 Competitive and Zero-Sum Interactions

A scenario in which an outcome $\omega \in \Omega$ is preferred by agent i over an outcome ω', if, and only if, ω' is preferred over ω by agent j, is said to be **strictly competitive**.
Formally, a competitive scenario takes the form:

strictly competitive

$$\omega \succ_i \omega' \qquad \Longleftrightarrow \qquad \omega' \succ_j \omega$$

Under this condition, it is straightforward to see that the preferences of the players are **diametrically opposed** to one another

diametrically opposed

Zero-sum encounters, similarly, are those in which, for any particular outcome, the utilities of the two agents sum to zero. Formally, these scenario is described by the condition:

Zero-sum encounters

$$u_i(\omega) + u_j(\omega) = 0 \qquad \forall \omega \in \Omega$$

Once again, any zero-sum scenario is strictly competitive, allowing for no possibility of cooperative behavior: "the best outcome for an agent is the worst outcome for its opponent. If an agent allows the opponent to get a positive utility, then it will get negative utility."[1]
Popular Zero-sum encounters are the games of chess and checkers as well as rock-paper-scissors.

Interesting enough, it is debatable that zero-sum games actually exists in real-world scenarios (apart from artificially forms of interaction like the games mentioned before). However, it appears that people interacting in many scenarios have a tendency to treat them as if they were zero sum.

2.1.5 The Prisoner's Dilemma

The **Prisoner's Dilemma** is as follows: "Two man are collectively charged with a crime and held in separate cells. They have no way of communicating with each other or making any kind of agreement. The two man are told that:

Prisoner's Dilemma

1. If one of them confesses to the crime and the other does not, the confessor will be freed, and the other will be jailed for three years
2. If both confess to the crime, then each will be jailed for two years

Both prisoners know that if neither confesses, then they will each be jailed for one year."[1] Under this conditions let us associate the act of confessing with defection D and not confessing with cooperating C.

There exists 4 possible outcomes to the prisoner's dilemma:

	i defects	i cooperates
j defects	2 2	0 5
j cooperates	5 0	3 3

Where the number displayed are NOT the year spent in prison.

From the provided payoff matrix we can come up with the preference ordering of each agent/prisoner:

$$\begin{cases} (D,C) \succ_i (C,C) \succ_i (D,D) \succ_i (C,D) & \text{for agent } i \\ (C,D) \succ_j (C,C) \succ_j (D,D) \succ_j (D,C) & \text{for agent } j \end{cases}$$

Thus, we can analyze the best response of an agent to the choice of action of the other:

- Assuming the other player cooperates. The best response is to defect
- Assuming the other player defect. The best response is to defect

From this we can conclude that defection for i is the best response to all possible strategies of the player j: by definition, defection is thus a dominant strategy for i.

Since the same conclusion can be drawn for agent j, the scenario under consideration is **symmetric**: this will result in both agent choosing to defect.
From an analysis of the problem under the concepts that we have seen previously, we can state that

symmetric

- the pair (D,D) is the only Nash equilibrium of the scenario, however intuition says that this is not the best the players can do.
- the pair of actions (D,D) is also the only one that is not Pareto efficient.
- The outcome that maximizes social welfare is (C,C)

"The fact that utility seems to be wasted here, and that the agents could both do better by cooperating, even though the rational thing to do is to defect, is why this is referred to as dilemma." [1]

Moreover, the prisoner's dilemma also seems to be the game that characterized the **tragedy of the commons**: which is concerned with the use of a shared, depletable resource by a society of

tragedy of the commons

self-interested individuals (e.g. overfishing in the seas, exploitation of bandwidth capacity on the Internet).

Many people find the conclusion of the analysis deeply upsetting: “the result seems to imply that cooperation can only arise as a result of irrational behavior, and that cooperative behavior can be exploited by those who behave rationally.”[1]
Binmore argues that the discomfort we have with the analsys of the prisoner’s dilemma is misplaced: “A whole generation of scholars swallowed the line that the prisoner’s dilemma embodies the essence of the problem of human cooperation. They therefore set themselves the hopeless task of giving reasons why [this analysis] is mistaken... Rational players don’t cooperate in the prisoner’s dilemma because the conditions necessary for rational cooperation are absent”.

2.1.6 Other Symmetric 2×2 Interactions

The prisoner’s dilemma and its variation are not the only type of multiagent interaction that exists.

In fact, if we restrict our attention to interaction in which there are:

- Two agents
- Each agent has two possible actions (C or D)
- The scenario is symmetric

Then 4! = 24 possible orderings of preferences, and as a consequence, 24 different games, can be constructed.

Scenario	**Preferences over outcomes**	**Comment**
1	$(C,C) \succ_i (C,D) \succ_i (D,C) \succ_i (D,D)$	cooperation dominates
2	$(C,C) \succ_i (C,D) \succ_i (D,D) \succ_i (D,C)$	cooperation dominates
3	$(C,C) \succ_i (D,C) \succ_i (C,D) \succ_i (D,D)$	
4	$(C,C) \succ_i (D,C) \succ_i (D,D) \succ_i (C,D)$	stag hunt
5	$(C,C) \succ_i (D,D) \succ_i (C,D) \succ_i (D,C)$	
6	$(C,C) \succ_i (D,D) \succ_i (D,C) \succ_i (C,D)$	
7	$(C,D) \succ_i (C,C) \succ_i (D,C) \succ_i (D,D)$	
8	$(C,D) \succ_i (C,C) \succ_i (D,D) \succ_i (D,C)$	
9	$(C,D) \succ_i (D,C) \succ_i (C,C) \succ_i (D,D)$	
10	$(C,D) \succ_i (D,C) \succ_i (D,D) \succ_i (C,C)$	
11	$(C,D) \succ_i (D,D) \succ_i (C,C) \succ_i (D,C)$	
12	$(C,D) \succ_i (D,D) \succ_i (D,C) \succ_i (C,C)$	
13	$(D,C) \succ_i (C,C) \succ_i (C,D) \succ_i (D,D)$	game of chicken
14	$(D,C) \succ_i (C,C) \succ_i (D,D) \succ_i (C,D)$	prisoner’s dilemma
15	$(D,C) \succ_i (C,D) \succ_i (C,C) \succ_i (D,D)$	
16	$(D,C) \succ_i (C,D) \succ_i (D,D) \succ_i (C,C)$	
17	$(D,C) \succ_i (D,D) \succ_i (C,C) \succ_i (C,D)$	
18	$(D,C) \succ_i (D,D) \succ_i (C,D) \succ_i (C,C)$	
19	$(D,D) \succ_i (C,C) \succ_i (C,D) \succ_i (D,C)$	
20	$(D,D) \succ_i (C,C) \succ_i (D,C) \succ_i (C,D)$	
21	$(D,D) \succ_i (C,D) \succ_i (C,C) \succ_i (D,C)$	
22	$(D,D) \succ_i (C,D) \succ_i (D,C) \succ_i (C,C)$	
23	$(D,D) \succ_i (D,C) \succ_i (C,C) \succ_i (C,D)$	defection dominates
24	$(D,D) \succ_i (D,C) \succ_i (C,D) \succ_i (C,C)$	defection dominates

For the sake of the analysis we will briefly look into two more of these games: stag hunt and the game of chicken.

- The stag hunt

	i defects	i cooperates
j defects	1 1	0 2
j cooperates	2 0	3 3

- The Game of chicken

	i defects	i cooperates
j defects	0 0	1 3
j cooperates	3 1	2 2

2.2 Voting

So far we have looked at the general setting of a multiagent encounter. In this section we will look at a specific class of protocols intended for making group decisions. This is the domain of social choice theory (aka voting theory).

2.2.1 Social Welfare Functions and Social Choice Functions

The general setting of a voting protocol is as follows:

- A set of agents or **voters** voters

  $$Ag = \{1, ..., n\}$$

  Ideally, this should be finite and the cardinality should be an odd number to avoid ties.
- A set of possible outcomes or **candidates** candidates

  $$\Omega = \{\omega_q, \omega_w, ...\}$$

  over which voters will make group decision. This set is assumed to be finite.
  The goal of the agents is to rank or order these candidates or simply to choose one from the set.

The problem that social choice theory tries to solve is,
"given a collection of preference orders, one for each agent, how do we combine these to derive a group decision?".

The answer is by using a **social welfare function** which maps the voter preferences to a **social preference order** social welfare function social preference order

$$f: \quad \Pi(\Omega) \times ... \times \Pi(\Omega) \rightarrow \Pi(\Omega)$$

The social preference ordering will be denoted with the notation $\succ^*$.

If the agents are required to choose just one candidate over the set Ω, we will use a **social choice function** or **voting procedure**: social choice function voting procedure

$$f: \quad \Pi(\Omega) \times ... \times \Pi(\Omega) \rightarrow \Omega$$

2.2.2 Voting Protocols

Simple majority voting

- Every voter submits their preference order
- The winner is the outcome that appears first in the preference orders the largest number of times
- This is generally applied to single choice, but it can be generalized to a social preference ordering.

Simple majority voting works relatively well with only two candidates, since it is straightforward to implement and understand by the voters.

Simple majority voting

However, when more than 2 voters are involved problems start to arise.

Let us consider that three voters submitted the following preferences:

$$\begin{array}{ll} 42\% & \omega_L \succ \omega_D \succ \omega_C \\ 14\% & \omega_D \succ \omega_L \succ \omega_C \\ 44\% & \omega_C \succ \omega_D \succ \omega_L \end{array}$$

Then the winner is selected based on the top preferences (i.e. C), even though the majority of the voters considered C as the least preferrable outcome.

This means that in principle the majority of the voters will be unhappy with the result of the voting protocol.

Furthermore, this simple majority protocol is sensible to insincere strategical voting (e.g. the 14% that voted D instead could have voted for L just to make C lose the voting).

Moreover, the simple majority voting is sensible to **Condorcet's paradox**:

Condorcet's paradox

$$\begin{gathered} \omega_1 \succ_i \omega_2 \succ_i \omega_3 \\ \omega_3 \succ_j \omega_1 \succ_j \omega_2 \\ \omega_2 \succ_k \omega_3 \succ_k \omega_1 \end{gathered}$$

In this case the winner is dependent on the agenda: which order we start to compute the outcome, but nonetheless $^2/_3$ of the voters will prefer another candidate.

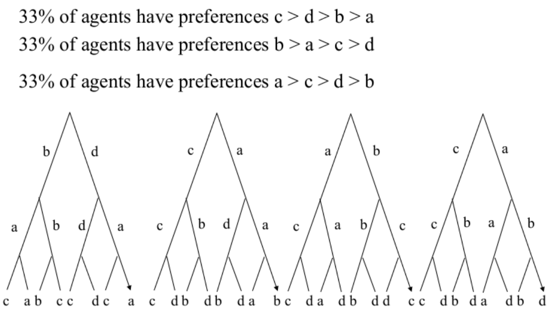
Binary protocol

Among the alternative to simple majority voting there is the **Binary protocol** (aka sequential majority elections).

Binary protocol

- Voters submit their preference ordering
- The element of the set of outcomes are compared pairwise
- the winner of the pairwise "election" is allowed to continue to the next pairwise election
- The order of pairwise elections (i.e. which candidates to compare first), is called **agenda**

agenda



33% of agents have preferences c > d > b > a
33% of agents have preferences b > a > c > d
33% of agents have preferences a > c > d > b

From the picture provided we can clearly notice that the winner of the overall election depends on the agenda, which imply that the protocol can be manipulated if the preference orderings in known a priori.

Borda protocol

One issue with simple majority voting and the binary protocol is that they completely ignore the information that is enconded in each submitted preference ordering (they only look at the top voted candidates).
The **Borda protocol** addresses such issue:

Borda protocol

- Each voter submit its preference ordering over the candidates
- a number equal to the cardinality of the set of candidates (i.e. $|\Omega|$) is assigned to the highest candidate in some agent's preference list
- a value of $|\Omega| - 1$ is assigned to the second and so on
- a the end we sum up the numeric value associated to each preference ordering and obtain the social choice result

This protocol is fast when compared to the binary protocol in the case of a large amount of candidates.

However it is not **indipendent of irrelevant alternatives**, i.e. removing one candidate will change the end result.

indipendent of irrelevant alternatives

Majority Graph: Condorcet's Winner and Slater ranking

An useful tool to analyse voting procedures or protocols is the **Majority Graph**. It can be considered as a succint representation of the voter preferences:

Majority Graph

- each node in the majority graph is a candidate
- each edge in the majority graph represents a preference (the edge will start from the node most preferred and will terminate in the node less preferred)
- a **possible winner** is a candidate that is the overall winner for at least one agenda. (occurs when there are loops in the majority graph)
- a **Condorcet's winner** is a candidate that is the overall winner for all possible agenda (occurs when there are no loops in the majority graph). In simple terms, is that candidate that will beat all others in a pairwise election.

possible winner

Condorcet's winner

While the overall winner determination is straightforward in the case of a majority graph with no loops (cfr. Condorcet's Winner), it is otherwise in the case of a majority graph with loops.

In that specific scenario, the **slater rule** or **slater ranking** is considered: it, in fact, tries to find a consistent ranking that does not contain any cycle that is as close as possible to the majority graph.

slater rule

slater ranking

In other terms, it tries to "minimize the number of disagreements between the majority graph and the social choice".

In practice, the Slater ranking considers the Condorcet's winners obtained by inverting the least amount of edges in the majority graph.

2.2.3 Desirable Properties for Voting Procedures

1. **Pareto condition**

   Pareto condition

   There is no other outcome that makes one agent better off without making another agent worse off.

   Borda and simple majority voting satisfy this condition.
   Binary protocol does not (the result depends on the agenda).

2. **Condorcet winner condition**

   Condorcet winner condition

   The Condorcet winner should be selected by the social choice function.

   Only Binary protol satisfies this condition.

3. **Indipendence of irrelevant alternatives**

   Indipendence of irrelevant alternatives

   Given a set of candidates of which ω_i and ω_j are part of.
   Let us assume that $\omega_i \succ^* \omega_j$.
   A change in preferences in any other candidate should not change the relative ranking of ω_i and ω_j.

   None of the protocol we have seen satisfies this property.

4. **Dictatorship**

   Dictatorship

   A social welfare function is said to be a dictatorship if for some voter i we have

   $$f(\bar{\omega_1}, \bar{\omega_2}, ...) = \bar{\omega_i}$$

   All protocols we have seen so far are non-dictatorship.

Theorem 1 (Arrow's theorem).

Arrow's theorem

If $|\Omega| > 2$ then the only voting procedure that satisfies Pareto condition, Condercet Winner condition and IIA is dictatorship.

2.2.4 Insincere Voters

- In reality it is seldom that all agent's preferences are known: usually agents have to reveal their preferences.
- Assuming knowledege of the preferences is equivalent to assuming that the agents reveal their preferences truthfully
- If an agent can benefit from insincerely declaring his preferences, it will do so.
- The goal is to generate protocols such that when agents use them according to some stability solution concept (dominant strategy equilibrium) the desirable social outcomes follow
- The strategies are not externally imposed on the agents, but instead each agent uses the strategy that is best for itself

However:

Let each agent i from A have some ordering ϑ_i from Θ which totally characterizes his preferences. A social choice function $f: \quad \Theta \to \Omega$ chooses a social outcome given the agent's orderings.

Theorem 2 (Gibbard-Satterthwaite impossibility theorem).

Gibbard-Satterthwaite impossibility theorem

Let each agent's ordering ϑ_i consists of a preference order $\succ_i$ on Ω.
Let there be no restictions on $\succ_i$ (i.e. each agent may rank the outcomes Ω in any order).
Let $|\Omega| > 2$.
Now, if the social function $f(\cdot)$ is truthfully implementable in a dominant strategy equilibrium (or is not manipulable), then $f(\cdot)$ is dictatorial, i.e. there is some agent who gets one of its most preferred outcome chosen no matter what types the other reveal.

Broadly speaking, the only procedure or mechanism that is immune to strategic manipulation (in the sense of untruthful report of an agent preference) is dictatorship in the case of more than 2 candidates.

There are ways to circumvent the impossibility theorem, by relaxing the assumptions made by the theorem itself.
The individual preferences, in fact, may happen to belong to some restrincted domain, thus invalidating the conditions of the impossibility theorem, and it is known that there are island in the space of agent's preferences for which non-manipulable non-dictatorial protocols can be constructed.

The solution in practice is to make agents precisely internalize the externality by imposing a tax on those agents whose vote changes the outcome. The size of tax is exactly how much its vote lowers the other's utility.
Such solution is known as the **Clark tax algorithm**:

Clark tax algorithm

- Every agent i from A reveals his valuation $v_i^*(g)$ for every possible g (which may be non-truthful)
- The social choice is

  $$g^* = \underset{g}{\text{argmax}} \sum v_i^*(g)$$

- Every agent is levied a tax:

$$tax_i = \sum_{j \neq i} v_j^* \left(\underset{g}{\text{argmax}} \sum_{k \neq i} v_k^*(g) \right) - \sum_{j \neq i} v_j^*(g)$$

It follows that

Theorem 3. *If each agent has quasilinear preferences, then, under the Clarke tax algorithm, each agent's dominant strategy is to reveal his true preferences, i.e.*

$$v_i^*(g) = v_i(g) \qquad \forall g$$

In conclusion, the Clarke tax algorithm:

- the mechanism leads to the socially most preferred g to be chosen
- because of truth telling, the agents need not waste effort in counter speculating each other preference declarations
- participation in the mechanism may only increase an agent's utility
- the mechanism does not mantain budget balance: too much tax is collected
- it is not coalision proof

Other way to circumvent the Impossibilty theorem:

- some fairness can be achieved by choosing the dictator randomly in the protocol
- to use a protcol for which computing an untruthful revelation (that is the better than the truthful one) is prohibitively costly computationally

2.3 Auctions

Auctions are mechanisms used to reach agreeements on one very simple issue: that of how to allocate scarce resources to agents.

It is important to understand that the resource in question is scarce and it is tipically desired by more than one agent (if one of the preconditions is not met than the allocation is trivial and straighforward).

Auctions provide a reasonable and principled way to allocate resources to agents, in particular they are effective at allocating resources efficiently, in the sense of allocating resources to those that value them the most.

2.3.1 Auction parameters for classification

In general terms:

- An auction takes place between an agent known as the **auctioneer** and a collection of agents known as the **bidders**
- The goal of the auction is for the auctioneer to allocate a good to one of the bidders
- The auctioneer desires to maximize the price at which the good is allocated.
  Hence the agent auctioneer will attempt to achieve its desire through the design of an appropriate auction
- The bidders desire to minimize the selling price.
  Hence the bidder agents will attempt to achieve their desires by using an effective strategy

auctioneer

bidders

There are several factors that can affect both the auction protocol and the strategy that agents use:

- The good has a **private value**, **public/common value** or a **correlated value** (i.e. an agent valuation of the good depends partly on private factors and partly on other agents' valuations of it).

private value

public/common value

correlated value

- **winner determination**: who gets the good that the bidders are bidding for and how much do they pay.
  In this setting, there are **first-price auctions** (the price goes to the highest bidder) and **second-price auctions** (the price goes to the highest bidder but it will pay the amount bid by the second highest bid)
- Whether or not the bids made by the agents are known to each other.
  In this setting, we will differentiate between **open-cry** and **sealed-bid**
- The mechanism by which bidding proceeds.
  In this setting, we will differentiate between **one shot**, **ascending auctions** or **descending auctions** (in which the price starts at a **reservation price**: in the case of ascending the reservation price is proposed by the first bidder, whereas in descending the reservation price is proposed by the auctioneer).

second-price auctions

open-cry

sealed-bid

one shot

ascending auctions

descending auctions

reservation price

2.3.2 English auctions

English auctions are: first-price, open cry, ascending auctions.

- The auctioneer sets a reservation price for the good (low price) and the good is allocated to the auctioneer for this amount
- Bids are then invited from agents, who must bid more than the current highest bid. (all agents can see the bid)
- When no agent is willing to raise the bid, then the good is allocated to the agent that has made the current highest bid at the amount of its bid.

The dominant strategy is to bid a small amount more than the current highest bid until the bid price reaches their private valuation and then to withdraw.

English auction's however are sensible to the **winner's curse**: should the winner feel "happy" that they have obtained the good for less than or equal to their private valuation or should they feel worried because no other agent valued the good so highly?

winner's curse

2.3.3 Japanese auctions

Japanese auctions are: open cry, ascending auctions

- The auctioneer sets a reservation price
- Each agent must choose whether or not to be in
- The auctioneer successively increases the price
- After each increase an agent must say if he stays or not. When agent drops out it is irrevocable
- The auction ends when exactly one agent is left
- The winner must buy the good for the current price

2.3.4 Dutch auctions

Dutch auctions are: open-cry, descending auctions:

- The auctioneer sets a reservation price
- The auctioneer then continually lowers the offer price of the good by some small value, until some agent makes a bid for the good
- The good is then allocated to the agent that made the offer

Dutch auctions are also susceptible to the winner's curse.

The dominant strategy is to shade bid a bit below the true willingness to pay.

2.3.5 First-price sealed-bid auctions

First-price sealed-bid auctions are: first-price, sealed-bid, one-shot auctions.

- A single round of proposals is considered
- Bidders submit to the auctioneer a bid for the good

- The good is awarded to the agent that made the highest bid
- The winner pays the price of the highest bid

The dominant strategy for an agent is to bid less than it true valuation. How much less will of course depend on what the other agents bid (hence there is no general solution).

2.3.6 Vickrey auctions

Vickrey auctions are: second-price, sealed-bid auctions.

- There is a single bidding round
- Each bidder submits a single bid.
- The good is awarded to the agent that made the highest bid
- The winning bidder pays the price of the second-highest bid

The dominant strategy is truth telling: a bidder's dominant strategy in a private value Vickrey auction is to bid their true valuation.

As such Vickrey auction are not prone to strategic manipulation, however it makes possible for **antisocial behaviour** to occur: one agent may bid close to the highest bid just to make the winner pay close to the amount that they have bid.

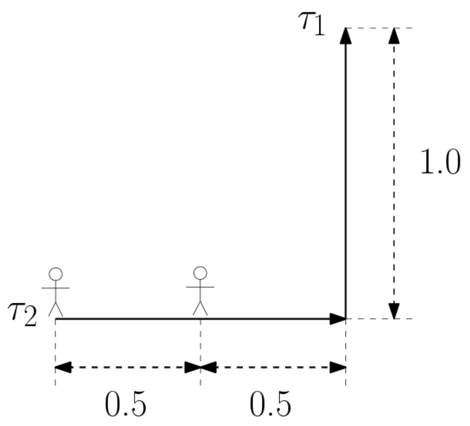
antisocial behaviour

2.3.7 Interralated auctions

Some auctions are interralated to each other:

- Two good are sold separately, but the bidders would like to acquire them together
- In this case the strategy to consider for more than one auction but for several interrelated auctions and then accomodate some valuations and bids according to this strategy but not for some particolar auction.

Let us consider the example of the proposed figure:



Two agents (agent 1 on the left and agent 2 on the right) are concurring for the acquisition of two tasks: τ_1 and τ_2.

It makes no sense for them to acquire just one of the two tasks. Please notice that in order to complete the full task an agent must go to the start of the arrow/task and fully traverse its edge.

Condering all of the above the valuation of the tasks for the two agents can be represented as a cost-to-go:

$$\begin{array}{llll} \text{Agent 1:} & c_1(\tau_1) = 2.0 & c_1(\tau_2) = 1.0 & c_1(\tau_1, \tau_2) = 2.0 \\ \text{Agent 2:} & c_2(\tau_1) = 1.5 & c_2(\tau_2) = 1.5 & c_2(\tau_1, \tau_2) = 2.5 \end{array}$$

However if both agents decided to follow this strategy, task τ_1 would go to agent 2 (lower cost to go), whereas task τ_2 would go to agent 1 (lower cost to go).

In order for agent 1 to win both tasks, since it is of no use for an agent to win just one task in an interrelated auction, it can incorporate the full look ahead in its bid.

If in fact, agent 1 wins τ_1, it will bid $c_1(\tau_2) = c_1(\tau_1, \tau_2) - c_1(\tau_1) = 0$ for the second task, otherwise it will bid $c_1(\tau_2) = 1$. So it makes sense to accomodate the first bid with this knowledge. If agent 1 wins τ_1, it will win τ_2 at the price of 0 and gets a payoff of $c_1(\tau_2) - 0 = 1$ that can redistribute to the first bid.

In conclusion the dominant strategy is to accomodate the payoff into the first bid:

$$c_1(\tau_1) - payoff = 2 - 1 = 1$$

In other terms this implies that if agent 1 realizes that by winning τ_2 it will automatically win τ_1, it will bid "higher" for τ_1 or say that the effort to complete task τ_1 will be lower by winning also τ_2.

2.3.8 Lies and Collusion

It is in the auctioneer interest to have a protocol that is immune to collusion by bidders, i.e. that made it against the bidder' best interests to engage in collusion with other bidders.

Similarly, as a potential bidder in an auction, we would like a protocol that made honesty on the part of the auctioneer the dominant strategy.

None of the auction discussed above is immune to collusion: for any of them the grand coalition of all agents involved in bidding for the good can agree beforehand to collude to put forward artificially low bids for the good on offer.
When the good is obtaines, the bidders can then obtain its true value and split the profits among themselves. A solution to collusion of the bidder is to modify the protocol so that the bidders cannot identify each other which however it's not possible in open cry auctions.

With regard to the honesty of the auctioneer, the main opportunity for lying occurs in the Vickrey auctions: the auctioneer can lie to the winner about the price of the second highest bid. A solution to this is to sign bids in some way so that the winner can independently verify the value of the second highest bid ro use a trusted third party to handle bids.

In open cry auction settings there is no possibility for lying by the auctioneer, nor in the first-price sealed-bid auctions.

Lastly, another possible opportunity for lying by the auctioneer is to place bogus bidders (aka **shills**), in an attempt to artificially inflate the current bidding price.

shills

Moreover, there are main limitation of auctions is that they are only concerned with the allocation of goods and as such are not adequate for settling agreements that concerns maters of mutual interest.

2.4 Negotiation parameters

The basic components that make up a negotiation setting are:

- A **Negotiation set**, which represents the space of possible proposals that agents can make
- A **protocol**, which defines the legal proposals that agents can make, as a funciton of prior negotiation history.
- A collection of **strategies**, one for each agent, which determine what proposals the agents will make.
- A **rule** that determines when a deal has been struck, and what this agreement deal is.

Negotiation set
protocol
strategies
rule

Negotiation usually proceeds in a series of rounds, with some proposal made at every round. The proposals that agents make are defined by their strategy, must be drawn from the negotiation set, and must be legal, as defined by the protocol.
If agreement is reaches, as defined by the agreement rule, then negotiation terminates with the agreement deal.

Several attributes may complicate negotiation:

- Multiple issues are involved: in case of a single attribute/price we have a symmetric scenario which is easy to analyze because it is always obvious what represents a concession.

On the other hand, in multiple-issue negotiation scenarios, agents negotiate over not just the value of a single attribute, but the values of multiple attributes, which may be interrelated. In such scenarios, it is usually much less obvious what represents a true concession.
Moreover multiple attributes also lead to an exponential growth in the space of possible deals. This means that, in attempting to decide what proposal to make next, it will be entirely unfeasible for an agent to explicitly consider every possible deal in domains of moderate size.

- Another source of complexity in negotiation is the number of agents involved in the process, and the way in which these agents interact. There are three obvious possibilities:
  1. **One-to-one negotiation**
  2. **Many-to-one negotiation**, as in the case of auctions or reverse auctions
  3. **Many-to-many negotiation**

One-to-one negotiation

Many-to-one negotiation

Many-to-many negotiation

2.4.1 Task Oriented Domain

The idea behind Negotiation for task allocation is that agents who have tasks to carry out may be able to benefit by reorganizing the distribution of tasks among themselves; but this raises the issue of how to reach agreement on who will do which tasks.

Formally this scenario is known under the name of **Task Oriented Domain (TOD)**. A task oriented domain is a triple:

Task Oriented Domain (TOD)

$$< T, Ag, c >$$

where

- T is the finite set of all possible tasks
- $Ag = \{1, ..., n\}$ is the finite set of negotiation participant agents
- $c : 2^T \rightarrow \mathbb{R}_+$ is a function which defines the cost of executing each subset of tasks: the cost of executing any set of tasks is a positive real number

The cost functino must satisfy two constraints:

1. It must be **monotonic**

monotonic

$$\text{If } T_1, T_2 \subseteq T \text{ are sets of tasks such that } T_1 \subseteq T_2, thenc(T_1) \leq c(T_2)$$

2. The cost of doing nothing is zero

$$c(0) = 0$$

We will restrict our attention as follows:

- One-to-one negotiation scenario, with two agents $\{1, 2\}$
- Given an encounter $< T_1, T_2 >$, a **deal** will be an allocation of the tasks $T_1 \cup T_2$ to the agents 1 and 2.

deal

- Three types of deal may happen under this conditions:
  1. **Pure deals**: agents are deterministically allocated exhaustive disjoint task set. Formally, a pure deal is a pair $< D_1, D_2 >$ where

Pure deals

$$D_1 \cup D_2 = T_1 \cup T_2$$

  2. **Mixed deals**: specify a probability distribution over partitions

Mixed deals

  3. **All-or-Nothing deals**: mixed deals where the alternatives only include partitions where one agent handles the tasks of all agents.

All-or-Nothing deals

- The **cost** to an agent i of a deal $\delta =< D_1, D_2 >$ is defined to be $c(D_i)$ and it will be denoted as $cost_i(\delta)$

cost

- The **utility** of a deal δ to an agent i is the difference between the cost of agent i doing the tasks T_i that it was originally assigned in the encounter and the cost $cost_i(\delta)$ of the tasks it is assigned in δ:

utility

$$utility_i(\delta) = c(T_i) - cost_i(\delta)$$

Thus the utility of a deal represents how much the agent has to gain from the deal (a negative utility means that the agent is worse off than it was originally)

- If the agents fail to reach an agreement they must perform the tasks that they were originally allocated (**conflict deal**, $\Theta =< T_1, T_2 >$)

conflict deal

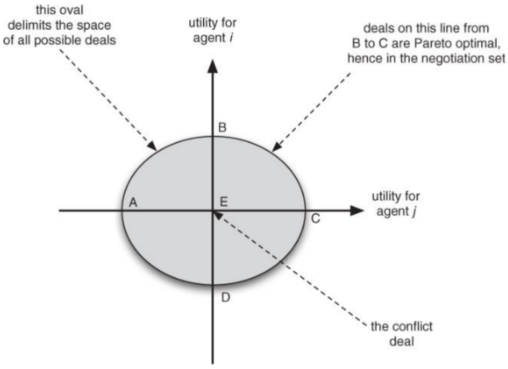
The properties that govern this agent interactions are:

- Dominance
  A deal δ_1 is said to be dominant if and only if:
  1. Deal δ_1 is at least as good for every agent as δ_2
  $$\forall i \in \{1, 2\}, utility_i(\delta_1) \geq utility_i(\delta_2)$$
  2. Deal δ_1 is better for some agent than δ_2
  $$\exists i \in \{utility_i(\delta_1) > utility_i(\delta_2)\}$$
  A deal is said to be **weakly dominant** if at least the first condition holds.

weakly dominant

- Pareto optimal
  A deal δ is Pareto optimal if there is no deal δ' such that $\delta' \succ \delta$
- Individual rationality
  A deal δ is said to be individual rational if it weakly dominates the conflict deal
  $$\delta \succeq \Theta$$
  If a deal is not individual rational, then at least one agent can do better by simply performing the tasks it was originally allocated (it prefers the conflict deal)

Given all of the above, we are in the position to define the space of possible proposals that agents can make: the negotiation set consists of the set of deals that are individual rational and Pareto optimal.
In the following figure is proposed the set of possible deals in a 2 agent encounter


this oval delimits the space of all possible deals
utility for agent *i*
deals on this line from B to C are Pareto optimal, hence in the negotiation set
B
A
E
C
utility for agent *j*
D
the conflict deal

Here, the first, third and fourth quadrant includes the subset of deals that are not individual rational. (there is no point for an agent to accept a deal that produces negative utility).
Similarly, all deals strictly inside the ellipsis are not Pareto Optimal. Which brings as to the boundary of the negotiation set in the second quadrant.
Agent i (ordinate axis) will start at its best possible deal (B), whereas agent j (abscissa axis) will start at its best possible deal (C).
Both agent will then concede in one or more rounds until a point of mutual agreement on the boundary of the negotiation set is reached.

Task Oriented Domains assume that agents have symmetric cost functions (i.e. $c_i(T') = c_j(T')$) and that every agent is capable of handling tasks of all agents.
However, there exists other subtypes of Task oriented domains that consider different relations in the agents cost functions:

- **Subadditive TODs (STODs)** are TODs where:
  $$c_i(T' \cup T'') \leq c_i(T') + c_i(T'')$$

Subadditive TODs (STODs)

- **Concave TODs (CTODs)** are subadditive TODs where:

Concave TODs (CTODs)

$$c_i(T' \cup T'') - c_i(T') \geq c_i(T'' \cup T''') - c_i(T'') \qquad \text{and} \qquad T' \subset T''$$

- **Modular TODs (MTODs)** are concave TODs where:

Modular TODs (MTODs)

$$c_i(T' \cup T'') \leq c_i(T') + c_i(T'') - c_i(T' \cap T'')$$

	TOD			STOD			CTOD			MTOD		
	Hid	Pha	Dec	Hid	Pha	Dec	Hid	Pha	Dec	Hid	Pha	Dec
Pure	L	L	L	L	L	L	L	L	L	L		
Mixed	L		L	L		L	L			L		
All-or-nothing						L						

Table 2.1: The L states that lying is profitable and possible

2.4.2 Worth Oriented Domain

Worth Oriented domains are domains where agents assign a worth to each potential state (of the environment), which captures its desirability for the agent.

Worth Oriented domains

- agent's goal is to bring about the state of the environment with highest value
- We assume that the collection of agents has available set of joint plans (a joint plan is executed by several different agents). In other terms there is no possibility to complete a task alone.

Contrary to TOD, where tasks could have been achieved separately by both agents, in worth oriented domain the outcome can be achieved only by joint efforts (e.g. buying and selling are part of the worth oriented domain).

Formally, Worth oriented domains can be defined as a tuple

$$< E, Ag, J, c >$$

where:

- E: is the set of possible environment states/outcomes
- Ag: is the set of possible agents
- J: is the set of possible joint plans
- $c(j, i)$: is the cost for agent i of executing the plan j

Moreover, we will denote as $W(e, i)$ the value of worth of agent i in state e.

Unlike, TODs, agents negotiating over Worth Oriented Domains are not negotiating about a single issue:

- They are negotiating over both the state that they wish to bring about (which has a different value for different agent)
- They are negotiating over the means by which they will reach the state

Since this domain deals with multiple set of attributes, we are interested to find the Pareto Optimal solution. In order to do so we calculate the utility by:

- Weighting each attribute
- Rating or ranking each attribute value
- Using constraints on an attribute

The negotiation process will proceed to in rounds where each agent concedes, ultimately reaching an agreement or finding no agreement.

If the utility graphs representing the rating of each agent intersects at some point, then a point of acceptance is reached, otherwise the negotiation terminates in conflict.

2.4.3 Monotonic Concession Protocol (MCP)

The rules of this negotiation protocol are:

- Negotiation proceeds in a series of rounds
- On the first round both agents simultaneously propose a deal from the negotiation set
- An agreement is reached if the two agents propose deals δ_1 and δ_2, respectively, such that either:
$$utility_1(\delta_2) \geq utility_1(\delta_1) \qquad || \qquad utility_2(\delta_1) \geq utility_2(\delta_2)$$
- If both agents offers match or exceed those of the other agent, then one of the proposals is selected at random
- If only one proposal exceeds or matches the other's proposal, then this is the agreement deal
- If no agreement is reached, then negotiation proceeds to another round of simultaneous proposals, where no agent is allowed to make a proposal that is less preferred by the other agent than the deal it proposed at the previous round
- If neither agent makes a concession in some round, then negotiation terminates with the conflict deal

Using such protocol negotiation is guaranteed to end (with or without agreement) after a finite number of rounds, however the protocol does not guarantee that the agreement (or lack of it) will be reached quickly.

2.4.4 The Zeuthen strategy

1. What should an agent's first proposal be?
2. On any given round, who should concede?
3. If an agent concedes, then how much should it concede?

With regard to the first question: an agent's first proposal should be its most preferred deal.

With respect to the second question: the idea of the **Zeuthen strategy** is to measure an agent's willingness to risk conflict (i.e. an agent is more willing to risk conflict if the difference in utility between its current proposal and the conflict deal is low).

Zeuthen strategy

Agent i's willingness to risk conflict at round t is:

$$risk_i^t = \frac{\text{utilty } i \text{ loses by conceding and accepting} j\text{`s offer}}{\text{utility } i \text{loses by not conceding and causing conflict}}$$

Until an agreement is reached, the value of risk is between 0 and 1 (the higher the value the higher the risk, which means that an agent has less to lose from conflict and as such it is more willing to risk conflict).
Formally:

$$risk_i^t = \begin{cases} 1 & \text{If } utility_i(\delta_i^t) = 0 \\ \dfrac{utility_i(\delta_i^t) - utility_i(\delta_j^t)}{utility_i(\delta_i^t)} & \text{otherwise} \end{cases}$$

Hence, the zeuthen strategy proposes that the agent to concede on round t of negotiation should be the one with the smaller value of risk.
Moreover, the agent in question should make the smallest concession necessary to change the balance of risk, so that on the next round, the other agent will concede.

Lastly, in the case of equal risk, a coin flip will decide which agent should concede, otherwise we are in the same situation of the prisoner's dilemma.

We notice that the Zeuthen strategy and the protocol itself:

- Does not guarantee success, but it does guarantee termination
- It does not guarantee to maximize social welfare
- If agreement is reached, then this agreement will be Pareto Optimal
- It is individual rational
- It is in Nash equilibrium (if an agent is using the Zeuthen strategy the other can do no better than using the same strategy)

2.4.5 Deception

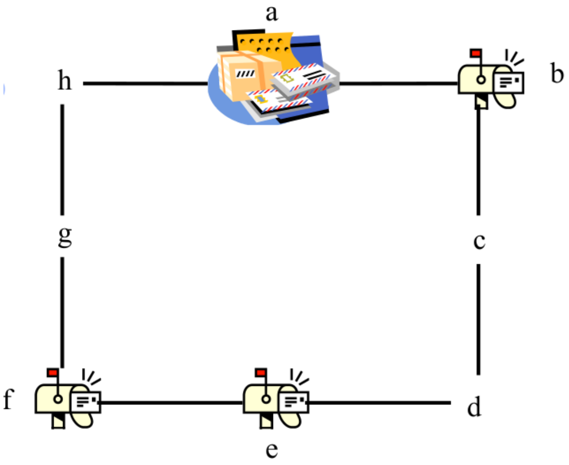
There are three obvious ways in which an agent can be deceitful in Task oriented domain:

- **Phantom tasks**
  Phantom tasks
  An agent can pretend to have been allocated a task that it has not been allocated.
  An obvious respoinse to these is to ensure that the tasks an agent has been assigned to carry out are verifiable by all negotiation participants.
- **Decoy tasks**
  Decoy tasks
  An agent can prodice an artificial task when asked for it.
  Detection of decoy tasks is esssentially impossible, mmaking it hard to be sure that deception will occur in such domains.
- **Hidden tasks**
  Hidden tasks
  An agent may benefit from deception by hiding tasks that it has to perform

Using mixed deals or all-or-nothing deals helps to get rid of deceptions:

- probability is assigned to each Agent task
- all or nothing deal is applied
- weighted coin (with probabilities) is used
- In case of phantom and decoy tasks, the deceiving agent can have a bigger probability to be assigned all the tasks, hence it has no advantage in lying
- However, it is possible that decoy tasks do not increase probability to be allocated all the tasks in some topology

This cas aspect can be further proven with an example: Two postmen need, for some obscure reason to redistribute their letters:



- Agent i, has to deliver letter b and f
- Agent j, has to deliver a letter to e

Agent i decides to deceive the other agent and hide the letter b.

However, if the two agents use a weighted coin and a all-or-nothing deal to reallocate their task, deceiving becomes disadvantegeous from a probabilistic standpoint. In order to understand why let us consider a simple model of the probability based on the edges that an agent has to traverse:

- Agent i (the deceiving one) have an expected cost that has the following form:

  $$\mathit{expected\,cost} = {}^{3}/_{8} \cdot 8 + {}^{5}/_{8} \cdot 2 = 4.25$$

  The expected cost components are derived in the followin way:

  - Compute the probabilities based on the claimed letters to deliver.

    Agent i claims to have only the letter f to deliver which is at a distance of 3 edges from the post office. Since there are a total of 8 edges, the associated probability is 3/8. As a result, the probability of winning the coin flip is $1 - {}^3\!/_8 = {}^5\!/_8$

  - Compute the cost of delivering the letter (which is the number of edges forward and back to traverse).
    In the case of agent i, losing the coin flip will imply that he has to deliver all the letters (all-or-nothing deals). The shortest path to do so implies traversing all of the 8 edges of the graph, hence the associated cost is 8.
    On the contrary if agent i wins the weighted coin flip, it has still to deliver the hidden letter b which would require him to traverse 1 edge forward and back, resulting in a total cost of 2.

- From the expected cost result we can then compute the utility that agent i obtained by hiding the letter: this value is compute as the difference of traversing the whole graph with the expected cost:
  $$utility = 8 - 4.25 = 3.75$$
- We can notice that if agent i would have not hidden the letter, it would have gotten a coin flip probability of ${}^4\!/_8 = 0.5$ since the sum of the edges that he would have had to traverse is 3 for the letter f and 1 for the letter b. Whereas the cost of winning the coin flip would have been 0 (no letters to deliver), and the cost of losing the coin flip is 8 (the postman has to deliver all the letters which means that it has to traverse the whole graph).
  Hence the expected cost of not lying would have been:
  $$expected\,cost = 0.5 \cdot 8 + 0.5 \cdot 0 = 4$$
  and as a result the utility would have been:
  $$utility = 8 - 4 = 4$$
  which is higher than the utility of hiding the letter b

2.5 Contract Net Protocol (CNP)

Cooperating experts metaphor:

- work independently
- exchange results
- ask help when enable to solve individually

The assumptions that CNP makes is that

- If the problem is too large then partition it and find other experts who can solve the task
- If the problem is outside of expertise then also findanother expert
- If expert is known then contact them directly, otherwise describe the task to the entire group of agents
- If somebody from the group agree then the agent will notifies it
- If more then one agree then choose one

The basic premise of the CNP is that, if an agent cannot solve an assigned problem using local resources/expertise, it will decompose the problem into subproblems and try to find other willing agents with the necessary resource/expertise to solve these subproblems

In this sense, each agent can take one of two roles: **manager** or **contractor**.

manager

contractor

The overall contracting mechanism goes as follows:

1. contract announcement by the manager agent

2. submission of bids by contracting agents in response to the announcement

3. evaluation of submitted bids by the manager

4. awarding a subproblem contract to the contractor with the most appropriate bids

Contrary to auctions, there is no limitations in respecting the bid that agent have made: this means that even though an agent wins a bid and accept a given task it is not forced to take it.

Formally, Let us consider:

- τ_i^t the set of tasks allocated to the agent i at time t
- e_i the resources available to i
- $\tau(ts)$ the announce task

The **marginal cost** proposed by Sandholm, takes the form:

marginal cost

$$\mu_i(\tau(ts)|\tau_i^t) = c_i(\tau(ts) \cup \tau_i^t) - c_i(\tau_i^t)$$

Hence the decision to bid happens if and only if

$$\mu(\tau(ts)|\tau_i^t) < (e(ts) + e_i)$$

where $e(ts)$ is reward for doing the new task

The problems with CNP are:

1. it does not detect conflicts

2. it assumes benevolent and non-antagonistic agents

3. it is still rather communication intensive

Chapter 3

Agent Communication

Communication has been a topic of extensive study in the field of computer science. Particularly, the problem of **synchronization** of multiple processes was of key interest in the 1970s and 1980s.

synchronization

This is of importance since processes have fundamentally the same behaviour of agents. In fact, the object oriented paradigm would require **communication via method invocation** to allow an object to message another and change its internal state (this is not allowed in an agent-oriented setting).

communication via method invocation

Let us consider, for example, two agents i and j, where i has the capability to perform an action α (similar to a method).
In an agent-oriented world, contrary to object-oriented, there is no such thing as method invocation. This is because both i is an **autonomous agent**, and as such it has control over both its state and behaviour.
Hence, it cannot be taken for granted that agent i will execute action α just because another agent j wants it to: performing the action may not be in the best interests of agent i.

autonomous agent

In general, agents can neither force other agents to perform some action, nor write data onto the internal state of other agents, but this does not mean that they cannot communicate.
What agents can do is perform **communicative actions** in an attempt to **influence** other agents.

communicative actions

influence

3.1 Approaches to software interoperation

We intiially consider **software interoperation**, i.e. the exchange of information and services with other programs, thereby solving problems that cannot be solved alone.
This is fundamentally different from communication (interaction between humans).

software interoperation

The main problem related with software interoperation is **heterogeneity**, in the sense that two or more software in need to interoperate might be written by different people, at different times and in different languages.

heterogeneity

Hence, in order for them to communicate successfully it should be provided:

- a standard communication language
- common libraries
- run time support

And particularly, we would like to answer the following questions:

- What is an appropriate agent communication language?
- How do we build agents capable of communicating in this language?
- What communication architectures are conductive to cooperation?

3.1.1 Components of a system for effective interaction and interoperability

The basic components of a system to achieve effective software interaction adn interoperability are:

- a common language
- a common understanding of the knowledge exchanged (common understanding of the components of the language as well as their meaning)
- the ability to exchange whatever is included in the previous items.

The reason why we cannot use natural language as a common language for agent communication is that it is not easy to develop system that understands it completely. Moreover, the use of other communication protocols such as xml or json it is not ideal since agents have conversations (as opposed to exchange of single messages).

In addition to this the communication behaviour should allow to express agents strategies, intentions, roles, ..., i.e. concepts that are high level than can be expressed in low level languages.

3.1.2 Three Important Aspects

There are three important aspects in communication:

1. **Syntax**: how the symbols of communication are structured. Syntax

   It includes the grammatical rules, how we write and so on.

2. **Semantics**: what the symbols denote Semantics

3. **Pragmatics**: How the symbols are interpreted, what is the context of the communication. Pragmatics
   The same sentence can be interpreted in different ways.

The combination of semantics and pragmatics is the **meaning** of the communication. meaning

From the above, we deduce that an agent communication language (ACL) should be:

1. Syntactic: should allow syntactic translation between the agents

2. Meaning: should allow meaning content preservation among applications.

3. Communication: should be able to communicate complex attributes about their information and knowledge.

Hence what distinguish ACLs from other languages is:

- Semantic complexity
- ACLs can handle propositions, rules and actions instead of simple objects with no semantics associated with them. Hence ACL does not deal with a simple exchange of data, but instead with the exchange of information that has a meaning.
- An ACL message describes a desired state in a declarative language, rather than a procedure or a method.

3.2 Speech Acts

Speech acts theory treats communication as action. It is predicated on the assumption that speech actions are performed by agents just like other actions, in the furtherance of their intentions.

Speech acts theory

As such, speech act theories attempt to account for how language is used by people every day to achieve their goals and intentions.

3.2.1 Austin

The theory of speech acts is generally recognized to have begun with the work of the philosopher John Austin. He noted that a certain class of natural language utterances, referred to as **speech acts**, had the characteristics of actions, in the sense that they change the state of the world in a way analogous to physical actions.

speech acts

Austin identified a number of **performative verbs**, which correspond to various different types of speech acts (e.g. request, inform, promise).
In addition, Austin distinguished three different aspects of speech acts:

performative verbs

1. **locutionary act**: act of making an utterance
2. **illocutionary act**: action performed in saying something
3. **perlocution**: effect of the act

locutionary act

illocutionary act

perlocution

Austin referred to the conditions required for the successful completion of performatives as **felicity conditions**, that are as follows:

felicity conditions

- There must be an accepted conventional procedure for the performative, and the circumstances and persons must be as specified in the procedure
- The procedure must be executed correctly and completely
- The act must be sincere and any uptake required must be completed insofar as is possible

We are particularly interested in the illocutionary act.
Formally, an illocutionary act $F(P)$ si composed from:

- **propositional content** P: what is the utterance or message exchanged
- **context**: context of the message
- **illocutionary force** F: what is the intention of the message

propositional content

context

illocutionary force

The illocutionary force divides speech acts into categories described by :

- **Illocutionary point**
- **direction of fit**, is the coveyed message describing the world or changing the world
- **sincerity conditions**

Illocutionary point

direction of fit

sincerity conditions

3.2.2 Searle

Searle extended the work of Austin in his 1969 book *Speech Acts.* Searle identified several properties that must hold for a speech act prefromed between a **HEARER** and a **SPEAKER** to succeed.

HEARER

SPEAKER

1. **Normal I/O Conditions**: the HEARER is able to hear the performative of the speaker and the act was performed in normal circumstances (not in a film or a play).
2. **Preparatory conditions**: state what must be true of the world in order that SPEAKER correctly choose the speech act (in a request to perform ACTION, the HEARER must be able to perform such ACTION and the SPEAKER must believe the HEARER is able to perform such ACTION)
3. **Sincerity conditions**:these conditions distinguish sincere performatives of the request; an insincere performance of the act might occur if SPEAKER did not really want ACTION to be performed.

Normal I/O Conditions

Preparatory conditions

Sincerity conditions

Searle also attempted a systematic classification of possible types of speech acts identifying the followin five key classes:

- **Representatives** (inform): a representative act commits the speaker to the truth of an expressed proposition
- **Directives** (request): a directive is an attempt on the part of the speaker to get the hearer to do something
- **Commissives** (promise): Commit the speaker to a course of action
- **Expressives** (thanking): Express some psychological state such as gratitude
- **Declarations** (declaring war): Effect some changes in an institutional state of affairs

Later some additional speech acts were introduces: **Permissives** and **Prohibitives**

Representatives
Directives
Commissives
Expressives
Declarations
Permissives
Prohibitives

3.2.3 Plan-based theory

In order for an AI to make a plan about how to achieve goals from the interaction with humans or other autonomous agents, such plans must include speech actions. Formally the aim of the application of such mechanism is to develop a theory of speech acts
“... by modelling them in a planning system as operators defined... in terms of speakers’ and hearers’ beliefs and goals. Thus speech acts are treated in the same way as physical actions”.

Cohen and Perrault came up with a formalism called the **STRIPS** notation, in which the properties of an action are characterized via **precondition**s and **postcondition**s.
Cohen and Perrault also demonstrated how the preconditions and postconditions of speech acts could be represented in a multimodal logic containing operators for describing the beliefs, abilities and wants of the participants in the speech act.

STRIPS
precondition
postcondition

In short, in order for a speech act to be successful the preconditions must be fullfilled; however the fulfillment of the preconditions are not enough in itself to guarantee that the desired action will actually be performed. This is because some speech act such as request or inform only models the illocutionary force of the act, not the perlocutionary force.

For this reason, Cohen and Perrault introduced some mediating act: Convince and CauseToWant; which respectively will make the hearer believe an inform act and believe that the speaker of a request act wants the hearer to do an action.

$Request(S, H, \alpha)$		
Precondition	Cando.pr	$(S\ BELIEVE\ (H\ CANDO\ \alpha)) \wedge$
		$(S\ BELIEVE\ (H\ BELIEVE\ (H\ CANDO\ \alpha)))$
	Want.pr	$(S\ BELIEVE\ (S\ WANT\ requestInstance))$
Effect		$(H\ BELIEVE\ (S\ BELIEVE\ (S\ WANT\ \alpha))$
$CauseToWant(A_1, A_2, \alpha)$		
Precondition	Cando.pr	$(A_1\ BELIEVE\ (A_2\ BELIEVE\ (A_2\ WANT\ \alpha)))$
	Want.pr	$\times$
Effect		$(A_1\ BELIEVE\ (A_1\ WANT\ \alpha))$
$Inform(S, H, \phi)$		
Precondition	Cando.pr	$(S\ BELIEVE\ \phi)$
	Want.pr	$(S\ BELIEVE\ (S\ WANT\ informInstance))$
Effect		$(H\ BELIEVE\ (S\ BELIEVE\ \phi))$
$Convince(A_1, A_2, \phi)$		
Precondition	Cando.pr	$(A_1\ BELIEVE\ (A_2\ BELIEVE\ \phi))$
	Want.pr	$\times$
Effect		$(A_1\ BELIEVE\ \phi)$

3.2.4 Speech acts as rational actions

While the plan-based theory of speech acts was a major step forward, it was recognized that a theory of speech acts should be rooted in a more general theory of rational action.
This led Cohen and Levesque to develop a theory in which speech acts were modelled as actinos performed by rational agents in the furtherance of their intentions.
The foundation upon which they built this model of rational action was their theory of intention.

3.3 Agent Communication Languages (ACL)

In the ACL we strive to communicate or exchange the meaning explicitly by saying what is the intention in the language. This is in contract with the human language in which the meaning is hidden by the words used, the intonation, the tone (it is not always explictly said).

There are some fundamental components of the language that may help to achieve a language that explicitly communicate meaning:

- An illocutionary force represented by a performative verb. (e.g. request, inform)
- Propositional content, what the agent said.

This makes it so that different propositional content can be interpreted in different ways by the hearer based on the associated illocutionary force.

The semantic of speech acts can be represented via plan-based theory, which treats speech acts as physical actions. Thus, each action is characterised by preconditions (conditions that must be fulfilled for successful communication) and postconditions (effect).

3.3.1 KSE

Speech act theories have directly informed and influenced a number of languages that have been developed specifically for agent communication.
In the early 1990s, the US-based DARPA-funded Knowledge Sharing Effort (KSE) was formed.

The KSE generated threemain deliverables:

- The **Knowledge Interchange Format (KIF)**.
  This language was explicitly intended to allow the representation of knowledge about some particular domain of discourse (aka the content of the message).

  Knowledge Interchange Format (KIF)
- The **Ontolingua**
  Ontology representation.

  Ontolingua
- The **Knowledge Query and Manipulation Language (KQML)**.
  This language defines an envelope format for messages, but is not concerned with the content part of the messages. In short, it defines the language for both message formatting and message handling protocols

  Knowledge Query and Manipulation Language (KQML)

The Basic assumption that KSE makes is that “Software agents are applications for which ability to communicate with other applications and share knoledge is of primary importance.” In the KSE case, the basic components of an ACL are represented as follows:

- Communication, through a interaction protocol, communication language and a transport protocol
- Representation, through knowledge bases and ontologies (way of representing organization of knowledge, how concept are organized and what is their meaning, set of symbols with their associated meaning)
- Supporting components, through planning activities, modeling other agents and environments, meta-knowledge (how we can reason about what we know) and reasoning

KIF

The Knowledge Interchange Format (KIF) was intended to:

- create a language for development of intelligent applications.
- create a common interchange format that allows to translate from any language to it and vice versa.
- express the contents of a message but not the message itself.

KIF was not intended to:

- to model the interaction with human user
- to be internal representation for knowledge within computer programs.

The interchange format that KIF uses is best described with an example.
Let us assume we have two languages (L_1 and L_2). In this scenario we need to develop two translations in order to have communication between the two agents.
If we have three languages, the number of translations is 6
If we have four languages, the number of translations is 12

We can notice that as the number of languages increases the number of translations necessary increases drastically, for this reason an intermediate language from and to other languages are translated was proposed under the name of KIF. It means the for each language we need to develop only two translations.

KIF is a prefix version of first order predicate calculus.

- The prefix version tells that first we place the operation then the operands
- KIF has a declarative semantics
  you do not need to say how it will be executed in the interpreter but instead you can present your statement in some arbitrary order and then an inference mechanism will find a way in which it can be operated
- KIF is logically comprehensive.
  allows to represent all necessary logic constraints
- KIF provides for representatino of knowledge about representation of knowledge (meta-level)

Additional features are:

- Translatability, easy to represent
- Readability, easy to read
- Usability as a representation language

Using KIF, it is possible to express:

- Properties of things in a domain
- Relationships between things in a domain
- General properties of a domain

In order to do so, KIF has introduced:

- Variables: ordinary variable represented with a ? prefix and list variables represented with a @ prefix
- Operators
  - term operators: e.g. listof, if, ...
  - sentence operators: e.g. not, and, / =, ...
  - rule operators: e.g. =>>, ...
  - definition operators: e.g. defobject, deffunction,...
- Constants
  - object constants
  - function constants

  - relational constants
  - logical constants

- Expressions/Sentences: word or a finite sequence of words

Ontologies

An ontology is a formal explicity specification of a shared conceptualization.

An ontology is a description of the concepts and relationships that can exist for an agent or a community of agents.

The reason why we need ontology is that agents need to have the same understanding of concepts that we use.

An example is the world VISA, which can be both a entering permit or a credit card. We therefore can specify an ontology for traveling and one for finance and distinguish between the two.

Hence, ontology is used to agree on a terminology to describe a domain, and by referencing this ontology we communicate the meaning of the communication.

Ontolingua is one of the first ontologies developed that was used to give notion or meaning of words that we use in KIF.

KQML

KQML is a message-based language for agent communication. Thus KQML defines a common format for messages.

Each KQML object has a **performative** and a number of **parameters**.

performative
parameters

An example of a KQML object is

```
{$performative$
  :content $content$
  :receiver $receiver$
  :language $language$
  :ontology $ontology$
}
```

KQML defines a set of standard performatives to choose from of which different parameters are required. In table 3.2 the list of 41 performatives are proposed with their meaning and in table 3.1 the list of the main parameters of KQML messages are summarized.

Parameter	**Meaning**
`:content`	content of the message
`:force`	whether the sender will ever deny the content of the message
`:reply-with`	whether the sender expects a reply, and, if so, an identifier for the reply
`:in-reply-to`	reference to the `:reply-with` parameter
`:sender`	sender of the message
`:receiver`	intended recipient of the message

Table 3.1: Parameters for KQML messages

To more fully understand these performatives, it is necessary to understand the notion of a **virtual knowledge base (VKB)** as it was used in KQML. The idea was that agents using KQML to communicate may be implemented using different programming languages and paradigms and any information that agents have may be internally represented in many different ways. However, for the purpose of communication, it makes sense for agents to treat other agents as if they had some internal representation of knowledge.

virtual knowledge base (VKB)

This attributed knoledge is known as the virtual knowledge base.

Despite the initial success, KQML was subsequently criticized on a number of grounds:

- The basic KQML performative set was rather fluid. It was never tightly constrained, and so different implementatinos of KQML were developed that could not, in fact, interoperate

- Transport mechanisms for KQML messages were never precisely defined
- The semantics of KQML was never rigorously defined. this is because the meaning of KQML performatives was only defined using informal, English language descriptions, open to different interpretations (it was impossible to tell whether to agents were using KQML language properly)
- The lenguage was missing an entire class of performatives, such as commissives (which are a requirement for agent coordination)
- The performative set for KQML was overly large and, it could be argued, rather ad hoc.

In an environment it is good to have language, but we need some supporting elements which are **Facilitators**. Facilitators are used to route the message in case an agent does not know exactly the address of the receiver.
In this sense, facilitators help to make communication protocol transparent.

Facilitators

We can think of facilitators as a special class of agents that perform useful communication services such as:

- Maintain registry of service names
- Forward messages to named services
- Routing messages based on content (not the content of the message but rather ontology and language)
- Provide matchmaking betweeen information providers and requesters
- Provide mediation and translation services
- ....

In particular KQML defines a standard protocol to be implemented by facilitator:

- **Point-to-Point protocol**
  Point-to-Point protocol
  If A is aware about B then it is appropriate to send query about X to B.
  In other terms if agent A knows exactly agent B then it is allowed to reach and communicate with it directly
- **Subscribe performative**
  Subscribe performative
  Request that Facilitator F helps find the truth of X. If B subsequently informs F that it believes X to be true, then F can in turn inform A
- **broker performative**
  broker performative
  A asks Facilitator to find another agent (not F) which can process a given performative. (agent A does not ask value of X, but rather someone who can process X)
- **recruit performative**
  recruit performative
  Asks Facilitator to find an appropriate agent to which an embedded performative can be forwarded. A reply is returned directly to the original agent
- **recommend performative**
  recommend performative
  Asks Facilitator to respond with the name of another agent which is appropriate for sending a particular performative

Another element of interest of KQML is its internal structure.
The communication architecture is built around three main elements:

1. facilitators.
   Facilitators are agents with own KQML routers, they are no different (in implementation) from other agents. The only things that makes them different from regular agents is that they have some predefined functionality about a particular facilitating communication.
   Tipically one facilitator for each local group of agents.

2. routers.
   Content independent message routers.
   Each KQML agent is associated with its own separate router process.
   router handles all KQML messages going to and from the associated agent.
   can try to find Internet address for service and deliver message to it.

3. Library of interfaces (KRIL)

Router Interface Library (KRIL)

Router Interface Library (KRIL)

It is a programming interface between router and agent.
It is embedded into application
The main role of KRIL to make access to the router as simple as possible for the programmer API are developed for different languages and then these API can be used directly in agent in order to make communication, tipically with primitives to send message and it defines some point of listening in which messages from other agents will be accomodated and processed.

In general, KQML allows us to implement the system via some independent routing mechanism without involving a particular agent in the knowledge of such mechanism.

It says that we can provide routers that can route network between different engines and in order to communicate with routers we provide some predefined interface that one can embed in you agent.

3.3.2 FIPA ACL

In 1995, the **Foundation for Intelligent Physical Agents (FIPA)** began its work on developing standard for agent systems. The centerpiece of this initiative was the development of an agent communication language (ACL).

Foundation for Intelligent Physical Agents (FIPA)

The FIPA ACL is superficially similar to KQML, since:

- It defines an outer language for messages
- It defines 20 performatives for defining the intended interpretation of messages,
- it does not mandate any specific language for message content

In additino the concrete syntax for FIPA ACL messages closely resembles that of KQML

```
{$performative$
  :sender $sender$
  :receiver $receiver$
  :content $content$
  :language $language$
  :ontology $ontology$
}
```

Hence the structure of messages is the same and the message attribute fields are also very similar, however the most important difference between the two languages is the collection of performatives they provide.

FIPA provides some standardization of agent interaction protocols.
Ongoing conversations between agents fall into typical patterns. In such cases, certain message sequences are expected and at any point in the conversation other messages are expected to follow. These typical patterns of message exchange are called protocols.

FIPA does not allow a protcol to be up to interpretation, for this reason they provide some standardized version of each multiagent interactions such as actions or voting or others and specify exactly how these interactions should be carried on.

Performative	**Meaning**
`achieve`	S wants R to make something true of their environment
`advertise`	S claims to be suited to processing a performative
`ask-about`	S wants all relevant sentences in R's VKB
`ask-all`	S wants all of R's answers to a question C
`ask-if`	S wants to know whether the answer to C is in R's VKB
`ask-one`	S wants one of R's answers to question C
`break`	S wants R to break an established pipe
`broadcast`	S wants R to send a performative over all connections
`broker-all`	S wants R to collect all responses to a performative
`broker-one`	S wants R to get help in responding to a performative
`deny`	the embedded performative does not apply to S (any more)
`delete-all`	S wants R to remove all sentences matching C from its VKB
`delete-one`	S wants R to remove one sentence matching C from its VKB
`discard`	S will not want R's remaining responses to a query
`eos`	end of a stream response to an earlier query
`error`	S considers R's earlier message to be malformed
`evaluate`	S wants R to evaluate (simplify)C
`forward`	S wants R to forward a message to another agent
`generator`	same as `standby` of a `stream-all`
`insert`	S asks R to add content to its VKB
`monitor`	S wants updates to R's response to a stream-all
`next`	S wants R's next response to a previously streamed performative
`pipe`	S wants R to route all further performatives to another agent
`ready`	S is ready to respond to R's previously mentioned performative
`recommend-all`	S wants all names of agents who can respond to C
`recommend-one`	S wants the name of an agent who can respond to a C
`recruit-all`	S wants R to get all suitable agents to respond to C
`recruit-one`	S wants R to get one suitable agent to respond to C
`register`	S can deliver performatives to some named agent
`reply`	communicates an expected reply
`rest`	S wants R's remaining responses to a previously named performative
`sorry`	S cannot provide a more informative reply
`standby`	S wants R to be ready to respond to a performative
`stream-about`	multiple response version of ask-about
`stream-all`	multiple response version of ask-all
`subscribe`	S wants updates to R's response to a performative
`tell`	S claims to R that C is in S's VKB
`transport-address`	S associates symbolic name with transport address
`unregister`	the deny of a register
`untell`	S claims to R that C is not in S's VKB

Table 3.2: Performatives for KQML messages

Performative	Passing Information	Requesting Information	Negotiation	Performing actions	Error handling
accept-proposal			×		
agree				×	
cancel		×		×	
cfp			×		
confirm	×				
disconfirm	×				
failure					×
inform	×				
inform-if	×				
inform-ref	×				
not-understood					×
propagate				×	
propose		×			
proxy				×	
query-if		×			
query-ref		×			
refuse				×	
reject-proposal			×		
request				×	
request-when				×	
request-whenever				×	
subscribe		×			

Chapter 4

Agent Coordination

Coordination is “the process by which an agent reasons about its local actions and the anticipated actions of other to try and esure that the community acts in a coherent manner.”

4.1 General models

In some sense coordination is the normal way of operating MASs, in which agents can plan their activities based on what they perceive around them. However, during this process agents can incur into some conflicts that are solved via Negotiation.
As a consequence, Coordination in the normal behaviour of the agents, whereas Negotiation is exception handling.

4.1.1 Motivation

The reasons why we need cooperation between agents is for:

- Preventing anarchy or chaos.
  If agents do not coordinate their behaviour than cause will arise in the system quite quickly
- Meeting global contraints.
  In particular, if a budget is given to some agents when reasoning about a project, there is a budget constraint that needs to be satisfied.
- Distributed expertise, resources or information.
  Because in many cases we have some sofisticated expertise by different actors and we would like them to work them together.
- Dependencies between agents' actions.
- Efficiency
  The exchange of information with other agents may increase the efficiency of the solution.

4.1.2 Coordination properties and mechanisms

Each coordination mechanism needs to ensure:

- **Coverage**. Coverage
  Given a common area of operation, all the portions of the problem are included into the activity of at least one agent.
- **Connectivity**. Connectivity
  Agents interact in a way that allow their activity to be integrated into the final overall solution.
- **Rationality**. Rationality
  Team members act in purpose in a consistent way.
- **Capability**. Capability
  All objectives must be achievable within available computational and resource limitations.

Moreover the activities that the mechanism needs, while operating, to perform is to:

- supply timely information to needly agents
- ensure synchronization
- avoid redundant problem solving

The question ramains on how a designer achieve coherent behaviour of the system and in particular of each agents. There are three approaches to achieve coherent behaviour:

- **complete knowledge**. complete knowledge
  Each agent knows completely what other agents do and based on this knowledge the agent will reason about its own activity
- **centralized control**. centralized control
  A designated agent will decide about the allocation of tasks and how these tasks can be used and synchronized
- **distributed control**. distributed control
  No agent has overview of tasks and agents reach coordination/coherent behaviour via negotiation or taking into consideration the activity of nearby agents

Fundamental coordination processes:

- **Mutual adjustment**. Mutual adjustment
  Agents will adjust their behaviour based on the behaviour of other agents.
- **Direct supervision**. Direct supervision
  An agent will tell nearby agents to change their behaviour.
- **Standardization**. Standardization
  Standardization of what can be done and what should be avoided, and based on that agents will try to adjust their own behaviour based on some rules.

4.1.3 Coordiantion problem

In order to tackle the coordination problem there are several steps that we must take:

- analysis of a perspective for coordination
    - external observation.
      Based on the behaviour of others each agent adjust its behaviour. However it may be not enough because:
        - behaviour can be coordinated but incoherent.
          An agent might try to reason about the course of actions but it does not interpret other agents' actions correctly.
        - behaviour can be coherent but non-coordinated.
    - examining internal goals and motivations is necessary.
      Which implies that having a model of behaviour of other surrounding agents can help establish coordination in MASs
- applying appropriate coordination model.
  An agent will need to have a general model of coordination that fits all the cases that might occur which will be a purely mathematical approach.

Control decisions and search

View to coordination as a matter of effective control of distributed search.
Almost every problem can be interpreted as a search problem, and it is a process of looking at a sequence of actions that react to a goal or look at some subgoal to reach a solvable goal.

In other terms, in any case an agent has complete knowledge of the current situation, has access to sequence of possible actions and it can try to combine every possible actions to reach a goal.

Thus, control in this sense is:

- control decisions are decisions about what actions to take next
- control decisions are choices
- copntrol knowledge is any knowledge that informs control decisions
- each control choice is the outcome of an overall control regime which includes knowledge about
    - what are the control alternatives? What are the agent choices?
    - What are the decision criteria?
    - what is the decision procedure?

Hence, in principle agents will deal with a search problem and they must have a control procedure over this search. And this control decision will be the underlying coordination mechanism.

Control choices become more complicated to the degree that control decisions have

- numerous choices
- asynchronous behaviour
- decentralized control

4.1.4 DAI and distributed search

In the distributed search problem the space of alternative problem states (goals) can be seen as a large search space investigated by a number of agents. Each agent has to make local control decisions.
These local control decisions have impacts on the overall efforts done by the collection of problem solvers.

The coordination process kickstarts when these local control area between agents overlap.

Two kinds of control in DAI

1. **Network control** or **Cooperative control**:
   Comprises decision procedures that lead to good overall performance
2. **Local control**:
   refers to decision procedures that lead to good local decisions, and that are based on local information only

Network control

Cooperative control

Local control

Network control

Sets contexts for agent's individual control decisions based on network level information (which can be the information the dependencies of two different goals or the intersection between two areas). Network-level information is:

- aggregatred from more than one agent or abstracted from data from more than one agent
- Information that concerns the relationships among a collection of agents

Network-level information can be utilized to influence:

- set of action alternatives to consider a control decisions
- the decision criteria applied to choose one of them
- the control decision procedure

One type of network control (or coordination) is allocation of search-space regions to agents.

- search space, alternative problem states (goals).
- establishing dependencies between nodes.
- search process explores that space by generating a tree of possibilities taking into account dependencies
- since a search process explores a tree of possibilities (and any tree is recursively composed of subtrees) it follows that any region of the search space can be characterized by a set of subtree roots
- allocation of search-space regions to age4nts takes place by allocating collections of search-subtree roots to agents
- dynamic nature of region allocation

Dependencies analysis

Two kinds of dependencies might occur in a tree:

1. **logical dependencies**, an action or goal depends on another action or goal
2. **resource dependencies**, both actions depends on a resource that goal consumes and the other produces.

logical dependencies

resource dependencies

A very important part of coordination is analysis of interdependencies because the behaviour of the agents involved in the coordination process is based on such analysis.

it may occur that some joint goals:

- team members are mutually responsible to one another

- the team members have a joint commitment to the joint activity
- the team members are committed to be mutually supportive of one another

Local control

Concerns the status and process of a single node in its own local environment and its own local search space region

Interaction with network control:

- network may increase or decrease local control uncertainty
- better local control may more efficiently uncover information that can focus network control decision

Control decision uncertainty

Control decision uncertainty: ambiguity in the next actino choice (often is characterized as the size of the set of next-states)
Resucing degree and/or reducing impact of uncertainty will have an

- Impact on arbitrariness of control decisions
- Impact of control uncertainty can be reduced by reducing common dependencies that agent share

Activities in coordination

- Defining the goal graph or search space
- Assigning regions of search space to different agents
- controlling decisions about which areas of the graph to explore
- traversing the goal structure satisfying dependencies
- ensuring report of the successful traversal

Determining the approach for each of the phases is a matter of system design and it depends upon:

- The nature of the domain
- The type of agents included into community
- the desired solution characteristics

4.2 Common coordination techniques

- Organisational structures
- Meta-level Information exchange
- Multi-agent Planning
- Explicity analysis and synchronization
- Norms and social laws
- Coordination models based on human teamwork (Joint commitments and mutual modelling)

4.2.1 Comparing common coordination techniques

The first four common techniques can be compared and classified using three main concepts:

- **Predictability**, how easy is it to predict what happens next
- **Reactivity**, how easy is it to react to change
- **Information Exchange**

Predictability

Reactivity

Information Exchange

Trade off between predictability, reactivity and information exchange.

4.2.2 Organizational structures

- It is the simplest coordination approach.
  Someone allocate responsibility to each agent to some particolar search space
- implicit coordination
- provides framework for defining
  - roles
  - communication paths
  - authority relationships
- pre-defined, long-term relations.
  the relations that you have do not change often.
- specifies the distribution of specializations among agents
- a precide way of dividing the problem space without specifying particular problems
- agents are associated with problem types and problem instances circulate to the agents which are responsible for instances of that type

Organisational structures may be: **functional**, **spatial**, **product-oriented**

`functional`

`spatial`

`product-oriented`

Hence, organizational structure models provide a pattern for decision-making and communication among a set of agents who perfrm tasks in order to achieve goals. The most well-known example of the application of organizational structure models is the automotive industry.
In fact, in such a field:

- Has a set of goals, i.e. to produce different lines of cars
- Has a set of agents to perform the tasks: designers, engineers and salesmen
- It bases its organizational structure based on various costs: production costs, coordination costs and vulnerability costs

Product Hierarchy

- several divisions for different product lines
- each division has a product manager
- each division has its own separate departments (agents) for different functions
- product manager assigns tasks o agents in its department
- communication links in product structure

- fails inside department do not affect other products
- one message to assign task and one message to notify result
- also model for set of separate companies

Functional Hierarchy

Missing figure

- a number of agents of similar types are pooled into functional departments
- each department has a functional manager
- reduce duplication of efforts
- executive office, production manager for all productshierarchilar task allocation
- 2 messages to assign task and 2 messages to notify about results
- failure of task aget results in a delay and task reallocation
- managers are critical

Missing figure

I did not understand why market are considered

Decentralized market

Missing figure

- all buyers are in contact with all possible suppliers
- buyers play a role of product managers
- each has a communication link with each supplier
- buyers can choose the best supplier
- given m suppliers, $2m + 2$ messages are requires
- if the processor fails the task is reassigned
- if manager fails the whole product fails

Centralized market

- brokers are in contact with possible sellers
- fewer connections and communications are required
- brokers play a role of functional managers
- broker chooses the best supplier
- 4 messages needed
- similar to functional model
- failure of one product manager does not affect others

Summary of organization structures

37,38, 39

- Organizational structures are useful when there are master/slave relationships in the MAS
- control over the slaves actions: mitigates against benefist of DAI such as reliability and concurrency
- Presumes that atleast one agent has global overview: an unrealistic assumption in MAS

4.2.3 Meta-level information exchange

- Exchange control level information about current priorities and focus
- Control level information
  - May change
  - Influence the decisions of agents
- Does not specify which goals an agent will or will not consider
- Imprecise
- Medium term: can only commit to goals for a limited amount of time

Among the most notable implementation of the Meta-Level Information exchange there is the **Partial Gloabl Planning (PGP)** proposed by Durfee

Partial Gloabl Planning (PGP)

Partial Global Planning (PGP)

The basic condition of PGP is the requirement that several distributed agents work on solution at the same time.

An agent therefore can observe actions adn relations between groups of other agents.

PGP involves 3 iterated stages:

1. Each agent decides what its own goals are and generates short-term plans in order to achive them

2. Agents exchange information to determine where plans and goals interact

3. Agents alter local plans in order to better coordinate their own activities

Hence PGP involves:

- Task decomposition.
  Starts with the premise that tasks are inherently decomposed.
  Assumes that an agent with a task to plan for might be unaware of what tasks other agents might be plannign for and how those tasks are related to its own.
  No individual agent might be aware of the global tasks or states.
  The purpose of oordination is to develop sufficient awareness
- Local plan formulation .
  Before an agent communicates with others it must first develop an understanding of what goals it is trying to achieve and which actions to perform.
  Local plans will most often be uncertain, involving branches of alternative actions depending on results of previous actions and changes in the environment
- Local plan abstraction.
  Alternative courses of action for achieving the same goal are important for an agent, however, the detail of alternatives might be unnecessary considering the agent-s ability to coordinate with others." An agent might have to commit to activitis at one level of details without committing to activities at more detailed levels.
  Agents are designed to identify their major plan steps that could be of interest to other agents
- Communication of local plan abstraction .
  Agents must communicate about abstract local plans in order to build models of joint activities.
  In PGP, the knowledge to guide this communication is contained in the meta-level-organization
- Partial global goal identification.
  The exchange of local plans gives an oppportunity to identify when the goal of one or more agents could be subgoals of a single global goal.
  Only portions of the global goal might be known to agents. Hence the name partial global goal
- PGP construction and modification.
  Local plans can be itegrated into PGP. PGP can in fact identify opportunities fo improved coordination (performing related tasks earlier, avoiding redundant task achievement). As a consequence, agents will undego a reordering of actions (usage of action rates). With **rates**, it si meant:

  rates

  - Whether the task is unlikely to have been accomplished already by another agent
  - how long it is expected to take
  - how useful its results will be to others in performing their tasks

  Hence the algorithm of action reordering goes as follow:

  1. For the current ordering, rate the individual actions and sum the ratings

  2. For each action, examise the later actions for the same agent and find the most highly-rated one. If it is higher rated, the swap the actions.

    3. If the new ordering is more highly rated than the current one, then replace the current ordering with the new one and go to step 2.
    4. Return the current ordering

- Communication planning.
  Agent must next consider what interactions should take place between agents.
  Interactions in the form of communicating the results of tasks are planned.
  By examining PGP an agent can determine when a task will be completed by one agent that could be of interest to another agent adn can explicitly plan teh communication action to transmit the result.
  Formally, the algorithm for communication plannign goes as follow:
    1. Initialize the set of partial task results to integrate
    2. While the set contains more than one element:
        (a) For each pair of elements: find the earliest time and agent at which they can be obtained
        (b) For the pair that can be combined earliest: add a new element to the set of partial results for the combination and remove the two elements that were combined
    3. Return the single element in the set
- Acting on PGP.
  Once a PGP has been constructed and the concurrent local and communication actions have been ordered, the collective activities of the agents have been planned. These activities must be translated back to the local level.
  An agent respons to a change in its PGP by modifying the abstract representation of its local plans.
  The modified representation is used by agent when choosing its next local action
- Ongoing modifications.
  As agents pursue their plans, their actions or events in the environment might lead to changes in tasks.
  A change in coordination is deciding when the changes in local plans are significant enough or defining a threshold (signiicant temporal deviations)
- Task trallocation.
  in case of disproportional task load, agents, through PGP, can exchange abstract models of their activities and detect whether they are overburdened.
  When this happens a possible task reallocation might occur.

In short: the main principle of PGP is that cooperating agents exchange information in order to raech common conclusions about the problem solving process.

It is said to be **partial** because an agent does not generate a plan for the entire problem.

partial

It is said to be **global** because agents form non-local plans by exchanging local plans and cooperating to achieve a non-local view of problem solving.

global

To summarize: PGP is a cooperatively generated data strcuture containing the actions and interactions of a group of agents.

It contains:

- **Objective**, the larger goal of the system
- **Activity map**, what agents are actually doing and the results generated by the activities
- **Solution construction graph**, a representation of how the agents ought to interact in order to successfully generate a solution

Objective

Activity map

Solution construction graph

PGP focuses on dynamically revising plans in cost-effective ways given an unertain world.
PGP is particularly situes to applications where some uncoordinated activity can be tolerated and overcome.
PGP works well for many tasks, but could be inappropriate for domains such as air-traffic control where guarantees about coordination must be made prior to any execution

4.2.4 Multi-agent planning

Agents generate, exchange and synchronize explicit plans o actions to coordinate their joint activity. they arrange a priori precisely which tasks each agent will take on.
Plans specify a sequence of actions for each agent.
Consideration of uncertainty is moved to the planning activity.
More specific than organizational structures or PGP and shorter time-horizon.

There are two basic approaches to Multi-agent planning:

1. Centralised: a central coordinator develops, decomposes and allocated plans to individual agents

2. Distributed: a group of agents cooperate to form either a centralised or a distributed plan

Centralized planning

- Given a goal description, a set of operators and an intial state description, generate a plan
- Decompose the plan into subprobelms such that ordering relationships between steps tend to be minimized across subprobelms
- Insert synchronization actions into subplans
- Allocate subplans to agents using task-passing mechanisms
- Initiate plan execution and optionally monitor progress

Distributed planning for centralised plans

In air traffic control domain, the aim is to enable each aricraft to maintain a flight plan that will aintain a safe distance with all aircrafts in its vicinity.
Hence each aircraft sends a central coordinator information about its intended actions. The coordinator builds a plan which specifies all of the agents' actions including the ones that they should take to avoid collision.

Another exaple is a general technological process such as manufacting: the plan is centralized but each step can be planned in parallel. As a consequence, there is a distributed expertize even though the control is centralized.

Distributed planning for distributed plans

Individual plans of agents, coordinate dynamically.
No individual with a complete view of all the agents' actions.
More difficult to detect and resolve undesirable interactions.

Agents in these different scenarios share and process a huge amount of information, hence these approaches require more computing and communication resources

4.2.5 Explicit anaylsis and synchronization

Analysis of situation in each decision-making step (possible-next-action-set).
Interacting with other agents:

- Exchange among all interdependent agents
- locks actions
- uses reply information to prune its next-action set
- send synchronization unlocking messages

A lot of time for communication during planning.
If the level of dependency is low and the granularity of actions is high, this approach can provide useful coordination.
Short time-horizon

4.2.6 Social Norms and Laws

Norm

Norm: an established, expected pattern of behaviour
Social Laws: similar to Norms, but carry authority.

Social Laws

Social laws in an agent system can be defined as a set of constraints:

- A constraint is formally represented as a tuple $< E\,\alpha >$, where:
  - $E' \subseteq E$ is a set of environment states
  - $\alpha \in Ac$ is an action (in this case Ac is the finite set of actions possible for an agent)
- If the environment is in some state $e \in E'$, then the action α is forbidden.
  An agent or plan is said to be legal wrt a social law if it never attempts to perform an action that is forbidden by some constraint in the social law.

4.2.7 Coordination Models based on human teamwork

In this section we will see that it is possible to achieve coordination without communication. Since some the method that we will see are based on human teamwork it is worth making a distinction:

- Coherent behaviour that is not cooperative, e.g. individual drivers in traffic following traffic rules
- Coordinated cooperative action, e.g. implicit sharing a common goal like dancers
- Teamwork action, e..g conovy-combining individual efforts

Formally teamwork is defined as a cooperative effort by the members of a team to achieve a common goal.
In such a scenario an individual intention towards a particular goal ay differ from being a part of a team with a collective intention towards a goal. However, teamwork is enforced with the notion of Responsibility towards the other members of the team

Mutual Modelling

In Mutual Modelling techniques, each agent builds a model of the other agents, in particular their beliefs and intentions.
Based on this model, the agent will coordinate its own activities and achieve cooperation without communication.

Joint Commitments

Joint Intentions models are based on human teamwork models: when a group of agents are engaged in a cooperative activity, they must have a joint commitment to the overall aim as well as their individual commitments.

The term commitment refers to a pledge or promise and carries around several properties:

- Commitments are persistent, in the sense that if an agent adopts a commitment, it is not dropped unitl for some reason it becomes redundant
- Commitments may change over time, due to a change in the environment

The main problem with joint commitment si that it is hard for agents to be aware of each others states at all times.

Conventions, refer to means of monitoring a commitment. The overall mechanism of conventions is nevessary to describe when to change a commitment, or more in particular:

- When to keep a commitment (retain)
- When to revise a commitment (rectify)
- When to remove a commitment (abandon)

Joint action by a team involves more than just the union of simultaneous individual actions. When a group of agents are engaged in a cooperative activity, they must have:

- Joint commitment to the overall activity
- Individual commitment to the specific task that they have been assigned to

More in particular Invividual conventions describe how an agent should monitor its commitments, but not how it should behave towards other agents. In this sense it is both asocial and suffiient for goals that are independent.

However, individual conventions do not work for inter-dependent goals, for which it is necessary to have social conventions, which specify how to behave with respect to the other members of the team.

Team members must be aware of the convention that govern their interaction.

Chapter 5

MAS Architecture

So far we have considered the main mechanisms of MASs: negotiation, communication and coordination.
But in these chapter we will focus on the technological aspect of MASs: how to build a system that employ such characteristics and mechanisms.

The context of MASs widens the notion of inteligent agents in two ways:

- An agent's user that imparts goals to it and delegates tasks might be not only human, but also another agent
- An agent6 must be desifgned with explicit mechanisms of communicating and interacting with other agents

This new notion of intelligent agents needs to be taken into account when analysizing the general architecture of a MAS.

Formally: "the infrastructure for a MAS is a set of services, conventions and knowledge that support complex social interactions (e.g. negotiations, agree on commitments...)" Among the services that a MAS architecture can support we can find the following

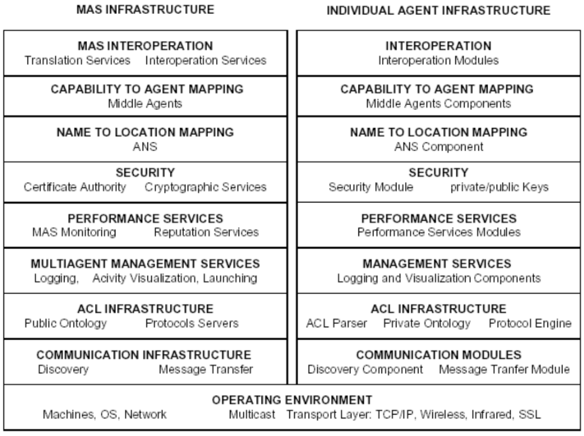
However, this is not a standard architecture, but rather a reference architecture that we can use in order to understand the kind of structures that can be included.
In particular, the structure of this architecture was proposed as a multi-layered architecture which includes a **MAS Infrastructure** and a **Individual agent infrastructure**.

MAS Infrastructure

Individual agent infrastructure

5.1 Low-level architectures

The low level of the proposed architectures/set of services, deals with communication and the communication infrastructure. And there are only two ways that this infrastructure can be implemented:



MAS INFRASTRUCTURE	INDIVIDUAL AGENT INFRASTRUCTURE
MAS INTEROPERATION Translation Services Interoperation Services	INTEROPERATION Interoperation Modules
CAPABILITY TO AGENT MAPPING Middle Agents	CAPABILITY TO AGENT MAPPING Middle Agents Components
NAME TO LOCATION MAPPING ANS	NAME TO LOCATION MAPPING ANS Component
SECURITY Certificate Authority Cryptographic Services	SECURITY Security Module private/public Keys
PERFORMANCE SERVICES MAS Monitoring Reputation Services	PERFORMANCE SERVICES Performance Services Modules
MULTIAGENT MANAGEMENT SERVICES Logging, Acivity Visualization, Launching	MANAGEMENT SERVICES Logging and Visualization Components
ACL INFRASTRUCTURE Public Ontology Protocols Servers	ACL INFRASTRUCTURE ACL Parser Private Ontology Protocol Engine
COMMUNICATION INFRASTRUCTURE Discovery Message Transfer	COMMUNICATION MODULES Discovery Component Message Tranfer Module
OPERATING ENVIRONMENT Machines, OS, Network Multicast Transport Layer: TCP/IP, Wireless, Infrared, SSL	

Shared memory and message sending/passing. In the former, we consider how we can communicate via some common channel where everybody can read and write something. In the latter, one agent send a message to exactly the recipient(s).

This is two basic approaches that can be implemented in very different ways and at different levels.

5.1.1 Blackboards

A possible shared memory approach to MAS communication there is a **Blackboard Architecture**. The metaphor at the core of this architecture is that a collection of intelligent agents gather around a blackboard, loot at pieces of information written on it, think about them, and add their conclusions.

Blackboard Architecture

It means that each agent can only communicate via the blackboard.

Some basic assumptions of this method are

- all of the agents can see all of the blackboard all the time, and what they see represents the current state of solution.
  All information in the blackboard is available to all agents.
- any agent can write his conclusions on the blackboard at anytime without gettin in anyone else's way.
  The write operation is atomic and asynchronious.
- the act of an agent writing on the blackboard will not confuse any other agents as they work.

The idea of blackboard architecture was originally implemented for translation of natural language.

The key ideas of the blackboard architecture is that 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.

  Incremental

  Agents do not just write whatever they want but rather they try to solve a problem (the solution to the problem is constructed incrementally).
- **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.

  Opportunistic

  Having just a blackboard is not enough to solve a given problem, but there must be some organization mechanism on how to read, write and choose the next action. In other terms in order to solve the problem, the MAS must choose the most informative and useful element written in the balckboard

In figure 5.1 a schematic representation of the blackboard architecture is proposed.

From the basic representation of the blackboard architecture we can identify the following main components:

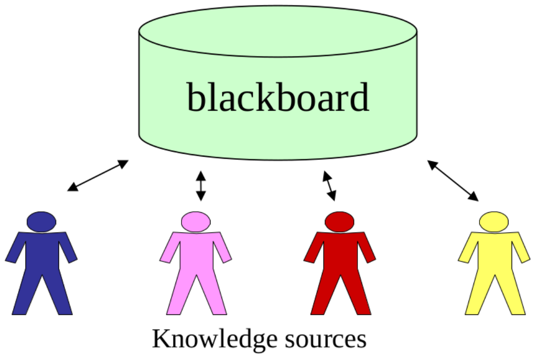


Figure 5.1

- A **blackboard**: a global database containing data and hypotheses (potential partial solutions).
- A set of **Knowledge Sources (KS)** or agents
- A **Control mechanism**, which will say what happens after an agents write something on the blackboard or that will decide who is going to be the next source of information.

blackboard

Knowledge Sources (KS)

Control mechanism

The main control problem then is that of selecting the best action to execute next.
Blackboard systems, in fact, must have some control mechanism:

- allow effective control requires to take into account
  - **Goal-directed factors**: based on what an agent wants.
  - **Data-oriented factors**: based on what an agent is best able to do. (based on what is written in the blackboard)
- 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.

Goal-directed factors

Data-oriented factors

Before going into depth of the application of the blackboard architecture there is one important hypothesis that need to be considered: the **Serialisation Hypothesis**.
In principle, the idea is that agents can read and write whenever they want, but this is not efficient because there must be some kind of order in which agents can do this.
Such order is ensured by some serialisation:

Serialisation Hypothesis

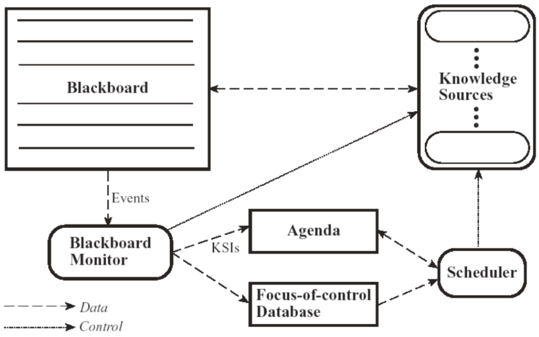
- agents are schedulable entities and only one can be running at any time.
- The control mechanism selects only the most productive agent at any given moment to work on the problem.
- The blackboard is not gloabally visible. Agents generally work on a limited area of the blackboard, known as the agent's context.
  In fact in principle agents have access to all the information of the blackboard, however for an efficiency point of view, it is better if agents look only into some locations.
- Implicit assumption: a knowledge source/agent operates within a valid, or consistent context.

Agenda-based Control

A first type of Blackboard architecture that we consider is the **Agenda Base Control**.

Agenda Base Control

1. Each agent can write into Blackboard, and the act of writing generates some event.
2. The event is then processed by some Blackboard monitor: a component that monitor what is going on in the blackboard and what should be the reaction.
3. When something new happears, the Blackboard monitor select what are other agents to whom this information may be useful.



4. Such information will be sent directly to the selected agents as a part of a preconditions for their execution.

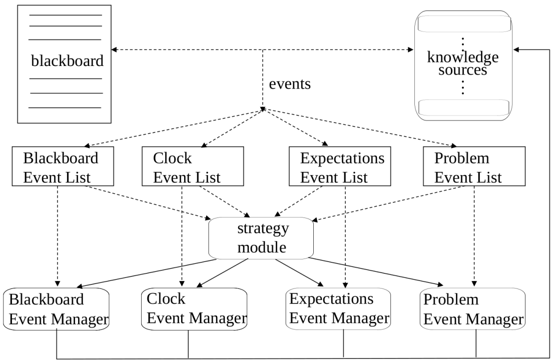
5. The selected agents will be put into an agenda to be scheduled for execution.

6. They way agents are selected from the agenda is through a rating which is attributed to each agent via a Focus-of-control database .
   The task of this component is to rate the execution of agents in the agenda based on the information and some other criteria.

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.

Event-based Control

There are some alternative and modification of this type of control. When there may be not just one but many Event list that are triggered by the blackboard.



1. An agent write to the blackboard and an event is generated

2. Such event is then classified based on some property. This differentiate and classify what the agents is writing on the blackboard, which might be a general information, an expectation or a problem.

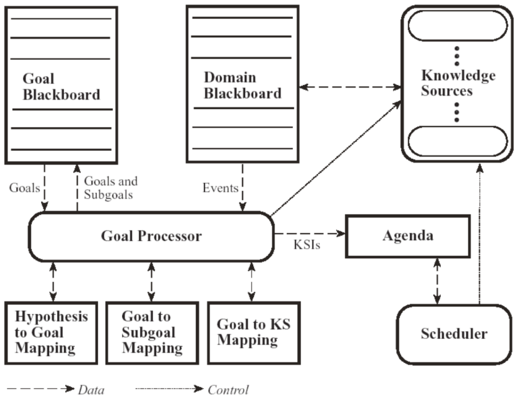
3. From the chosen amount of blackboard monitor there will be some strategy module, who decides from which event list to choose from.

4. A particular monitor/manager will handle the resulting event and transmit it to knowledge sources.

Goal-directed Control

Goal-directed control

Goal-directed control considers the role and the ultimate value of actions in satisfying the system's goals.



Domain Blackboard

Goal Blackboard

In this case there are two blackboard, namely a **Domain Blackboard** and a **Goal Blackboard**

1. An agent writes on the Domain Blackboard and generates an event
2. The goal processor processes information in the blackboard and transmit it to a particular agent and schedule such agent to execute via a scheduler.
   In addition, the goal processor maps the event into a goal balckboard, whenever it cannot identify a specific agent to convey such information to.
   From the goal blackboard some subgoals can be generated until an agent is found according to what we need.

   This is some sort of reasoning mechanism that yields a more elaborated processing of each goal.

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

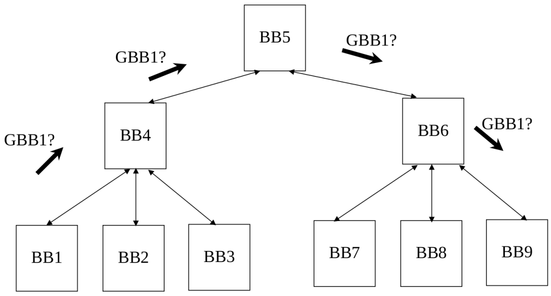
When a goal is inserted onto the goal blackboard, it may trigger KSs that can achieve the goal identified by the goal-to-KS map.
If KS is likely to generate a hypothesis to achieve the goal, KS is added to the agenda.
A possible problem may emerge: subgoaling needs to be carefully controlled.

Hierarchy of Blackboard Servers

Moreover, we can consider that the blackboard architecture is in reality a distributed blackboard, since keeping a centralized blackboard for every agent in the MAS can be a bottleneck.

However, by creating an hierarchy or a distributed system of blackboard. Between which some relations and dependecies exist.

Whenver a goal gets written into a blackboard but there are no sources that can be find to understand what the next action should be, it will be forwarded to the upper level blackboard. Such parent blackboard will see information and identify a blackboard that can process the information. If yet there is no one that can process the information it will forward the goal to its parent blackboard and repeat the process.



5.1.2 ACTORs

ACTOR model approach is a pure message passing architecture that has the following properties

- **Social**, can send messages to other ACTORs
- **Reactive**, carry out computation in response to a message received from another actor

Social

Reactive

In the ACTOR model, computation itself is viewed as message passing. It can be considered as consisting of:

- An address
- A behaviour which specifies what the ACTOR will do upon receipt of a message

In other term, we have some agents to be considered as a reactive entity, that has an address and a behaviour. Whenever an agent receives a message it executes its behaviour. As such agents are programs that can be triggered and have some address that we can invoke directly.

The ACTOR approach is formulated around three main design objectives:

1. Shared, mutable data.
   Not shared channel but shared channels. This can be some kind of common data that all agents need in order to perform some kind of communication.

2. Reconfigurability.
   New agent can be created and it is possible to communicate with such new agents.

3. Inherent concurrency.
   Asynchronius behaviour is possible.

An ACTOR is an object that carries out its actions in response to communication it receives. ACTOR may perform only three basic actions:

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

The behaviour of an ACTOR model system is as follow:

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

ACTOR computation is therefore 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 an important part of the model.

The communication event in actor model is called a task and it has 3 parts:

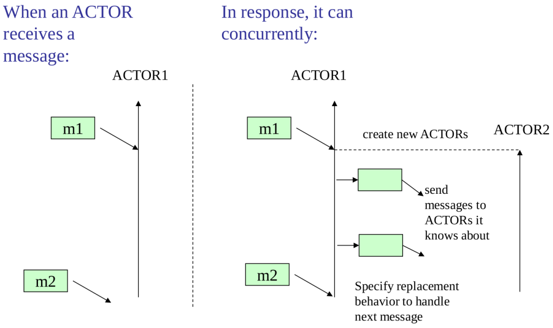
1. A unique tag, distringuishing it from tasks in the system
2. A target, the mail address of intended receiver
3. A communication which is the data passed by this communication event

Actor model is asynchronous.
Communication is:

- explicity, through mail addresses, without shared variables
- buffered and asynchronous

Weak fairness is assumed:

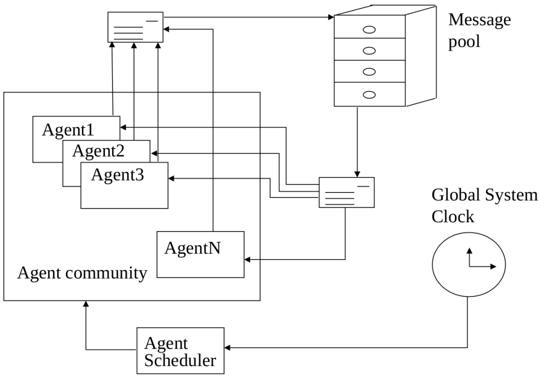
- every message which is sent is eventually delivered
- every computation eventually progress (no starvation)


When an ACTOR receives a message:
In response, it can concurrently:
ACTOR1
ACTOR1
m1
m1
create new ACTORs
ACTOR2
send messages to ACTORs it knows about
m2
m2
Specify replacement behavior to handle next message

Message sending based architecture: Agent Factory

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: the content of the message does not change in transaction, the 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



5.2 Testbeds

Testbeds

Testbeds are high-level architectures, which where developed specifically in the DAI community. The main purpose of DAI testbeds is to support the implementation of ideas so that they can be evaluated in a useful context. In other terms, if a designer would like to propose a new algorithm for communication and negotiation, it can compare it with other algorithms via a testbed: this will allow to use the same example, the same tasks and the same system.

In order to have such a possibilities most DAI testbeds provide three classes of facilities to compare different class of algorithm:

- **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

Domain facilities

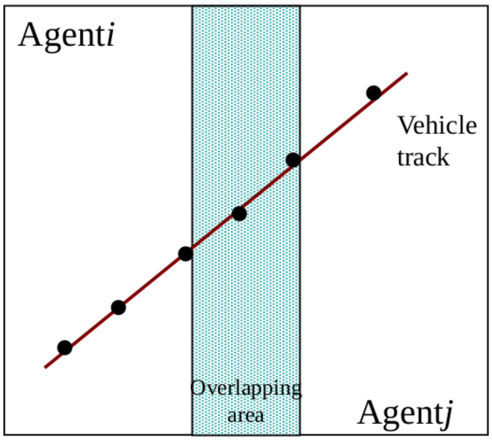
Development facilities

Evaluation facilities

5.2.1 DVMT

The **Distributed Vehicle Monitoring Testbed (DVMT)** is a DAI testbed developed to successfully track a number of vehicles that pass within the range of a set of distributed sensors (agents).

Distributed Vehicle Monitoring Testbed (DVMT)



DVMT is a problem dependent testbed and agents (problem solvers) have a blackboard architecture.

5.2.2 MACE

MACE provides tools for constructing DAI systems at different level of abstraction with agents that have an ACTOR model:

- Each rule can be an agent
- coarse-grained systems with large scale agents

MACE had no fixed domain for problem solving activity, however it have some fixed examples.

5.3 Infrastructures that use wrappers: ARCHON

In many industrial applications, a substantial amount of time, effort and money was spent on developing complex and shophisticated software systems (e.g. expert systems and databases).
The basic idea is that we have a legacy system and we would like that these systems operate in a

new environment. In order to do so, the designer could provide them with a wrapper that enable them to use in any other agent system.

This is what was implemented in **ARchitecture for Cooperative Heterogeneous ON-line systems (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:

ARchitecture for Cooperative Heterogeneous ON-line systems (ARCHON)

- **Framework**: which provides assistance for interaction between constituent subcomponents
- **Methodology**: which provides a means for structuring these interactions

Framework

Methodology

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 Layer (AL)

Intelligent System (IS)

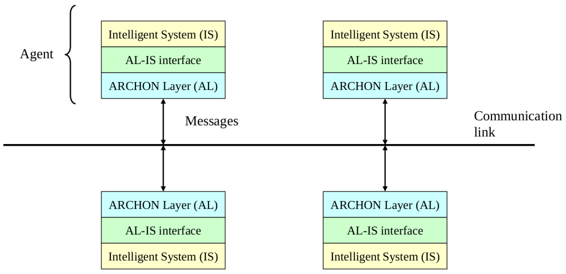


Figure 5.2

Figure 5.2 shows the basic ARCHON community structure. Each agent has an:

- ARCHON layer that is responsible for the implementation of interagent communication and other services that are necessary for communication
- An interface layer that allow the designer to wrap the Intelligent System into the overall MAS.

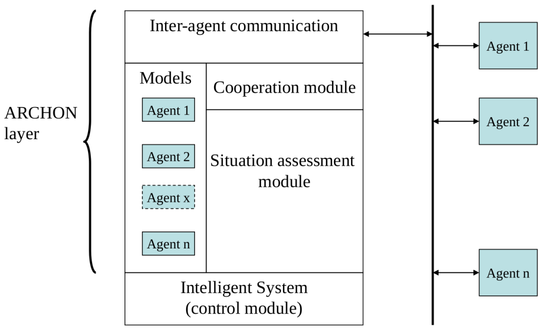
Communication happens via a single communication link or bus. This is fairly similar to KQML system where the Archon layer was the router, the AL-IS interface is similar to KRIL and lastly the intelligent system was the system itself.

This is a more general way of implementing a communication system where we define a set of general features that can be used by other system to communicate with the ARCHON Layer. This implies that it does not matter how we implement our Intelligent System, the communication will be consistent.

The most interesting part is how the AL-IS interface is structured, because it does not consist of a simple API, but more advanced techniques.

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 gloabl contraints.
Such interactions are controlled by the ARCHON layer which functions are to:

- 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 layer
Inter-agent communication
Models
Agent 1
Agent 2
Agent x
Agent n
Cooperation module
Situation assessment module
Intelligent System (control module)
Agent 1
Agent 2
Agent n

Looking into the three basic layers of the ARCHON architecture/approach we can notice that the ARCHON Layer can be devided into several modules:

- An Inter-agent communication layer/module that is responsible for supporting communication
- An interface which is a set of components that can be adjusted in order to plugin the Intelligen System.
  Among these modules we might find a **Cooperation module**, a **Situation assessment module** (something that you can tune in order to adjust your system), a set of Models of other agents or of itself.

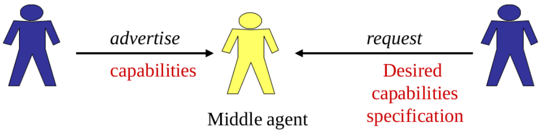
Cooperation module

Situation assessment module

In other terms this is a basic implementation that you can adjust in order to plugin the desired legacy system or intelligent system. It is a smooth translation model to some generic agent architecture.

5.4 Infrastructures that use Middle Agents: RETSINA

The idea of Middle agents is that instead of having a direct mechanism, Middle agents is a highly intelligent blackboard that it is implemented as shared memory but rather as an agent that can keep relationships between tasks and requests and proposals.



In an open system, the set of agents is not known a priori.
The infrastructure should provide ways for its agents to locate each other based on name, functionality or capability.

Agents that provide this service are called Middle agents which may also take the role of Facilitators or Matchmakers.

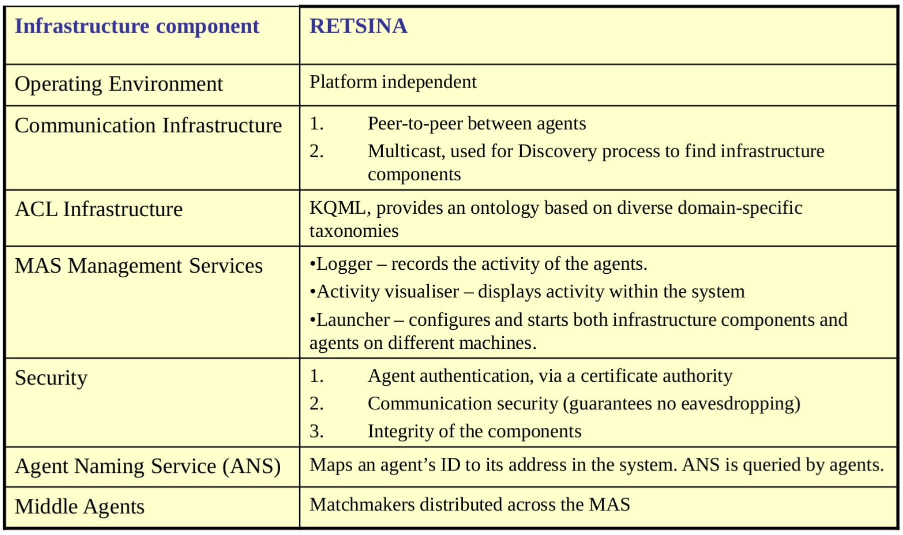
REusable Task Structure-based Intelligen Network Agents (RETSINA) is an open MAS infrastructure that supports communities of heterogeneous agents.
The main idea of RETSINA is that agents should form a community of peers that engage in peer-to-peer relations.
There is no central control and it implements distributed infrastructural services that facilitate the relation between agents.

REusable Task Structure-based Intelligen Network Agents (RETSINA)

RETSINA actually implements all the layers proposed in the reference architecture at the beginning of this chapter except for the interoperation layer.

RETSINA middle agents are called Matchmakers, i.e. 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: advertisements and requests



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

- the task of matchmakers is to match advertisements to requests
- Two types of protocols: single shot and monitor

5.5 Market-based architectures

Online market places offer an opportunity to buyers and sellers to meet electronically and conduct trade. In some sense, these architecture consider a Middle agent to conduct their activity, but a special one which is the market place itself.

Offers benefits for both buyers and seller:

- Buyers: ease to process of searching for and comparing sellers
- Sellers provide access to much broader customer bases

The major challenges are:

- 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 failure in contract execution

The requirements of a Market-based Architecture are to:

- Provide support for a variety of transaction types (buying and selling, complex multi-agent contract negotiation, auctions)
  This mechanism should be oriented towards different type of transactions.
- Control fraud and misrepresentation
- Discourage counterspeculation
- Provide for secure and private credit and payment mechanisms
- Provide a language in which the rich array of semanting 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

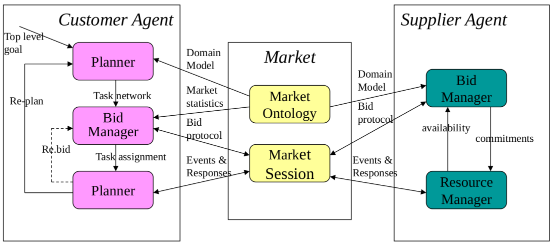
So the idea is that in addition to the standard Matchmaking propose-request we need to implement the aforementioned features.

5.5.1 MAGNET

The **Multi-AGent NEgotiation Testbed (MAGNET)** is a testbed to support multiple agents in negotiating contracts.
Supports negotiation of contract for tasks that have temporal and precedence constraints.
Distinguishes between the agents:

Multi-AGent NEgotiation Testbed (MAGNET)

- 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



In this architecture there are:

- A special element called market that implements market sessions and may implement also Market Ontology or knowledge about such market place.
- Customer agents, that are provided with a Planner in order to plan its task and a Bid Manager.
- Supplier agent will have a Bid Manager and a Resource Manager.

In the general case, the bidding process goes as follows:

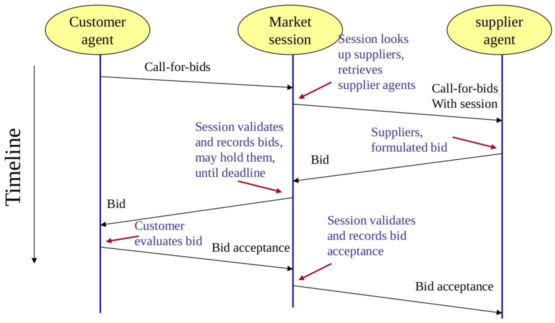
1. Customer agent plans a set of task
2. Getting information about the Market Ontology, the Customer Agent can create bids via the Bid Manager and publish such tasks to be performed into the Market Session
3. The Supplier Agent gets information about the Market Ontology and via the communication between the Bid Manager and the Resource Manager, decides which of the resources of interest are available.
4. Based on the information of availability derived from the Resource Manager, the Supplier Agent will publish into the Market Session
5. The bid protocol will coordinate the rules of the action or negotiation.
6. Depending on whether or not the bid was accepted or rejected, the Customer Agent may decide to rebid or replan its tasks.
7. Upon agreement the Supplier Agent will commit via the Resource Manager to provide the agent with the resources needed.

The interaction among agents goes as follows

1. A customer issues a Request for Quotes (RFQ), which sp[ecifies tasks, the precedence relations and a timelien for the bidding process
2. 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
3. The customer decides which bid to accept based on cost, risk and time constraints
4. The customer awards bid, notifies the suppliers of their commitments and specifies the work schedule

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


Customer agent
Market session
supplier agent
Call-for-bids
Session looks up suppliers, retrieves supplier agents
Call-for-bids With session
Timeline
Session validates and records bids, may hold them, until deadline
Suppliers, formulated bid
Bid
Bid
Customer evaluates bid
Bid acceptance
Session validates and records bid acceptance
Bid acceptance

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
- Can act as a trusted auctioneer

5.6 Conclusion

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 do not exist then to general tendencies)

Chapter 6

Agent Oriented Software Engineering

As MASs become more established in the collective consciousness of the computer science community, we might expect to see increasing effort devoted to devising methodologies to support the development of agent systems.

6.1 When is an Agent-Based Solution Appropriate?

There are a number of factors which point to the appropriateness of an agent-based approach:

- The environment is open or at least highly dynamic, uncertain or complex
- Agents are a natural metaphor.
  Many environments are naturally modelled as societies of agents, either cooperating with each other to solve complex problems, or else competing with one another.
- Distribution of data, control or expertise.
  In some environments, the distribution of either data, control or expertise means that a centralized solution is at best extremely difficult or at worst impossible.
- Legacy systems.
  A problem increasingly faced by software developers is that of **legacy software**, that is technologically obsolete but functionally essential to an organization.
  Such software cannot generally be discarded, because of the short-term cost of rewriting. And yet it is often required to interact with other software components which were never imagined by the original designers. One solution to this problem is to wrap the legacy components, providing them with an agent layer functionality, enabling them to communicate and cooperate with other software components.

legacy software

6.2 Agent-Oriented Analysis and Design

An analysis and design methodology is intended to assist first in gaining an understanding of a particular system and secondly in designing it.

Methodlogies generally consist of a collection of models, and associated with these models, a set of guidelines. The models are intended to formalize understanding of the system being considered.

Analysis and design methodolgies of agent-based systems can be broadly divided into two groups:

1. Those that take their inspiration from OO development, and either extend existing OO methodologies or adapt OO methodologies to the purposes of **Agent-Oriented Software Engineering (AOSE)**

Agent-Oriented Software Engineering (AOSE)

2. Those that adapt knowledge engineering or other techniques

6.2.1 The AAII methodology

The **Australian AI Institute (AAII)** developed a range of agent-based systems using their PRS-based belief-desire-intention technology and the distributed multiagent reasoning system (DMARS).

Australian AI Institute (AAII)

This methodoly draws primarly upon OO methodologies and enhances them with some agent-based concepts. The methology itself is aimed at the construction of a set of models which, when fully elaborated, define an agent system specification.

The AAII methology provides two types of model:

1. The **External model** presents a system level view: the main components visible in this model are agents themselves.

   External model

   The external model is thus primarly concerned with agents and the relationships between them.
   Particularly the external model is intended to define inheritance relationships between agents classes, and to identify the instance of these classes that will appear at run time.

   It is itself composed of two further models: the **agent model** and the **interaction model**.

   Furthermore the agent model is divided into an **agent class model** and an **agent instance model**.

   agent model

   interaction model

   agent class model

   agent instance model

2. The **Internal model** is concerned with the internals of agents: their beliefs, desires and intentions

   Internal model

These two models define the agents and agent classes that can appear, and relate these classes to one another via inheritance, aggregation and instantiation relations. Each agent class is assumed to have at least three attributes: beliefs, desires and intentions. The analyst is able to define how these attributes are overridden during inheritance. Details of the internal model are not given but it seems clear that developing an internal model correspons fairly closely to implementing a PRS agent.

The AAII methodology is aimed at elaborating the models described above. It may be summarized as follows:

1. Identify the relevant roles in the application domain and on the basis of these, develop an agent class hierarchy

2. Identify the responsibilities associated with each role, the services required by and provided by the role and then determine the golas associated with each service.

3. For each goal, determine the plans that may be used to achieve it and the context conditions under which each plan is appropriate

4. Determine the belief structure of the syste, the information requirements for each plan and goal.

Note that the analysis process will be iterative, as in more traditional methologies. The outcom eiwll be a model that closely corresponds to the PRS agent architecture. As a result, the move from end-design to implementation using PRS is relatively simple

6.2.2 The GAIA methodology

The **GAIA methology** is intended to allow an anlyst to go systematically from a statement of requirements to a design that is sufficiently detailed that it can be implemented directly.

GAIA methology

In applying GAIA, the analyst moves from abstract to increasingly concrete concepts. Each successive move introduces greater implementation bias and shrinks the space of possible systems that could be implemented to satisfy the original requirement statement. Gaia borrows some terminology and notation from OO analysis and design. However, it is not simply a naive attempt to apply such methods to Agent-oriented development. Rather, it provides an agent-specific set of concepts through which a software engineer can understand and model a complex system. In particualr, Gaia encourages a developed to thing of building agent-based systems as a process of **organizational design**.

organizational design

The main Gaia concepts can be divided into two categories:

- **abstract** entities are those used during analysis to conceptualize the system, but which do not necessarily have any direct realization within the system

abstract

- **concrete** entities are used within the design process, and will tipically have direct counterparts in the run-time system.

concrete

The objective of the analysis stage is to develop an understanding of the system and its structure without reference to any implementation detail. In the Gaia case, this undestanding is captured in the system's **organization**.

organization

An organization is viewed as a collection of roles that stand in certain relationships to one another and that take part in systematic, institutionalized patterns of interactions with other roles.

The idea of a system as a society is useful when thinking about the next level in the concept hierarchy: **roles**. A role is defined by four attributes:

roles

1. **Responsibilities**

   Responsibilities

   They determine funcitonality and as such are perhaps the key attribute associated with a role. Responsibilities are divided into two types:

   (a) **Liveness properties** descirbe those states of affair that an agent must bring about, given certain environmental conditions.

   Liveness properties

   (b) **Safety properties** are invariants, which in other terms states that an acceptable state of affairs is maintained across all states of execution.

   Safety properties

2. **Permissions**

   Permissions

   A role has a set of permissions which identify the resources that are available to that role in order to realize its responsibilities. Permissions tend to **information resources**.

   information resources

3. **Activities**

   Activities

   The activities of a role are computations associated with the role that may be carried out by the agent without interacting with other agents. activities are thus private actions.

4. **Protocols**

   Protocols

   A role is also identified with a number of protocols, which define the way that it can interacti with other roles.

6.2.3 The Tropos methodology

The **Tropos methodology** aims to give an agent-oriented view of software throughout the software development lifecycle.

Tropos methodology

It provides a conceptual framework for modelling systems based on the following concepts:

- **Actor**

  Actor

  An entity with strategic goals and intentionality within the system or the organizational setting.
  Associated with actors are the notions of role and position.

  - A role is an abstract characterization of the behaviour of a social actor within tsome specialized context
  - A position is a set of roles, typically played by one agent

- **Goal**
  A goal represent sthe actors; strategic interests. Tropos distinguish **hard goals** (system funcitonal requirements) and **soft goals** (non-functional requirements)
- **Plan**
  A recipe for achieving a goal.
- **Resource**
  Physical entities or information
- **Dependency**
  A relationship between two actors, indicating that one agent needs the other to carry out some part of a plan, deliver some resource, or similar
- **Capability**
  The ability of an actor to achieve some goal or carry out some plan
- **Belief**
  Knowledge that actors have about their environment.

Goal

hard goals

soft goals

Plan

Resource

Dependency

Capability

Belief

The first phase of analysis in Tropos involves developing an :

- **actor model**, which captures the key stake holders in the system and their strategic intersts in the form of their goals
- **dependency model**, which documents the dependencies between these actors

actor model

dependency model

A **goal model** is then developed, which decomposes goals into their constituent parts, and, associated with this, a **plan model** captures the structure of recipes for achieving goals.

goal model

plan model

6.2.4 The Prometheus methodology

The **Prometheus methodology** consists of three main stages:

Prometheus methodology

1. **System specification**, which focuses on identifying the goals and basic functionalities of the system, and specifies the interface between the system and its environment in terms of actions and percepts

2. **Architectural design**, which focuses on identifying agent types, the system structure and the interactions between agents

3. **Detailed design**, which first involves refining agents into their capabilities and specifying the processes ion the system and then involves the detailed design of capabilities

System specification

Architectural design

Detailed design

Prometheus provides a rich collection of models for each stage and detailed guideline for each step. It also has software tool support

6.2.5 Agent UML

The **Undefined Modelling Language (UML)** is an attempt by three of the main figures behind object-oriented analysis and design to develop a single notation for modelling OO systems.
It is important to note the UML is NOT a methodology, but rather a language for documenting models of systems; however, associated with UML is a methodology known as the rational unified process.

Undefined Modelling Language (UML)

When looking for agent-oriented modelling languages and tools, many researchers felt that UML was the obvious place to start. The result has been a number of attempts to adapt the UML notation for modelling agent systems.
The proposed modifications include:

- support for expressing concurrent threads of interactino thus enabling UML to model such well-known agent protocols as the the Contract Net

- A notion of role that extends that provided in UML and in particular allows the modelling of an agent playing many roles

Both the Object Management Group and FIPA are currently supporting the development of UML-based notations for modelling agent systems, and there is therefore likely to be considerable work in this area.

6.2.6 Discussion

The predominant appraoch to developing methodologies for MASs is to adapt those developed for OO analysis and design. There are several disadvantages with such approaches:

- the kind of **decomposition** that OO methods encourage is at odds with the kind of decomposition that agent-oerinted design encourages. `decomposition`
- OO methodologies simply do not allow us to capture many aspects of agent systems. The extensions to UML proposed address some, but by no means all of these deficiencies.
- At the heart of the problem is the problem of the relationship between agents and objects which has not yet been satisfactorily resolved.

6.3 Pitfalls of Agent Development

- **You oversell agent solutions, or fail to udnerstand where agents may usefully be applied**
- **You get religious or dogmatic about agents**
- **You do not know why you want agents**
- **You want to build generic solutions to one-off problem**
- **You believe that agents are a silver bullet**
- **You forget that you are developing software**
- **You forget that you are developing multithreaded software**
- **Your design does not exploit concurrency**
- **You decide that you want your own agent architecture**
- **Your agents use too much AI**
- **You see agents everywhere**
- **You havce too few agents**
- **You spend all your time implementing infrastructure**
- **Your agents interact too freely or in a disorganized way**

Chapter 7

Agent Theory

Previously we mainly considered agent group perspective in which agents communicate, negotiate and coordinate their activities.
In this section, we will focus on individual perspective of agents, in the sense of how agents operate in terms of theoretical model architecture and mobility for implementing their particular task.
The first element of this individual perspective is called **Agent Theory**.

Agent Theory

The reason why it is necessary to consider agent theory and formalisms is because formal theory has arguably had little impact on the general practice of software development, however, they are relevant in agent-based systems because we need to be able to give a semantic to the architecture, languages and tools that we use (i.e. a meaning).
Moreover without a semantic, it is never clear exactly what is happening and why it works.
But most importantly we need a theory to reach any kind of profound understanding the tools.

On the other hand the formalization of agents can be seen from 2 distinct perspective/purposes:

1. As **internal specification language** to be used by the agent in its reasoning or action
2. As **external metalanguage** to be used by the designer to specify, design and verify certain behavioural properties of agents situated in a dynamic environment.

internal specification language

external metalanguage

Agent theory gives us both an overview of the ways in which an agent is conceptualised and semantics to the architecture, language and tools: in other terms, we strive to formalise and conceptualise what is an autonomous entity and what is an autonomous behaviour.
Of course in order to have a theory of agent we need to introduce or consider a model of agents. For this purpose we will consider agents as**intentional systems**: the behaviour of an agent is explained in terms of attitudes such as believing and wanting.

intentional systems

7.1 Agents as intentional systems

Theorists start from the notion of an agent as an intentional system.
So, agent theorists start with the view of agents as intentional systems: where agent's behaviour is explained in terms of **attitudes**.

attitudes

7.1.1 Attitudes

Attitudes represent a summary evaluation of a psycological object (e.g. oneself, other people, issue, plan, behaviour). We do not know for sure how it works, but it is our perception and psychological understanding. For this reason attitudes are formed throughout the interaction with the surrounding environment and help to manage individual's cognitive resources to deal with uncertainties of complex dynamic domains.

In fact, the knowledge of the environment is seldom precise, agents may not know for sure or the environment may change. In this sense the agent knowledge is based on some uncertain element which prevents us to use plane knowledge base system as a foundation for agent. Moreover, when dealing with autonomous entities we need a way to generate goals. And attitudes enable agents to express the knowledge about the world, as well as intentions and goals, similarly to humans.

Formally an attitude is the use of a folk psychology, by which human behaviour is predicted and explained through the attribution of attitudes.
Human however, do not act on attitudes, but rather use the notion of attitudes to explain their action. This is the reason why this formalism was chosen as a way to describe agent behaviour.

The attitudes employed in such folk psychological descriptions are called **intentional notions**. An approach is to describe agent's behaviour in terms of intentional systems, "whose behaviour can be predicted by the method of attrivutring belief, desire and rational acumen".

intentional notions

An intentional system could be quite complex: first-order (attitudes), second-order (reasoning about attitudes),... etc.

Given all of the above, we might ask ourselver: is it useful to consider agents (similar to humans) as intentional systems? John McCarthy claims that:

- It is legitimate when it express the same information about the machine that it expresses about a person.
- It is useful when it helps us understand the structure of the artifact
- It perhaps never logically required even for humans.
  In fact, humans do not operate according to intentional notions but rather use attitudes and intentional notions to explain their action.
- Ascription of mental qualities is most straightforward for machines of known structure but most useful for entities whose structure is incompletely known.
  In simpler terms, if we completely know how a system works, then intentional systems are NOT useful. But the point is that agents do not and in fact cannot know exactly how the system works.
  The more we know about a system the less we need to rely on intentional explanations of its behaviour.

An autonomous agent is a system that is most conveniently described by the intentional stance since it tries to mimic the behaviour of human which we do not know.

7.1.2 Theories of Attitudes

We want to be able to design and build computer systems in terms of mentalistic notions. Before we can do this, we need to identify a manageable subset of these attitudes and a model of how they interact to generate system behaviour. So first, which attitudes?

There exists 2 categories of attitudes:

1. **information attitudes**: which express the perception of the world (e.g. belief and knowledge)

information attitudes

2. **Pro-attitudes**: which guides agents behaviour (e.g. desire, intention, obligation, commitment, choice...)

Pro-attitudes

In principle, a good approach would be to choose at least one attitude from both of the categories.

The manipulation of these attitudes falls in the study of knowledge.

7.1.3 Study of Knowledge

The study of knowledge tries to answer a series of open questions:

- What do we know?
- What can we know?
- What does it mean to say someone knows something?
- What does an agent need to know in order to perform an action?
- How does an agent know whether it knows enough to perform an action?
- At what point does an agent know enough to stop gathering information and make a decision?

In other terms, it is of our desire to formalise the intentional system in such a way that agent can reason about it.

Moreover, we need to distinguish between two different dimensions of knowledge:

- Individual perspective
- Group perspective, which encompasses the knowledge of other agents in the group, **common knowledge** (e.g. traffic rules) and **distributed knowledge** (agents hold only a small portion of knowledge but only together can come up with the whole knowledge)

`common knowledge`

`distributed knowledge`

7.2 Foundation of formal logic

The formalisation of attitudes and knowledge requires some logical theory at its core.
A **formal logic** is a game for producing symbolic objects according to given rules. It can be interpreted as a sort of language with some general syntax or alphabet which include the following notation:

`formal logic`

- Variables $(X, Y, ...)$
- Constants $(a, abc, 15, ...)$
- Functors (f/n) or functional symbol
- Predicate symbols $(p, q, ...)$
- Logical connectivities $(\neg, \vee, \wedge, \rightarrow, \iff)$
- Quantifiers $(\forall, \exists)$
- Auxiliary symbols (commas, brackets and so on)

From this alphabet we can combine these elements to create words or **term**s (T):

`term`

- Any constant in A is in T
- any variable in A is in T
- if f is an n-ary functor in A and $t_1, ..., t_n \in T$ then $f(t_1, ..., t_n) \in T$

Terms represent the basic building block of formal logic since they represents object to reason about, and as such they could be constant (constant term), variable (variable term) or structured object (functor).

Once we have defined what are the words or objects to discuss about, we can start to build up sentences as a set of terms or **formulae**.
Let T be the set of terms over alphabet A and F is the set of formulae:

`formulae`

- If p is an n-ary predicate symbol and $t_1, ..., t_n \in T$ then $p(t_1, ..., t_n) \in F$
- If H and $G \in F$ so are

$$\neg H \quad H \vee G \quad H \wedge G \quad H \rightarrow G \quad H \iff G$$

- If $H \in F$ and X is a variable then $\forall X \in H$ and $\exists X \in F$ are formulae

Formulae of the form $p(t_1, ..., t_n)$ are called **atomic formulae**.

`atomic formulae`

7.2.1 Interpretation

Until now we considered the pure syntax and components of formal logic, the meaning or semantic in formal logic is given by the interpretation of an alphabet.
An interpretation T of alphabet A is a non-empty domain D and a mapping that associates:

- $c \in Const$ with an element $c_t \in D[Const \subseteq A]$
- $f/n \in Func$ with an element
$$f_t : D_n \to D[Func \subseteq A]$$
- $p/m \in Pred$ with an element
$$p_T \subseteq D_m(= D \times ... \times D)[Pred \subseteq A]$$

In summary, both constant terms, functors (or structure object) and predicates (or relationship between objects) make up a domain.
The difference between predicate and functors is that predicate returns either true or false, the actual result of the predicate has to be deducted from the interpretation.
Under an interpretation, each term has a value and each atomic formula is either true or false.

We therefore defined the semantics of these objects or set of terms in the interpretation and equivalently we can formalize the semantics of the formulae or connectives of formal logic:

- **Negation** Negation
  $\neg A$ is true if A is false
- **Conjunction** Conjunction
  $A \wedge B$ is true if both A and B are true
- **Disjunction** Disjunction
  $A \vee B$ is true if either A or B is true
- **Implication** Implication
  $A \to B$ is true if either $\neg A$ or B are true
- **Universal quantifier** $\forall X$ Universal quantifier
  $A(x)$ is true if A is true for every X
- **Existential quantifier** $\exists X$ Existential quantifier
  $A(x)$ is true for some X
- **Propositional logic** is the logic of connectives (i.e. do not includes quantifier)
- Adding quantifiers give first-order logic, sometimes called **predicate calculus**.
- Adding quantifiers over formula variables give Higher Order Logic Propositional logic

We do not need classical formal logic to express something, we want to exploit this language to create a formalism that allow us to create new formulae from a given formula. There are two different ways to achieve this: predicate calculus

- **Models** and **Logical Consequence** Models Logical Consequence
  An interpretation T is a model of a set of formulae P iff every formulae of P is true in T:
  - every P has infinitely many interpretations (because there exists infinite interpretation to the same symbol)
  - Not every P has a model (e.g. $F \wedge \neg F$ cannot be true in the same interpretation)
  - A set of formulae is unsatisfiable if it has no model
  - A set of formulae is satisfiable if it has at least one model

  Let P be a set of formulae. F is a logical consequence of P ($P| = F$) iff F is true in every model of P.
  The symbol $| =$ indicates the logical consequence.
  In some cases proving logical consequence is not necessary since, there already are some formulae that are equivalent in every intepretation. F and G are **logically equivalent** logically equivalent

$(F \equiv G)$ iff F and G have the same truth value for every interpretation

$$\begin{aligned}
F \rightarrow G &\equiv \neg F \vee G \\
F \rightarrow G &\equiv \neg G \rightarrow \neg F \\
\neg(F \vee G) &\equiv \neg F \wedge \neg G \\
\neg(F \wedge G) &\equiv \neg F \vee \neg G \\
\neg \forall X H(X) &\equiv \exists X \neg H(X) \\
\neg \exists X H(X) &\equiv \forall X \neg H(X) \\
&\ldots
\end{aligned}$$

- **Logical inference** does not consider the truth value but tries to process and manipulate a given set of formulae to create new ones.
  Reasoning can be seen as a process of manipulation of formulae, which from a given set of formular, called **premises**, produces a new formula, called the conclusion using inference rules.
  For instance **Modus Ponens**, states that if F is true and $F \rightarrow G$ then we can deduce that G is true.
  In simpler terms, we use a set of given formulae to deduce a new formula via inference rules.
  Such deducability is represented with the notation: $P \vdash F$ means F is derivable from P.
  We can say that if the inference rules are sound then $P \vdash F$ (deducability) implies $P| = F$ (logical consequence)
  If the inference rules are complete then $P| = F$ (logical consequence) implies $P \vdash F$ (deducibility)
  In other terms **soundness** states that everything that is deducible is true, whereas **completeness** says that everything that is true is deducible.

Logical inference

premises

Modus Ponens

soundness

completeness

7.3 Formalising attitudes

Turns out that when we try to formalize attitudes or intentional systems, the application of formal logic makes them referentially opaque: hence standard substitution rules of first-order logic do not apply (intentional notions are not truth functional). Hence in an interpretation the truth value of a formula depends on the truth value of each component, but in intentional systems belief can be true without proving the truth of what you believe.

We can conclude that this formalisation is not good enough for attributes.
There are 2 sorts of probelms to be addressed in developing a logical formalism for intentional notions:

1. Syntactic
2. Semantic

With respect to the syntactic problem, there are two fundamental approaches:

1. Use a **modal language**, which contains modal operators, which are applied to formulae
2. use a **meta-language**: a first-order language containing terms that denote formulae of some other object language

modal language

meta-language

Likewise, two basic approaches to the semantic problem:

1. **Possible worlds semantics**
2. **Interpreted symbol structures**

Possible worlds semantics

Interpreted symbol structures

Of the four combination we will focus on the possible world semantics and modal logic

7.3.1 Possible worlds

The intuitive idea behind possible worlds is that besides the true states of affairs there are a number of states of affairs or "worlds".
Each world represents one state of affairs.
Given his current information, an agent may not be able to tell which of a number of possible worlds describes the actual state of affairs. Consequently, an agent's beliefs can be characterized as a set of possible worlds, which can be represented using modal logic.

The main advantages of such approach is:

- Mathematical theory is appealing
- Neutral on the subject of the cognitive structure

Consider an agent playing a card game, who possessed the ace of spades. How could the agent deduce what cards were possessed by the opponents?
First, calculate all the various ways that the pack of cards could possibly have been distributed among the various players
Then, systematically eliminate all those configurations which are not possible given what the agent knows (any configuration in which the agent did not possess the ace of spades could be rejected).
Each configuration remaining after this is a world: a state of affairs considered possible given what the agent knows (aka **epistemic alternatives**).
We can notice that something true in all our agent's possibilities is known by the agent.

epistemic alternatives

This approach/representation can be represented with a new form of logic known as **Modal logic**.

Modal logic

7.3.2 Modal Logic

Possible worlds can be incorpored into the semantic framework of modal logic. In fact, modal logic was used by philosophers to investigate different **modes of truth** (e.g. possibly true and neceessarility true).

modes of truth

In the study of agents, it is used to give meaning to concepts such as belief and knowledge.
Modal logic can be considered as the logical theory of necessity and possibilty and it is essentially classical propositional logic extended by two operators:

- **Necessity** ($\Box$)
- **Possibility** ($\Diamond$)

Necessity

Possibility

And as a consequence, the syntax will be extended. Let $S = \{p, q, ...\}$ be a set of atomic propositions

- If $p \in S$ then p is a formula
- If A, B are formulae, then so are $\neg A$ and $A \wedge B$
- if A is a formulae, then so are $\Box A$ and $\Diamond A$

other connectives can be expressed by abbreviation:

$$F \rightarrow G \equiv \neg F \vee Gq \qquad \neg(F \wedge G) \equiv \neg F \vee \neg G$$

This new operators can be expressed one from another, this is called duality of operators:

$$\Box A \equiv \neg \Diamond \neg A \qquad \Diamond\, A \equiv \neg \Box \neg A$$

In addition there are two basic properties that are true in this system:

1. **K axiom schema**
$$\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$$
2. **Necessitation rule**: if A is valid, then $\Box A$ is valid (A is true in all interpretation)

K axiom schema

Necessitation rule

The semantics of modal logic is traditionally given in terms of possible worlds:

- The formula $\Box A$ is true if A is true in every world accessible from the current world
- The formula $\Diamond A$ is true if A is true in at least one world accessible from the current world

With sets of worlds as primitive, the structure of the model is captured by relating the different worlds via a binary accessibility relation.

Given all the above formalizing possible worlds via **Kripke structure**: Kripke structure

$$(S, \pi, K_1, ..., K_n$$

- S is the set of possible worlds
- π is the set of formulae true at a world
- K_i is a binary accebility relation on S (a set of pairs of elements of S)

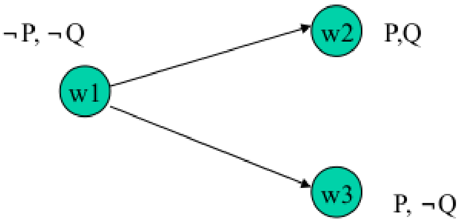


Figure 7.1: Worlds w_2 and w_3 are accessible from world $w1$: modal logic states $\Box P$ and $\Diamond Q$. We can say nothing about world w_1 since it is not accessible/visible

This accessibility relation is given by a set of possible properties related to the worlds:

- **Reflexive** (a world can see itself) Reflexive
  For all $s \in S$, we have $(s, s) \in K$
- **Symmetric** (two worlds are mutually accessible) Symmetric
  For all $s, t \in S$, we have $(s, t) \in K \iff (t, s) \in K$
- **Transitive** Transitive
  For all $s, t, u \in S$, we have that if

$$(s,t) \in K \qquad \wedge (t,u) \in K \quad \text{then} \quad (s,u) \in K$$

- **Serial** (a world can access at least another world) Serial
  For all $s \in S$ there is some t such that $(s, t) \in K$
- **Euclidean** Euclidean
  For all $s, t, u \in S$ whenever

$$(s,t) \in K \quad \wedge \quad (s,u) \in K \quad \text{then} \quad (t,u) \in K$$

From the above we deduce that:
If K is Reflexive and Euclidean, then K is symmetric and transitive
If K is Symmetric and transitive, then K is Euclidean

The following are equivalent:

- K is reflexive, simmetric and transitive
- K is symmetric, transitive and serial
- K is reflexive and Euclidean

Hence we determine which world are accessible by analysing the properties of the relations between such worlds.

Properties of accessibility relation can be expressed by axiom schemas:

- **T axiom**: corresponds to reflexive accessibility relation T axiom

$$\Box A \rightarrow A$$

- **D axiom**: corresponds to serial accessibility relation D axiom

$$\Box A \rightarrow \Diamond A$$

- **4 axiom**: corresponds to transitive accessibility relation 4 axiom

$$\Box A \rightarrow \Box\Box A$$

- **5 axiom**: corresponds to Euclidean accessibility relation 5 axiom

$$\Diamond A \rightarrow]square \Diamond A$$

Depending on which axiom we include in our logic we can derive different types of logic which are referenced with the notations:

$$S5(KT5) \qquad S4(KT4) \qquad T(KT) \qquad weak - S5(KD45)$$

This is important since for belief and knowledge logic there will be different sets of axioms.

7.4 Logic of Knowledge

Finally, we have all the tools to formalise knowledge, and this will be done by changing the meaning or interpretation of the symbols, not the axiom of the modal logic.

The formula $\Box A$ is read "It is known that A" of "agent knows A" and denoted as K
For group knowledge we have an indexed set of modal operators:

$$K_1, ..., K_n \quad \text{for } \Box$$

In this frame of reference $K_1 A$ is read as "agent1 knows A"

As a consequence, we can interpret the 4 axiom of accessibility relation:

- T axiom (Knowledge axiom) $K_i A \rightarrow A$
  "what is known is true"
- D axiom $K_i A \rightarrow \neg K_i \neg A$
  "If agent i knows A then agent i does not know $\neg A$"
- 4 axiom (positive introspection) $K_i A \rightarrow K_i K_i A$
  "If agent i knows A then agents i knows that it knows A"
- 5 axiom (negative introspection) $\neg K_i \neg A \rightarrow K_i \neg K_i \neg A$
  "agent i is aware of what it does not know"

Knowledge is often defined as true belief, i.e. an agent knows A if the agent believs A and A is true.
Axioms KTD45 are often chosen as a logic of knowledge.
Axioms KD45 are often chosen as a logic of belief (T does not hold since what is believed is not necessarily true).

How well does normal modal logic serves as a logic of knowledge and belief? There are two main problems related to the K axiom and the necessitation rule. In fact, the interpretation changes from the logic of belief and the logic of knowledge

- Necessitation rule: an agent knows all valid formulae (an agent will have an infinite number of items of knowledge)
  If A is valid, then $\Box A$ is valid
- K axiom: agent's knowledge is closed under implication

$$\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$$

This leads to the logical **Logical omniscience problem**: constituted by that of knowing all valid formulae and that of knowledge/belief being closed under consequence (must know all logical consequences of one's knowledge or belief).

Logical omniscience problem

This means that in this system logic, agents believe/know all valid formulae and agents' beliefs/knowledge are closed under logical consequence.

Other approaches that were propesed to avoid the logical omniscience problem are:

1. **Levesque**
   In order to avoid logical omniscience problem a distinctrion is made between explicit (what the agent knows by experience and perception) and implicit belief (what the agent deduce from what is known).

   Levesque

2. **Konolige**
   Deduction model of belief
   The deduction model defines a belief system as a tuple

   $$d = (\Delta, \rho)$$

   containing a base set of formulae in some internal, logical language of belief and a set of deduction rules (may be incomplete) for deriving new beliefs
   Agent with such a belief system believe A if A can be derived from its base set using its deduction rules

   Konolige

7.5 BDI architecture

We have all the tools to apply this logic and consider the combinations of a small set of attitudes and see how they can be expressed in formal theory.
Systems and formalisms that give primary importance to intentions are often referred to as **BDI architecture**.
Formalization of intentions based on branching-time possible worlds future and single past model.

BDI architecture

In addition to possible worlds semantics Rao and Georgeff, decided to consider what happens inside a single world and they find out the relations of what happens in one world between different attitudes

Crucial elements are :

- Intentions are treated on a pair with beliefs and goals
- Distinguishes between choices and the possibilities of the different outcomes of actions
- Interrelationship between belief, goals and intentions are specified (goals are consisten desires of an agent)

They formulated some informal semantics:

- The world is modeled by using a temporal structure with a branching time future and a single past, this is called a **time tree**
- A particular time point in a particular world is a **situation**
- Event types transform one time point to another
- Primitive events are those events directly performable by the agent and uniquely determine the next time point
- The brances in a time tree represent the choices available to an agent

time tree

situation

To model this informal semantics they chose to use 2 modal operators:

- **Optional**: a path formula (path along a time tree) is said to be optional if, at a particular time point in a time tree, it is true of at least one path emanating from the point
- **Inevitable**: a path formula is said to be inevitable if it true of all paths emanating from that point

Optional

Inevitable

In addition, they provided some additional **Temporal operators**: next, eventually (appears at the end of a path in time tree), always (appears in every node of a path) and until.
A combination of these modalities can be used to describe the options available to an agent.

Temporal operators

BDI Architecture states that there are three basic attitudes:

1. **Beliefs**
   In each situtation a set of belief-accessible worlds is associate. Those worlds that the agent

   Beliefs

believes to be possible.
Each belief-accessible world is a time-tree

2. **Goals/Desires** Goals/Desires

   For each situation a set of goal-accessible worlds is associated. Those represent the goals of the agent.
   Goals are chosen desires of the agent that are consistent and agent should believe that the goal is achievable; goals must be compatible with beliefs.
   For each belief-accessible world w in time t, there must be a goal accessible sub-worlds of w at time t.

   In particular goals are desires that we believe to be achievable.
   In each world, there is a time tree that can be structured as a belief structure. Goals/desires are substructure of such structure because we choose only the paths/goals that we believe achievable (scuh substructure is called goal tree).

3. **Intentions** Intentions

   Intentions are represented by a set of intention-accessible worlds
   These worlds are ones that an agent has committed to attempt to realize
   The intention-accessible worlds of an agent must be compatible with its goal-accessible worlds
   For each goal-accessible world w in time t, there must be an intention accessible sub world of w at time t

An agent perceivs the world and understand what is believable. From what is believable some of these points can be chosen as goals. And from this we can choose some goals (intentions) to which we commit to execute.

Chapter 8

Agent Architecture

8.1 Abtract Architectures

Let us make formal the abstract view of agents presented so far. First, let us assume that the evironment may be in any of a finite set E if discrete instantaneous states:

$$E = \{e, e', ...\}$$

This is a fairly standard modelling assumption, which we can justify by pointing out that any continuous environment can be modelled by a discrete environment to any desired degree of accuracy.

Agents are assumed to have a repertoire of possible actions available to them, which tranform the state of the environment.
Let

$$Ac = \{\alpha, \alpha', ...\}$$

be the finite set of actions.

The basic model of agents interacting with their environments goes as follow:

1. The environment starts in some state
2. The agent begins by choosing an action to perform on that state

3. As a result of this action, the environment can respond with a number of possible states, however, only one state will actually result, though, of course, the agent does not know in advance which will it be.

4. On the basis of this second state, the agent again chooses an action to perform.

5. The environment responds with one of a set of possible states

6. ...

A **run** (r) of an agent in an environment is thus a sequence of interleaved environment states and actions

run

$$r : e_0 \rightarrow \alpha_0 \, e_1 \rightarrow \alpha_1 \, e_2 \rightarrow \alpha_2 \, e_3 \rightarrow ... \rightarrow \alpha_{u-1} \, e_u$$

Let

- $\mathcal{R}$ be the set of all such possible finite sequeces (over E and Ac)
- $\mathcal{R}^{Ac}$ be the subset of these that end with an action
- $\mathcal{R}^{E}$ be the subset of these that end with an environment state

We will use $r, r', ...$ to stand for members of $\mathcal{R}$.

In order to represent the effect that an agent's actions have on an environment we introduce a **state tranformer function**

state tranformer function

$$\tau : \mathcal{R}^{Ac} \rightarrow 2^{E}$$

Thus a state tranformer function maps a run (assumed to end with the action of an agent) to a set of possible environment states, those that could result from performing the action.

There are two important points to note about this definitions:

- Environments are assumed to be **history dependent**, i.e. the next state of an environment is not solely determined by the action performed by the agent and the current state of the environment. The actions made earlier by the agent also play a part in determining the current state
- This definition allows for **non-determinism** in the environment. There is thus uncertainty about the result of performing an action in some state

history dependent

non-determinism

If $\tau(r)0\emptyset$ where r is assumed to end with an action, then there are no possible successor states to r. In this case, we say that the system has ended its run. We will also assume that all runs eventually terminate.

Formally, we say that an environment Env is triple

$$Env = \langle E, e_0, \tau \rangle$$

where:

- E is a set of environment states
- $e_0 \in E$ is an initial state
- τ is a state transformer function

We now need to introduce a model of agents that inhabit systems. We model agents as functions which map runs (assumed to end with an environment state) to actions

$$Ag : \mathcal{R}^{E} \rightarrow Ac$$

Thus an agent makes a decision about what action to perform based on the history of the system that it was witnessed to date.

Notice that while environments are implicitly non-deterministic, agents are assumed to be deterministic.

We say a **system** is a pair containing an agent and an environment. Any system will have associated with it a set of possible runs; we denote the set of runs of agent Ag in an environment Env by

system

$$\mathcal{R}(Ag, Env)$$

which, for simplicity, contains only terminated runs.

Two agents Ag_1 and Ag_2 are said to be **behaviourally equivalent wrt environment** Env is and only if

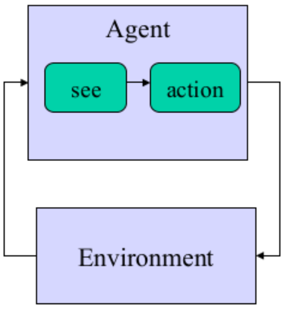
behaviourally equivalent wrt environment

$$\mathcal{R}(Ag_1, Env) = \mathcal{R}(Ag_2, Env)$$

And **behaviourally equivalent** if they are behaviourally equivalent with respect to all environments

behaviourally equivalent

8.1.1 Purely reactive agents



Certain types of agents decide what to do without reference to their history. They base their decision-making entirely on the present, with not reference at all to the past.
We will call such agents **purely reactive** or **tropistic**, since they simply respond directly to their environment.

purely reactive

tropistic

Given that tropistic agents makes decision based only on the present state of the environment, there is no reason for them to hold a state.

Formally, the behaviour of a purely reactive agent can be represented by a function

$$Ag : E \to Ac$$

A thermostat agent is an example of a purely reactive agent

8.1.2 Agents with state

A purely reactive agents is of no help to construct and implement agents, since its representation is extremely simple and abstract. For this reason, we need to refine our abstract model of agents, and make design choices that mostly relate to the sybsystems that go to make up an agent (i.e. what data and control structures will be present).

An **agent architecture** is eesentially a map of internals of an agent, its data structures, the operations that may be performed on these data structures and the control flow of between these data structures.
We are not partiularly interested in the content of an agent's state, but rather on the overall role of state in the agent's decision making loop.

agent architecture

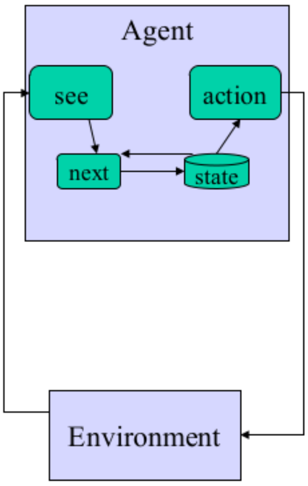
Thus agents have some internal data structure, which is tipically used to record information about the environment state and history. Let I be the set of all internal state of the agent. An agent's decision-making process is then based, at least in part, on this information.

The perception function *see* represents the agent's ability to obtain information from its environment. The output of the *see* function is a *percept* (perceptual input).
Let Per be a non-empty set of percepts. Then *see* is a functionof the form:

$$see : E \to Per$$

The action-selection function *action* is defined as a mapping:

$$action : I \to Ac$$



from internal states to actions. An additional function *next* is introduced, which maps an internal state and percept to an internal state:

$$next : I \times Per \rightarrow I$$

The behaviour of a state-based agent can be summarized in the following way:

1. The agent start in some initial internal state i_0
2. It observes its environment state e and generates a percept $see(e)$
3. The internal state of the agent is then updated via the *next* function

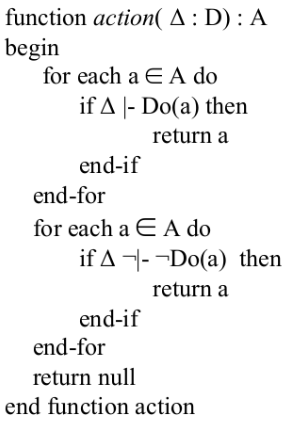
$$next(i_0, see(e))$$

4. The action selected by the agent is then

$$action(next(i_0.see(e)))$$

5. This action is then performed, and the agent enters another cycle

It is worth observing that state-based agents as defined here are in fact no more powerful than the standard agents we introduced earlier. In fact they are identical in their expressive power: every state-based agent can be transformed into a standard agent that is behaviourally equivalent.



```
function action( Δ : D) : A
begin
    for each a ∈ A do
        if Δ |- Do(a) then
                return a
        end-if
    end-for
    for each a ∈ A do
        if Δ ¬|- ¬Do(a)  then
                return a
        end-if
    end-for
    return null
end function action
```

8.2 Deliberative Architectures

8.2.1 Practical Reasoning

The particular model of decision-making is known as **practical reasoning**. Practical reasoning is reasoning directed towards actions, or, in other terms, the process of figuring out what to do.

practical reasoning

“Practical reasoning is a mattter of weighting conflicting considerations for and against competing options, where the relevant considerations are provided by what the agent desires/values/cares about and what the agent believes”.

It is important to distinguish practical reasoning from **theoretical reasoning**. Theoretical reasoning is directed towards beliefs.

theoretical reasoning

Human practical reasoning appears to consist of at least two distinct activities:

1. **Deliberation**: deciding what state of affairs we want to achieve

Deliberation

2. **Means-ends reasoning**: deciding how we want to achieve these states of affairs

Means-ends reasoning

Hence, the output of the deliberation process are intentions, whereas the end result of means ends reasoning is a **plan** or **recipes** os some kind for achieving the chosen state of affairs.

plan

recipes

Deliberation

First notice that it is possible to distinguish several different types of intention: actions and states of mind.

In this section, we will characterize intentions as states of mind or **future-directed intentions**, i.e. intentions that an agent has towards some future state of affairs.

future-directed intentions

The most obvious role of intentions is that they are pro-attitudes which means that they tend to lead to action.

The second main property of intentions is that they persist, i.e. If an agent adopt an intention, then the agent should persist with this intention and attempt to achieve it.

The third main property of intentions is that once an agent has adopted an intention, the very fact of having this intention will contrain its future practical reasoning. This means that an agent while holding a particular intention will not entertain options that are inconsisten with that intention. This is a highly desirable property from the point of view of implementing rational agents, because in providing a **filter of admissibility**, intentions can be seen to constrain the space of possible intentions that an agent needs to consider.

filter of admissibility

Finally, intentions are closely related to beliefs about the future.

Form this discussion, we can identify the following closely related situations:

- Having an intention to bring about ϕ, while believing that you will not bring about ϕ, is called **intention-belief inconsistency**, and is not rational
- Having an intention to achieve ϕ without believing that ϕ will be the case is **intention-belief incompleteness**, and is an acceptable property of rational agents

intention-belief inconsistency

intention-belief incompleteness

The distinction between these two cases is known as **asymmetry thesis**.
Summarizing, we can see that intentions play the following important roles in practical reasoning

asymmetry thesis

- Intentions drive means-ends reasoning
- Intentions persists
- Intentions constrain future deliberation
- Intentions influence belief upon which future practical reasoning is based

The process of deliberation or generation of intentions can be modelled via two functions:

- an **option generation** function
  This function takes the agent’s current beliefs and current intentions and on the basis of these produces a set of possible options and desires.
- a **filtering** function
  In order to select between competing options, an agent uses a filter function. Intuitively, the filter funciton must simplty select the best options for the agent to commit to.

option generation

filtering

Addiionally an agent's belief update process is modelled though a **belief revision function**

belief revision function

Means-Ends Reasoning

Means-ends reasoning is the process of deciding how to achieve an end/intention using the available means/actions.
Means ends reasoning is perhaps better known in the AI community as **planning**: planning is essentially automatic planning, in which a planner is a system that takes as input representations of the following:

planning

- A goal, intention or a task
- The current state of the environment: the agent's beliefs
- The actions available to the agent

As output, a planning algorithm generates a plan.
The first real planner was the STRIPS system. The two basic components of STRIPS were :

1. a model of the world as a set of formulae of first-order logic

2. a set of action chemata, which describe the preconditions and effects of all the actions available to the agent.

The STRIPS planning algorithm was based on a principle of finding the difference between the current state of the world and the goal state, and reducing the difference by applying an action. Unfortunately, this proved to be an inefficient process for formulating plans, as STRIPS tended to become lost in low-level plan detail.

Deductive Reasoning Agents

The traditional approach to building artificially intelligent systems, known as symbolic AI, suggests that intelligent behaviour can be generated in a system by giving the system a symbolic representtion of its environmenta and its desired behaviour, and syntactically manipulating this representation.
Hence we define a **deliberative agent** or **deliberative agent architecture** to be one that:

deliberative agent

deliberative agent architecture

- Contains an explicitly represented, symbolic model of the world
- Makes decision via symbolic reasoning

In order to build such an agent, it seems we must solve two key problems:

- The **transduction problem**
  The problem of translating the real world into an accurate, adequate symbolic description of the world, in time for that description to be useful
- The **representation/ reasoning problem**
  The problem of representing information symbolically, and getting agents to manipulate/reason with it, in time for the results to be useful.

transduction problem

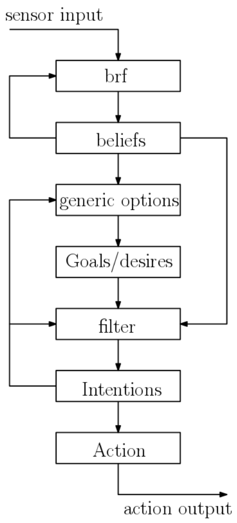
representation/ reasoning problem

8.2.2 BDI - Belief, Desire, Intentions

We can now discuss the overall control structiure of a practical reasoning agent. The basic strucutre of the decision-making process is a loop in which the agent continually:

- Observes the world and updates beliefs
- deliberates to decide what intention to achieve
- uses means-ends reasoning to find a plan to achieve these intentions
- executes the plan

However, this basic control loop is complicated by a number of concerns. The first of these is that of commitment and, in particular, how committed an agent si to both ends (intentions) and means (the plan to achieve the intention)

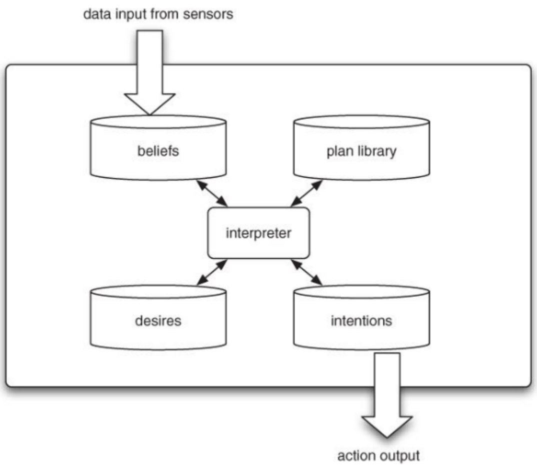


```
function action(p : E) : A
begin
  B := brf(B,p)
  D := options(B, I)
  I := filter(B,D,I)
  return execute(I)
end function action
```

8.2.3 PRS - Procedural Reasoning Systems

The **Procedural Reasoning System (PRS)** was perhaps the first agent architecture to explicitly embody the BDI paradigm, and proved to be the most durable agent architecture developed to date.

Procedural Reasoning System (PRS)



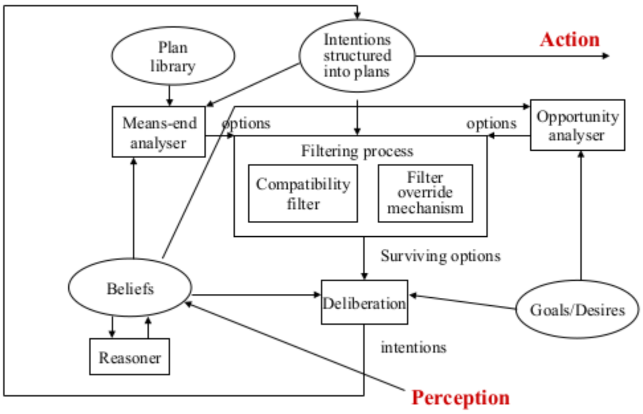
In the PRS, an agent does no planning from first principles. Instead, it is equipped with a library of pre-compiled plans (which effectively represent the agent's procedural knowledge). These plans are manually constructed, in advance, by the agent programmer.

Plans in the PRS each have the following components:

- A goal: the postcondition of the plan
- A context: the precondition of the plan
- A body: the recipe part of the plan or the course of action to carry out.

8.2.4 IRMA - Intelligent Resource-Bounded Machine Architecture

IRMA has 4 key symbolic data structures:



1. A plan library
2. Beliefs: information available to the agent
3. Goals/Desires: those things that the agent would like to make true
4. Intentions: goals/desires that the agent has chosen and committed to.

Additionally, the architecture has:

- A reasoner: for reasoning about the world, an inference engine
- A means-end analyser: determines which plans might be used to achieve the intentions
- An opportunity analyser: monitors the environemnt and as a result of changes, generates new options
- A filtering process: determines which options are compatible with current intentions
- A deliberation process: responsible for deciding upon the best intentions to adopt

8.3 Reactive Architectures

The many problems with symbolic/logical approaches to building agent led some researchers to question and unltimately reject, the assumptions upon which such approaches are based.
In the mid to late 1980s, these researchers began to investigate alternatives to the symbolic AI paradigm. It is difficult to neatly characterize these different approaches, since their advocates are united mainly by a rejection of symbolic AI, rather than by a common manifesto. However, certain themes do recur:

- The rejection of symbolic representations and fo decision making based on syntactiv manipulation of such representations
- The idea that intelligent, rational behaviour is seen as innately linked to the environment an agent occupies: intelligent behaviour is not disembodies but is a product of the interaction the agent maintains with its environment
- The idea that intelligent behaviour emerges from the interaction of various simpler behaviours

Alternative appraches to agency are sometimes referred to as **behavioural situated** and **reactive** (because such systems are often perceived as simply reactive to an environment, without reasoning about it).

behavioural situated

reactive

8.3.1 The subsumption architecture

Subsumption architecture is arguably the best-known reactive agent architecture.
It was developed by Rodney Brooks, who ahs propounded three key theses that have guided his work:

Subsumption architecture

1. Intelligent behaviour can be generated without explicit representation of the kind that symbolic AI proposes
2. Intelligent behaviour can be generated wihout explicit abstract reasoning of the kind that symbolic AI proposes
3. Intelligence is an emergent property of certain complex systems

Brooks also identifies two key ideas that have informed his research.

- **Situatedness and embodiment**: real intelligence is situated in the world, not in disembodies sytems such as theorem provers or expert systems
- **Intelligence and emergence**: intelligent behaviour arises as a result of an agent's interaction with its environment. Also, intelligence is in the eye of the beholder, it is not an innate isolated property.

Situatedness and embodiment

Intelligence and emergence

These ideas were made concrete in the subsumption architecture. There are two defining characteristics of the subsumption architecture. The first is that an agent's decision making is realized througha set of task-accomplishing behaviours; each behaviour may be thought as an individual action seleciton function which continually takes perceptual input and maps it to an action to perform.

An important point to note is that these task-accomplishing modules are assumed to include no complex symbolic representation and are assumed to do no symbolic reasoning at all. In many implementations, these behaviours are implemented as rules which simply map perceptual input directly to actions. In this sense, the architecture is **reactive** and **situated** (since agents do not take past events into account and cannot forsee the future).

The second defining characteristic of the subsumption architecture is that many behaviours can fire simulataneously. There must obviously be a mechanism to choose between the different actions selected by these multiple actions.

Brooks propsoed arranging the modules into a **subsumption hierarchy**, with the behaviours arranged into **layers**. Lower players in thew hierarchy are able to inhibit higher layers (the lower the layer the higher the priority)

subsumption hierarchy

layers

Steel's Mars explorer experiments

To illustrate the subsumption architecture in more detail, we will look at the Steels's Mars explorer experiments: "The objective is to explore a distant planet – more concretely, to collect samples of a particular type of precious rock. The location of the rock samples is not known in advance, but they are typically clustered in certain spots. A number of autonomous vehicles are available that can drive around the planet collecting samples and later re-enter a mother ship spacecraft to go back to Earth. There is no detailed map of the planet available, although it is known that the terrain is full of obstacles – hills, valleys, etc. – which prevent the vehicles from exchanging any communication"

Steels's showed that the subsumption architecture would be a near optimal architecture for the individual agents.

The solution makes use of two mechanisms:

- The **gradient field**.
  In order that agents can know in which direction the mother ship lies.
- an **indirect communication method**

gradient field

indirect communication method

The behaviour of an individual agent is then built up from a number of behaviours.

For individual (non-cooperative) agents, the lowest-level behaviour is obstacle avoidance. The behaivour can be represented in the rule:

- Avoid obstacles
- Any samples carried by agents are dropped back at the mother-ship
- Agents carrying samples will return to the mother-ship
- Agents will collecty samples they find
- An agent with nothing better to do will explore randomly

```
function action(p : P) : A
begin
  fired := $\{(c_i, a) | (c_i, a) \in R$ and $p \in c_i\}$
  for each $(c_i, a) \in$ fired do
    if $\neg(\exists(c_{i-1}, a') \in$ fired such that $(c_{i-1}, a') < (c_i, a))$ then
      return a
    end if
  end for
  return null
end function action
```

Where :

- (c, a) is the condition action pair
- R is the set of condition-action pairs
- $c_1 < c_2$ is read as c_1 is lower in hierarchy than c_2

8.3.2 Situated automata

Rosenchein and Kaelbling pointed out that, just because we might conceptualize, of specify the behaviour of an agent in terms of concepts like beliefs and goals and indeed might use a logical representation of beliefs and goals to specify the agent, this does not implyt hat an agent must be implemented as a theorem prover for a logic of beliefs and goals.

In their **situated automata** paradigm, an agent is specified in a logic of beliefs and goals, but this specification is then compiled down to a digital machine, which satisfies the beliefs, goal specification in a precise formal sense.

situated automata

The resulting digital machine can operate in a provably time-bounded fashion; it does not do any symbol manipulation, and in fact no symbol expressions are represented in the machine at all. The logic used to specify an agent is essentially a logic of knowledge.

An agent is specified in terms of two components: perception and action.

Perception is specified by three components:

1. semantics of agent's input
2. a set of static facts
3. a specification of state transitions

whereas action is specified by semantics of output.

The compiler syntheses a circuit whose output will have the correct semantics but that does not represent or manipulate symbolic expressions. This means that all symbolic manipulation is done at compile time.

8.4 Hybrid Architectures

Given the requirement that an agent be capable of reactive and proactive behaviour, an obvious decomposition involves creating separate subsystems to deal with these different types of behaviours. This idea leads naturally to a class of architectures in which the various subsystems are arranged into a hierarchy of interacting layers.

Tipically, there will be at least two layers: to deal with reactive and proactive behaviours respectively. In principle, there is no reason why there should not be many more layers.
It is useful to characterize such architectures in terms of the information and control flows withing the layers. Broadly speaking, we can identify two types of control flow within layered architectures as follows:

- **Horizontal Layering**
  In horizontally layered architectures, the software layers are each directly connected to the sensory input and action output. In effect each layer itself acts like an agent producing suggestions as to what action to perform

Horizontal Layering

Vertical Layering

- **Vertical Layering**
  In vertically layered architectures, sensory input and action output are each dealt with by at most one layer

mediator

The great advantage of horizontally layered architectures is their conceptual simplicity: if we need an agent to exhibit n different types of behaivour then we implement n different layers. However, because the layers are each in effect competing with one another to generate action suggestions, there is a danger that the overall behaviour of the agent will not be coherent. In order to ensure conistency a **mediator** function, must decide about which layer has control of the agent at any given time.

Such central control is clearly difficult form a design point of view in any but the most simple system. The introduction of a central control system also introduces a bottleneck into the agents' decision making.

These problems are partly allievated in a vartically layeres achitecture. We can subdivide vertically layered architectures into:

one-pass architectures

two-pass architectures

- **one-pass architectures**
  Control flows sequenctially through each layer, until the final layer generates action output
- **two-pass architectures**
  Information flows up the architecture (the first pass) and control then flows back down.

In both these architectures the complexity of interactions between layers is reduced and as a consequence it is much simpler than the horizontally layered case. However, this simplicity comes at the cost of some flexibility: in order for a vertically layered architecture to make a decision, control must pass between each different layer. This is not fault tolerant, i.e. failures in any one layer are likely to have serious consequences for agent performance.

8.4.1 Touring Machines

Touring Marchine architecture

actively producing layers

reactive layer

The **Touring Marchine architecture**, consists of three **actively producing layers**, i.e. each layer continually produces suggestions for what actions the agent should perform:

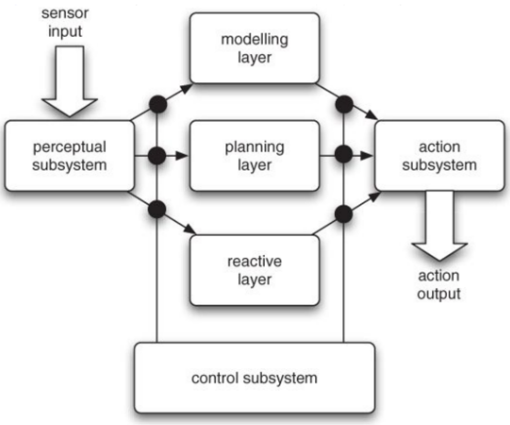
- the **reactive layer** provides a more-or-less immediate response to changes that occur in the environment. It is implemented as a set of situation-action rules similarly to the subsumption architecture.
  These rules map sensor input directly to effector output. The rules can only make references to the agent's current state, but they cannot do any explicit reasoning about the world.
  It is worth pointing out that the result of rules are actions not predicates, which means that if the rule fired, it would not result in any central environment model being update, but would just result in an action being suggested by the reactive layer.
- the **planning layer** achieves the agent's proactive behaviour. Specifically, the planning layer is responsible for the running of the agent under normal circumstances.
  The planning layer employs a library of plan 'skeletons' called **schemas**, which are in essence hierarchically strcutured plans that the touring machines plannign layer elaborates at run-time in order to decide what to do.
  So, in order to achieve a goal, the planning layer attempts to find a schema in its library which matches that goal. The schema will contain subgoals, which the planning layer elaborates by attempting to find other schemas in its plan library that match these subgoals
- the **modelling layer** represents the various entities in the world (including the agent itself, as well as other agents). The modelling layer thus predicts conflicts between agents and generates new goals to be achieved in order to resolve these conflicts. Such goals are then passed to the planning layer.

planning layer

schemas

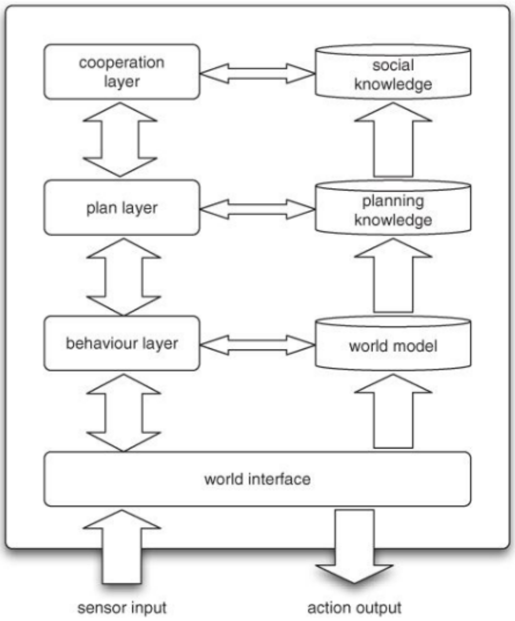
modelling layer

The three control layers are embedded within a control subsystem which is effectively responsible for deciding which of the layers should have control over the agent. This control subsystem is implemented as a set of control rules.
Control rules can either suppress sensor information between the control rules and the control layers, or else censor action outputs from the control layers.



8.4.2 InteRRaP

InteRRaP is a vertically layered two pass agent architecture that contrains three control layers. The purpose of each InteRRaP layer apperas to be rather similar to the purpose of each corresponding Touring Machines layer.



Thus :

- The lowest (**behaviour-based**) layer deals with reactive behaviour
- The middle (**local planning**) layer deals with every day planning to achieve the agent's goals
- The uppermost (**cooperative planning**) layer deals with social interactions.

behaviour-based

local planning

cooperative planning

Each layer has associated with it a knowledge base, i.e. a representation of the world appropriate for that layer. The se different knowledge bases represents the agent and its environment at different levels of abstraction.

Thus:

- The highest-level knowledge base represents the plans and actions of other agents in the environment
- The middle-level knowledge base represents the plans and actions of the agent itself
- The lowest-level knowledge base represents raw information about the environment

Layers interact with each other to achieve the same end. The two main types of interaction between layers are **bottom-up activation** and **top-down execution**. The former occurs when a lower layer passes control to a higher layer because it is not competent to deal with the current situation, the latter occurs when a higher layer makes use of the facilities provided by a lower layer to achieve one of its goals.

bottom-up activation

top-down execution

The basic flow of control in InteRRaP begins when perceptual input arrives at the lowest layer in the architecture.
If the reactive layer can deal with this input, then it will do so, otherwise, bottom-up activation will occur, and control will be passed to the local planning layer.
If the local planning layer can handle the situation, then it will do so, tipically by making use of top-down execution, otherwise, it will use bottom-up activation to pass control to the highest layer.

In this way, control in InteRRaP will flow from the lowest layer to higher layers of the architecture, and then back down again.

It is worth noting that each layer implements two general functions:

1. **situation recognition and goal activation** function which maps a knowledge base and current goals to a new set of goals

situation recognition and goal activation

2. **planning and scheduling** function whcih is responsible for selecting which plans to execute, based on the current plans, goals, and knowledge base of that layer.

planning and scheduling

Chapter 9

Mobile Agents

Mobile agents are agents that are capable of transmitting themselves, their program and their state, across a computer network, and recommencing execution at a remote site. Mobile agents

The program in this way can choose when and where to migrate, therefore suspending its execution at an arbitrary point and transport itself to another machine where it would resume execution.

The original motivation behind mobile agents is fairly simple: the idea was that mobile agents would replace remote procedure call as a way for processes to communicate over a network.

9.1 Remote Procedure Calls vs Mobile Agents

9.1.1 Remote Procedure Calls

With remote procedure calls the idea is that one process can invoke a procedure (method) on another process which is remotely located.

Suppose one process A invokes a method m on process B with arguments $args$; the value returned by process B is to be assigned to a variable v. This mechanism can be summarized with the notation:

$$v = B.m(args)$$

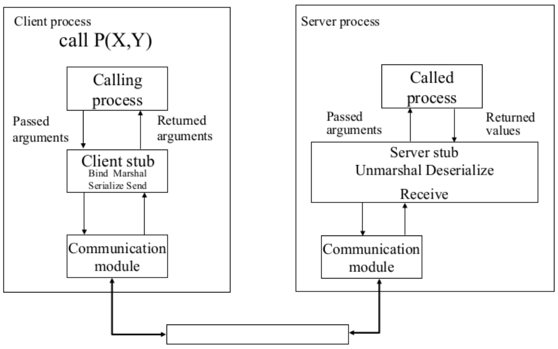
Crucially, in remote procedure calls, communication is **synchronous**, i.e. process A blocks from the time it starts executing the instruction until the time that B returns a value. If B never returns a value, because the network fails, then A may remain indefinitely suspended, waiting for a reply that will never come. synchronous

The network connection between A and B may well also remain open, and even though it is largely unused (no data is being sent for most of the time), this may be costly.

In other terms, remote procedure calls enable one computer to call procedures in another. In order to do so however, the two computers must agree in advance upon a protocol: the effects of each remotely accessible procedure and the types of its arguments and results.

Each interaction entails two acts of communication: request and acknowledge, which, consequently, means that the ongoing interaction requires ongoing remote communication.

The overall general architecture is depicted in the following figure:



9.1.2 Remote execution

The idea of mobile agents is to replace the remote procedure call by sending out an agent to do the computation. Thus instead of invoking a method:

1. Process A sends out a program/a mobile agent to process B.

2. the program then interacts with process B

3. Since the agent shares the same address space as B, these interactions cas be carried out much more efficiently than if the same interactions were carried out over a network.

4. When the agent has completed its interactions, it returns to A with the required result.

During the entire operation, the only network time required is that to send the agent to B, and that required to return the agent to A when it has completed its task.

This is potentially a much more efficient use of netwrok resources than the remote procedure call alternative.

One computer not only calls procedures on another computer, but also provides the procedures. Hence each exchanged message contains the procedure and its arguments.

The two computers agree in advance upon a language, i.e. instructions and the types of data that are allowed.
A user agent and a server can interact without using the network once the agent is transported which means that the ongoing interaction does not require ongoing remote communication.

In the frame of reference of remote executing a mobile agent is a program that can migrate from machine to machine in a heterogeneous network. Moreover, the program is able to choose when and where to migrate and in doing so it will suspend its execution at an arbitrary point, transport itself to another machien and resume execution.

The distintion at the core between pure mobile agents and remote execution is that:

1. A program which is sent without execution state to remote CPU executes there, possibly communicating with other CPUs and then terminates (remote execution)

2. A program which carries execution state with it and is sent to a remote CPU executes there possibly communicating with other CPUs and then moves again to a third CPI or return to its origin (mobile agent)

9.1.3 Advantages of RE over RPC

We can distinguish to main classes of advantages of remote execution over remote procedure calls:

1. **Tactical**:

   Tactical

   - **Performance**: due to less message passing over the network

     Performance
   - **Less connection time**: need network conneciton only to transport the agent

     Less connection time
2. **Strategic**:

   Strategic

   - **Customization**: agents let manufacturers of client sw extend the funcitonslities of the server sw.

     Customization
   - In a RPC application, the server component needs to be statically installed by the user. In RE, they are dynamically installed by the application itself which is an agent.
   - The use of new RPC-based applications must be a decision that has to be taken by the providers.
     The use of new RE-based applications must be a decision that has to be taken by the user.
   - A public network becomes like a platform

9.2 Mobile Agent Environment

A mobile agent environment is a software system which is distributed over a network of heterogeneous computers. Its primary task is to provide an environment in which mobile agents can execute.
It implements the majority of models which appear in the mobile agent definition.

A mobile agent environment is responsible to:

- Support services related to the mobile agent environment itself
- Support services pertaining to the environments on which the mobile agent environment is built
- Provide services to support access to other mobile agent systems
- Provide supp9ort for openness when accessing non-agent-based software environments

9.3 Issues with Mobile agents

There are a number of technical issues that arise when considering mobile agents:

- **Serialization**, how is the agent serialized (i.e. encoded in a form suitable to be sent across the network), and, in particular, what aspects of the agent are serialized, the program, the data, or the program and its data?

  Serialization
- **Hosting and remote execution**,what the agent arrives at its destincation, how is it executed, for example ifg the original host of the agent employs a different operating system or processor from the desitination host?

  Hosting and remote execution
- **Security**, when the agent freom A is sent to the computer that hosts process B, there is obvious potential from the agent to cause trouble. It could potentially do this in a number of ways:

  Security

  - It might obtain sensitive information by reading filestore or RAM directly
  - It might deny service to other processes on the host machine, by either occupying too much of the available processing resource (processor cycles or memory) or else by causing the host machine to malfunction (e.g. writing over the machine's RAM)
  - It might simply cause irritation and annoyance , for example by causing windows to pop up on the user's GUI

Many different answers have been developed to address these issues.

With respect to the first issue, that of how to serialize and transmit an agent there are several possibilities:

- Both the agent and its state are transmitted and the state includes the program counter. In this way, the agent "remembers" where it was before it was transmitted across the network, and when it reaches its destination, it recommences execution at the program instruction following that which caused it to be transmitted
- The agent contains both a program and the values of variables, but not the program counter, so the agent can remember the values of all variables, but not where it was when it trransmitted itself across the network.
- The agent to be transmitted is essentially a script, without any associated state (although state might be downloaded from the original host once the agent has arrived at its destination).

9.3.1 Security

The issue of security dominates discussions about mobile agents, in fact we can say that security is a significant concern with mobile agent-based computing.
The key difficulty is that, in order for an agent to be able to do anything useful when it arrives at a remote location, it must access and make use of the resources supplied by the remote host. However, provinding access to these resources is inherently dangerous: it lays open the possibility that the host will be abused in some way.

Some general issues in using agents are:

- **Delegation**: you are delegating to the agent some of your authority. This means that agents are doing things that you cannot always see. Delegation
- **Mobility**: they may be doing it on the other side of the planet. Or, an agent from the other side of the planet may be doing it on your server Mobility
- **Viruses**: mobile agents share many characteristics with viruses. In creating an environment for agents, there is the additional risk that we expose weaknesses that may enable viruses to breed Viruses
- **Trust**: humans have classified their co-workers into those who are reliable and those who are not. Trust

Delegation

The purpose of an agent is to perform some tasks that would otherwise be performed by its user.
The agent may need many, if not all, of the access rights of the user.
In a security environment, this can be readiliy achieved by passing the copy of the user's certificate to the agent.
In this regard, the agent is indistinguishable from any other applications employed by the user.
The certificates arte valid for a finite period, defined by the security administrators.

Performing the full delegation is not a big problem. However, a problem is to limit delegated authority.
An agent is more flexible and unpredictable than a traditional application.

Viruses

It is impossible, in principle, to verify with complete cenrtainty if an arbitrary program is a virus or not.
In practice, the problem of writing a program that can verify the correct behaviour of another program is unsolved.
It is difficult to define the necessary and sufficient tests that an agent must pass in order to determine its intentions.
Some prevautions that can be enforced are:

- restriction of access to critical resources
- restriction on altering other programs

Security for hosts

In order to enforce security of hosts, the designer needs to design a system that:

- **Limits delegation**.
  In other terms give the agent and the user separate identities.
  A way to enforce this limitation is via Secure co-processors: have a physically separate processor on which the agent runs execute the agent in a padded cell.
  As a result, the approach allows the agent to interact with the system environment only in a language with limited system level expressiveness

  Limits delegation
- **Limits resource consumption**.
  In particular, enforcing a limit on the amount of each resource that an agent is permitted to consume or a limit on the amount of budget/CPU time an agent can access.

  Limits resource consumption

Security for Agents

On the contrary we need to protect mobile agents form malicious hosts because:

- agents have a right to privacy
- We often do not want to send our programs to a remote host, as to do so might enable the recipient to determine, its purpose and hence our intent
- the agent might be modified (sabotaged) in some way, without the owner's knowledge or approval

In practive, however, the current consensus is that it is computationally impossible to protect a mobile agent form malicious host.

In fact:

- the use of authentication of the executing server is easy to implement for a single host but quickly becomes difficult for multiple dynamic hosts
- enforcing agent privacy would require its execution in a highly trusted environment, which consequently would require encapsulating all methods and data and high protection in a Multi agent environment

Still there are some possibilities for protection:

- **Data integrity**
  an agent can be protected in transit by using conventional encryption techniques

  Data integrity
- **Origin authentication**
  certification

  Origin authentication
- **Access itinerary control**
  restriction on visiting some environments

  Access itinerary control
- **Privacy and integrity of gathered information**
  via stateless gathering or stateful gathering

  Privacy and integrity of gathered information

9.4 Requirements for Mobile Agents

From the previous sections we can conclude that the general requirements to mobile agent environments are:

- Expressiveness as a programming language
- ability to execute remotely or to transport state
- Support for agent communication language
- Security support
- Management support

Bibliography

[1] Michael Wooldridge. *An introduction to multiagent systems*. John wiley & sons, 2009.