Agents working together

▼ Introduction

Agents are autonomous and make decisions at run time. They don't know what they're going to do until they are in front of a problem, so they must have some dynamic coordination, which should include sharing tasks and information with each other.

Based on their motivations we can distinguish between two types of agents: benevolent or self-interested.

- A Benevolent Agent is a "good guy", will always try and help. It's good if we own the entire system (environment and all the agents in it). This way, all agents will "help out", simplifying the design of the system. This way we can use Cooperative Distributed Problem Solving (CPDPS) for solving problems.
- A self-interested agent is selfish (they only care about themselves and their own goals). Therefore, it will further it's own interests. This will lead to potential conflict and complicate the design task. We're going to need to have a strategic behaviour towards them, since they can even be adversarial towards us.

▼ Cooperative Distributed Problem Solving

Sophisticated agents may work together to solve a problem greater than their capabilities. We can measure it by either coherence or coordination.

- Coherence: how good is the solution created by a given group of agents, how efficiently does it use resources for problem-solving
- Coordination: how much do they get in each other's way, do they have to synchronise with each other.

This process has multiple stages:

▼ Stage 1: Problem Decomposition

Similar to parallel computing, we divide a large problem into smaller subproblems. This process is often recursive and hierarchical.

The granularity of a resulting problem is important: if it's too coarse, an agent might be doing all of it, while if it's too fine, agents will end

up doing atomic operations (like addition or subtraction), so we must know when to stop the subdivision of problems.

Another important question is who does this decomposition (is it centralized?), has knowledge of the task structure and assigns tasks to different agents.

As an example, we may have the following equation

$$Answer = f(x) + f(y)$$

We could assign f(x) to be computed by agent A and f(x) to be computed by agent B.

▼ Stage 2: Sub-Problem Solution

At this stage subproblems identified in Stage 1 are solved. Agents may need to share information, synchronise and cooperate in order to solve these smaller subproblems (dependencies between agents may appear).

As an example, we could have an agent A push a box towards another agent B, and then B carrying it on.

▼ Stage 3: Solution Synthesis

At this point, partial solutions to subproblems are integrated into the answer to the problem (a larger solution), a process often hierarchical in nature.

An important question in this stage is who does this composition, similarly to "who does the decomposition" at Stage 1.

Information sharing is required at this stage.

▼ Task sharing and result sharing

Task sharing is where we allocate work to other agents during problem decomposition and cooperation while solving the subproblems. We must take into consideration agents having different capabilities or their own interests.

Result sharing is the process of sharing answers to solved subproblems. They can be solved **proactively** (an agent tells the answer to another agent thinking they want to know it), or they can be

shared **reactively (**upon request). This also includes the problem of how to assemble partial answers into a complete solution.

▼ Contract Net

It's a task-sharing protocol used for task allocation (once we've broken down a large task into subtask). It revolves around a manager-contractor relationship.

- Manager: agent that wants some work done.
- Contractor: agent that attempts to do some work.

▼ There are several stages involved in the Contract Net protocol:

▼ Stage 1: Recognition

A manager realises it has a problem. Agent 0 (manager) recognises that it has a problem that needs solving and can't or doesn't want to achieve it's goal in isolation.

▼ Stage 2: Announcement

A manager announces it has a problem and the tasks it wants agents to help it with. It contains a description of the task and any constraints (deadlines or required resources). The manager broadcasts this announcement to the contractors. It can be either a selective broadcast or a full broadcast.

- Selective broadcast: the manager knows only a subset of contractors are able to complete the task or has a particular contractor in mind.
- Full broadcast: the manager shares the announcement to all contractors.

▼ Stage 3: Bidding

The contractors have received an announcement from the manager and decide if they're capable of performing the task, does it want to do it and if it is worth it to do it.

If it decides to do it, it submits a bid for that work.

▼ In order to decide, it uses Marginal Cost.

▼ Notation:

▼ Current task, with time *t*

$$\tau_i^t$$

▼ Available resources to the agent

$$e_i$$

▼ Cost of a task for the agent

$$c_i(\tau_i^t)$$

▼ Task specification

$$\tau(ts)$$

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

It is the union of the cost of doing the contractor's current with the cost of the new task minus the cost of doing solely the new task. This is done because the new task could be cost-free (if it's objectives are already achieved with the other agent's tasks).

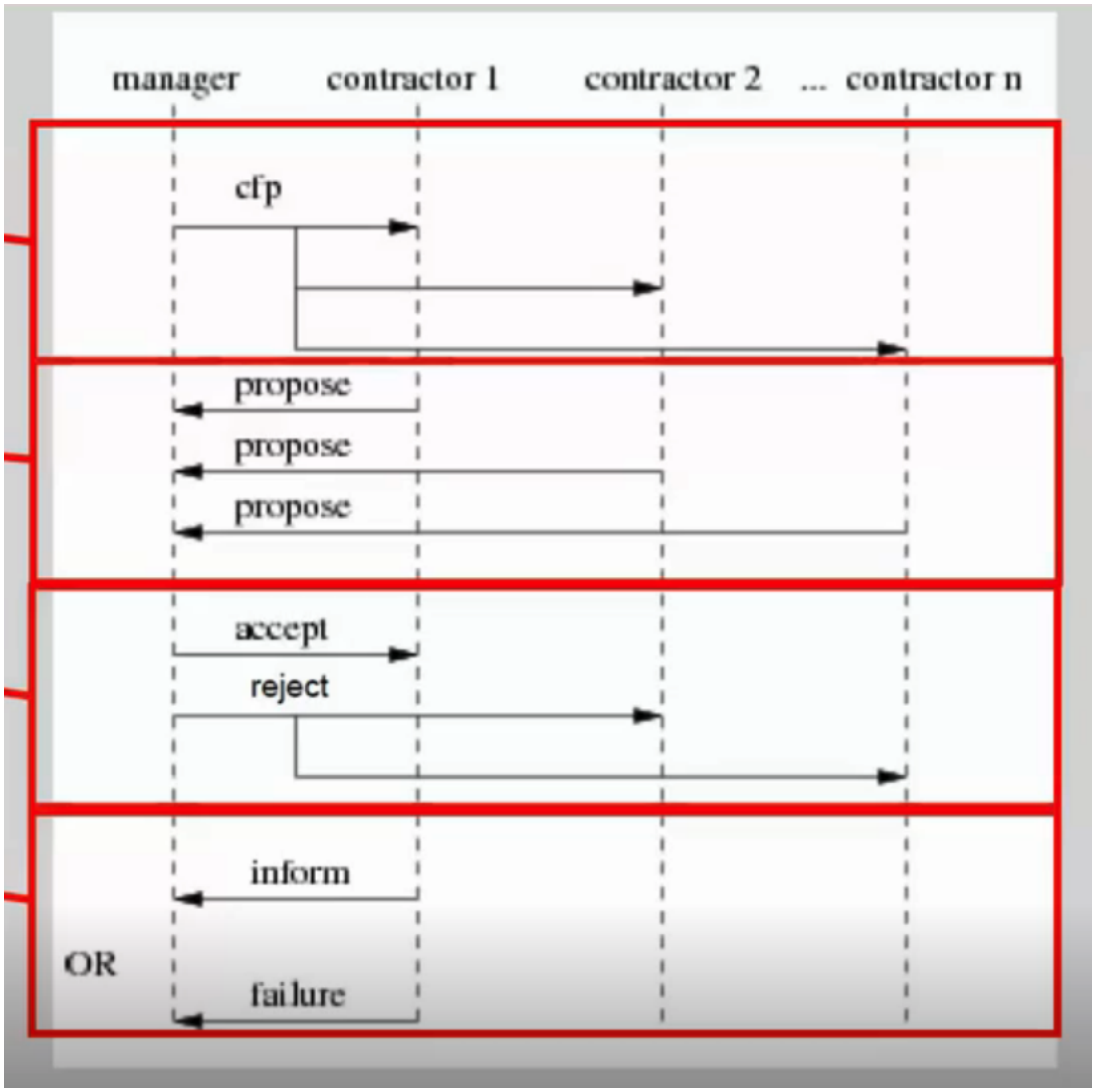
▼ Stage 4: Awarding

The manager collects all the bids and decides to which agent it's going to award the task. Then it sends an acceptance notification to the selected agent and a rejection one to every unsuccessful bidder.

All rejected agents delete every information related to the bid from their database. The awarded agent will perform the task and report back on it wether it succeeds or fails to do it.

▼ Contract Net in FIPA

Outlined are Announcing, Bidding, Awarding and Reporting.



▼ Coordination Relationships

Coordination is essential in a multi agent system if it's agents are going to interact in any way whatsoever. It's key activity is managing the interdependence between agent activities.

▼ Situations that require coordination

- A resource allocation problem: two people want to leave the room at the same time but they won't fit through the door. If one of them let's the other one go first, they've coordinated that activity.
- Dependency between activity: an agent has written an application but requires the signature of another one. Therefore, the activity of submitting the proposal depends on the activity of signing by the other agent.

- Non-requested cooperation: an agent reads an article online, thinks another one will like it and prints and hands it to another agent. An agent increases another one's utility with extra time, effort...

Coordination can be explicitly requested or implicit. It's important that at least a bit of benefit comes out of this coordination. This is known as a positive coordination relationship, of which there are several types.

▼ Types of positive coordination relationships

- Action Equality Relationship: both agents' plans have identical outcomes, so it can be agreed that one of them performs those actions, saving the other one efforts.

  ▼ Example

  Agent 1's plan:

  $$\pi_1 = \{\alpha_1, \alpha_2, \alpha_3\}$$

  Agent 2's plan:

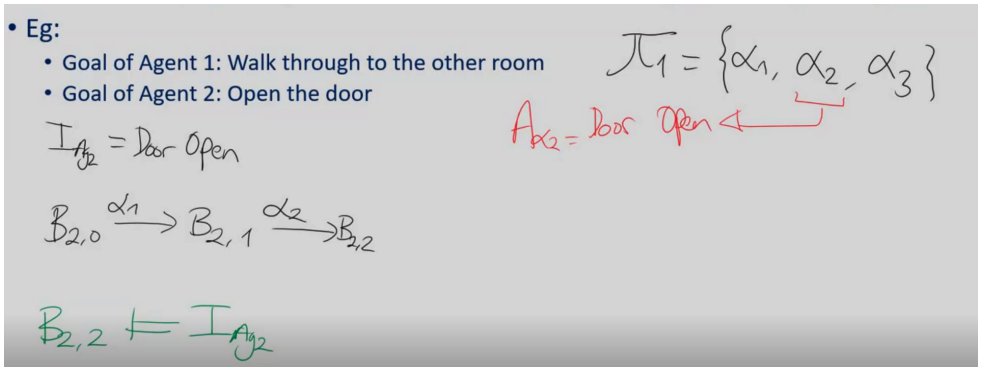
  $$\pi_1 = \{\alpha_4, \alpha_3, \alpha_5\}$$

  Since both do action alpha 3, it can be agreed that only one of them will perform it.

- Consequence Relationship: the actions in one agent's plan have a side effect of achieving another agent's goals (not necessarily intended)

  ▼ Example

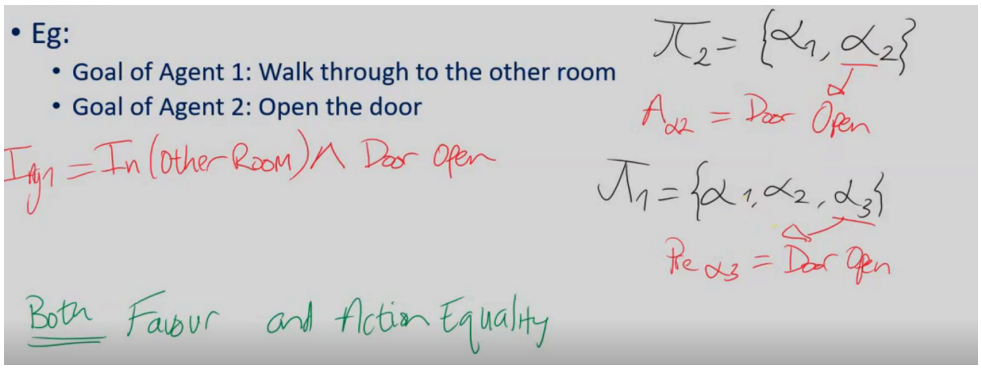
  Agent 1's goal is to walk through to the other room and Agent 2's goal is to open the door. If Agent 1 finishes task alpha 2 (opening the door) before Agent 2 attempts to open the door, Agent 1 will have achieved Agent 2's goal without Agent 2 having done anything.



- Favour Relationship: part of one agent's plan has the side effect of contributing to another agent's goals (not completely achieving it)

  ▼ Example

  Agent 1's goal is to walk through to the other room and Agent 2's goal is to open the door. The difference is in this case Agent 2 opens the door before Agent 1 attempting to do so, therefore contributing to Agent 1's goal.



Coordination happens at runtime, so agents should recognise and deal with these relationships dynamically (agents can't anticipate what will happen like a computer program does).

▼ Handling Coordination

There are multiple ways in which agent coordination can be handled:

▼ Coordination by Partial Global Planning

Based on the idea that agents exchange information and reach common conclusions about a problem.

- Each agent decides goals and generates local plans to achieve such goals

- Each agent exchanges plans with other agents, seeing all the agents as a collective and where their plans interact
- Agents will alter their local plans to better coordinate with all the other agents

To enable this, there is a meta-level structure that's contributed to in a cooperative way, called the Partial Global Plan. Partial because the system doesn't generate a plan for the entire system, and Global because the agents form non-local plans (not just for them, non-local = global).

It contains 3 key components:

- Objective: main goal that the entire system is working towards
- Activity maps: mapping of what each agent is doing in their plans
- Solution construction graph: tells us when and how the agents will interact

▼ Coordination by Joint Intentions

It occurs when agents are actively cooperating and working as a team. They have to have a single joint commitment to the same goal. This joint commitment has to persist amongst the agents, and has to be monitored by a Social Convention.

This is **not** a situation where multiple agents have the same intentions and that's it. It is more similar to a group of dancers whose next move is to get under a tree, rather than multiple people getting under a tree when it starts raining.

▼ Social Convention

Outlines how agents should behave to one another (what's expected of them) and the conditions under which a joint commitment can actually be dropped.

For example, if an agent thinks the goal is unreachable and so it drops it, it has to convince the other agents to drop is as well.

Using SC's, every agent knows what is going to be expected of it.

All agents are committed to a goal PHI, and the motivation for such goal is PSI.

▼ Example

PHI is to "move a box", and PSI is "a boss wants the box to be moved".

All agents believe PHI is not satisfied but it is possible.

Every agent has goal PHI until it believes:

- PHI is satisfied
- PHI is impossible
- PSI is not present

▼ Coordination by Mutual Modelling

Agents build a model of each other and try to make predictions about other agents' actions. Then they'll use those predictions to coordinate their activities.

▼ Example

Agents 1 and 2 want to cross a door but realise it's not big enough for the both of them. Agent 1 has built a model of Agent 2 and predicts it'll wait for 1 to go through the door.

▼ Coordination by Social Laws

Social norms and laws are established and expected patterns of behaviour.

Social norms are not enforced, but allow agents to regulate behaviour.

Social laws are more formal, and are enforced with some more authority.

Both are going to tell an agent or help an agent discern what they are allowed or not to do.

▼ There are 2 approaches for Coordination by Social Laws:

- Offline Approach: until now we've considered agents as a function that maps runs ending in an environmental state to an action. This allows any action that's possible to be done. We avoid this with constraints, a tuple containing E' (a set of states) and alpha (an action), such that the action alpha cannot be done while in a state contained in E'. A social law is a set of constraints. A focal state is an "important" state the agent has

to get to. A social law is **useful** if it does not prevent the agent from getting to a focal state.

- Allowing social norms to emerge: agents can adopt social laws they are told about or see having a high success rate

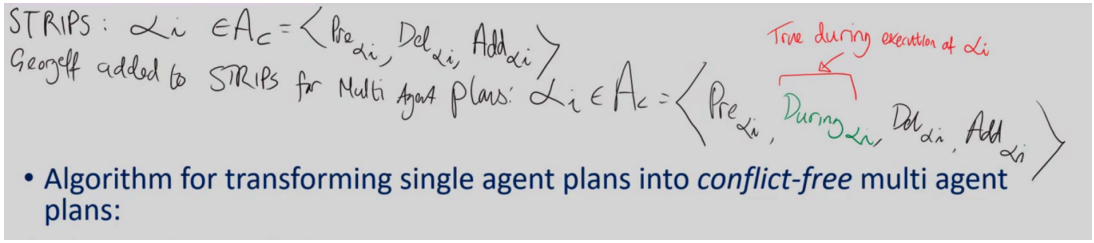
▼ Multi-Agent Planning

Planning can be used for a single agent and incorporated into practical-reasoning agents. This can also be done for multiple agents, with some changes.

▼ Possibilities for MA planning:

- Centralised planning for distributed plans: the division of labor is predefined and is a central Master that distributes work to multiple slaves.
- Distributed planning: a group of agents cooperate to form a centralised plan. These plans are often carried out by specialists in their fields (agents that are good at one particular thing). This method only generates the plans, the agents don't actually have to execute them.
- Distributed planning for distributed plans: groups of agents cooperate and form individual plans in those groups, and then coordinate along the way. The disadvantage is that there may never be a global plan with this.

▼ Single Agent Plans to Multi Agent Plans



1. Interaction analysis: Do plans affect others? Do plans interact with each other? Wether actions can be executed without invalidating preconditions, can things happen in parallel, should they be ordered...
2. Safety analysis: identify any interactions that can be considered unsafe (from last phase)

3. Interaction Resolution: any problematic or unsafe interaction will be treated as a critical area (like in distributed computing, only one thing can execute in it at a time).