CHAPTER 8: WORKING TOGETHER

Multiagent Systems

http://www.csc.liv.ac.uk/~mjw/pubs/imas/

Working Together

- Why and how to agents work together?
- Since agents are autonomous, they have to make decisions at *run-time*, and be capable of *dynamic coordination*.
- Overall they will need to be able to share:
  - Tasks
  - Information
- If agents are designed by different individuals, they may not have common goals.

- Important to make a distinction between:
  - *benevolent agents* and
  - *self-interested agents*.

Benevolent Agents

- If we “own” the whole system, we can design agents to help each other whenever asked.
- In this case, we can assume agents are *benevolent*: our best interest is their best interest.
- Problem-solving in benevolent systems is *cooperative distributed problem solving* (CDPS).
- *Benevolence simplifies the system design task enormously!*
- We will talk about CDSP in this lecture.

Self-Interested Agents

- If agents represent the interests of individuals or organisations, (the more general case), then we cannot make the benevolence assumption:
- Agents will be assumed to act to further there own interests, possibly at expense of others.
- Potential for *conflict*.
- May complicate the design task enormously.
- Strategic behavior may be required — we will cover some of these aspects in later lectures.
  - Game theory

Coherence and coordination

- Criteria for assessing an agent-based system.
- *Coherence*

  how well the [multiagent] system behaves as a unit along some dimension of evaluation (Bond and Gasser).

- We can measure coherence in terms of solution quality, how effiently resources are used, conceptual clarity and so on.

- *Coordination*

  > the degree. . . to which [the agents]. . . can avoid "extraneous" activity [such as] . . . synchronizing and aligning their activities (Bond and Gasser).

  If the system is perfefctly coordinated, agents will not get in each others' way, in a physical or a metaphorical sense.

Task Sharing and Result Sharing

- How does a group of agents work together to solve problems?
- There are three stages:
  - Problem decomposition
  - Sub-problem solution
  - Answer synthesis
- Let's look at these in more detail.

Problem decomposition

- The overall problem to be solved is divided into smaller sub-problems.
- This is typically a recursive/hierarchical process.
  - Subproblems get divided up also.
  - In ACTORS, this is done until we are at the level of individual program instructions.
- Clearly there is some processing to do the division. How this is done is one design choice.

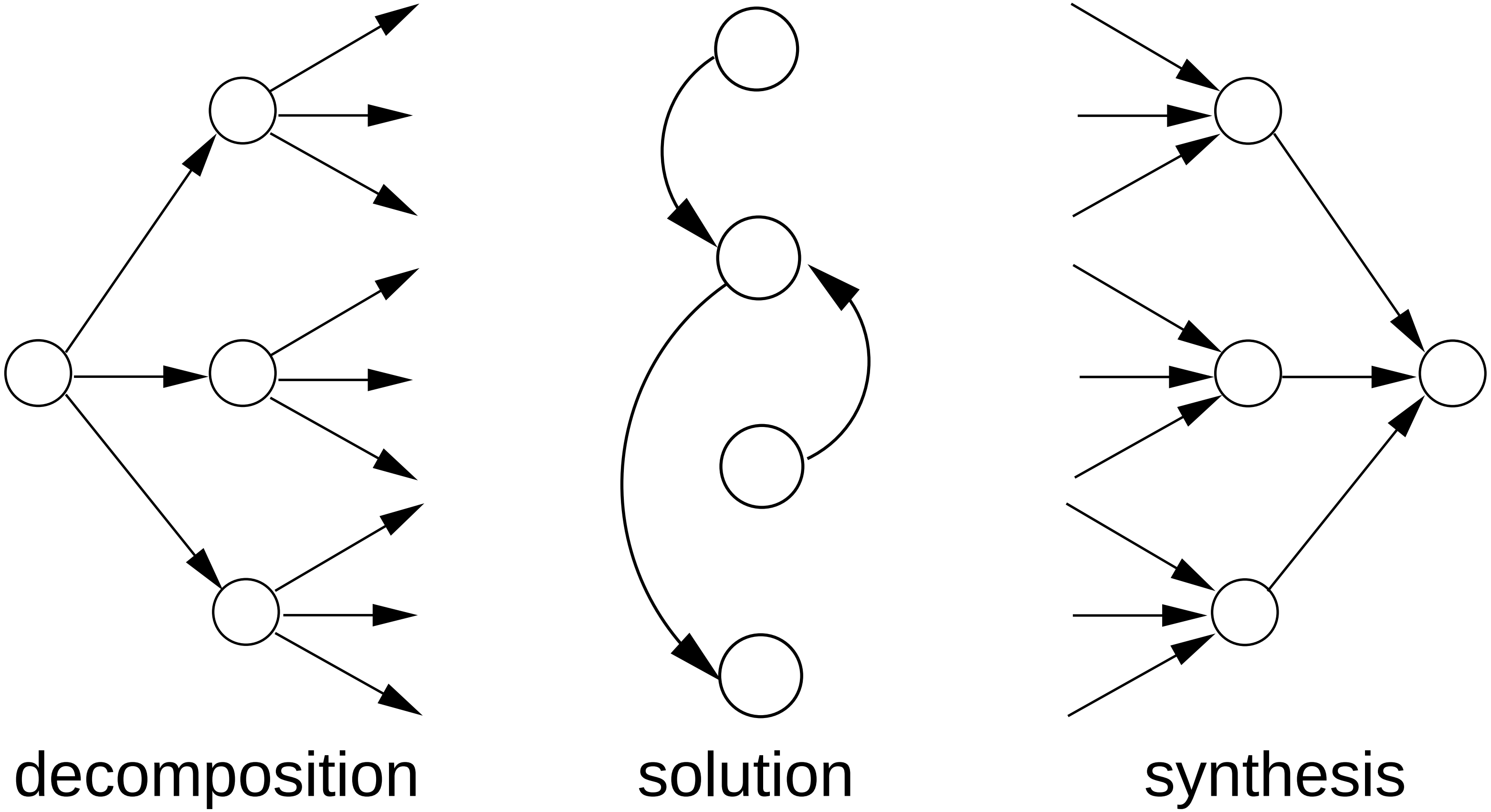
- Another choice is *who* does the division.
  - Is it centralized?
  - Which agents have knowledge of task structure?
  - Who is going to solve the sub-problems?

Sub-problem solution

- The sub-problems derived in the previous stage are solved.
- Agents typically share some information during this process.
- A given step may involve two agents synchronizing their actions.

Solution synthesis

- In this stage solutions to sub-problems are integrated.
- Again this may be hierarchical
  - Different solutions at different levels of abstraction.


decomposition
solution
synthesis

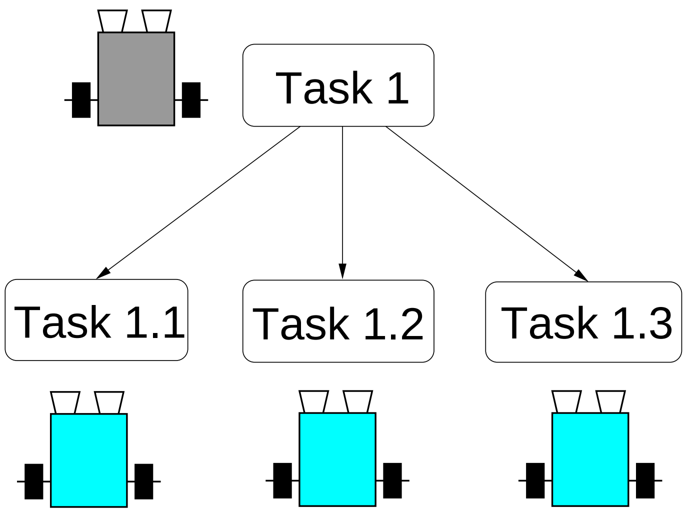
- Given this model of cooperative problem solving, we have two activities that are likely to be present:
  - *task sharing*:

    components of a task are distributed to component agents;

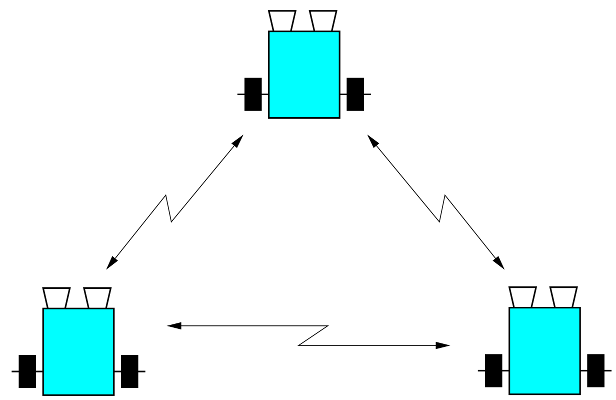
    how do we decide how to allocate tasks to agents?
  - *result sharing*:

    information (partial results etc) is distributed.

    how do we assemble a complete solution from the parts?



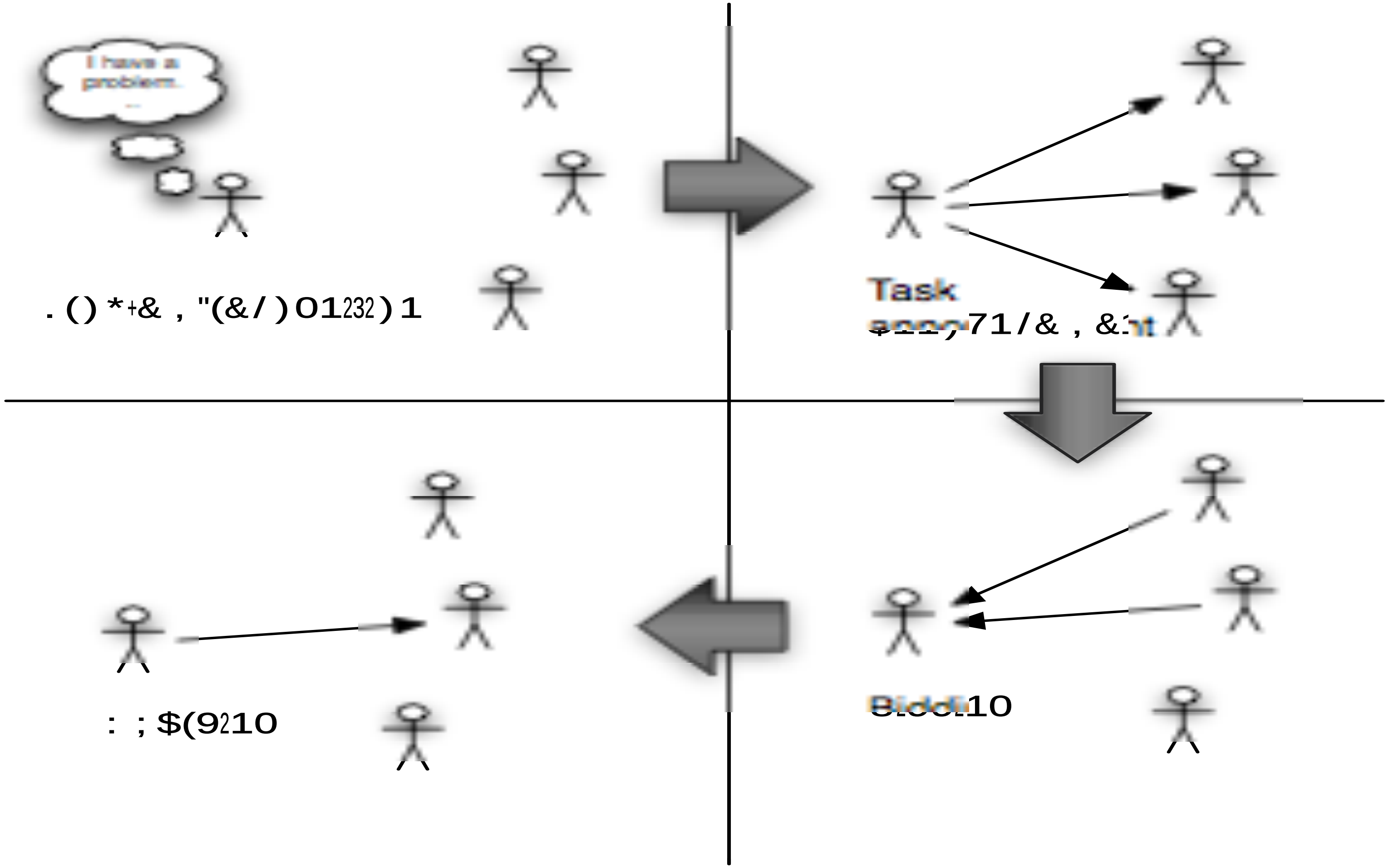
task sharing



result sharing

The Contract Net

- Well known task-sharing protocol for *task allocation* is the *contract net*.
- The contract net includes five stages:
  1. Recognition;
  2. Announcement;
  3. Bidding;
  4. Awarding;
  5. Expediting.
- The textbook describes these stages in procedural terms from the perspective of an individual agent.


Task

Recognition

- In this stage, an agent recognises it has a problem it wants help with.
- Agent has a goal, and either. . .
  - realises it cannot achieve the goal in isolation — does not have capability;
  - realises it would prefer not to achieve the goal in isolation (typically because of solution quality, deadline, etc)
- As a result, it needs to involve other agents.

Announcement

- In this stage, the agent with the task sends out an *announcement* of the task which includes a *specification* of the task to be achieved.
- Specification must encode:
  - description of task itself (maybe executable);
  - any constraints (e.g., deadlines, quality constraints).
  - meta-task information (e.g., “bids must be submitted by. . . ”)
- The announcement is then *broadcast*.

Bidding

- Agents that receive the announcement decide for themselves whether they wish to *bid* for the task.
- Factors:
  - agent must decide whether it is capable of expediting task;
  - agent must determine quality constraints & price information (if relevant).
- If they do choose to bid, then they submit a *tender*.

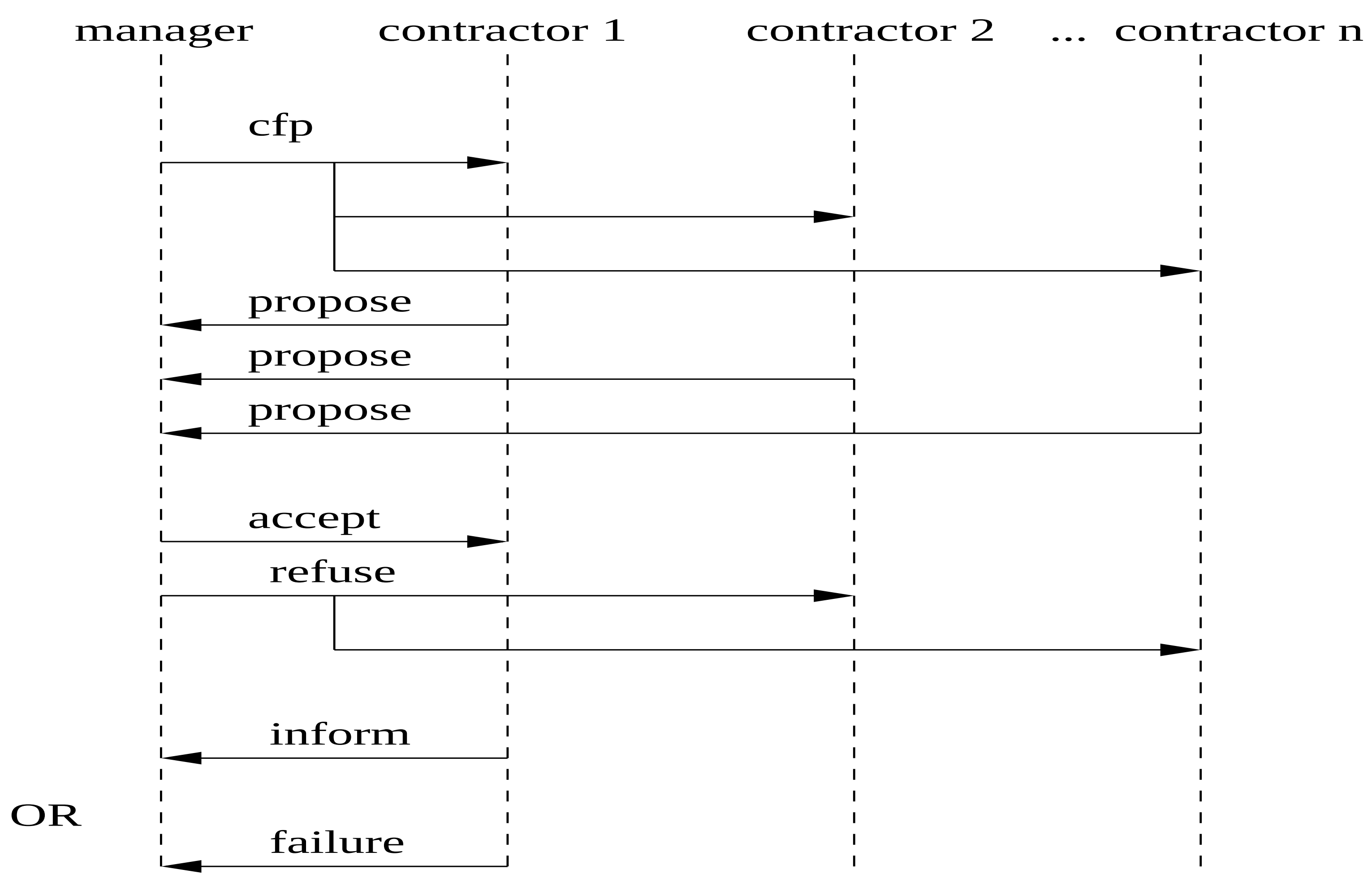
Awarding & Expediting

- Agent that sent task announcement must choose between bids & decide who to “award the contract” to.
- The result of this process is communicated to agents that submitted a bid.
- The successful *contractor* then expedites the task.
- May involve generating further manager-contractor relationships: *sub-contracting*.
  - May involve another contract net.

The Contract Net via FIPA Performatives

- The FIPA ACL was designed to be able to capture the contract net.
- `cfp` (*call for proposals*):
  Used for *announcing* a task;
- `propose`, `refuse`:
  Used for making a proposal, or declining to make a proposal.
- `accept`, `reject`:
  Used to indicate acceptance or rejection of a proposal.

- `inform`, `failure`:
  Used to indicate completion of a task (with the result) or failure to do so.


manager
contractor 1
contractor 2
...
contractor n
cfp
propose
propose
propose
accept
refuse
inform
OR
failure

Issues for Implementing Contract Net

- How to. . .
  - . . . specify *tasks*?
  - . . . specify *quality of service*?
  - . . . decide how to *bid*?
  - . . . select between competing offers?
  - . . . differentiate between offers based on multiple criteria?

Deciding how to bid

- At time t a contractor i is scheduled to carry out τ_i^t.
- Contractor i also has resources e_i.
- Then i receives an annoucement of task specification ts, which is for a set of tasks $\tau(ts)$.
- These will cost i $c_i^t(\tau)$ to carry out.
- The *marginal cost* of carrying out τ will be:

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

that is the difference between carrying out what it has already agreed to do and what it has already agreed plus the new tasks.

- Due to synergies, this is often not just:

$$c_i(\tau(ts))$$

  in fact, it can be zero — the additional tasks can be done for free.

- Think of the cost of giving another person a ride to work.
- As long as $\mu_i(\tau(ts) \mid \tau_i^t) < e$ then the agent can afford to do the new work, then it is rational for the agent to bid for the work.
- Otherwise not.

Result Sharing

- In results sharing, agents provide each other with information as they work towards a solution.
- It is generally accepted that results sharing improves problem solving by:
  - Independent pieces of a solution can be cross-checked.
  - Combining local views can achieve a better overall view.
  - Shared results can improve the accuracy of results.
  - Sharing reuslts allows the use of parallel resources on a problem.

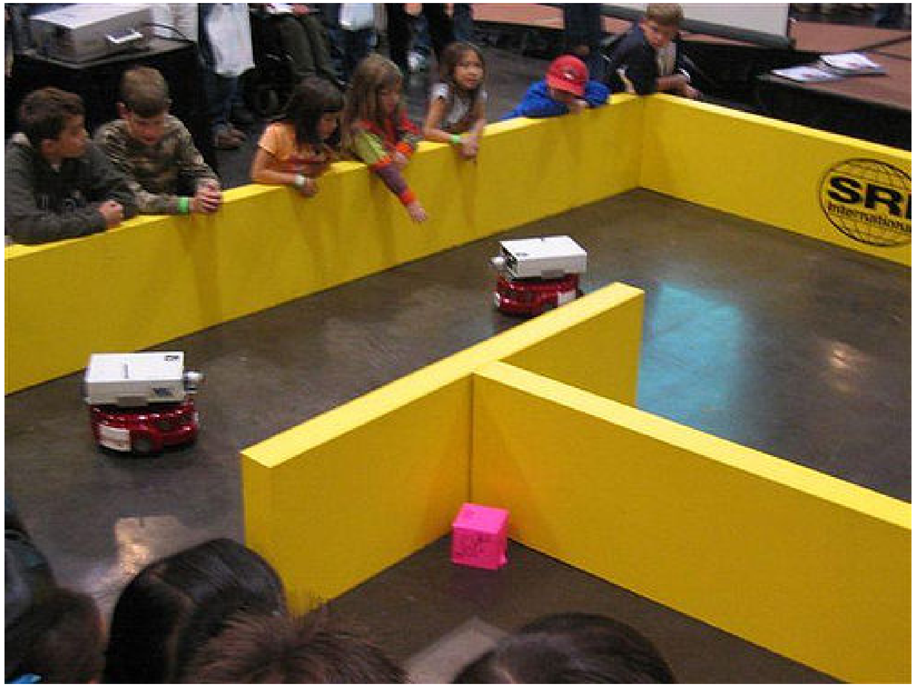
Result Sharing in Blackboard Systems

- The first scheme for cooperative problem solving: was the *blackboard system*.
- Results shared via shared data structure (BB).
- Multiple agents (KSs/KAs) can read and write to BB.
- Agents write partial solutions to BB.
- BB may be structured into hierarchy.
- Mutual exclusion over BB required $\Rightarrow$ bottleneck.
- Not concurrent activity.
- Compare: LINDA tuple spaces, JAVASPACES.

Result Sharing in Subscribe/Notify Pattern

- Common design pattern in OO systems: *subscribe/notify*.
- An object *subscribes* to another object, saying "tell me when event e happens".
- When event e happens, original object is notified.
- Information pro-actively *shared* between objects.
- Objects required to know about the *interests* of other objects $\Rightarrow$ inform objects when relevant information arises.

Result Sharing



- The Centibots robots collaborate to map a space and find objects.

Handling inconsistency

- A group of agents may have inconsistencies in their:
  - Beliefs
  - Goals or intentions
- Inconsistent beliefs arise because agents have different views of the world.
  - May be due to sensor faults or noise or just because they can't see everything.
- Inconsistent goals may arise because agents are built by different people with different objectives.

- Three ways to handle inconsistency (Durfee at al.)
- Do not allow it

  For example, in the contract net the only view that matters is that of the manager agent.
- Resolve inconsistency

  Agents discuss the inconsistent information/goals until the inconsistency goes away.

  We will discuss this later (argumentation).
- Build systems that degrade gracefully in the face of inconsistency.

Coordination

- Coordination is managing dependencies between agents.
- Example

  We both want to leave the room through the same door. we are walking such that we will arrive at the door at the same time. What do we do to ensure we can both get through the door?

  We both arrive at the copy room with a stack of paper to photocopy. who gets to use the machine first?

- Von Martial suggested that *positive* coordination is:
  - Requested (explicit)
  - Non-requested (implicit)
- Non-requested coordination relationships can be as follows.
- Action equality: we both plan to do something, and by recognizing this one of us can be saved the effort.
- Consequence: What I plan to do will have the side-effect of achieving something you want to do.
- Favor: What I plan to do will make it easier for you to do what you want to do.

Social norms

- Societies are often regulated by (often unwritten) rules of behavior.
- Example:

  A group of people is waiting at the bus stop. The bus arrives. Who gets on the bus first?
- Another example:

  On 34th Street, which side of the sidewalk do you walk along?
- In an agent system, we can design the norms and program agents to follow them, or let norms evolve.

Offline design

- Recall how we described agents before:

$$Ag : R^E \rightarrow Ac$$

  a function which, given a run ending in a state, gives us an action.

- A *constraint* is then a pair:

$$\langle E', \alpha \rangle$$

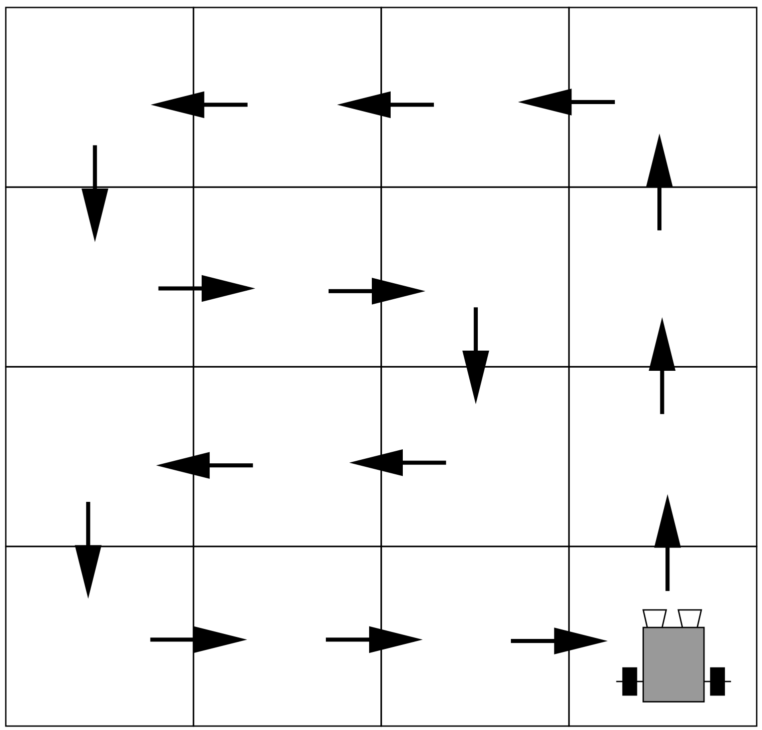
  where $E' \subseteq E$ and $\alpha \in Ac$.

- This constraint says that α cannot be done in any state in E'.

- A *social law* is *a set of constraints*.

- We can refine our view of an environment.
- *Focal* states, $F \subseteq E$ are the states we want our agent to be able to get to.
- From any focal state $e \in F$ it should be possible to get to any other focal state $e' \in F$ (though not necessarily right away).
- A *useful* social law is then one that does not prevent agents from getting from one focal state to another.

- A useful social law that prevents collisions :



- Not necessarily efficient (On^2 steps to get to a specific square).

Emergence

- We can also design systems in which social laws emerge.
- T-shirt game (Shoham and Tennenholtz):

  Agents have both a red t-shirt and a blue t-shirt and wear one. Goal is for everyone to end up with the same color on. In each round, each agent meets one other agent, and decides whether or not to change their shirt. During the round they only see the shirt their pair is wearing — they don't get any other information.

- What *strategy update function* should they use?

Strategy Update Functions

- *Simple majority*:

  Agents pick the shirt they have seen the most.

- *Simple majority with types*:

  Agents come in two types. When they meet an agent of the same type, agents pass their memories. Otherwise they act as simple majority.

- *Highest cumulative reward*:

  Agents can "see" how often other agents (some subset of all the agents) have matched their pair. They pick the shirt with the largest number of matches.

Joint intentions

- Just as we have individual intentions, we can have joint intentions for a team of agents.
- Levesque defined the idea of a *joint persistent goal* (JPG).
- A group of agents have a collective commitment to bring about some goal ϕ, “move the couch”.
- Also have motivation φ, “Simon wants the couch moved”.
- The mental states of agents mirror those in BDI agents.

- Agents don't believe that ϕ is satsified, but believe it is possible.
- Agents maintain the goal ϕ until a termination condition is reached.

- The terminations condition is that it is *mutually believed* that:
  - goal ϕ is satisfied; or
  - goal ϕ is impossible; or
  - the motivation φ is no longer present.
- You and I have a mutual belief that p if I believe p and you believe p and I believe that you believe p and I believe that you believe that I believe p and . . . .

- The termination condition is achieved when an agent realises that, the goal is satisfied, impossible and so on.
- But it doesn’t drop the goal right away.
- Instead it adopts a new goal — to make this new knowledge mutually believed.
- This ensures that the agents are coordinated.
- They don’t stop working towards the goal until they are all appraised of the situation.
- Mutual belief is achieved by communication.

Multiagent planning

- Another approach to coordinate is to explicitly plan what all the agents do.
- For example, come up with a large STRIPS plan for all the agents in a system.
- Could have:
  - Centralized planning for distributed plans
    One agent comes up with a plan for everybody
  - Distributed planning
    A group of agents come up with a centralized plan for another group of agents.

– Distributed planning for distributed plans
  Agents build up plans for themselves, but take into account the actions of others.

- In general, the more decentralized it is, the harder it is.
- Georgeff propsed a distributed version of STRIPS.
- New list: *during*
- Specifies what must be true while the action is carried out.
- This places constraints on when other agents can do things.
- Different agents plan to achieve their goals using these operators and then do:
  - Interaction analysis: do different plans affect one another?

- Safety analysis: which interactions are problematic?
- Interaction resolution: treat the problematic interactions as *critcal sections* and enforce mutual exclusion.

Summary

- This lecture has discussed how to get agents working together to do things.
- We discussed a number of ways of having agents decide what to do, and make sure that their work is coordinated.
- A typical system will need to use a combination of these ideas.