Planning and Scheduling Ingredients for a Multi-Agent System

Federico Pecora and Amedeo Cesta [*]

ISTC-CNR
Institute for Cognitive Science and Technology
Italian National Research Council
Viale Marx 15, I-00137 Rome, Italy

fpeco@tiscali.it, cesta@ip.rm.cnr.it

Abstract

The aim of this paper is to show some work on applying planning and scheduling technology in a multi-agent setting. The context of the research presented herein is the development of an *active supervision system* for a multi-agent environment. With the idea of using off-the-shelf components, we have implemented a first version of the control framework. Specifically, the creation of distributed plans is achieved by using BlackBox in a domain in which time and resource features have been abstracted away, and cascading it with O-Oscar to accommodate this further information. The result, although preliminary, shows how such planning and scheduling components can contribute an interesting coordination service for realistic applications.

1 Introduction

Thanks to recent advancements in technology, it is possible to find a substantial amount of potential applications for robotic technology including multi-agent systems. Our work focuses on studying the issues and problematics involved in the design of multi-agent systems with fixed and mobile heterogeneous agents. These agents can be robots, intelligent sensors and possibly even humans. At present, we are conducting a preliminary analysis of the technology that can be used to design such a system. This research is part of a larger project aimed at developing a distributed system in which software and robotic agents contribute to the common goal of generating active services in a civil environment such as a health care facility [2]. The project, which is due to start officially in 2003, shares some aspects with projects for the assistance of elderly people (e.g., [12] and [15]) but addresses the particular goal of creating a heterogeneous multi-agent environment for generating user services.

We are concerned with the development of a control infrastructure, also referred to as *supervision framework*, which manages and supervises robotic systems in an environment and supports a certain level of human-machine interaction. Also, we are developing a simulated environment to be used in conjunction with a preliminary implementation of the supervision framework, in order to evaluate the applicability and efficiency of our approach.

The scope of this paper is to introduce the reader to some issues and initial results that we are starting to encounter in our research. Specifically, we will focus on the planning and scheduling functionalities implemented in the control (supervision) framework, and show how it is possible to achieve high levels of coordination among the agents during task execution.

The (necessarily) distributed nature of the application is achieved through the use of CORBA middleware. Having such a transparently distributable system makes it possible to separate logically-grouped functionalities on different platforms, thus enabling us to allocate computational resources more efficiently.

In the following sections we will briefly illustrate the complete architecture of the supervision framework, composed of two levels of hierarchy. We will then focus on the second level, showing the assumptions we make at the planning stage that enable us to tackle the multi-agent planning issue as a classical single-agent problem. We will show how this approach induces a team of agents to cooperate in order to achieve a common goal. Specifically, we demonstrate how to make multi-agent task planning problems more tractable

[*] Federico Pecora is also affiliated with Universitá degli Studi di Roma "La Sapienza" — Computer Science Engineering School

by spreading the computational load over two distinct modules: the planning module and the scheduling module.

Finally, we would like to note that the integration of planning and scheduling has already been dealt with, namely with software tools that can interpret a domain description language (DDL) that is capable of expressing time and logics, and is able to reason about both aspects of the domain. This approach is described in [6, 9, 11], just to name a few implementations. Others have dealt with an approach that separates the planning and scheduling phases, e.g., [16]. We are currently investigating the applicability of an approach similar to the latter with an additional idea, namely to integrate well built off-the-shelf components such as the BlackBox planner [13] and the O-Oscar scheduler [4].

The paper is organized as follows: Section 2 introduces the multi-agent problem that is the object of our interest and states some assumptions that bound the scope of our current investigation. Section 3 describes the software architecture that we are building to address the problem, which implements the idea of using off-the-shelf components, and it isolates the part of the system we will discuss in more detail in the rest of the paper. Section 4 presents the main solving idea that we are currently pursuing: separating solving responsibilities between a planner and a scheduler. Section 5 covers aspects related to the integrated planning and scheduling subsystem while Section 6 contains a complete running example of the architecture. Some comments on the system and research prospectives are given in a concluding section.

2 Planning for Multi-Agent Systems

In order to clarify the problems that arise from this type of application and our approach to solve them, we will start with a simple example.

Let's suppose that we have deployed a team of agents in the north wing of a hospital. At this moment there are a certain number of beds that have to be made and a certain number of patients that need to be fed.

We will assume that each mobile agent is endowed with a set of built-in reactive behaviors (avoiding obstacles etc.) and capabilities, and a set of actuators through which it performs actions on the environment (serving meals, going from one room to another etc.) The agents are also supplied with sensors through which they obtain information on the environment and other agents.

The agents are capable of doing the following:

- travel between adjacent rooms
- retrieve a meal (*iff* they are in the kitchen)
- serve a meal
- make a bed (*iff* the bed is clear)
- clear a bed

In order to successfully contribute to the operation of daily tasks in the hospital, the agents have to be able to *find out* and *interpret* the needs of the facility, and to *devise and enact a plan* to successfully fulfill these needs.

To this end, we would like to be able to tell the team of agents which persons need to be fed and which beds need to be made, and then simply sit back and watch them carry out the necessary tasks without the need for any further human intervention. Moreover, we would like the plan the agents devise to fulfill our request to be efficient, possibly better than what a team of humans would do.

But what to we really mean by "efficient"? For now, all we know is that we are interested in "minimizing how long it takes for the agents to fulfill our request", which has to be translated into more specific terms. This boils down to finding some specific parameter to minimize (or maximize). The answer to this question depends on the details of the planning procedure we are going to use. Thus, this question will only be answered at the end of section 4.1.

In any case, let us consider two options for the choice of planning procedure:

- One planning procedure could be to divide the problem among the agents and have them deliberate independently upon how to satisfy the goals assigned to them. We can then try to increase the level of cooperation by implementing some form of plan-merging and/or negotiation, for instance like in [1].
- On the other hand, we could think of using a centralized planning procedure. Because of its centralized nature, such a procedure would take into account the state of every agent *contextually* and propagate the evolution of the *entire* system during the plan generating phase.

It is clear that the first strategy would inevitably yield a less efficient global performance than the centralized approach would. For this reason, our work deals mainly with finding methods and tackling problems that arise from the use of this centralized planning paradigm.

Our initial desire can be thus rephrased as follows: given a current state of the environment and a desired state that we specify, we would like a *supervisor agent* to generate a plan for all the robotic agents *in one planning step*. Again, the motivation behind the choice of centralized planning is that we want to achieve a high level of cooperation among the agents without explicitly telling them *how* to do things together.

2.1 Assumptions

The task of managing the operation of a community of agents involves addressing issues that rage from symbolic planning to allocation in time and execution of planned tasks. The basics of multi-robot control architecture design require for a bottom-up approach, in the sense that any eventual cooperative behavior must be addressed after that of designing a control architecture for the autonomous robots.

Not only: we cannot expect to build a complete system without implementing a robust subsumption architecture over the agents of the colony. In fact, the system design must be addressed in an incremental fashion in order to achieve intelligent capabilities for the agents. For this level of intelligence, explicit models of the environment are counter-productive, and at this level, the world is used as its own model.

The context in which we are designing a control infrastructure is concerned with a higher level of abstraction. Our goal is to achieve, in an world in which we have abstracted away problems that are dealt with by the basic subsumption architecture, a high degree of cooperation among the agents. The output of the planning procedure we describe in the following sections is to be dispatched to the highest layer of the agents' behavioral architecture. Upon receiving these "commands", the activity producing system the agents are endowed with may generate further cooperative actions, such as negotiating the passage through a door and so on.

The scope of the research we are presenting here is limited to the issue of generating high level directives for a colony of agents in order to obtain a well-coordinated sequence of behaviors otherwise difficult to generate in the context of a more detailed representation of the environment. We will show how, even in this simplified domain representation (indeed, analogous to a multi-hand blocks world or a multi-truck/airplane logistics domain), it is necessary to break down the problem in order to obtain plans in a timely and reliable fashion.

Assuming that we are free of computational issues (which we aren't, as we will show in section 4), this approach suffers from another deficiency, namely that we are obliged to generate unconditional plans. In other words, the plans are actually generated off-line with respect to their execution. As a consequence, the planning procedure cannot make use of sensing actions, but must rely only on the initial conditions specified in the problem instance that has been generated. In the next sections, however, it will become clear that this factor is not crippling thanks to the execution monitor and to the fast rate at which we are able to generate new plans.

Another important assumption that we make is that we will not have to deal with error-prone sensors. This assumption is justified by two factors. First, we are building a system that has to operate in an "agent-friendly" environment: therefore, we assume that we can deploy a variety of environmental sensors to suit the system's needs. Second, it is possible to take on the appropriate measures to counter an eventual erroneous observation made in the pre-planning phase through the execution monitor - again, we rely on the speed of executable plan generation.

With the expression "executable plan" we intend a sequence of directives to be dispatched to the agents which will be mapped to behaviors (through a potentially complex subsumption architecture). Again, control mechanisms involved in the execution of the actions are outside the scope of this paper.

Now for what might be the most restrictive precondition we pose: we assume that compiling a logically sound plan does not depend on time or resource constraints. In other words, we are making the assumption that during the plan generation phase, no action is discarded or gains priority because of time- or resource-based considerations. The reason for this assumption will become clear in the next section.

Finally, it is important to notice that this assumption is not extendable to any domain. For instance, if the logical deduction step depends on how much time it takes for an action to be completed, it is necessary to integrate logical deduction and time-based reasoning in one single planning procedure. An example in which this is true could be the RoboCup Rescue domain, in which a plan produced without any time-based deliberation would probably be "logically incorrect".

3 The Active Supervision Framework

As we have mentioned earlier, the module that is placed highest in the hierarchy is the supervisor agent. Its role is to instruct the teams of agents upon the tasks they have to carry out. It does so by "interpreting" the needs of the environment (which are acquired through a human operator and/or through environmental sensors) into a high level plan.

All agents are grouped into teams. The role of the team supervisors (called *AgentBrokers*) is to coordinate teams of agents upon receiving a high level plan for the team they supervise.

Thus, in the current preliminary implementation of the control framework we envisage a two level decisional hierarchy (see *fig. 1*):

1. compilation of high level plans, such as "team 3: clean north wing"
2. interpretation of high level plans by team supervisors (AgentBrokers) and compilation of team plans, such as "(agent A: goto room 2), (agent C: make bed 4)"

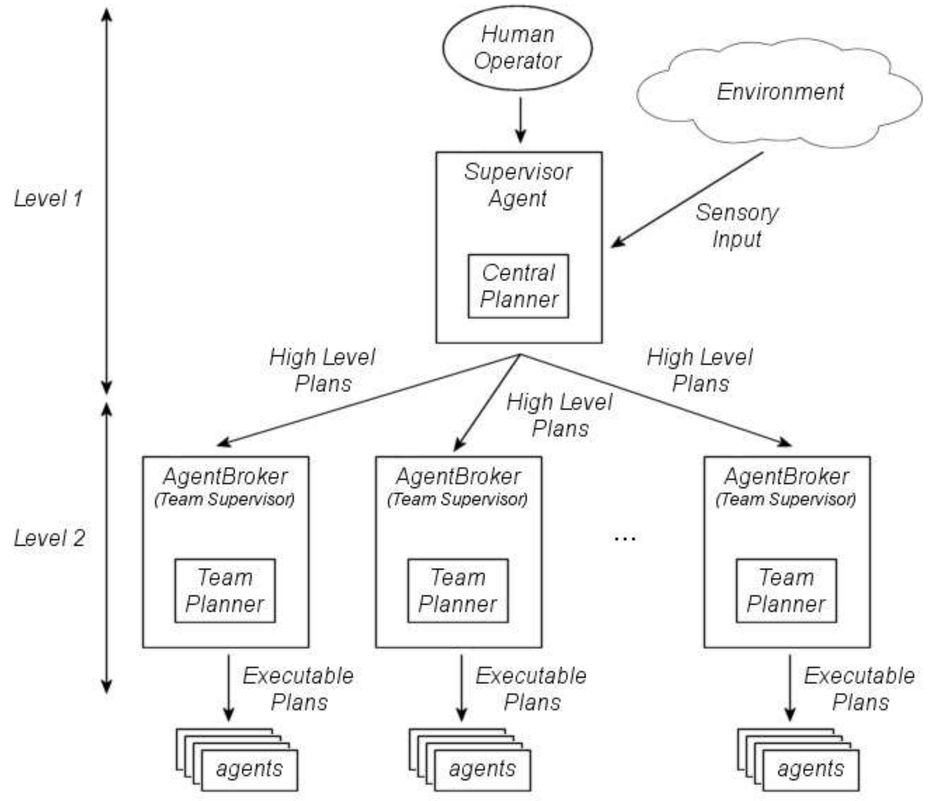

Figure 1: System structure

During system operation, the supervisor agent periodically observes the environment and formulates a set of goals to be achieved. Using the central planner, it compiles high level plans for each of these sets, which represent sets of directives to be dispatched to each AgentBroker. In turn, each AgentBroker breaks these high level directives into a set of tasks for the agents to execute.

The supervisor agent maps teams to high level plans, according to some objective function which induces associations based upon the relative convenience of assigning each task to this or that team of agents. Therefore, the supervisor agent's function is actually to derive a description of the state of the environment through its sensors, and to figure out which measures have to be taken in order to maintain the desired state. We are somehow abusing the term "plan" to characterize the output produced by the supervisor agent, in the fact that it does not really contain information on "how" to act upon the environment. Rather, this entity compiles directives which result from a matching algorithm, similarly to a work-flow management utility. The matching heuristic can be based on skill, geographical position, workload and many other criterions.

We will not discuss the criterions with which teams are chosen, i.e. Level 1. Indeed, the following sections are dedicated to the description of the second level in the above mentioned hierarchy, namely *how the AgentBrokers generate an executable plan for the team of agents they supervise.*

4 Separating Logical and Time-Based Reasoning

The first attempt we can make to implement planning functionalities into the agentBrokers is to equip them with a symbolic planner and model the situation in its domain description language (DDL). This domain independent planner would be capable of finding a *global* plan for the team of agents.

We can model the actions in the domain in the form (in PDDL format)

```
(:action_name
:parameters (?a - agent ...)
...
)
```

where `?a` is the agent that is to carry out the action, and specify a *pool* of agents (the team agents) in the initial conditions. Such a planning architecture implements an "implicit" form of coordination in the sense that the agents don't need to do any explicit negotiation (in fact, the agents aren't even involved in the planning procedure).

The following sections will show that this centralized planning procedure is made possible by conducting a series of simplifications on the domain in order to make the problems tractable for the planning subsystem. We will describe the assumptions that we make and the measures that we take, and how these emphasize the importance of a robust scheduling system for a correct execution of the agents' plans.

As long as it is possible to express all the physical and logical concepts necessary to model the environment and the agents' behaviors in the planner's DDL, the only problem that can arise is the complexity of the planning procedure. Alas, generating an efficient plan (i.e. with few steps) given a realistic goal requires, in general, far too much computing power than we can dispose of. In the hospital domain of the previous example, already giving the goal of feeding 2 people and making 7 beds, makes the problem very difficult for GraphPlan.

As noted by several authors (e.g., [7]), planning is concerned with figuring out *which* actions to carry out, while a scheduling procedure decides *when* to carry them out. The distinction is often not very clear, but for our specific domain it is possible to draw a line between the two steps.

Reasoning about time is a difficult task for most general purpose planners. In fact, their DDLs do not explicitly support time-based expressions. Trying to model even simple time-related concepts, such as "only do one action at a time", makes the complexity of the domain rise dramatically.

On the other hand, schedulers are tools that are completely incapable of doing any logical deduction but are very efficient in reasoning about time. With the term "scheduler", we intend a software tool that, given a set of partially ordered tasks with their expected durations and resource usage, produces an efficient allocation in time of these tasks.

These considerations have lead us to choose a component-based approach in the design of the system, which we implement by using two efficient off-the-shelf components. The choice of these components, in particular of the symbolic planning subsystem, is biased by precise considerations on the type of application we are dealing with. These issues will be discussed in more detail in the following sections, after we have described how we separate solving responsibilities between a planner and a scheduler.

What follows is not a formal method to separate logical and time-based preconditions, rather it can be seen as a reference guideline. In fact, the concept is illustrated with the use of an example.

4.1 Precondition Analysis

Fig. 2 shows an example of some action specifications in PDDL.

Notice that, even if there is no explicit reference to time, through the `(not (busy ?a))` predicate in the action's precondition and the `finish` action, we are trying to force the agents to *not* carry out two actions in the same logical step (they first have to "finish" doing something before they can start doing something else). In other words, we are giving them a logical "excuse" to obey the physical rule that they can only make one bed at a time. A partial order planner would in fact generate a plan similar to this one:

```
1 (goto_room A R1 R2)
1 (make_bed C B1 R3)
1 (make_bed B B2 R3)
2 (finish B)
2 (goto_room A R2 R4)
2 (finish C)
3 (make_bed C B3 R3)
3 (finish B)
3 (goto_room B R3 R4)
4 (clear_bed A B5 R4)
...
```

```
...

(:action make_bed
:parameters (?a - agent ?b - bed ?r - room)
:precondition (and (not (busy ?a)) (unmade ?b)
   (clear ?b) (is_in_agent_room ?a ?r) (is_in_bed_room ?b ?r))
:effect (and (not (unmade ?b)) (busy ?a))
)

(:action finish
:parameters (?a - agent)
:precondition (busy ?a)
:effect (not (busy ?a))
)

...
```

Figure 2: PDDL extract with time-based dependencies

Eliminating this precondition in the action has no impact on the logical correctness of the generated plan. For example, the partial order planner's output given this new domain specification (see *fig. 3*) would become something like:

```
1 (goto_room A R1 R2)
1 (make_bed C B1 R3)
1 (make_bed B B2 R3)
1 (make_bed C B3 R3)
2 (goto_room B R3 R4)
2 (goto_room A R2 R4)
3 (goto_room C R3 R7)
4 (clear_bed A B5 R4)
...
```

It is clear that the plan is certainly not executable in this form, but it is logically correct. Notice that there is no "logical" reason for which an agent cannot carry out more than one task at a time.

```
...

(:action make_bed
:parameters (?a - agent ?b - bed ?r - room)
:precondition (and (unmade ?b) (clear ?b)
   (is_in_agent_room ?a ?r) (is_in_bed_room ?b ?r))
:effect (not (unmade ?b))
)

...
```

Figure 3: PDDL extract without time-based dependencies

In so doing, we are transferring the burden of time-based reasoning onto the scheduler. This is possible because we have made the assumption that logical deduction is not time-critical for our domain (section 2.1). Notice also that, similarly to the BlackBox spirit ([13]), our aim is to integrate two systems (the planner and the scheduler) which "know nothing" about each other - they see each other as "black boxes".

At this point we are able to answer the question as to what we want to optimize in our planning procedure. Since we are interested in minimizing the time it takes to successfully complete the execution of the agents' activities, but we *do not* consider time in the planning phase, we will assume that the less steps a plan contains, the more it is efficient. Notice that according to this definition we also have that a plan with a high degree of parallelism is efficient.

5 The Planning and Scheduling Subsystems

Let us now focus on the off-the-shelf components we have selected to address the solving problem we have to deal with, and the reasons behind this choice of systems.

For the purpose of generating a logically sound and efficient plan we use the BlackBox planner. BlackBox is a planning system that works by converting problems specified in PDDL notation into Boolean satisfiability problems, and then solves the problems with a SAT solver (the version we are using employs the Chaff solver by default). The reason for this choice of planner resides in the fact that it has proved to be a very powerful tool to solve problems in our multi-agent setting. In fact, the powerful search (extraction) algorithms implemented in the SAT engine make this planner particularly good at solving problems in domains that require a high degree of parallelism (see [10]). For a detailed discussion on these issues please refer to [13] and [14].

The scheduling subsystem is mainly made up of O-Oscar (see [4]), a complete scheduling package that implements the ISES algorithm for schedule generation [5]. It is integrated with an execution monitor whose function is to dynamically adapt the schedule whenever an "unexpected" event occurs. Such an event could be:

- a task is delayed,
- the resources a task uses are diminished or made unavailable.

The input provided to the scheduler is a partially ordered plan consisting of tasks to be carried out, resources and their capacity, and the tasks' resource usage. The input for the execution monitor is given through a set of sensors that are capable of detecting the time it is taking to execute the current tasks and the instantaneous usage and capacity of resources.

We will now discuss how to link the planning and scheduling subsystems, namely how to

1. convert the partially ordered plan generated by BlackBox into input for O-Oscar (in the form of a graph),
2. interpret what is going on in the agents' world in a form that is meaningful for the execution monitor.

5.1 Integrating Planning and Scheduling

Fig. 4 expands Level 2 of the control architecture. The block-schema shows the information flow through the integrated planning and scheduling subsystem and the control signals (dotted lines) generated by the various modules, along with its interfaces towards the higher layer (Level 1 - the supervisor agent) and the lower layer (the agents).

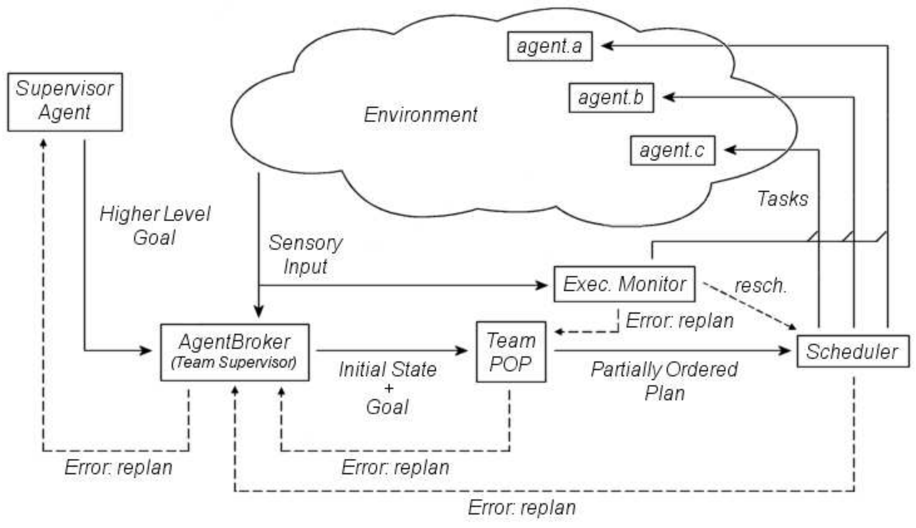

Figure 4: Complete planning and scheduling architecture for Level 2

In the domain description used for the planner, we do not represent any resource or time related concepts. In fact, the result obtained from the planning step must be adapted and integrated with time and constraint information.

The adaptation procedure is composed of four phases. Referring to the pseudo-code in *fig. 5*, notice that the input for the adaptation procedure is a partially ordered plan (POP) obtained from the symbolic planner, and the output is a partially ordered plan complete with time and resource constraints (CompletePOP).

```
function AdaptPlan(POP planner_output):  CompletePOP scheduler_input
{
  ;; PHASE 1 ;;
  CompletePOP result ← new CompletePOP(planner_output);

  ;; PHASE 2 ;;
  result ← SampleDurations(GetTasks(planner_output));

  ;; PHASE 3 ;;
  forall Agent a ∈ GetAgents(planner_output) do
  {
    Resource r ← new Resource(a);
    SetCapacity(r, 1);
    result ← r;
    SetResourceUsage(result, r, 1);
  }

  ;; PHASE 4 ;;
  forall Task t1 ∈ GetTasks(planner_output) do
    forall Task t2 ∈ GetTasks(planner_output) do
      if ((t2 ≻ t1) && (ResourceContention(t1, t2)))
      {
        CausalLink l ← new CausalLink(t1, t2);
        result ← l;
      }
  return result;
}
```

Figure 5: The adaptation procedure

First of all, the AdaptPlan function computes an initial plan, based upon the logical steps described in the planner's output. This plan contains a set of tasks, each of which is an action performed by an agent in the original plan. This plan already contains a subset of the necessary time constraints which are implicit in the sequence of actions in the input plan. In fact, if T is the set of tasks (actions) in a plan, $agent(t_i)$ is the agent which carries out task t_i and $t_2 \succ t_1$ means that t_2 comes after t_1, the following relation holds:

$$\forall\, t_1, t_2 \in T$$
$$(agent(t_1) = agent(t_2)) \;\wedge\; (t_2 \succ_{POP} t_1) \;\Rightarrow\; t_2 \succ_{CompletePOP} t_1$$

In other words, the "constructor" can easily generate basic precedence constraints among the plan's tasks because *if two tasks are logically consequent (they are in two distinct logical steps) and are carried out by the same agent, they are also ordered in time.*

In phase two, the algorithm determines the "expected" durations of the various activities. Although these parameters are sampled from previous executions, they do not have an important role in the successful completion of the plan. This will become clearer when we discuss the execution monitor.

Third, in order to enforce that the agents carry out at most one task at a time, each agent in the plan is modelled as binary capacity resource and a resource constraint is added to the (logically) parallel tasks that involve each agent. Notice that the most critical resources to manage in any real world problem are human- (in our case, agent-) resources.

In the last phase, extra causal links must be added to the plan in order to take into account dependencies between the agents' actions. These dependencies are certainly satisfied in the plan because of the soundness assumption, but they are implicit. In fact, the planner generates plans that represent *one* solution to a given problem, and not the set of all possible plans of the same degree of efficiency.

The overall complexity of this three-step procedure is polynomial in the length of the plan.

Once again, since we are adding time and resources only after the plan has been generated, we are assuming that the correctness of the logical steps that make up the plan do not depend on time or resources. For our application, this procedure generates executable plans with a relatively high degree of optimality because our aim is to maximize the degree of coordination (parallelism) in the agents' missions.

5.2 The Execution Monitor

The execution monitoring phase is one of the most delicate parts of the system we are designing. In fact, the proper functioning of the system depends very strongly on the decisions made during the execution of the agents' plans. The considerations we make in this section are to be taken as a preliminary study of the issues involved in the design of an execution monitor for a system that integrates a planner and a scheduler.

One of the basic roles this component plays is to act as a diagnostics-agent. Its main function is to "tidy up" the execution of the plan by handling the intrinsic uncertainty of the environment. Notice that we are not dealing with sensorial uncertainty because of the assumptions made earlier; rather, we are taking into account uncertainty on the *outcome* of the tasks the agents perform (section 2.1).

Once the scheduler has generated a suitable schedule for the plan specification, the agents' tasks are dispatched. The execution of the team's mission is followed by the execution monitor, which is responsible for generating schedule adjustments and, eventually, re-planning directives when particular situations arise. Indeed, the execution monitor is responsible for diagnosing the current situation and maintaining system-wide coherence. These operations rely upon feedback from the lower levels at which the execution is actually taking place, namely the interface between the agents and Level 2 of the hierarchy we have described.

It is quite safe to say that we cannot count on successful completion of the plan without having to adjust some parameters at execution time. Indeed, the very fact that the initial executable plan is compiled based upon *expected* task completion times makes the schedule unreliable. Moreover, since the plans that are generated do not contain sensing actions (we cannot compile conditional plans - see section 2.1), we will never be sure of the environmental conditions in which the agents carry out their tasks; therefore we will never be sure that an agent is even *capable* of completing a task.

Specifically for these reasons, there are at least four types of situations the execution monitor must be able to recognize:

1. an agent anticipates the end-time for the execution of a task
2. an agent exceeds a task's predicted duration
3. an agent fails to carry out a task but is capable of executing its next task
4. an agent fails completely

To cope with the first type of situation, the execution monitor simply updates the plan constraints and propagates them through the temporal network so as to obtain a new schedule. This is a backward-shifting operation which also takes into account the new resource contention situation which derives from shifting the tasks.

The second type of occurrence induces a similar behavior (forward shifting), except for the fact that it recognizes situations in which it may be more convenient to inhibit the operation of the delaying agent rather than propagating the delay (see case four).

As for the third case, the monitor must eliminate from the plan all tasks that depend on the failed one and propagate the new situation through the temporal network. Again, the monitor must decide whether it is better to issue a re-planning directive or to try to adapt the current schedule.

As we pointed out earlier, all agents are modelled as resources, and each task "uses" the resource corresponding to the agent that is executing the task. In fact, if a situation of type four arises, the monitor collapses the agent-resource and computes a new time allocation for the tasks given the new situation. Notice that the execution is now taking place with one less agent, therefore all tasks relative to that agent (and those that depend on these tasks) will be skipped. Clearly, the monitor must be able to determine whether their execution is "important enough" to issue a re-planning signal so as to fulfill the initial goal of the team without the malfunctioning agent. Moreover, this decision is rather critical also because it may be more convenient to have another team with more agents take care of the situation.

In this context, it is obvious that many more situations can arise than those we have categorized. Nonetheless, we feel this is a first approach in formulating an albeit partial methodology to approach the execution monitoring problem.

Along with a variety of approaches to the execution problem, namely reactive, predictive and mixed approaches (described in [3]), we are also focusing on the specific issues that arise in multi-agent settings. These issues include, but are not limited to, the problems mentioned earlier in this section, such as deciding when a schedule is "over-weakened" by the exogenous events in order to issue a re-planning directive.

6 An Application: the Health-Care Institution Domain

Before concluding, we would like to show some aspects of the system's performance in an example domain we think captures some issues concerning real world deployment and practical use of multi-agent systems.

The scenario we present is a health-care institution in which the agents are involved in facility administration and human assistance. During a day's work, the agents are given tasks to carry out which contribute to the maintenance and administration of the facility. These tasks also include aiding the patients of the institution.

All agents in the facility are capable of moving from one room to another. Basic obstacle avoidance and path planning functionalities are implemented as reactive behaviors. Furthermore, depending on the type of agent, the following other reactive behaviors are implemented:

- walking patients from one room to another
- making beds
- clearing beds from objects
- loading meals onto a built-in rack
- serving meals to patients

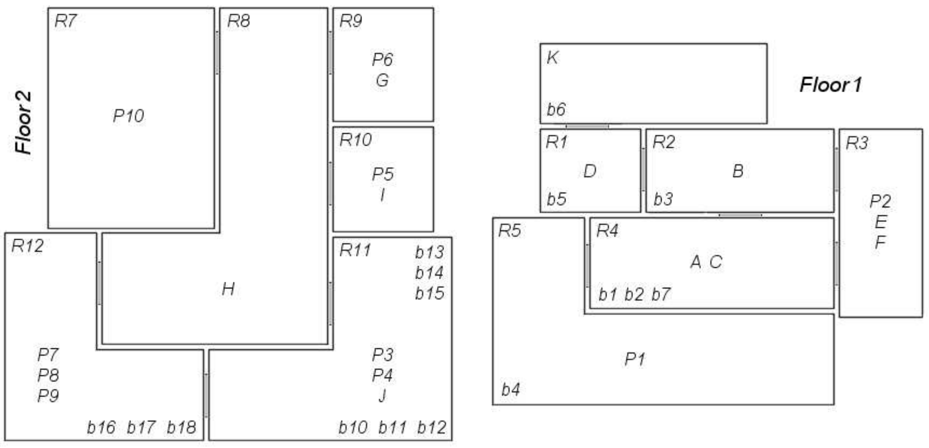

Figure 6: Two floors of the health-care institution

There are many teams in our hospital, all of which are made up of "specialized" agents. Given the environment depicted in *fig. 6*, we will follow the activities of two teams:

Team 1:
  location: floor 1
  agents: A, B, C, D, E, F
  capabilities: make/clear bed, load/serve meal
Team 2:
  location: floor 2
  agents: G, H, I, J
  capabilities: make/clear bed, walk patient

Let us watch what happens during one period of the agent supervisor's activity, specifically when it detects that the following "environmental requirements" are not fulfilled: some beds need making on both floors (mainly on floor 1), some patients on floor 1 need to be fed and some patients on floor two need to be walked to medical examination appointments.

1. The supervisor agent dispatches the following messages:
   `To: AgentBroker(Team 1) <make beds, feed patients - floor 1>`
   `To: AgentBroker(Team 2) <make beds, do walking - floor 2>`
2. The AgentBrokers refine the directives contained in the messages, and obtain the following set of goals:
   Team 1:
   `(not (needs_meal P1))` $\wedge$ `(not (needs_meal P2))` $\wedge$
   `(not (unmade B2))` $\wedge$ `(not (unmade B3))` $\wedge$

```
(not (unmade B1)) ∧ (not (unmade B4)) ∧
(not (unmade B5)) ∧ (not (unmade B6)) ∧
(not (unmade B7))
```
Team 2:
```
(is_in_person_room P3 R7) ∧ (is_in_person_room P4 R7) ∧
(is_in_person_room P5 R8) ∧ (is_in_person_room P6 R8) ∧
(is_in_person_room P7 R11) ∧ (is_in_person_room P8 R11) ∧
(is_in_person_room P9 R11) ∧ (not (unmade B12)) ∧
(not (unmade B17)) ∧ (not (unmade B14))
```

The compilation of team goals into logically correct and efficient plans occurs through the team planners. When BLACKBOX is called upon, it is given the goals and a description of the environmental conditions in which the actions to satisfy the goals are to take place. The above mentioned goals yield the two plans shown in *fig. 7*.

task	action	task	action
1	1: make-bed a b7 r4	1	1: goto-room g r9 r8
2	1: clear-bed a b1 r4	2	1: goto-room i r10 r8
3	1: goto-room b r2 r1	3	1: walk-person j p5 r10 r8
4	1: goto-room e r3 r2	4	1: goto-room h r8 r9
5	1: make-bed a b2 r4	5	2: goto-room g r8 r12
6	1: goto-room f r3 r2	6	2: goto-room j r8 r12
7	1: goto-room c r4 r5	7	2: goto-room i r8 r12
8	1: goto-room d r1 k	8	2: walk-person h p6 r9 r8
9	2: goto-room b r1 k	9	3: clear-bed g b17 r12
10	2: clear-bed c b4 r5	10	3: walk-person i p7 r12 r11
11	2: make-bed a b1 r4	11	3: walk-person j p9 r12 r11
12	2: clear-bed e b3 r2	12	4: walk-person j p4 r11 r8
13	2: get-meal d p1 k	13	4: walk-person i p3 r11 r8
14	2: clear-bed d b6 k	14	4: goto-room h r8 r12
15	2: goto-room f r2 r1	15	4: walk-person g p8 r12 r11
16	3: clear-bed f b5 r1	16	5: make-bed g b14 r11
17	3: make-bed c b4 r5	17 (← 12)	5: walk-person i p4 r8 r7
18	3: make-bed e b3 r2	18	5: make-bed g b12 r11
19 (← 4)	3: make-bed b b6 k	19 (← 9)	5: make-bed h b17 r12
20	3: get-meal b p2 k	20 (← 13)	5: walk-person j p3 r8 r7
21	3: goto-room d k r1		
22	4: goto-room b k r1		
23	4: goto-room d r1 r2		
24	4: make-bed f b5 r1		
25	5: goto-room d r2 r4		
26	5: goto-room b r1 r2		
27	6: goto-room b r2 r3		
28	6: goto-room d r4 r5		
29	7: serve-meal d p1 r5		
30	7: serve-meal b p2 r3		

Figure 7: Plans for Team 1 (left) and Team 2 (right)

In order to obtain a correct and efficient allocation in time of the agents' actions, the two plans are integrated with time and resource constraints. The resulting partial order plans, which are represented in the form of graphs in *fig. 8*, are submitted to O-OSCAR. These figures show the tasks and the causal links between them (arrows), i.e. precedences between tasks.

Some details of the "adaptation" phase are annotated in *fig. 7*, namely the implicit causal links found in BLACKBOX's output (in the task column). Note that logical parallelism of actions is dealt with by adding resource constraints. For instance, agent A in Team 1 (top graph) does not carry out tasks 1, 2 and 5 *at the same time* because they all use the binary capacity resource relative to agent A.

The initial schedule which is computed for the two teams is shown in *fig. 9*. At this point, we have obtained an executable plan, in other words, an allocation in time of activities the two teams of agents

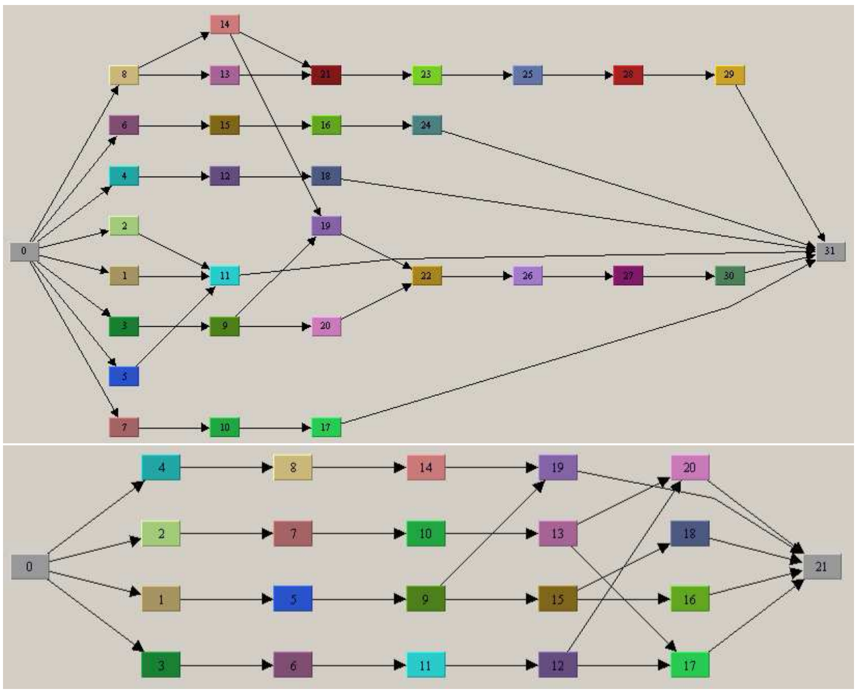

Figure 8: Plans with time and resource constraints for Teams 1 (above) and 2 (below)

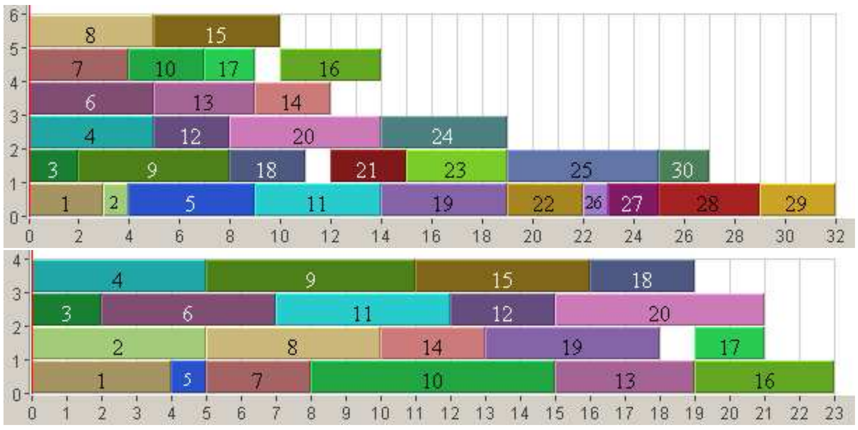

Figure 9: Initial execution schedule for Teams 1 (above) and 2 (below)

have to perform in order to satisfy the initial demands of the facility which the supervisor agent singled out through its sensors.

Notice that the plans are *executable*, which does not mean they will necessarily *terminate*. Specifically for the reasons we have discussed in section 5.2, we can see that the level of reliability of the system depends strongly on the execution monitor.

Fig. 10 shows the time consumption for the computational phases.

Phase	**Platform**	**CPU**
Planning (BlackBox)	AMD Athlon 1.2 GHz	203ms (Team 1) 6773ms (Team 2)
Scheduling (O-Oscar)	AMD Athlon 1.6 GHz	77000ms (Team 1) 31000ms (Team 2)

Figure 10: Time consumption for the computational phases

7 Conclusions and Future Work

In this paper we have partly described the high levels of a framework for controlling the activities of multi-agent systems. We have presented a hierarchical structure which implements the high-level decisional and coordination mechanism for a colony of robotic agents whose function is to produce active services in a civil environment such as a health care facility.

We have discussed how the two levels of the hierarchy we are implementing (in a preliminary, simulated version of the system) can contribute to the coordination of the agents by periodically generating plans and task assignments, through a serialized planning and scheduling procedure. The feasibility of a centralized symbolic planning procedure for an entire team of agents is obtained by means of delegating a scheduler to ensure the satisfaction of time- and resource-bound constraints, which are expunged from the planning phase.

An execution monitor is responsible for diagnosing the state of execution of plans by deriving diagnostic information from the agents' feedback and/or direct observation of the environment, and issuing the appropriate directives when needed.

From the earliest stage of development, we have set ourselves the goal to use pre-existing components, in particular BlackBox and O-Oscar. In fact, there is an overwhelming quantity of general-purpose planners which have yet to find an application other than that of participating in the AIPS competition. Also thanks to the high degree of competition that has been introduced, many high performance symbolic planners have been developed, and should be taken into consideration for real-world applications.

Through our work we have come into contact with a series of issues specific to the multi-agent nature of the domains in which we want to plan. In fact, the intrinsic "parallelism" contained in multi-agent domains complicates the task of finding a plan. We are beginning to gather clues as to how to tackle this problem using a centralized planning paradigm. In our application we have found that using SAT-based planning can help to keep complexity-related hazards under control. In fact, our work has given a partial confirmation to the fact that SAT-based planning can be very efficient when working in parallel (non-sequential) domains (see [13]), as is the case in a multi-agent context.

Our system relies upon the assumption that it is possible to carry out the two phase planning and scheduling procedure relatively fast. In fact, even if overloading the two subsystems is partially avoided during execution monitoring - the execution monitor is capable of "solving" some situations without issuing re-planning or re-scheduling directives - BlackBox and O-Oscar are invoked relatively often (similar concepts are expressed in [8]). Thanks to

- the separation of the planning procedure in a "strictly" logical planning phase and a scheduling phase that takes into account time- and resource-related constraints,
- the use of efficient off-the-shelf components which particularly suit the requirements of our domain

we are able to tackle even complex problems in an acceptable time-frame.

The fine-tuning of the execution monitoring phase is particularly important to obtain good performance. Moreover, it is becoming clear to us that the policies implemented in the execution monitor are extremely dependant on the execution conditions of specific domains. Thus, devising general techniques for execution monitoring will require a substantial research effort. We are looking forward to expanding the number of domains in which execution monitoring can be successfully applied in conjunction with symbolic planning.

Due to the distributed nature of the multi-robot context, the current implementation of the system is based upon the CORBA infrastructure. Specifically, since CORBA makes distribution completely transparent to the programmer, we have been able to de-couple modules that are logically atomic. This has a series of advantages, most important of which is probably the fact that it is possible to spread the computational efforts of the planning and scheduling phases over many platforms.

In conclusion, we have (partially) presented the health-care institution domain. In our opinion, this domain certainly forebodes further refinements, chiefly because it contextualizes multi-agent planning and scheduling in a realistic scenario.

Acknowledgments. This research is supported by MIUR (Italian Ministry for Education, University and Research) under project "Sistema multi-agenti con componenti fisse e robotiche mobili intelligenti" (A Multi-Agent System with Intelligent Fixed and Mobile Robotic Components), and also by ASI (Italian Space Agency). The Authors are part of the Planning and Scheduling Team [PST] at ISTC-CNR and would like to thank the other members of the team for their continuous support. They are particularly grateful to Riccardo Rasconi for his recent work on O-Oscar, which has been instrumental to the research described in this paper. Finally, Federico Pecora expresses his most sincere gratitude to his supervisor, Prof. Luigia Carlucci Aiello, for having inspired him to pursue his studies in AI and granting him the opportunity to join the PST.

References

[1] Botelho, S., and Alami, R. Robots that Cooperatively Enhance their Plans. In *Proceedings of DARS-2000, Tennessee, USA* (2000).

[2] Cesta, A. A Distributed Environment with Software and Robotic Agents for Active User Support. Tech. rep., ISTC-CNR [PST], Italian National Research Council, 2002. In preparation.

[3] Cesta, A., Cortellessa, G., Oddi, A., Policella, N., and Rasconi, R. Approaching the Schedule Execution Problem with O-Oscar. Internal Note PST@ISTC-CNR, October 2002.

[4] Cesta, A., Cortellessa, G., Oddi, A., Policella, N., and Susi, A. A constraint-based architecture for flexible support to activity scheduling. In *Proceedings of 7th Congress of the Italian Association for Artificial Intelligence* (2001).

[5] Cesta, A., Oddi, A., and Smith, S. A Constrained-Based Method for Project Scheduling with Time Windows. *Journal of Heuristics 8* (2002), 109–136.

[6] Currie, K., and Tate, A. O-Plan: The Open Planning Architecture. *Artificial Intelligence 52* (1991), 49–86.

[7] Dean, T., and Kambhampati, S. Planning and Scheduling. In *CRC Handbook of Computer Science and Engineering*. CRC, 1995.

[8] desJardins, M. E., Durfee, E. H., Ortiz, C. L., and Wolverton, M. J. A Survey of Research in Distributed, Continual Planning. In *AI Magazine* (1999), pp. 13–22.

[9] Ghallab, M., and Laruelle, H. Representation and Control in IxTeT, a Temporal Planner. In *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)* (1994).

[10] Huang, Y.-C., Selman, B., and Kautz, H. Control Knowledge in Planning: Benefits and Tradeoffs. In *Proc. AAAI-99, Orlando, FL* (1999).

[11] Jonsson, A., Morris, P., Muscettola, N., Rajan, K., and Smith, B. Planning in Interplanetary Space: Theory and Practice. In *Proceedings of the Fifth Int. Conf. on Artificial Intelligence Planning and Scheduling (AIPS-00)* (2000).

[12] Kautz, H., Fox, D., Etzioni, O., Borriello, G., and Arnstein, L. An Overview of the Assisted Cognition Project. In *Proceedings of the AAAI Workshop on Automation as Caregiver* (2002).

[13] Kautz, H., and Selman, B. Unifying SAT-based and Graph-based Planning. In *Proc. IJCAI-99, Stockholm* (1999).

[14] Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the Design Automation Conference* (2001).

[15] Pollack, M. E., Engberg, S., Matthews, J. T., Thrun, S., Brown, L., Colbry, D., Orosz, C., Peintner, B., Ramakrishnan, S., Dunbar-Jacob, J., McCarthy, C., Montemerlo, M., Pineau, J., and Roy, N. Pearl: A Mobile Robotic Assistant for the Elderly. In *Proceedings of the AAAI Workshop on Automation as Caregiver* (2002).

[16] Srivastava, B., Kambhampati, R., and Do, M. B. Planning the Project Management Way: Efficient Planning by Effective Integration of Causal and Resource Reasoning in RealPlan. *Artificial Intelligence 131* (2001), 73–134.