A Constraint-Based Architecture for Flexible Support to Activity Scheduling

Amedeo Cesta[1], Gabriella Cortellessa[1], Angelo Oddi[1], Nicola Policella[1], and Angelo Susi[1,2]

[1] IP-CNR, National Research Council of Italy, Viale Marx 15, I-00137 Rome, Italy
{cesta,corte,oddi,policella}@ip.rm.cnr.it
[2] Automated Reasoning Systems (SRA), ITC-IRST, Via Sommarive 18 - Loc. Pantè, I-38050 Povo, Trento, Italy
susi@irst.itc.it

Abstract. The O-OSCAR software architecture is a problem solving environment for complex scheduling problem that is based on a constraint-based representation. On top of this core representation a problem solver module and a schedule execution system guarantee a complete support to address a scheduling problem. Furthermore, a rather sophisticated interaction module allows users to maintain control on different phases of schedule management.

1 Introduction

In several real application domains it is important that planning architectures support the user in all the phases a plan may go through, e.g., planning problem definition, solution synthesis, and execution monitoring. This kind of support is increasingly useful when the applications become critical, like for example in space missions support systems. We refer to all the phases of a plan with the term *"plan life-cycle"*. This paper is aimed at describing how artificial intelligence techniques for planning and scheduling can be integrated in a software architecture for offering services to users during plan life-cycle. We will show features of the software architecture O-OSCAR (for Object-Oriented SCheduling ARchitecture) and will underscore several of its recent developments aimed at creating a more complete supporting environment.

A key aspect in O-OSCAR design is the constraint-based reference model that we have intensively used to shape all the services it is able to offer. We have targeted a class of scheduling problems that involve quite complex time and resource constraints and applications domain related to automated management of space missions. Time constraints may model set-up times for instruments, target visibility windows, transmission times (e.g., to represent memory dump), deadline constraints, etc. Resource constraints can represent capacity of on board memory (e.g., tape recorder or solid state recorder), transmission channel capacity, and energy bounds (e.g., limitation on the number of active instruments in a space probe). From the scheduling literature point of view target problems have

F. Esposito (Ed.): AI*IA 2001, LNAI 2175, pp. 369–381, 2001.

been the so-called Resource Constrained Project Scheduling Problem (RCPSP) [2] and its variants with Generalized Precedence Relations (RCPSP/max) [7, 4] and with Inventory Constraints [9]. Such problems contain sophisticated constraints like maximum temporal separation between activities, and a variety of resource constraints for activities. In addition, specific attention has been deserved to the interaction of users with the scheduling systems. We have studied what kind of interaction services may contribute to enhance acceptance of the whole approach in real contexts.

This paper is organized as follows: Section 2 introduces the O-OSCAR architecture while the following sections describe specific aspects of the main software components: the constraint data base (Section 3), the problem solver (Section 4), the execution monitor (Section 5), and the user-interaction module (Section 6).

2 A CSP-based Software Architecture

In developing a complete solution to a planning/scheduling problem several basic aspects need to be integrated. They are sketched in Figure 1 that shows the different modules of O-OSCAR.

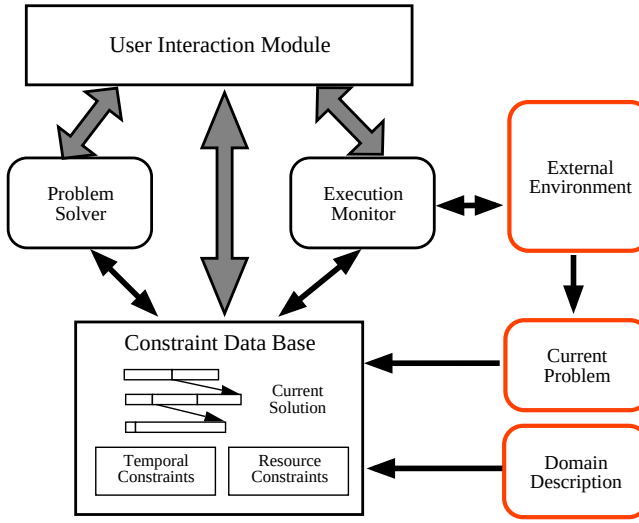

Fig. 1. A high level description of the O-OSCAR architecture

First of all we see (as typical in any computerized task) a box representing the "external environment" that is the part of the real world that is relevant for the problem the software architecture is aimed at supporting. Such an environment is modeled in the architecture according to two distinct aspects:

Domain Representation. The relevant features of the world (the domain) and the rules that regulate its dynamic evolution should be described in a symbolic language. This is the basic knowledge that allows the system to offer services.

Problem Representation. A description of the goals of a current problem is given to specify states of the real world that are "desired" and to be achieved starting from the current world state.

The core component of an architecture that uses a CSP (constraint-based problem solving) approach is the:

Constraint Data Base (CDB). This module offers an active service that automatically takes care of checking/maintaining the satisfaction of the set of constraints representing the domain and the problem. It is in charge of two strictly interconnected aspects:

Domain and Problem Representation. The Domain Representation Language allows the representation of classes of problems and the peculiar domain constraints in the class. At present O-OSCAR is able to solve the class of scheduling problems RCPSP and its variants mentioned before. The Problem Representation Language consists of a set of constraints specifying the activities and their constraints requirements specified in a RCPSP.

Solution Representation and Management. The CSP approach to problem solving is centered on the representation, modification and maintenance of a solution. Such solution in O-OSCAR consists of a representation designed on top of specialized constraint reasoners. The constraint model represents particular aspects of the domain (e.g., temporal features, resource availability) and is called into play when changes are performed by a problem solver, the execution monitor, or a user. The solution manager is usually endowed with a set of primitives for communicate changes and formulate queries.

The CDB is the key part of the approach and should be designed taking efficiency into account. Special functionalities built on it allow to obtain a complete support to plan life-cycle. Three modules contribute to complete the scenario and are in charge of the following tasks:

Automated Problem Solving. This module guides the search for a solution. It is endowed with two main features: (a) an open framework to perform the search for a solution; (b) heuristic knowledge (it is able to identify and represent heuristics to guide search and avoid computational burden).

Execution Monitoring. Once a solution to a given problem is obtained, this module closes the loop with the real world. It is responsible for dispatching the plan activities for execution and sensing the outcome of this execution and of the relevant aspect of the world. Sensed information (e.g., successful or delayed execution of activities, machine safe or fault status) is used to update the CBD and maintain a representation synchronized with the evolution of the real world.

User-System Interaction. This module allows the interaction of the user with the CDB and the two previously described services. The interaction functionalities may vary from more or less sophisticated visualization services, to a set of complex manipulation functionalities on the solution allowed to the user.

The advantage of using CSP as the basic representation mechanism is that additional services can be added relying on the same representation. Furthermore, the same basic service can be incrementally updated inserting more sophisticated functionalities (for example a portfolio of solution algorithms relying on different search strategies). In this way we have been able to incrementally evolve from an architecture for solving only disjunctive scheduling problems [5], to the current tool that solves more sophisticated problems.

3 The Constraint Data-Base

In developing the CDB for resource constrained scheduling problems two domain features need support: (a) quantitative temporal constraints where both minimum and maximum separation constraints can be specified and (b) multi-capacity resources (e.g., capacity greater than one). As shown in Figure 1 the CDB contains two specialized constraint reasoners, the first for temporal constraint management (with an expressivity correspondent to the Simple Temporal Problem [8]), the second for representing resource constraints and their status of satisfaction. In particular data structures called *resource profiles* are dynamically maintained that represent the current resource consumption. This lower layer is hidden to the external user by a more external layer that essentially represent the features of the Domain Description Language. For the current purposes the interface language at the higher level identifies the typical aspects involved in a schedule, namely activities, resources, constraints and decisions. Choosing a general representation allows us to interface our work not only with constraint-based algorithms but also with typical abstractions from Operations Research (the representation features are similar to those defined in [12] even if with somehow different semantics).

3.1 Multi-capacity Resources

A peculiar feature of O-OSCAR is the type of resources and the corresponding constraints that is possible to represent. We can represent both disjunctive and cumulative resources and among the latter we can model both reusable and consumable resources.

Reusable Resources. Reusable resources can serve multiple requests at the same time but have a maximum level of resource requirement (the capacity level) that cannot be violated. Examples are very common in manufacturing, and supply chain management but also when describing human resources. Typical behavior of reusable resources is that the resource requirement of an activity

happens during activity execution but when the activity ends the resource level is re-established at the previous level and it can be "reused".

Consumable resources. Resources are consumable when an activity can permanently destroy or produce a quantity of it. The fuel level in a car tank or the amount of money in a bank safe are examples of this kind of resources. The resource constraint to represent in such a case is that the level of resource cannot exceed a maximum value or assume values below a minimum level. The complexity introduced by this kind of resources is quite high although they are very common in nature and few works have explicitly dealt with them (exception being the work on inventory constraints [9, 1]).

It is worth specifying that the resource bounds are constant values in these problems and that a resource consumption is associated with activity start times, and a resource production with activity end times. We have studied the synthesis of specific resource constraint propagation rules to prune the search space of inconsistent solutions. Following an approach introduced in [6] and extending it, propagation rules are obtained computing upper and lower bounds for the resource consumption and reasoning about these bounds in any time-point in which a change in the resource level happens. Given t one of these time points, the activities can be subdivided in three sets: the set of activities that for sure precedes t, the set that follows, and the set that currently overlaps with t. Reasoning on the content of this last set allows to synthesize propagation rules that add new necessary temporal constraints to the problem description.

> **Example:** Given a single resource and three activities: a_1 that consumes 2 units of resource; a_2 that produces 1 unit; a_3 that produces 2 units. The resource is consumable with bounds $[0, 5]$. Let us suppose that according to current temporal constraints the possible start-time of activities may be in the following intervals:
> - a_1 $[0, 2]$
> - a_2 $[2, 8]$
> - a_3 $[1, 10]$
>
> If we consider the time instant $t = 2$, the activity a_1 precedes or coincides with t, while the other two activities are not ordered with respect to t. If we assume that a_3 happens after t the resource constraints are violated so we can detect the implicit constraints that the upper bound of a_3 $ub_3 \leq 2$. Imposing explicitly this constraint we prune the search space of useless potential assignments.

This is just an example of the propagation rules that are fully described in [11].

4 Constraint-Guided Problem Solving

The generic problem solving algorithm that is used within O-OSCAR follows the *profile-based schema* introduced in [3] and shown in Figure 2. It is a basic greedy algorithm that works on a temporally consistent solution, computes the resource violations (the conflicts) using the information on the resource profiles,

and then tries to build a solution that is "resource consistent" by imposing some additional ordering constraints between activities to level the resource contention peaks. This is iterated until either a resource consistent solution is obtained or a dead-end encountered. The greedy algorithm can be inserted in an optimization schema for obtaining multiple, increasingly better, solutions. A technique we have used in various domains integrates the greedy search in a random sampling loop. The technique, introduced in [10], allows to escape local minima, taking advantage of the random restart biased by the heuristic estimation.

```
CreateSchedule(Problem)
1.    CDB ← BuildCspRepresentation(Problem)
2.    loop
3.        Conflicts ← ComputeConflicts(CDB)
4.        if (Conflicts = ∅)
5.        then return(CDB)
6.        else
7.            if ExistUnsolvableConflict(Conflicts)
8.            then return(Failure)
9.            else
10.               Conflict ← SelectConflict(Conflicts)
11.               Precedence ← SelectPrecedence(Conflict)
12.               Post&Update(Precedence,CDB)
13.   end-loop
14.   end
```

Fig. 2. A Basic One-Pass Profile-Based Algorithm

Relevant to study are not only the techniques for computing conflicts (function `ComputeConflicts` in Figure 2) but also the heuristic estimators for the choice points represented by `SelectConflict` and `SelectPrecedence`. In O-OSCAR, specific knowledge has been inserted from our own work fully described in [4]. It is worth noting that different search spaces are designed according to the kind of resources needed to represent a domain. Different heuristics are shown to be effective in capturing the structure of the search space of different problems like for examples problems with maximum separations and reusable resources (see [4]) and problems with consumable resources (see [11]).

5 The Plan Execution Monitor

When a schedule has to be executed in a real working environment, differences are possible between the modeled approximate reality used to produce the schedule and the real world. This can arrive to cause, in extreme cases, the impossibility to use the computed plan. To minimize the risk, two possible kinds of strategies are applicable: one predictive and another reactive. Predictive strategies act during problem solving and are based on the computation of schedules

that are robust with respect to a specified set of unexpected events. Reactive strategies are based on methods that, during the execution phase, dynamically adjust the current schedule according to the evolution of its execution in the real environment. In O-OSCAR we have currently followed a reactive approach based on schedule repair. We have implemented an execution control module that continuously updates the constraint data-base representing the current status of the schedule to describe its variations at execution time. The execution module contains a basic portfolio of repair strategies that specifies the reaction methods to undesired events.

The Execution Algorithm. Figure 3 gives a very high level description of the regular activities of the execution monitor. Its input parameters represent the main communication channels that are open when it is working, namely the constraint data base (`CDB`) that contains the current solution to a problem, the external world (`extEnvironment`), and a new knowledge source, called `RepairStrategies`, that is internal to the execution monitor but can be object of specific interaction with the user.

```
ExecuteSchedule(CDB,extEnvironment,repairStrategies)
1.   t ← 0;
2.   currentSchedule ← EstractSolution(CDB)
3.   loop
4.      if(ExecutionFinished(currentSchedule)) then Return(TerminationSuccess)
5.      else
6.         SensingReport ← SenseEnv(extEnvironment)
7.         ConflictSet ← ConflictCheck(SensingReport, currentSchedule)
8.         if (ConflictSet ≠ ∅) then
9.            CDB ← Repair(CDB, ConflictSet, RepairStrategies)
10.           if (Inconsistent(CDB)) then Return(TerminationFailure)
11.           currentSchedule ← ExtractSolution(CDB)
12.        DispatchSet ← ComputeExecutableActions(currentSchedule)
13.        Dispatch(DispatchSet, extEnvironment)
14.     t ← t + 1;
15.  end-loop.
```

Fig. 3. A Sketchy View of the Execution Algorithm

The executor, according to increasing time stamps, performs a basic cycle that consists of: (a) sensing the environment (Step 6); (b) computing conflicts with respect to the current represented scenario (Step 7); (c) in case of conflicts, updating the CDB and attempting at repairing the current schedule (steps 8-11); (d) in case of successful repair, continuing execution by dispatching actions in the external environment (steps 12-13).

The quite general abstraction of this presentation hides most of the work realized for constructing an executor for O-OSCAR. For the sake of space we add only few words aimed at clarifying the use of the CDB and the current repairing actions. After sensing the environment the executor generates a conflict report

that is used to update the constraint data base. Information in the execution report is used to synchronize the representation in the CDB and the real word. At present the update information is purely temporal. It represents either delays in the start time of activities (additional separation constraint with respect to the beginning of the time-line) or delays in the termination of an activity (change of duration constraints). When the CDB contains the new propagated situation, if this is inconsistent an unrecoverable failure is declared, otherwise an updated schedule status is used by the monitor to continue execution. The interaction module updates the information visualization highlighting the differences with respect to nominal schedule (see next section for visualization features). In case the update of the schedule ends up in a consistent scenario the execution of the updated schedule continues. If the resulting schedule contains some conflicts (shifting activities may cause new contention peaks in the solution) the solution needs to be fixed before continuing execution. The user can influence the monitor fixing a particular repair strategy or leave complete initiative to the monitor to choose the repair policy. The current solution in this concern is rather simple but it can be seen as an enabling infrastructure to realize further features. On one hand we have to experiment different protocols for exchange of initiative between the user and the execution monitor to choose repairing actions, on another we need further investigation to understand better how to make the user more comfortable in doing direct modifications on the current schedule according to a problematic evolution of the execution process.

6 An Interaction Module for Continuous Support

As we said before, we aim at *putting the user in the loop* by creating supporting tools that allow the user to maintain control on the evolving situation represented in the CDB. Our aim is to study the possibility offered by the mixed environment in which human being and automated systems contribute to solving/managing complex problems (an area also referred to as *mixed-initiative problem solving*). In this view a scheduling architecture should be endowed with a specific module that takes care of the continuous communication between users and system. Goal of this module is to favor collaboration and to allow a synergetic interaction between the intuitions and specific knowledge of human beings from one hand and the computational power of the automated system from another.
Figure 4 gives an high level view of the kind of functionalities the interaction module is supposed to have. The idea is that users are supported in the whole plan life-cycle, having specialized support for Domain and Problem Definition, Problem Solving and Execution Monitoring.

The current status of the Interaction Module of O-OSCAR offers a first suit of functionalities for interacting with the different aspects of the systems. During each phase a set of specific supporting tools helps the user to perform basic tasks. We are evolving from an interaction module aiming at showing smart problem solving features [5], toward a tool in which the Interaction Module is a first citizen in the architecture adding value to the system as a whole.

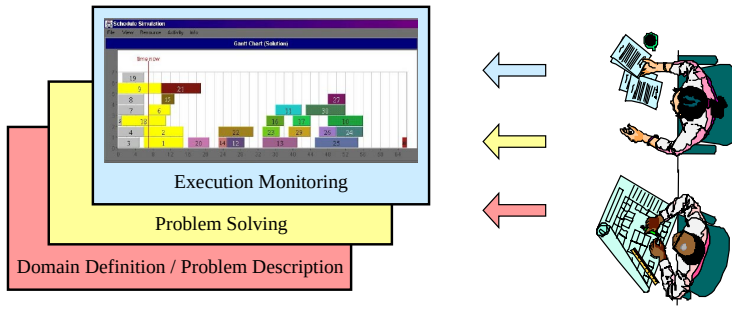

Fig. 4. Supporting Plan Life-Cycle

The most important goals of such a module are: (a) to give support to the scheduling problem definition; (b) to create a first set of features for mixed-initiative problem solving; (c) to take advantage of the flexibility of the CSP representation for offering services to the user for incremental and continuous intervention on the constraints represented in the CDB also during the execution phase, reacting to changes in the scheduling domain. (d) to offer basic inspection mechanisms to allow the user to monitor the plan execution phase.

A feature of the O-OSCAR interaction module is the ability to hide the system complexity to the user maintaining for him a level of intervention on the system. This is because it represents the constraint based schedule representation maintained in the system in a way understandable for the user.

The interaction module has also an object oriented design, is entirely written in Java and interacts with the C++ implementation of the rest of the architecture through a client-server protocol.

In the next subsection we describe how the big picture of Figure 4 is currently implemented in O-OSCAR and describe for each phase in the plan life-cycle which are the implemented services.

6.1 Current Visual Tools

The O-OSCAR Interaction Module made possible for the user to directly manipulate the graphical objects that represent the entities of the scheduling problem during the three phases described before. As said, the user can have distinct graphical environments for every phase but may also maintain a parallel view of the entire scheduling process. The interactive approach based on graphical object gives the system a level of "usability". The coordinated vision of all the available resources, in relation with all the three interaction phases, gives the user a "feeling of maintained control" on the system favoring the intermixing/shift of initiative.

Domain and Problem Definition. This graphical environment allows the user to manage the first part of the scheduling process. According with the CSP approach, used in the O-OSCAR system, a problem is represented as a graph of

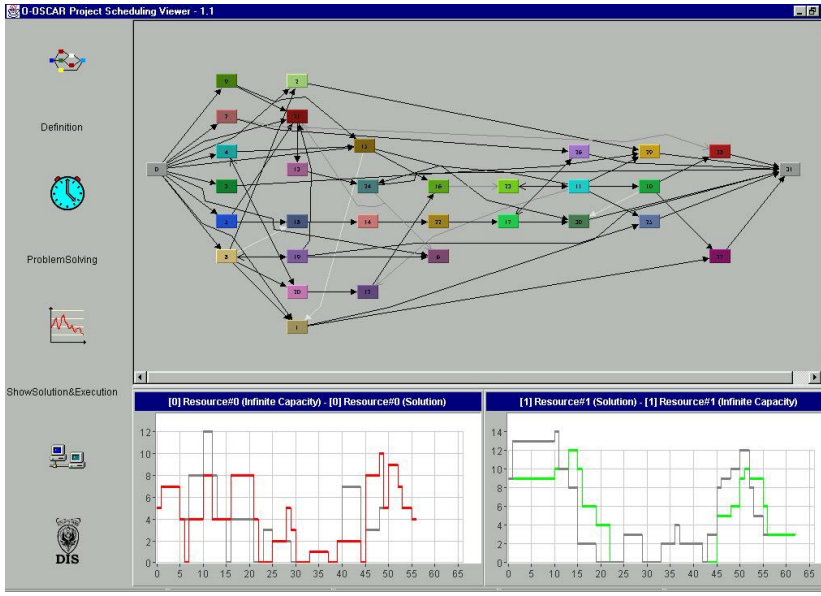

Fig. 5. Interaction for Problem Definition and Resolution

temporal constraints between activities. The system offers a graph manipulation window that allows the user to load and save the problem, add and remove activities, constraints and resources, modify the characteristics of every scheduling entity directly manipulating the graphical representation. In this way the user can manage dynamical situation, quite frequent in real contexts, in which either new activities arrive on-line and their accommodation in the existing schedule should be taken care of or resources breakdown and the schedule should be fixed in order to minimize total disruption.

Figure 5 shows the current interface for Problem Definition phase intermixed with features of the Problem Solving phase. In the upper part of the figure the graphical representation for a 30 activities scheduling problem. The problem graph is made of active graphical objects, and allow the user to change the problem interactively, for example adding new activities, modifying the temporal constraints network, etc.

Problem Solving. The Interface allows the user to ask for the solution of the represented problem. It is possible to select one solution strategy in a portfolio of specified strategies. Once a solution has been retrieved the system allows the user to choose between a certain number of possible Gantt diagrams, resource allocation diagrams or resource profile diagrams. The possibility of observing the solution from different perspectives, increase the user knowledge and stimulate in different ways his perception with respect to the problem at hand. In this way the user may have support for inserting his own intuition and experience in taking part in the decisional problem.

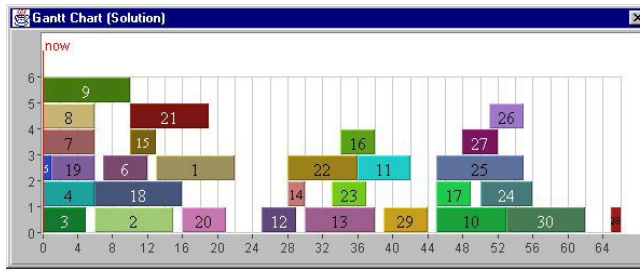

Fig. 6. Gantt Chart of a solved RCPSP problem

In addition the user can manually modify the solution while the system incrementally update the internal representation, automatically verifying its consistency. Figure 6 shows one of the Gantt diagrams available for the problem specified before. Additional information disaggregated in single resources is given in the lower part of Figure 5 (the two small windows showing resource profiles) and complete the picture of the current solution status.

Execution Control. Plan execution is dealt with as a separate working environment and represents the final step in the schedule life-cycle. The Interaction Module allows to analyze the execution process in every instant until all operation are executed. In particular the Gantt window is used as the main source of information (see also the upper part of the pictures in Figure 7. The user has as a basic information the current time (a bar with label "time now" in the Figures 6 and 7. The bar evolve dynamically with time on the Gantt and activities change colors according to their status of executed (grey) under execution (yellow) and to be executed (original color). Figure 7 shows the execution status in different time instant on the same problem.

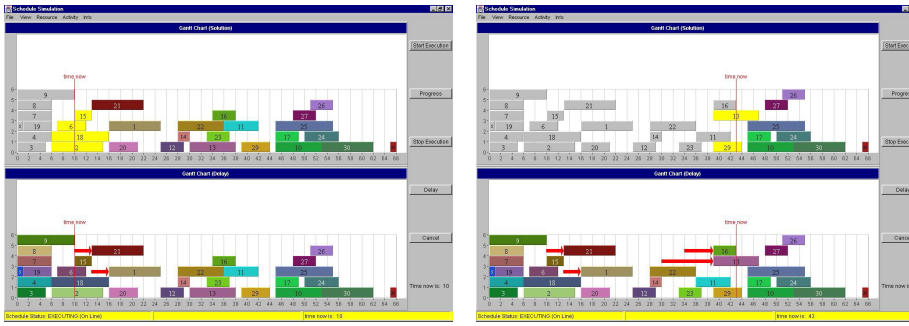

Fig. 7. Visual Inspection of the Execution

During the execution it is possible for the user to verify the differences between the schedule computed during the solution phase and the real operation sequence

that is possible to execute in a given instant. This is shown in the lower windows in the pictures of Figure 7 where little (red) arrows identify delays of an activity with respect to the original schedule. It is possible to select the visualization of the operations execution delay that can be distinguished in two sets:

1. automatic delay, produced by the reactive strategies of the execution algorithm as a reaction to the working environment unexpected events;
2. manual delay, introduced by the user and manually added to the schedule representation using the graphical interfaces editing functionalities.

Further work is underway to endow the Execution Inspection interface with further views on the current problem. A further feature consists of the possibility of re-running off-line the execution of a schedule. In this way we create a sort of "simulation environment" in which the user can learn off-line the effects of certain choices he has made on-line on previous problems. These functionalities are very useful in a real dynamical working environment where decisions have to be taken very quickly and is very important that users has to be rapidly informed about the execution status and failures.

7 Concluding Remarks

In this paper we have reported a set of functionalities we have inserted in the O-OSCAR architecture. With respect to the initial work on O-OSCAR (see [5] for an overview) in our more recent work we have maintained the basic idea in this research project that is choosing constraints as the basic representation vehicle for problem solving, but have deeply scaled up in the kind of functionalities we are able to offer. In particular the main improvements we have described in this paper are:

- the representation of consumable resources adds a modeling capability particularly relevant. In fact it enlarges the class of scheduling problems addressable by O-OSCAR to include problems with power consumption, inventory constraints, etc.
- the execution monitor has allowed us to close the loop with the external world. This opens quite an amount of possibility for addressing real problems. Although at present the executor deals with a limited set of situations (mainly represented as delays in the temporal plan) it is a building block particularly important for actually using this technology.
- a rather sophisticated interaction module that allow the user to perceive the differences among three phases of the plan life-cycle, having different tools available to maintain control of the system behavior.

A lot of possibilities are opened by the current status of O-OSCAR, but a single feature is worth mentioning before closing the paper: the flexibility in manipulation the basic constraint representation with all the module that use the CDB. The fact that the constraint manager offers fully dynamic services has

been a constant focus in our research work and is now somehow emphasized in O-OSCAR where they take the shape of additional service to the users grounded on the constraint representation.

Acknowledgments. This research has been supported by ASI (Italian Space Agency) as part of a joint effort among Dipartimento di Informatica e Sistemistica dell'Universita' di Roma "La Sapienza", Dipartimento di Informatica e Automazione della Terza Universita' di Roma, and IP-CNR [PST], Consiglio Nazionale delle Ricerche, Roma. O-OSCAR developments are currently partially supported by European Space Agency (ESA-ESOC) under contract No.14709/00/D/IM.

References

[1] J.C. Beck. Heuristics for Constraint-Directed Scheduling with Inventory. In *Proceedings of the Fifth Int. Conf. on Artificial Intelligence Planning and Scheduling (AIPS-00)*, 2000.

[2] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operations Research*, 1998.

[3] A. Cesta, A. Oddi, and S.F. Smith. Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In *Proceedings of the Fourth Int. Conf. on Artificial Intelligence Planning Systems (AIPS-98)*, 1998.

[4] A. Cesta, A. Oddi, and S.F. Smith. A Constrained-Based Method for Project Scheduling with Time Windows. *Journal of Heuristics*, 2002.

[5] A. Cesta, A. Oddi, and A. Susi. O-OSCAR: A Flexible Object-Oriented Architecture for Schedule Management in Space Applications. In *Proceedings of the Fifth Int. Symp. on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS-99)*, 1999.

[6] A. Cesta and C. Stella. A Time and Resource Problem for Planning Architectures. In *Proceedings of the Fourth European Conference on Planning (ECP 97)*, 1997.

[7] B. De Reyck and W. Herroelen. A Branch-and-Bound Procedure for the Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations. *European Journal of Operations Research*, 111(1):152–174, 1998.

[8] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[9] K. Neumann and C. Schwindt. Project Scheduling with Inventory Constraints. Technical Report WIOR-572, Universität Karlsruhe, 1999.

[10] A. Oddi and S.F. Smith. Stochastic Procedures for Generating Feasible Schedules. In *Proceedings 14th National Conference on AI (AAAI-97)*, 1997.

[11] N. Policella. Problemi di scheduling con risorse consumabili, risoluzione con approccio a vincoli in una architettura software orientata agli oggetti. Master's thesis, Università di Roma "La Sapienza", March 2001. In Italian.

[12] G. Wolf. Schedule Management: An Object-Oriented Approach. *Decision Support Systems*, 11:373–388, 1994.