# Monitoring Domestic Activities with Scheduling Techniques

**Amedeo Cesta, Gabriella Cortellessa, Federico Pecora and Riccardo Rasconi**

Institute for Cognitive Science and Technology
Italian National Research Council

name.surname@istc.cnr.it

**Abstract**

In this article we describe an application where state-of-the-art Planning & Scheduling techniques are used in the context of the ROBOCARE Project. The project is aimed at responding to the deeply felt issue of ensuring poorly invasive automatic monitoring on the daily activities of elderly and/or impaired people directly in their homes.

## 1 Introduction

In this article we describe an application where state-of-the-art Planning and Scheduling techniques are used in the context of the ROBOCARE Project [1]. The project is aimed at responding to the deeply felt issue of ensuring poorly invasive automatic monitoring on the daily activities of elderly and/or unpaired people directly in their homes. As system acceptance is a primary concern and privacy is generally a very important requirement, the system is designed to keep at a minimum its level of home invasiveness.

The design of the supervision system we present in this article focuses on the idea of utilizing an existing scheduling architecture, namely O-OSCAR (Object-Oriented Scheduling Architecture), extended with a Schedule Execution Monitoring Module (Cesta & Rasconi 2003), which makes it possible to follow the real time execution of a predefined set of scheduled activities, so as to continuously check the adherence of what is being monitored to the initial constraints/requirements. The proved ability of our scheduling technology to keep control of rather general and sophisticated temporal relations among the activities, led us to choose it as a solid platform upon which to base the entire system development.

Similar attempts at using core solving technology in domestic and health-care environments have been made (e.g. (McCarthy & Pollack 2002; Pollack *et al.* 2002)).

Our system is conceived as an innovative application in the field of Pervasive Computing, and it is mainly designed to represent a valid aid for caregivers (the system users), in that it allows them to easily specify the desired behaviour of the assisted person through the introduction of behavioural prescriptions, i.e. preferential timetables for daily meals as well as for medicine taking, prescribing particularly healthy activities whose execution should be checked, and so on. Once the set of activities which represent the desired behaviour of the assisted person has been set by the caregiver, the system is responsible for monitoring its execution; in this context, this basically means to automatically recognize the actions performed by the patient, checking that their execution falls between the prescribed temporal boundaries, and, in case this does not happen, setting off a predefined combination of events, such as suggestions, alarms, or pro-active behaviours of the domotic environment, depending on the specific importance of the event occurred. Non invasiveness is guaranteed by the possibility of managing a high degree of flexibility in the specification of the temporal relationships among the activities involved in the monitoring process. In other words, the assisted person does not have to execute the prescribed tasks according to a strict timing: it is generally possible to specify some “temporal allowance” before a constraint is deemed violated and the appropriate reaction is triggered. The ultimate goal of the monitoring process is thus to witness the proper execution of meaningful tasks performed by the patient as well as to recognize a set of particularly dangerous situations, modeled through pre-specified behavioural patterns.

As system acceptance is a primary concern and privacy is generally a very important requirement, the system is also designed to keep at a minimum its level of home invasiveness.

All the required information on the activities which are actually carried out by the assisted person is provided by periodically polling a set of intelligent sensors, e.g. stereo cameras, spread accross the monitored area. At each polling cycle, the data gathered from these sensors are used to decide if and when a certain action has commenced: the information is immediately passed on to the monitor, which adjusts the schedule accordingly, checking any possible constraint violation.

One of the major problem that has to be tackled in order to make the whole system comfortably usable, is the one of building a suitable front-end for the end-user. In other words, it is necessary to design a modeling framework which allows a user not familiar with general purpose scheduling technology, such as the O-OSCAR scheduling and execution

[1] See the ROBOCARE project home page at http://robocare.istc.cnr.it.

monitoring suite we have deployed, to easily create the plans to be submitted to the subsequent monitoring phase. Moreover, the proposed modeling service should be promptly reconfigurable so as to be directly customizable, depending on the different working environments where it must be employed.

This article is organized as follows: in section 2 we give a general overview of the whole ROBOCARE Domestic Environment (RDE); section 3 describes in detail the Front-end Modeling Framework, while section 4 provides a real-world example; finally, section 5 draws some conclusions.

## 2 The ROBOCARE Domestic Environment

The intelligent system developed for the RDE is devised as a multiplicity of hardware (i.e. sensors) and software agents. All agents can be thought of as peripheral components of a single complex system, whose aim is to create an intelligent and supporting environment. In particular, the beneficiaries of this application are elderly people whose every-day independence may be improved by such a monitoring infrastructure.

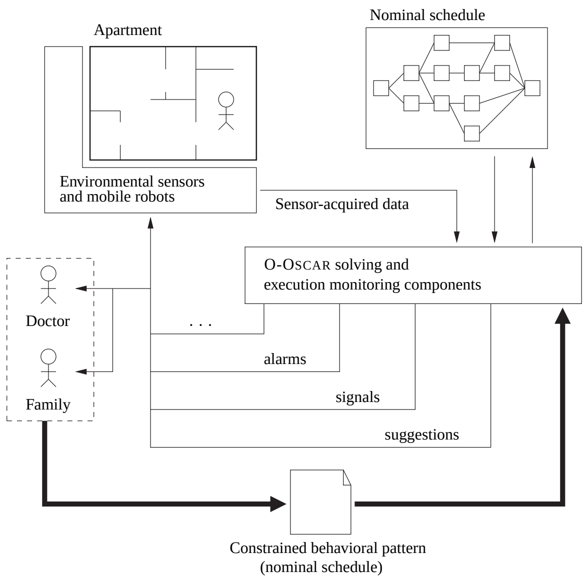
Apartment
Nominal schedule
Environmental sensors and mobile robots
Sensor-acquired data
O-OSCAR solving and execution monitoring components
Doctor
Family
. . .
alarms
signals
suggestions
Constrained behavioral pattern (nominal schedule)

Figure 1: Scheme of principle of the ROBOCARE Domestic Environment.

Fig. 1 shows a system in which environmental sensors, (for instance stereo-cameras) are "coupled" with an execution monitoring module. Together, these two components provide basic services for monitoring the daily schedule of an assisted elderly person in his or her apartment. On one hand, the stereo-camera observes the environment, and by means of its 3D localization capabilities (Bahadori *et al.* 2004), compiles symbolic information (such as the presence, position and permanence of persons and/or objects) determining, to a certain extent, the current state of the environment. This information is then processed by a CSP [2] based execution monitoring module. By means of an internal representation of the assisted person's *nominal* schedule, this module is capable of recognizing inconsistencies in the actual execution of the activities as they are performed by the assisted person. The loop is closed through the output of the execution monitoring service, which can consist in suggestions, alarms and other such signals.

[2] Constraint Satisfaction Problem

Creating such tightly coupled intelligent systems presents an important challenge as it requires the development of intelligent agents which are capable of complex symbolic reasoning tasks and high levels of interaction with humans. The system must provide more than loose co-ordination between its components. It must provide a complete supervision framework which implements a solid infrastructure and maintains a global view of the system and provides control functions for human operators. And, as caring for elderly people is often all about managing routinary tasks such as scheduling walks and daily social events, the system should be capable of planning and scheduling about such events, as well as interpreting them when they occur. The major responsibility of the supervisory system is to *monitor* the daily activities of the assisted human(s), and to recognize if any particular situations occur. For instance, if an assisted elderly person decides to perform an activity which contrasts with a particular medication requirement (e.g., no eating for two hours after having taken a particular medication), the system would recognize the inconsistency and deduce a contingent plan to solve the situation (suggestions, alarms, and so on).

More specifically, the execution monitor accepts the representation of the set of activities that have to be executed in the environment, and basically plays the role of a *supervisor*, that is, attempts to continuously maintain situation awareness. This is achieved for instance by checking that all activities, as well as the global plan, are indeed executed within a specified time; the interface with the environment is realized through a routine which periodically interacts with a set of sensors and checks whether all activities are correctly executed or not. In the negative case, the module is able to assess the criticality of the anomaly and take the most proper action. Representation and management of all the components of the schedule is performed by the O-OSCAR scheduling and execution monitoring suite, which is described in the following paragraphs.

### 2.1 The Execution Monitor

Informally, a schedule consists of a certain number of activities, each requiring some resources in order to be executed. One key point is the fact that activities can in general be temporally constrained, either individually or among one another: for instance, some operation in a schedule might be constrained not to start before, or not to finish after, a certain instant; in addition, there might be several *precedence constraints* between any two activities in the schedule: for instance, activity B might not be allowed to start before the end of activity A, and so forth.

As shown in fig. 1, the execution monitor is interfaced with (a) the core scheduling system, through which it has access to the constraint representation of the schedule (Constraint Data Base - CDB) in its current state, and (b) with the real world, through the environmental sensory services which are responsible for signaling the exogenous events. The Schedule Execution Monitor has been developed as an integration to the O-OSCAR (Object-Oriented SCheduling ARchitecture) tool, an existing constraint-based software architecture for the solution of complex scheduling problems.

As explained in the previous section, the detected information is used to update the CDB in order to maintain the world representation perfectly consistent with the evolution of the real environment; the main issue is that updating the data stored in the representation module in accordance to the information gathered from the environment may introduce some inconsistency in the schedule representation.

We have implemented an Execution Monitor which reacts to these inconsistencies as they are detected, namely attempting to take the schedule back to a consistent state, so as to maintain its executability. The repair action is performed by exploiting state-of-the-art scheduling techniques; traditionally, two main strategies are used for scheduling: the *predictive* and the *reactive* strategy. The predictive approach is used during the solution search process and it is based on the synthesis of *robust* schedules. Robustness is the characteristic which makes a plan able to absorb, to a certain extent, the disturbances possibly produced by exogenous events, with no need of any schedule revision (Policella *et al.* 2004).

The reactive approach, mostly suitable for our purposes, is instead based on a dynamical modification of the schedule during its execution phase, consistently with the evolution of the monitored environment (Cesta & Rasconi 2003).

Independently of the chosen approach, some preventive action is to be taken before firing the rescheduling procedure in order to have the necessary control on schedule repair choices. In other words, we can guide the revision process by preventively constraining the activities, depending on the strategies we want to realize. A number of primitives have been developed which make the dynamic insertion and deletion of temporal constraints possible.

Each primitive directly reflects the most frequent needs encountered in real time schedule management: for instance, the ability to easily impose temporal relationships between different activities in the schedule (*Precedence Constraints*) is invaluable; furthermore, it is often necessary to model typical situations where activities can not start before or end after a specified instant (hence the need of, respectively, *Release Time Constraints* and *Deadline Constraints*) primitives; finally, the two last requirements may be simultaneously needed: we face this situation through the use of a *FixTime Constraints* primitive.

The introduction of the previous primitives allows to manage any temporal relationship among the schedule activities in accordance to the possible changes which may naturally occur in the environment, thus keeping the description of the schedule under execution coherent with the real world situation at all times. More specifically, the utilization of each primitive modifies the structure of the Constraint Data Base, which is the place where all temporal constraints are kept; each time the CDB is changed, specific propagation algorithms are triggered with the aim of (a) checking if the imposition of the latest constraints clashes with previously inserted ones, or (b) resetting the situation to the closest consistent state, in case the constraints are to be removed. The scheduling algorithms utilized to maintain plan consistency greatly exploit these time management features, which are also extensively used to bias the schedule in order to satisfy the caregivers' preferences. By exerting this kind of preventive control, it is possible to impose behavioural biases on each activity, therefore obtaining a solution which best suites specific desires, for instance, by modulating the *temporal slack* of some activities, such as maximum delays, preferred anticipations, and so on, before schedule revision.

A more detailed description of the Execution Monitor is outside the scope of this article. See (Cesta & Rasconi 2003) for a deeper insight.

## 3 A User-Oriented Modeling Framework

One open problem which was left to be solved was that of making the technology involved in the project easily manageable by users who generally might not have any Planning & Scheduling specific knowledge, and at the same time, guaranteeing a high degree of system customization and reconfigurability.

The end-user must be capable of specifying the "real-world" problems which define the assisted person's desired behaviour according to the terminology which best matches the domain of application; on the other hand, the scheduling expert must gather the necessary domain knowledge for casting the end-user's problem into a well formed problem specification suited for the particular solving technology.

While attempts have been made to creating a common terminology for problem specification which relied on technology-specific ontologies aimed at making problem specification more comprehensible for the end-user, we deem such efforts not yet sufficient for our purposes. In fact, the introduction of such single-layer ontologies do not succeed in fully re-casting the scheduling-specific technicalities in domain-specific terms, which is essential in our case.

For this reason, in order to facilitate to the caregiver the task of using our scheduling technology, the modeling framework we have designed consists in two ontological levels: The first layer consists in a technology-specific ontology which provides a low-level interface for problem specification. As well as defining the basic elements of scheduling problems, this low-level ontology must provide a scheduling problem definition formalism $\mathcal{L}$ whose expressiveness allows to specify all and only the scheduling problems belonging to the low-level ontology.

Clearly, the larger the problem category described by the ontology is, the more versatile the specification formalism will have to be (see, e.g., the ProGen/max formalism for specifying RCPSP/max problems (Brucker *et al.* 1998)). But a versatile problem specification formalism comes a cost. It is often the fact that building a scheduling problem through the low-level, solver-specific formalism can be a demanding task even for a scheduling expert. Moreover, the more elaborate modeling needs encountered when casting a specific real-world problem by means of this formalism would turn out to be not only tedious but also definitely out of reach for someone not proficient in scheduling.

For this reason, we build a second layer in our modeling framework, aimed at providing an interface between the low-level ontology and the domain-specific concepts which compose the end-user's problem. This layer should provide the technologist with a formalism to describe domain-specific "building blocks" (later referred to as *constructs*) for the definition of scheduling problems within the particular applicative context. These building blocks are agreed upon during the knowledge acquisition phase, and their aim is twofold: first, they restrict the expressiveness of the end-user's problem definition formalism $\mathcal{L}$ to a subset of the scheduling problems expressible in the low-level ontology; second, they represent a collection of domain-specific terms which are close to the end-user's usual way of thinking. Thus, we refer to a collection of these building blocks as a *domain*, since they encapsulate elements of the end-user's domain knowledge, and describe how these elements of knowledge are transformed into elements of a scheduling problem. This can be stated in more precise terms: a

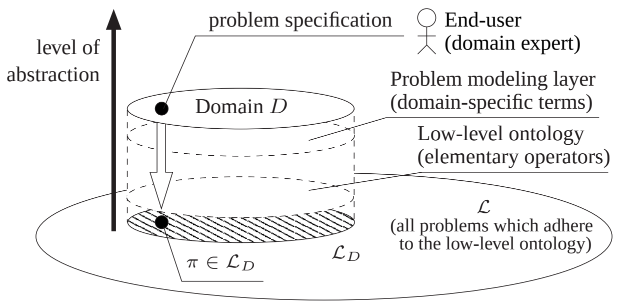

Figure 2: A scheduling domain definition corresponds to the specification of a language $\mathcal{L}_\mathcal{D}$ which restricts problem definition to the subset of all scheduling problems which are expressible within the application domain.

scheduling domain $D$ for a problem definition language $\mathcal{L}$ is a grammar which defines the language $\mathcal{L}_D \subseteq \mathcal{L}$ containing all scheduling problems which are expressible within the specific applicative context modeled in $D$.

As shown in figure 2, our two-layer modeling framework allows to cast RDE-specific terms (such as meals, medications and so on), into elements of the underlying scheduling problem representation of the caregivers' prescriptions.

The remainder of this section describes the two layers of the modeling framework: we first describe the scheduling ontology on which our current technology is grounded; we then describe the second layer by providing the RDE domain definition which allows caregivers to specify the behavioural requirements of the assisted person, while disregarding how these requirements are cast in terms of the low-level ontology.

## 3.1 The Low-level Scheduling Ontology

Providing a low-level interface for problem specification requires the identification of the elements which define a scheduling problem. We shall informally define a scheduling problem as an element $\pi$ of a scheduling problem definition language $\mathcal{L}$, whose symbols are contained in the following mutually disjoint sets: a set $\mathcal{T}$ of *task* symbols; a set $\mathcal{P}$ of *precedence constraint* symbols, and a set $\mathcal{R}$ of *resource* symbols.

Each task $\mathtt{t}$ is characterized by two events (or *time points*), namely its start time $[st(\mathtt{t})]$ and end time $[et(\mathtt{t})]$. All the time points belonging to the existing tasks (constituting the *temporal network*) can be bound to one another by precedence constraints, which in fact impose a restriction in the mutual distance among the time points (and therefore, among the tasks). In general, every task needs some resources in order to be executed; each resource $\mathtt{r}$ is characterized by a maximum capacity $\mathcal{C}(\mathtt{r})$ and each task may require $\mathcal{U}(\mathtt{t}, \mathtt{r})$ instances of one or more resources $\mathtt{r}$.

We can identify the basic procedural elements for defining a scheduling problem which adheres to the previous definition as shown in figure 3, where the semantics of four elementary operators are detailed by means of the notation introduced in the scheduling problem description. The `create_task()` operator allows for the specification of tasks with variable durations by creating lower and upper bounds between the start and end time-points of the task which is being created; the `create_res()` operator allows to introduce a resource by specifying its maximum capacity; the `set_res_usage()` operator allows to associate a task to a resource, by specifying the resource quantity required by that task; finally, the `create_pc()` operator allows to constrain any two time points belonging to any two tasks, by a specified distance.

| *Operator* | *Semantics* |
|---|---|
| `create_task(t,min,max)` | $\mathcal{T} = \mathcal{T} \cup \{\mathtt{t}\}$<br>$\mathcal{P} = \mathcal{P} \cup$<br>$\{\{st(\mathtt{t}) \xrightarrow{\mathtt{min}} et(\mathtt{t})\},$<br>$\{et(\mathtt{t}) \xrightarrow{-\mathtt{max}} st(\mathtt{t})\}\}$ |
| `create_res(r,cap)` | $\mathcal{R} = \mathcal{R} \cup \{\mathtt{r}\}$<br>$\mathcal{C}(\mathtt{r}) = \mathtt{cap}$ |
| `set_res_usage(t,r,use)` | $\mathcal{U}(\mathtt{t}, \mathtt{r}) = \mathtt{use}$ |
| `create_pc(t1,p1,`<br>`t2,p2,x)` | $\mathcal{P} = \mathcal{P} \cup$<br>$\{\mathtt{p1}(\mathtt{t1}) \xrightarrow{\mathtt{x}} \mathtt{p2}(\mathtt{t2})\}$<br>where<br>$\mathtt{p1}, \mathtt{p2} ::= \mathtt{st} \vert \mathtt{et}$ |

Figure 3: The four elementary operators for building scheduling problem instances.

We will use these operators as atomic elements for defining the semantics of the domain-specific terminology for the end-user. In addition to the operators, we also define three utility functions for specific data retrieval. The description of such functions is outside the scope of this article; for further details, see (Cesta *et al.* 2005).

## 3.2 The Domain Modeling Layer

Defining a scheduling domain thus equates to specifying the language of all scheduling problems which adhere to a particular structure. The building blocks which are used to define this structure are called *constructs*. A construct has two main characteristics: (1) it represents a domain-specific term (i.e., an element of the end-user's terminology), and (2) it defines how this term is mapped into the underlying scheduling problem specification formalism given by the four operators of figure 3. As a consequence, a domain is defined as a collection of constructs.

A domain specification begins with the optional definition of domain-specific constants. In addition to the user-specified constants, there are two predefined constants, `source` and `sink`, which represent, respectively, the *origin* and *horizon* time-points. Notice also that the first and third arguments of the `create_pc()` operator represent task IDs. These arguments have no meaning in the event that the relevant time point is `source` or `sink`, and are thus ignored.

After defining the constants, the domain designer proceeds with the definition of constructs. Each construct has a list of parameters and a sequence of operators which define how the construct is translated in terms of the scheduling problem specification.

Once the problem definition syntax is defined (see (Cesta *et al.* 2005) for further details), the end-user can easily use the syntax to build a problem by (a) indicating a reference domain and (b) properly instantiating the constructs specified in the domain definition.

In the following section we show an example which puts this top layer to work in the context of a typical ROBOCARE scheduling application.

## 4 Building and Monitoring a Plan for ROBOCARE

In this section we give an example of application of our scheduling tools in the context of the ROBOCARE Domestic Environment (RDE). While in most cases of real-world deployment the scheduling tool is used to solve a scheduling problem (eventually optimizing with respect to some criteria on the duration of the tasks or on resource usage), in the RDE context, the high performance optimizing functionalities of O-OSCAR are absolutely secondary, as we employ our technology primarily for execution monitoring. The need for an advanced tool such as a scheduler for this task can be appreciated in the fact that often the constraints consist of complex relationships in time between the daily tasks of the assisted person. Also, given the high degree of uncertainty in the exact timing of task execution (a person never has lunch at the same time every day, etc.), it is necessary to model flexible constraints among the tasks, while admitting the possibility of hard deadlines or fixed time-points. Overall, the aim is not to control task execution, nor to impose rigid routines, rather it is to monitor the extent to which the assisted person adheres to a predefined routine, defined together with a physician or family member.

We make the assumption that the knowledge-sharing phase between the scheduling expert and the end-user has yielded a complete description of the ROBOCARE domain in the form of the following set of domain constructs:

```
(:construct assisted_person
  :parameters (name)
  :operators (create_res(name,1)))
```

which defines a binary resource corresponding to the assisted person. Every task will require the assisted person as a necessary resource for execution;

```
(:construct breakfast
  :parameters (person start end)
  :operators (
    create_task(bfast_task,(end-start),
      (end-start))
    set_res_usage(bfast_task,person,1)
    create_pc(null,source,bfast_task,st,
      start)
    create_pc(bfast_task,et,null,sink,0)))
```

which defines the breakfast-type activity, characterized (see the `:parameter` list) by a *start* and *end* time (i.e., a duration);

```
(:construct lunch
  :parameters (person start end min_bfast
    max_bfast)
  :operators (
    create_task( lunch_task,(end-start),
      (end-start))
    set_res_usage( lunch_task,person,1)
    create_pc(null,source, lunch_task,st,
      start)
    create_pc( lunch_task,et,null,sink,0)
    create_pc(bfast_task,et, lunch_task,
      st,min_bfast)
    create_pc( lunch_task,st,bfast_task,
      et,-max_bfast)))
```

which defines the lunch-type activity, characterized by a *start* and *end* time, as well as a minimum and a maximum distance which binds the start time of the lunch with the end time of the breakfast (in this example we will also use the construct `dinner`, having the same characteristics of the construct `lunch`);

```
(:construct medication
  :parameters (person product dur min_time
    max_time)
  :operators (
    create_task(product,dur,dur)
    set_res_usage(product,person,1)
    create_pc(null,source,product,st,0)
    create_pc(product,et,null,sink,0)
    create_pc(null,source,product,st,
      min_time)
    create_pc(product,st,null,source,
      -max_time)))
```

which defines the medication-type activity, characterized by a minimum and a maximum start time (respectively, *min_time* and *max_time*), and by a duration;

```
(:construct meal_bound_medication
  :parameters (person product dur meal
    min max)
  :operators (
    create_task(product,dur,dur)
    set_res_usage(product,person,1)
    create_pc(product,et,null,sink,0)
    create_pc(meal,et,product,st,min)
    create_pc(product,st,meal,et,-max)))
```

which defines a medication-type activity which is temporally bound to some other task, characterized by the minimum/maximum distance (*min*, *max*) between its start time and the end time of the task to which it is bound, and by a duration;

Now that the constructs which define the ROBOCARE domain are known, let us suppose that the caregiver has the task of synthesizing a specific behaviour for a patient, consisting of a number of activities (corresponding to *breakfast*, *lunch*, *dinner*, as well as the taking of three different medicines) that the assisted person should perform and whose execution should be monitored; due to medical requirements, let us also suppose that such activities must satisfy the following temporal requirements:

1. *breakfast*: can not begin before 8:00 hours; its nominal duration should be 30 minutes;
2. *lunch*: can not begin before 13:00 hours; can not begin before at least 4 hours have passed from the end of *breakfast*; can not begin later than 6 hours after the end of *breakfast*; its nominal duration should be 1 hour;
3. *dinner*: can not begin before 19:30 hours; can not begin before at least 5 hours have passed from the end of *lunch*; can not begin later than 6 hours after the end of *lunch*; its nominal duration should be 2 hours;
4. *taking herbs*: can not begin before 12:00 hours; can not begin after 20:00 hours; its nominal duration should be 10 minutes;
5. *taking laxative*: can not begin before 17:00 hours; can not begin after 21:00 hours; its nominal duration should be 5 minutes;
6. *taking aspirin*: can not begin before the end of *dinner*; can not begin later than 20 minutes after the end of *dinner*; its nominal duration should be 5 minutes;

It is easy to see that the previous nominal behaviour can be perfectly captured by the set of constructs presented above; according to what stated at the end of section 3.2, a problem specification which fully satisfies all the requirements can be therefore synthesized as follows:

```
(define (problem test_prob)
   (:domain RDE)
   (:specification
      (assisted_person jane)
      (breakfast jane 480 510)
      (lunch jane 780 840 240 360)
      (dinner jane 1170 1290 300 360)
      (medication jane herbs 10 720 1200)
      (medication jane laxative 5 1020
         1260)
      (meal_bound_medication jane aspirin 5
         dinner 0 20)))
```

The above list of instantiated constructs in the `:specification` section fully defines our desired behaviour: the first construct instantiation `(assisted_person jane)`, defines the patient *Jane*; the other instantiations define all the tasks whose execution should be monitored: for example, `(lunch jane 780 840 240 360)` prescribes that Jane should have lunch from 13:00 hours (all values in the parameter list are expressed in *minutes*), that the lunch should last $840 - 780 = 60$ minutes, and that it should start not before 240 minutes and not after 360 minutes that breakfast has ended; the instantiation `(meal_bound_medication jane aspirin 5 dinner 0 20)` prescribes that Jane should take aspirin, that the action should last 5 minutes and that it should start between 0 and 20 minutes after the end of dinner.

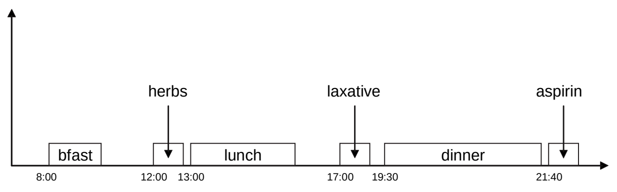

Figure 4: A possible solution for the given problem.

One possible solution to the problem defined above is presented in figure 4. The schedule represents a solution because, as it can be easily confirmed by inspection, all the temporal constraints in the original problem specification are respected. The above schedule represents the behaviour the patient should adhere to, and contains all the information regarding the temporal relationships among the activities whose consistency should be constantly checked during execution.

Once the monitoring starts, the sensors are periodically queried and the nominal schedule is adjusted in accordance with the patient's detected behaviour. At each detection cycle, the execution status of each activity is checked: among the possible events, some activities may be reported as under execution *before* their nominal start time, the execution of other activities may be reported as delayed, the duration of some activities may exceed the nominal value, and so on; each deviation from the nominal schedule may entail a conflict which has to be reacted upon.

As an example (see figure 4), let us suppose that the patient, after having dinner, sits on the sofa and starts watching TV: at each new sensor detection cycle, the system assesses the situation and delays the *aspirin-taking* activity. But since, according to the medical requirements, the aspirin should be taken no later than twenty minutes after dinner, the attempt to delay the aspirin activity beyond such limit is recognized by the execution monitor as a conflict and therefore is not accepted. The system responds to such situation by triggering a warning to the patient as a reminder for the forgotten action.

## 5 Conclusions

In this article we have described our approach in tackling the problem of how to monitor the execution of activities through the development of an intelligent supervision system. We believe that this project represents an original way of deployment of state-of-the-art Planning & Scheduling technology.

Great attention has been paid to maintaining very low the required level of environment engineering, as well as the level of system invasiveness, both for privacy-related issues and in order to leave as much freedom of action as possible to the assisted person, within the limits imposed by health-preserving necessities and/or personal safety issues.

Future research work will be dedicated to enhancing the degree of interaction between the supervision system and the assisted person, from an initial interaction of purely sensor-based nature (thus providing a merely *cognitive* help), towards the introduction of active robotic components in the environment, aimed at offering a more "usefully intrusive" assistance.

## 6 Acknowledgements

This research is partially supported by MIUR under project ROBOCARE. The Authors thank the other members of the PST, particularly Marcelo Oglietti and Simone Fratini for having fostered interesting common research, as well as the people at the Dept. of Computer and Systems Science (DIS) of the University of Rome "La Sapienza" for their work on the sensor-provided ROBOCARE environment.

## References

Bahadori, S.; Cesta, A.; Iocchi, L.; Leone, G.; Nardi, D.; Pecora, F.; Rasconi, R.; and Scozzafava, L. 2004. Towards Ambient Intelligence for the Domestic Care of the Elderly. In Remagnino, P.; Foresti, G.; and Ellis, T., eds., *Ambient Intelligence: A Novel Paradigm*. Springer. To appear.

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

Cesta, A., and Rasconi, R. 2003. Execution Monitoring and Schedule Revision for O-OSCAR: a Preliminary Report. In *Proceedings of the Workshop on Online Constraint Solving at CP-03, Kinsale Co. Cork*.

Cesta, A.; Cortellessa, G.; Pecora, F.; and Rasconi, R. 2005. Mediating the Knowledge of End-users and Technologists: a Problem in the Deployment of Scheduling Technology. In *Proceedings of the 23rd IASTED International Multi-Conference on Artificial Intelligence and Applications, Innsbruck, Austria*.

McCarthy, C., and Pollack, M. 2002. A Plan-Based Personalized Cognitive Orthotic. In *Proceedings of the 6th International Conference on AI Planning and Scheduling*.

Policella, N.; Oddi, A.; Smith, S.; and Cesta, A. 2004. Generating Robust Partial Order Schedules. In *Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming (CP-04)*. To appear.

Pollack, M.; McCarthy, C.; Ramakrishnan, S.; Tsamardinos, I.; Brown, L.; Carrion, S.; Colbry, D.; Orosz, C.; and Peintner, B. 2002. Autominder: A Planning, Monitoring, and Reminding Assistive Agent. In *Proceedings of the 7th International Conference on Intelligent Autonomous Systems*.