ELSEVIER

Information and Software Technology 39 (1997) 185–193

INFORMATION AND SOFTWARE TECHNOLOGY

Rule handling in the day-to-day resource management problem: an object-oriented approach

K.X. Thrampoulidis*, C. Goumopoulos, E. Housos

Department of Electrical & Computer Engineering, Computer Laboratory, University of Patras, GR-265 00 Rio Patras, Greece

Received 8 October 1995; revised 21 June 1996

Abstract

The day-to-day resource management problem is caused by a set of unexpected events which disturb the planned daily activities and thus change the long-term optimal schedule. The solution to this problem presupposes that all the regulations for the handling of resources in the particular application domain have been satisfied. In this paper, a new approach to handling the rules in the resource management problem is presented. An object-oriented application specific language that allows the flexible expression of the rules, as well as the corresponding rule handling subsystem are presented. The design of the whole system is based on a generic meta-model derived from the object-oriented paradigm. This makes the system applicable to a wide range of problem domains such as repairs management, airline and other transportation scheduling, school scheduling, etc. The system has been developed and tested as a subsystem of the DAYSY system, a day-to-day resource management system for the airline domain.

Keywords: Rule handling; Legality checking; Object-oriented application specific language.

1. Introduction

The confrontation of problems which occur in the daily application of a resource schedule is a difficult and tedious process. Due mainly to unexpected events, like resource weakness to support it, the realization of the planned timetable is impossible. This leads to the necessity to change in real-time the planned schedules of the resources. The calculation of the optimal changes in such cases involves the solving of difficult combinatorial problems [1–3] without violating the set of enterprise regulations the solution must conform to. In the context of the DAYSY/ESPRIT (Day-to-Day Resource Management Systems) project, we faced the problem of day-to-day resource management in the airline problem domain, with Lufthansa German Airlines as the pilot user. However, the development of a generic system, capable of being applied to a wide range of problem domains, was one of our main targets.

The daily rescheduling problem in most airlines is still a manual process without the use of a computer system to optimize the solution with respect to both quality and cost. However, the consolidation of airline companies and their consequent increase in size has made it mandatory for such a computer system to be used in the daily rescheduling of their resources. Such a system is a valuable tool for the persons (planners in the airline domain), who are responsible for the realization of the timetable. Special purpose techniques, different from those used in long-term planning must be used, as the time available for the computation process is very limited.

One of the main required functions of a day-to-day resource management system is the testing of the legality of the produced solution in compliance with the complete set of the enterprise regulations. All airlines must conform to a set of *International Government Regulations* (*IGR's*), which are intended to minimize crew fatigue and ensure passenger safety. In addition to the IGR's, most major airlines must also conform to a complex set of regulations imposed by employee unions. These regulations, called rules for simplicity, vary according to crew type (pilot or flight attendant), crew size, aircraft type, time zone distance from base, etc. They include, among others, work rules concerning maximum duty periods, maximum flying time and maximum number of flights permitted during regular and irregular operations. All these regulations are continuously changing, so there is a need for a user-friendly system to express and manage them.

Most of the existing scheduling and rescheduling systems test the legality of the produced solution using a few external parameters and embedding the rules within the

* Corresponding author. e-mail: {thrambo,goumop,Housos}@ee.upatras. gr

PII 0950-5849(95)0000-0

application software. Other systems, such as Volvo Data CARMEN [4] (Lufthansa, Alitalia, SAS, KLM) and SBS RuleTalk [5] (United, British Airways, Delta) are using a special purpose language for the expression and subsequent management of rules. In this case, using a language as an interface, the user is able to change not only the data but also the structure of the rules. In this way it is possible to investigate future scenarios and rule extensions without changing the application programs. Finally, a third category includes systems such as CESAR of Air France and that of Cathay Pacific Airways [6] which are using artificial intelligence techniques to analyse, interpret and finally represent, by the use of rules, both the expert's knowledge of the scheduling process as well as the enterprise regulations. All the above systems do not deal with the representation of the rules in the general case as imposed by different application domains. They also require skilled programmers so as to be used effectively.

In this paper, we focus on the subsystem responsible for the legality of the produced solution according to the set of enterprise regulations. These regulations were classified for the purpose of the present work into three main categories: *activity composition* rules, *property calculation* rules and *property constraint* rules. This categorization proved to be very useful in the confrontation of this kind of problem. To manage the rules of the above categories, a rule language was defined and the underlying subsystem, which checks the legality of the rules, was developed. The system contains reusable components in such a way that although its primary application is in airlines, it can easily be adopted in other problem domains. An extension [7,8] of the OMT methodology [9,10] was used for the design and development of the system. Use cases and object interaction diagrams are the main enhancements of this OMT extension.

In the next section, the day-to-day resource rescheduling problem in the airline problem domain is presented. The third section briefly presents the architecture of the DAYSY system. The fourth section introduces the Legality Checking Subsystem; discusses the design characteristics of the system, its ability to be used in different problem domains and the object-oriented meta-model on which the design of the whole system was based. The fifth section illustrates, by use of an example, our high level object-oriented rule handling DAYSY language. Finally, the last section concludes the work.

2. Problem description

Planning airline operations is a highly complex process that uses costly resources — aircraft and staff — under difficult technical and legal constraints, to produce an airline schedule. The smooth execution of this schedule is usually disrupted by unexpected events of several types. Additional flights, flight cancellations (bad weather, low booking), changes in aircraft type, changes in the time limits of flight duties, personal disruptions (sickness, qualifications not renewed), etc. can occur at any time. They create the need for a fast and effective rescheduling of resources in order to have optimal service for the flying public, without violating the contractual and safety rules of the crews. The task of repairing the schedules in real-time, known as day-to-day resource management, includes the following:

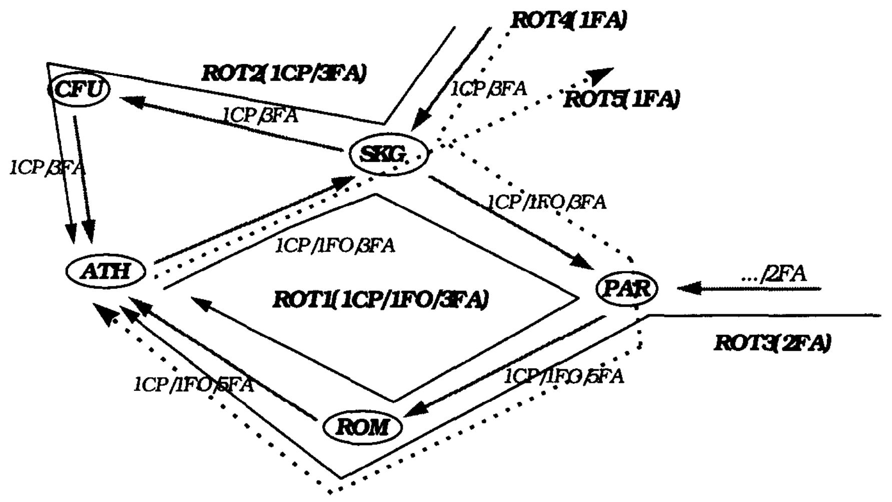
- recognition of the event;
- identification of all the affected components of the schedule;
- generation of one or a set of solutions;
- selection of the best solution, that can be realized.

In the field of crew scheduling, a rotation is a round trip during which a number of legs/flights are covered. The idea is to start from a station with a set of crews, cover a number of flight legs during a few days and finish the trip at the same station. This station is called home base for the specific crew. Fig. 1 gives graphically a portion of a schedule of an airline. The flights shown are: ATH-SKG, SKG-PAR, PAR-ROM, ROM-ATH, SKG-CFU and CFU-ATH. Each flight is supported by a specific crew composition (crew complement). So, the flight ATH-SKG requires a crew composition of 1CP/1FO/3FA, i.e. 1 captain, 1 flight officer and 3 flight attendants. The planned rotations ROT1, ROT2 and ROT3 with their crew complements are given with solid lines. For example, ROT1 with crew complement 1CP/1FO/3FA consists of the following legs: ATH-SKG, SKG-PAR, PAR-ROM, and ROM-ATH. Notice that the crew requirements of a flight can be satisfied with a combination of two or more rotations.

The schedule is performed normally unless an unexpected event appears. Such an event is the following: The FA 'Petrou' gets sick during the flight ATH-SKG. The planner informs the system about this event and he/she issues the actions that will ensure the normal operation of the airline schedule. A possible solution for the above problem includes the following actions:

(a) Modify the crew complement 1CP/1FO/3FA or ROT1 to 1CP/1FO/2FA. This is due to Petrou's absence.
(b) Modify the crew complement 1CP/3FA of ROT2 to 1CP/2FA. The FA 'X' is taken from ROT2 to cover Petrou's absence. It is assumed that the flight that brings 'X' arrives at SKG airport at about the same time as the flight that brings Petrou. A choice is made to satisfy the needs of an international flight with priority over the corresponding domestic flight.
(c) Create a new rotation, ROT4 with crew complement 1FA. This rotation is assigned to the dispatched FA 'X'.
(d) Create a new rotation, ROT5 with crew complement 1FA. This rotation is assigned to Petrou.
(e) Update the monthly schedule of FA Petrou.
(f) Update the monthly schedule of the FA 'X'.

This solution is valid only if there are no rule violations for the monthly schedules and the rotations that are affected.



ATH: Athens, SKG: Thessaloniki, CFU: Corfu, PAR: Paris, ROM: Rome

BASE ATH
Rotation 1: ATH - SKG - PAR - ROM - ATH
Rotation 2: ... - SKG - CFU - ATH
Rotation 3: ... - PAR - ROM - ATH
Rotation 4: ... - SKG - PAR - ROM - ATH
Rotation 5: ATH - SKG - ...

$\longrightarrow$ *: flight segment*
$\longrightarrow$ *: planned rotation*
$\cdots\rightarrow$ *: new rotation*

Fig. 1. A portion of the daily schedule of an airline.

Day-to-day resource management is not limited to airlines. It is also applicable to other companies involved in air transportation: catering companies, airports, fuel providers. Transportation networks have very similar problems, and competition is forcing them to react in real-time to market demands. Many other areas, such as the daily scheduling of technical crews for radio and television, live haul transportation in the food industry, etc. need efficient resource management.

3. The DAYSY system

In the context of the DAYSY/ESPRIT project we faced the problem of day-to-day resource management, in the airline problem domain with Lufthansa German Airlines as the pilot user. The DAYSY system, that was developed, is a toolbox of reusable software components that makes it possible to develop and deploy day-to-day resource management systems at a fraction of current cost. In Fig. 2 the basic architecture of the DAYSY system is given.

According to DAYSY, for the production of a solution, the Daysy-Planner uses the constraint logic programming based Automatic Rescheduling [11,12] module (AR), the Legality Checking subsystem (LC) and a set of additional tools (planning tools), all of which have access to the company's Oracle database. The Daysy-Planner, having the responsibility for reprogramming, guides the entire solution process by choosing the strategy and defining a set of priorities. The LC module checks the legality of the crew

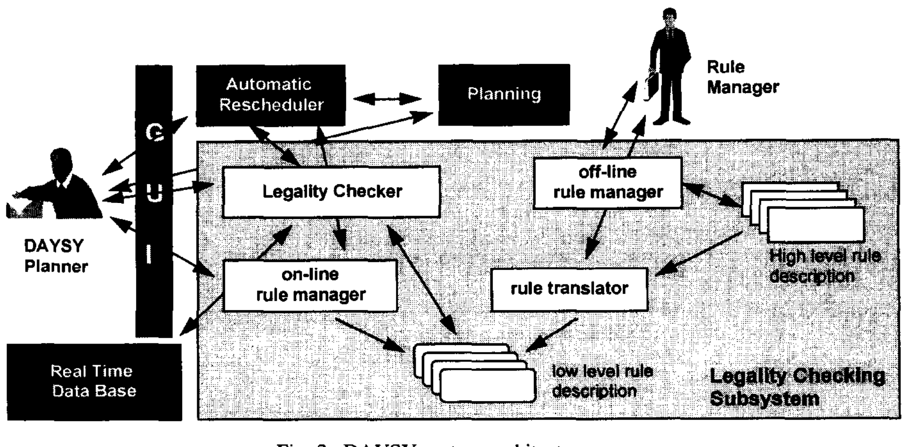


Fig. 2. DAYSY system architecture.

assignments, as well as new or modified trips according to the set of rules.

The Rule Manager, using the off-line rule manager subsystem, creates new rules and updates the existing ones using a high-level special purpose language. This high-level rule description is then translated into a low-level one, which was designed to achieve the best computational performance for the legality checking mechanism, using state of the art practices, e.g. object-oriented approach, constraint programming etc.

The basic concepts involved in the rescheduling process are the concepts of activity and available resources. An activity requires a specific set of resources in order to be in the ready state and thus able to be executed. Fig. 3 shows part of the object model of the DAYSY system, containing the basic objects of the airline crew-rescheduling problem. The basic activities in the airline area are: leg, duty, trip, rest, education, training, vacation, etc. A crew member can be associated with many activities with link attributes position and requested_by. He/she probably has constraints to specific airports and is qualified for a specific airplane with link attributes position, start_date and end_date.

A subset of the rules captured in the object model of Fig. 3 do not change in time and are embedded in the enterprise Database Management System, mainly in the form of stored procedures and triggers. Instead, rules concerning activity composition, property calculation and property constraint, change over time and must be decoupled from the initial application build stage. These rules include among others rest rules, duty and transit rules, departure/landing rules and cost rules [13]. They are provided to the application subsequently via a high-level special purpose language in a way that greatly increases the company's reaction to changes in enterprise regulations.

The LC subsystem is based on the above-mentioned object model. Consequently, in the low-level rule description of Fig. 2 one can find rules concerning constraints on properties of the crew member class of Fig. 3. Also, through the on-line rule manager module the DAYSY-Planner is able to on-line change attributes of briefing and debriefing classes presented in the object model of Fig. 3.

4. Legality Checking Subsystem

Every solution produced either manually or automatically must satisfy a set of rules. These rules can be roughly divided into static and dynamic. Static rules refer to time and space conservation, and are implemented within the Automatic Rescheduler [14] subsystem implemented with CHIP (Constraint Handling In Prolog). For example, according to static rules, no crew can arrive at one place and depart from another or arrive after the departure of the next flight. Dynamic rules change over time and are encapsulated within the Legality Checking Subsystem. The principal reason for encapsulating the rules within the LC subsystem was to protect against extensive modifications of the complete system whenever a change in the

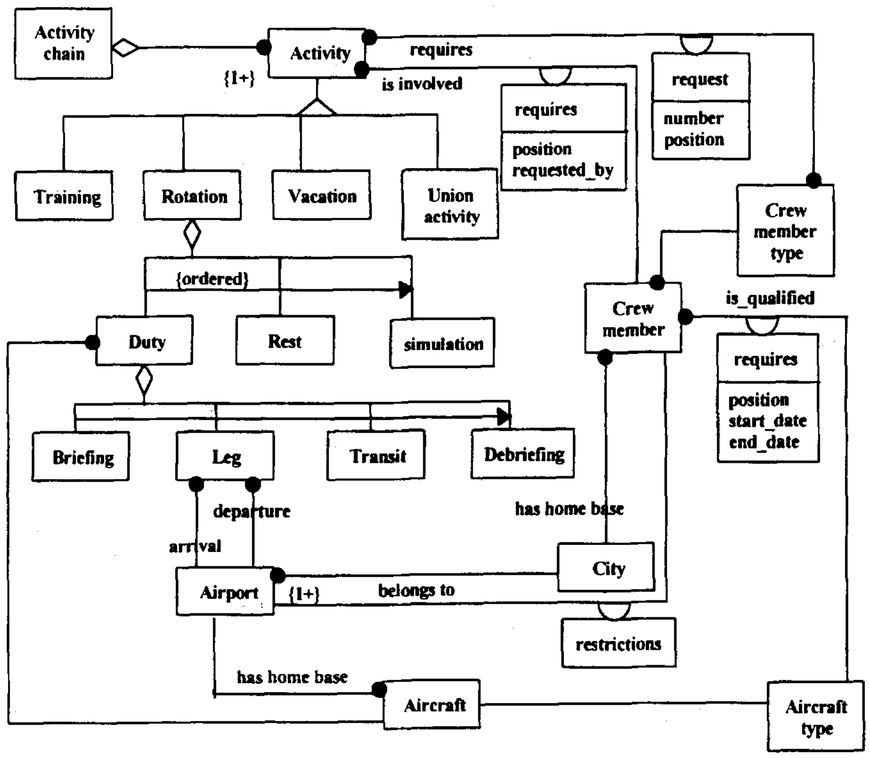


Fig. 3. Part of the DAYSY system Object model.

rules happens. The LC subsystem must handle any rule governing the scheduling process without the involvement of the other components of the DAYSY system. Contract changes must be incorporated into the scheduling process by the daysy users without risking system integrity. The LC subsystem must also be flexible and user friendly, as it is used by airline planners without special skills in computer languages. These requirements were satisfied through the creation of an object-oriented language for the definition of the rules, the allowance for the on-line management of these rules and the development of a fast and efficient legality-checking mechanism.

The *Rule-Manager*, using the *off-line rule management* subsystem, describes the rules and creates rulesets. A ruleset is composed of a set of logically cohesive rules for a particular application and exists both at a high-level representation (DAYSY language) and a low-level one, that is used by the LC subsystem (Fig. 2). The translation from high level into low level is done by the rule translator, designed and implemented using the Lex & Yacc Unix tools [15,16]. The *rule translator* implements the front end of the DAYSY language and produces an intermediate C++ code from the original DAYSY language source. Then, the back end of the compilation process is assigned to the corresponding C++ compiler of the target machine. This scheme enables portability of the rule translator over different platforms. The low-level representation was defined in terms of C++, through the facilities that come with it (e.g. class libraries for manipulating complex data structures), in such a way as to achieve the required performance and to support late binding in the ruleset construction process.

The *on-line rule manager* allows for the on-line management of a significant part of the knowledge represented by the rules. This gives the DAYSY planner the ability to test several different scenarios and find the most favourable solution to each specific planning problem.

In an effort to improve the flexibility of the DAYSY system and allow it to cover other scheduling areas as well (e.g. hospital staff scheduling, school-timetabling, other transportation scheduling, etc.), the meta-model of Fig. 4 was created, which has the concepts of *activity* and *rule* as its basic building elements. ''*Primitive activities* in the scheduling process are the set of basic non-overlapping and indivisible activities which are typically performed in order to produce the deliverables'' [17]. Primitive activities are characterized by *primitive properties* such as: start time, end time, activity type. They provide the basis for the construction of other, non-primitive activities, that we call *composite activities*. A composite activity, from an external viewpoint, is conceptually composed of other activities. These 'other activities' are referred to as component activities. For example, the trip in the airlines problem domain, is a composite activity that consists of shifts, rests and optional training and simulation. Every shift in its turn is a composite activity composed of flight activities (legs). For each composite activity there is *an activity composition rule*. Activity

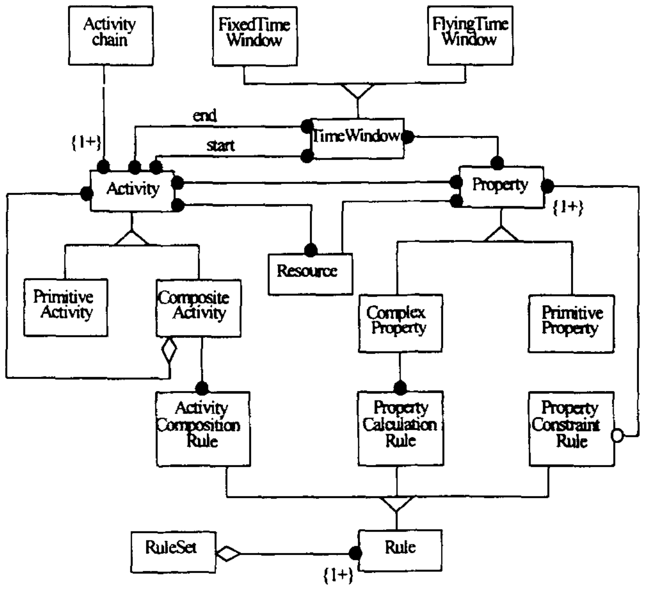


Fig. 4. Object model of LC subsystem.

composition rules are used in the recognition phase of the various composite activities.

Each activity may be characterized by a set of *complex (computed) properties*. For each complex property there must be a *property calculation rule* to evaluate it. Each property calculation rule is expressed by use of the basic constructs of the DAYSY language, e.g. operators, data types, functions and statements. Finally, *property constraint rules* are the heart of the ruleset and specify restrictions or requirements concerning instances of a property type.

The creation of the object model for the specific application and recording of the activities and rules is based on problem domain knowledge and application domain knowledge (Fig. 5). In every different application domain, the user defines the primitive and composite activities either from scratch or as extension of the primitive ones. The EXPANDS operator of the DAYSY language, allows the activities of the application to inherit similar characteristics

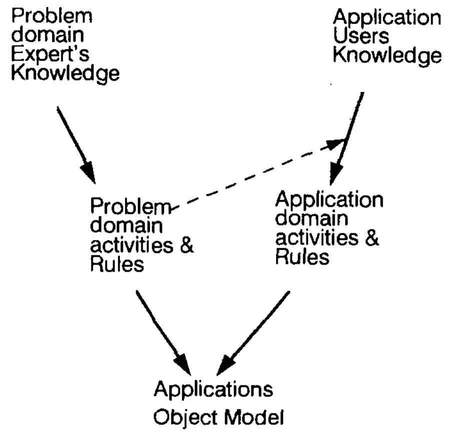

Problem domain Expert's Knowledge
Application Users Knowledge
Problem-domain activities & Rules
Application domain activities & Rules
Applications Object Model

Fig. 5. Generation process of the application's object model.

from the basic predefined activities of the problem domain. It also simplifies the integration of the LC to existing scheduling systems, and provides a uniform access to the characteristics of the activities. For the Lufthansa case, adaptation was made to existing activities in an Oracle server. Meanwhile new activities like rest, transit and days-off were created.

The simpler search method for a trip-building system, like the AR of the DAYSY system, is the generate and test approach, according to which a complete solution is generated and then its legality is tested. However, the most efficient search is realized by the test and generate approach. In order for the LC subsystem to support this approach, *incremental checks* are performed during the trip-building process. A *caching* mechanism is used in such a way that the later incremental expansion of the partial solution does not need to fire the calculations from scratch, but rather pre-computed results can be used.

To support *chronological backtracking* [18], the most widely known search method, the LC subsystem resets its activity aggregation hierarchy in memory and checks the constraints only on the affected activities. In addition, a number of useful values denoting properties of the activities are available for the searching procedure (AR). This kind of information is used by the Automatic Rescheduler implementing the search method to reach the final solution quickly while removing the search tree branches that turn out to be invalid.

```
//DECLARATION SECTION
INCLUDE      file-name
:
INCLUDE      file-name
CONST
  const-name = scalar-value,
  :
  const-name = scalar-value;
VAR
  variable-name : type,
  :
  variable-name : type;

ACTIVITY activity-name [EXPANDS activity-name]
  PROPERTIES:
    property-name : type [TEMP],
             :
    property-name : type [TEMP];
  COMPONENTS:
    activity-name,
    :
    activity-name;
  RELATED KEYWORDS
    keyword-name,
    :
    keyword-name;
  RULES:
    rule-name,
             :
    rule-name;
END

TIMEWINDOW timewindow-name EXPANDS CalWin
  PROPERTIES:
    property-name : type,
    :
    property-name : type;
  RULES:
    rule-name;


END
```

```
//INITIALIZATION SECTION
INITIALIZE
  [(activity-name timewindow-name)::]variable-name =
  scalar value;
  :
  [(activity-name timewindow-name)::]variable-name =
  scalar-value;
END

//DEFINITION SECTION
PROPERTY property-name OF (activity-name timewindow-
  name)
  VAR: /* list of variables used for the property calculation
       */
       variable-name,
       :
       variable-name;
  CONST:
       const-name = scalar-value,
       const-name = scalar-value,
       :
       const-name = scalar-value;
  RULE:
       expression;
       :
       expression;
END

RULE rule-name OF (activity-name timewindow-name)

       [STATUS: ON/OFF]
       [GROUP: group-name]
       [PRIORITY: priority-number]
       [CONDITION: boolean-expression]
  [CONST:
       const-name = scalar-value,
       const-name = scalar-value,
       :
       const-name = scalar-value;]
  BODY:
       expression;
END
```

Fig. 6. DAYSY language file outline.

5. The DAYSY language

The DAYSY language is an object-oriented special purpose language for the expression and handling of rules inherent in scheduling problems. It was designed and implemented during the execution of the DAYSY/ESPRIT project [19,20]. In this section, the basic elements of the language are presented using simple examples. A DAYSY program is organized as is shown in Fig. 6. It is composed of three major distinct sections: the **declaration section**, the **initialization section**, and the **definition section**.

(1) **Declaration section**. The declaration section of a DAYSY program consists of four different types of declarations:

(a) *The CONST Declaration*: The CONST declaration assigns a symbolic name to a scalar constant. These names act like constants in the sense that they do not appear on the left-hand-side of any assignment statement of the language, but they can be altered from the external environment. It is possible to attach new values through the on-line rule manager nodule while the legality system is running, without having to recompile the DAYSY source. This facilitates the running of different scenarios from which we can find the most favourable solution to each specific planning problem. For example, the following declaration declares the briefing and debriefing constants:

```
CONST
   briefing = 00:30,
   debriefing = 00:45;
```

(b) *The VAR Declaration*: Every variable that is used by the program but does not belong to a specific activity or time window must be declared within a VAR section. For example, the following VAR declaration declares two variables of type Boolean.

```
VAR
   activity_detected: BOOLEAN,
   penalty_defined: BOOLEAN;
```

(c) *The ACTIVITY Declaration*: The ACTIVITY keyword declares an activity type that is associated directly or indirectly with rules. For each activity, its properties and components types, in the case of a composite activity, are defined. Also, the rules and the related keywords for each activity are declared. A rule is associated with an activity if one of the following holds:

- the rule defines the computation of a property of the activity (for an *inference or computational rule*);
- the rule defines a constraint on the value of a property of the activity (for a *constraint rule*);
- the rule supports the process of recognizing an activity (for a *stimulus/response rule*);

For each activity property there is a regulation for the calculation of its value. Its expression is

```
//DECLARATION SECTION
:
:

ACTIVITY shift EXPANDS basic_shift
        PROPERTIES
          tz_diff: trel; /* ''Work start for a shift'' */
          TEMP
          max_transit_time    :    trel,
          duty_start          :    tabs,
          duty_end            :    tabs,
          duty_period         :    trel;
        COMPONENTS:
          leg, simulation, training;
        RELATED KEYWORDS:
          is_first_shift, is_last_shift, flight_carrier;
        RULES:
          shift_in_a_low, /* stimulus rule */
          duty_before_and_after_break,/*constraintrule*/
          timezone_diff,          /* constraint rule */
          maximum_flight_duty; /* constraint rule */
END

//DEFINITION SECTION
:
:
PROPERTY duty_start OF shift
        RULE:
          departure of first leg-briefing;
  END

PROPERTY duty_end OF shift
     RULE:
        arrival of last leg + debriefing;
END

PROPERTY duty_period OF shift
     RULE:
        duty_end — duty_start;
END

RULE maximum_flight_duty OF shift
     STATUS:        ON
     GROUP:         MTV
     PRIORITY:      1
     CONDITION:     season_period == SUMMER;
     CONST:
        max_duty_period = 14:00;
     BODY:
        duty_period <= max_duty_period;
END

//where
//trel : declares a relative time constant that is used to
//represent an elapsed amount of time between two activities
//or a time of a day, in hours and minutes and
//tabs : declares an absolute time constant that is used to
//represent the time and date of some event.
```

Fig. 7. Activity declaration and property definition examples.

given in the definition section. As an example of activity declaration, the activity shift is declared in Fig. 7.

(d) *The TIMEWINDOW Declaration*: The TIMEWINDOW keyword is used to declare every time window that is used by at least one rule. There are two types of time windows: the *Fixed Time Window* and the *Flying Time Window*. The DAYSY language has several built-in time windows, e.g. CalYear, CalMonth, CalDay, etc. This is a totally different and more user-friendly approach for the expression of rules that refer to calendar intervals in comparison with existing systems. For example, the language proposed by the CARMEN [4] system, evaluates properties and constraints within the activity aggregation hierarchy and not within a specific time interval. Thus the property flight-time in 28 consecutive days is considered to be an attribute of the rotation activity. A better approach is to consider it as an attribute of a 28ConsDays TimeWindow object (see example below), since it constitutes a natural property of such a window. In our system, the time window declaration encapsulates complex properties and their associated property calculation and property constraint rules.

TimeWindow Example

The following is an example of the implementation of a rule from *European Flight Regulations* [21]. The rule states: ''No operator shall schedule a crew member for flight duty and no crew member shall accept an assignment for a flight duty if his/her block time of the flights in which he/she was an operating crew member is more than 100 hours in any 28 consecutive days.''

```
TIMEWINDOW 28ConsDays EXPANDS CalDay
  INITIALIZE:
    twduration = 28;
  PROPERTIES:
    FlightTime;
  RULES:
    MaxFlightTimeRule;
END

PROPERTY FlightTime OF 28ConsDays
  RULE:
      SUM BlockTime OVER LEG;
END

RULE MaxFlightTimeRule OF 28ConsDays
  STATUS:     ON
  PRIORITY:  2
  CONST:
      MaxFlightTime = 100:00;
  BODY:
      FlightTime < MaxFlightTime;
END
```

The time window 28ConsDays is declared to inherit the built in CalDay timewindow, with *FlightTime* and *MaxFlightTimeRule* as new property and constraint names respectively. Its duration is set to 28 calendar days. In the definition of *FlightTime* the aggregation operator SUM is used to evaluate *FlightTime* as the summation of *BlockTimes* over all leg activities during the 28 days time frame, where *BlockTime* is a property defined for the leg activity. The definition of *MaxFlightTimeRule* states that the rule is active and its limitation (*MaxFlightTime*) may be changed dynamically in run-time through its local CONST declaration. The rule returns a true or false value depending on the evaluation of the Boolean expression given at the BODY of the rule definition. The legality checking mechanism is responsible for checking the above rule in a per calendar day step, for the current activity chain that is checked.

(2) **Initialization section**. All variables are assigned initial or default values in the initialization section INITIALIZE before the start of the execution of the rules.

(3) **Definition section**. The definition section contains the definition of the properties and rules that were declared in the declaration section. In Fig. 7, the definition of some of the properties and rules for the shift activity are given.

For the expression of the property calculation rules, a set of operators [20] was defined, e.g.:

- *expression* OF FIRST *activity-name*
  This expression is an example of the specifier operator. Specifiers are used for referencing a specific activity in the aggregation hierarchy and either obtaining one of its properties or evaluating an expression. Specifier FIRST references the first component object of the activity-name type and calculates the expression on it. There are three other specifier operators with similar semantics: LAST, NEXT and PREV.
- SUM *expression* OVER *activity-name*
  This expression is an example of the aggregation operator. Aggregations are used for referencing all the component activities and evaluating an expression on each one that contributes to the net result. Examples of aggregations are SUM, AVG, MAX, MIN, COUNT, etc.

6. Conclusions

The object-oriented approach is a powerful methodology for the analysis and design of complex systems. An extension of the OMT methodology was used in the development of the legality checking subsystem of a day-to-day resource management system. A high-level programming language was defined for the expression and management of the rules. The language and the legality checking system were used in the expression and integration of both EEC and Lufthansa rules.

The main advantage of the DAYSY LC system compared with existing systems is its independence from a particular application and its potential to be used in a wide range of scheduling problems. The user defines the activity types and the activity aggregation tree suitable to the specific application and is able to run what-if scenarios by changing on-line a significant part of the knowledge represented by the rules. In general, the user has the ability to express the rules having all the benefits of the object-oriented approach. Specific information can be defined as exported from the system and this becomes immediately usable from other subsystems, for example, the Automatic Rescheduler in its effort to reduce its search space. In addition, the two systems co-operate for the backtracking needs of the Automatic Rescheduler, in order to reduce the computational effort and improve the performance.

From the user point of view, comments recorded in favour of the DAYSY LC system compared with other ones, are the following: user friendliness, improved flexibility in the expression of the regulations, guaranteed overall system integrity by encapsulating the management of the rules in the LC system, flexibility in the creation of what-if scenarios, and the improved productivity of the Rule Manager.

References

[1] R. Anbil, E. Gelman, B. Patty and R. Tanga, Recent advances in crew-pairing optimization at American Airlines, *INTERFACES*, 21 (1991) 62–74.

[2] S. Lavoie, M. Minoux and E. Odier, A new approach for crew pairing problems by column generation with an application to air transportation, *European Journal of Operational Research*, 35 (1988) 45–58.

[3] E. Housos and T. Elmorth, Automatic subproblem optimization for airline crew scheduling, *INTERFACES* (accepted for publication).

[4] *CARMEN GPC — User's Reference Manual*, Volvo Data AB, Gothenburg Sweden (December 1993).

[5] C. Boegner, New advances in crew planning, management and operations, *Airline Information Systems*, ATTIS 94 Paris France (April 94).

[6] L. Kwok, S. Hung and C.J. Pun, Knowledge-based cabin crew pattern generator, *Knowledge-Based Systems*, 8 (February 1995).

[7] K. Thrampoulidis, N. Agavanakis and V. Makios, General object-oriented design methodology, *Technical Report*, Department of Electrical Engineering, University of Patras, April 1991.

[8] K. Thrampoulidis and N. Agavanakis, Object interaction diagram, a new technique in OO analysis and design, *Journal of Object-Oriented Programming (JOOP)* (June 1995).

[9] J. Rumbaugh *et al. Object-Oriented Modeling and Design*, Prentice-Hall, 1991.

[10] G. Booch, *Object-Oriented Design with Applications*, 2nd edn, Benjamin, 1994.

[11] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf and F. Berthier, The constraint logic programming language CHIP, in *Proc. Int. Conf. Fifth Generation Computer Systems, FGCS-88* Tokyo, Japan, December 1988.

[12] G. Baues, P. Kay and P. Charlier, Constraint based resource allocation for airline crew management, *Airline Information Systems*, ATTIS 94 Paris, France, April 94.

[13] K. Thrampoulidis, E. Housos, C. Goumopoulos and G. Thomopoulos, *Analysis and Classification of Enterprise Regulations* ESPRIT PROJECT EP8402, DAYSY Technical Report D4.1, Patras, Greece, November, 1994.

[14] G. Baues, Detailed design specification — automatic rescheduler *DAYSY Technical Report*, Orsay, France, January 1995.

[15] M. Lesk, Lex — a lexical analyzer generator. *Computing Science Technical Report 39*, AT & T Bell Laboratories, Murray Hill, NJ, 1975.

[16] S.C. Johnson, Yacc — yet another compiler compiler. *Computing Science Technical Report 32*, AT & T Bell Laboratories, Murray Hill, NJ, 1975.

[17] Andrew T. Hutt, *Object Analysis and Design: Comparison of Methods*, Object Management Group, John Wiley, 1994.

[18] P. Van Hentenryck, *Constraint Satisfaction in Logic Programming*, MIT Press, 1989.

[19] N. Diamantopoulos, E. Housos, C. Goumopoulos and K. Thrampoulidis, Rule Translator & Language. *ESPRIT PROJECT EP8402 — DAYSY Technical Report*, Patras, Greece, March 1995.

[20] C. Goumopoulos, K. Thrampoulidis, N. Diamantopoulos and E. Housos, Syntactic and semantic definition of DAYSY rule language, ESPRIT PROJECT EP8402, DAYSY Technical Report D4.2, Patras, Greece, March 1995.

[21] *Flight and Duty Time Limitations and Rest Requirements*, Der umstrittene JAR_Draft Nr. 19.