USING UML FOR STATING AND HANDLING REGULATIONS IN THE RESOURCE (RE)SCHEDULING PROBLEM

K. Thrampoulidis, C. Tranoris
Electrical & Computer Engineering Dept., Computer Laboratory, University of Patras, 265 00 Patras, Greece
Tel: +30-61-997325; Fax: +30-61-997316 E-mail: thrambo@ee.upatras.gr, ctranoris@acm.org

ABSTRACT

In the class of (re)scheduling problems where humans constitute the main resource, the scheduling process is influenced by a great number of regulations. The complexity and the dynamic nature of these regulations impose the need for an efficient, flexible and user-friendly way to express and manage them. Our approach, called REDOM, in the form of a Regulations Handling System and a special purpose language, fulfils the above requirements and is well accepted by users. To improve REDOM we considered the Unified Modeling Language and its specific language for constraint definition known as Object Constraint Language. We developed visual REDOM for the visual representation and handling of regulations.

INTRODUCTION

The daily application of a resource schedule is a difficult and tedious process. Due mainly to unexpected events, like resource weakness to support the schedule, the realization of the planned timetable is impossible. Resource rescheduling which is the only choice, is a difficult real time problem, especially when human resources are involved. The major reason of this difficulty emerges from the complexity and the dynamically changing nature of the regulations that restrict the problem's solutions. In the context of the Airline Domain where we faced the problem, these regulations include among others, complex sets of regulations imposed by employee unions, International Government Regulations, etc. All these regulations, called rules for simplicity, are continuously changing, so there is a need for a user-friendly system to state and handle them. Related regulations exist in other areas too, such as school-timetabling, transportation scheduling, hospital staff scheduling, etc.

Most of the existing scheduling and rescheduling systems, in the airline domain, which seams to be the most complicated, test the legality of the produced solution hard-coding the rules within the application software. Other systems use a special purpose language for the expression and subsequent management of the rules. However these systems, with CARMEN and RuleTalk as the most dominant, can't deal with the representation of rules in the various scheduling application domains and they also require special programming skills in order to express complex rules.

As partners of the DAYSY/ESPRIT (Day-to-day resource management systems) project, we were engaged in addressing the problem of stating and solving regulations for scheduling applications involving human resources. We have researched and suggested a new approach that is based on an Object-Oriented (OO) meta-model [3], and a special purpose OO language called REDOM (REgulations Definition and On-line Manipulation)[4]. The system we developed is currently being utilized by the DAYSY resource management system, which is currently used by Lufthansa Airlines.

In the context of this work, we decided to redesign the entire REDOM meta-model using the Unified Modeling Language (UML) which provides a standard modeling notation[5]. Our next attempt was the expression of REDOM constructs and syntax rules using the Object Constraint Language (OCL)[5][6].

In the next section, a brief description of the REDOM approach is given. Section three briefly refers to the main points of the OCL as a language to state constraints on the object model. Section four describes by use of an example the way we use OCL in cooperation with Visual REDOM to embed rules in the airline object-model and finally the last section concludes the work.

THE REDOM APPROACH

The project team of the rescheduling system, in order to fulfil the efficiency requirements, was guided to adopt the Constraint Logic Programming paradigm (CLP). In particular, the CHIP language was used for the development of the rescheduling algorithm in order to obtain efficiency close to imperative programming and considerably reduced development time[2]. Unfortunately, the statement of regulations was extremely hard and resulted in a large semantic gap between the implied formalization and the problem regulations, as perceived by the end-user. To reduce this semantic gap and make the system "acceptable", we decided to develop a Regulations Handling Subsystem (RHS) to carry out these tasks.

For the development of the RHS, we have researched and suggested a new approach that is based on an OO meta-model [3]. For the defined meta-model to be easily applied to a wide range of application domains a two step process was defined. In a first step, specific problem domain experts (e.g., airline experts) apply Domain Analysis to create the airline object-model as an instance of the generic meta-model. This model contains among

others declarations of airline generic activities, properties, rules and problem domain keywords. In a second step, the specific application rule-manager (e.g., Lufthansa's rule-manager) specializes and refines the airline object-model in order to produce the specific user's application object-model. However, from the user's point of view, the definition of these object-models in terms of a general-purpose Object-Oriented programming language was very difficult. The absence of a language to state constraints on the object model was also evident. These were the main reasons for defining REDOM. By using REDOM, the non-skilled programmer specializes the generic model, by embedding in a way close to his terms, specific notions from the context of his problem and effectively experiments with new regulation and environment changes.

The REDOM approach can be applied to different scheduling application domains with a minimum degree of effort. An application programming interface facilitates REDOM's integration into existing scheduling systems.

REDOM's Metamodel In Unified Modeling Language Notation.

We decided to use UML to re-design the DAYSY meta-model. UML is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. It is already an OMG standard for OO analysis and design. UML represents a collection of best engineering practices that have proved successful in the modeling of large and complex systems. The interoperability of the UML models among several OO CASE tools and IDE's, will ease the use and extension of the REDOM approach making the meta-model easily adopted into other application domains.

As shown in figure 1, where part of the REDOM meta-model that constitutes the base of our approach is given, primitive activities provide the basis for the construction of other, non-primitive activities, that are called composite activities. A composite activity is conceptually composed of other activities, which are referred to as component activities. For each composite activity, there must be an activity composition rule. Each activity may be characterised by a set of complex properties. For each complex property there must be a property calculation rule to evaluate it. Property constraint rules are restrictions concerning mainly values of properties. Finally, the concept of the time-window was introduced to cope with regulations, which express constraints on specific time intervals.

However the meta-model is not enough for stating the whole problem. There are many subtleties and nuances of meaning that a diagram cannot convey by itself: uniqueness, derivation, limits, constraints, etc. It is clear that a careful combination of a diagrammatic and a formal language would offer the best solution. This was the basic motivation of the creators of OCL: the creation of an expression language that enables one to describe constraints on the object models as well as on the other object modeling artifacts [6].

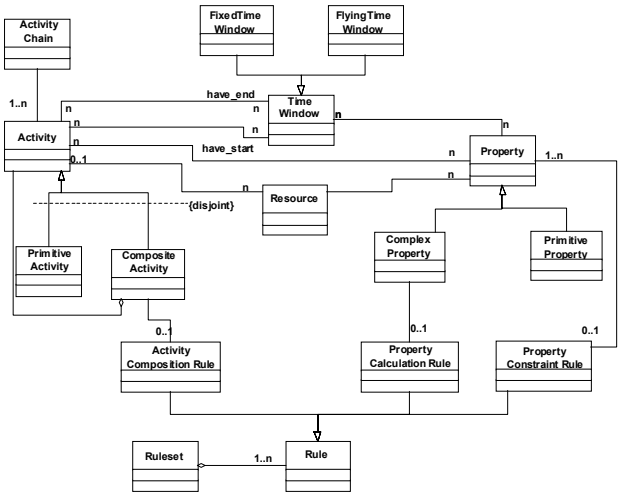

Figure 1. Part of the REDOM meta-model.

OBJECT CONSTRAINT LANGUAGE

OCL is a formal language to express side effect-free constraints and is formally defined in [6]. Users of the UML and other languages can use OCL to specify constraints and other expressions attached to their models. Such constraints are usually described in natural language, but practice has shown that this is a source of many ambiguities. A number of formal languages have been developed, in order to write unambiguous constraints, but since they require a mathematical background, they are difficult for the average business or system modeler to use. OCL has been developed to fill this gap. It is a formal language that still remains easy to read and write. Among the primary requirements of the language we discriminate:

-OCL can express extra information on the design models. It is a precise, unambiguous language that can easily be read and written by all practitioners of object technology.

-OCL is a declarative language. Its expressions have no side effects. The state of the system will not change because of an OCL expression.

-OCL is a typed language and so all expressions can be checked during modeling and before execution.

OCL has various uses in a UML model. It can be used in expressing invariants. An invariant is a constraint that can be associated with a class, a type or an interface in a UML model. An invariant means that the result of the expression must be true for all instances of the associated class. OCL can also be used to specify the pre- and post conditions of operation methods on all classes, types and interfaces and to express the guards of transitions. A guard on a transition is a boolean expression that determines whether or not the transaction is enabled. The context of an OCL expression is always a specific

element of a UML model.

OCL comes with a basic set of constructs with which someone can write constraints from simple expressions to complex navigations through the associations on a class diagram. Collections, Sets, Bags and Sequences, are the language's predefined collection types, used to specify the exact results of navigations through associations. A number of standard operations such as select, reject, and collect are offered by the language to handle the elements of the above collections. We use the part of the airline object model of figure 2 to give some representative OCL expressions.

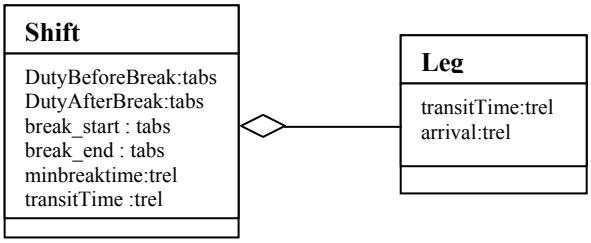

Figure 2. Part of the airline domain Object-model.

The following OCL expression specifies the value of the *Break_start* attribute of the *shift* activity.

Break_start() = self.leg.Arrival→
select(self.leg.transitTime>=min_break_time)
→asSequence→Last

By use of the *select* construct on the component activity *Leg*, we select only those instances where the property *transitTime* of *Leg* is equal or larger than *min_break_time*. From this selection we get the last element, by applying the operation *Last*, and afterwards specify its property *Arrival.*

Constraints on attributes usually result in more simple expressions like the following:

self.DutyBeforeBreak<=max_duty_before_break and
self.DutyAfterBreak<=max_duty_after_break

The above invariant, applies to all instances of Shift and it shows that at any time and for all instances the properties DutyBeforeBreak and DutyAfterBreak must be less than or equal of max_duty_before_break and max_duty_ after_break respectively.

EXPRESSING RULES IN OCL

OCL being a standard constraint language, allows the rule manager to write the rules making use of a simple yet formal language with a well-defined syntax and grammar. We support the use of OCL in two different ways:

1. OCL can be used instead of the REDOM, for the statement of the rules and
2. OCL can be used to refine or embed in the ruleset constraints not supported by REDOM.

For the second option to be possible, we guided to define an intermediate layer between REDOM and the low-level rule representation. This layer is an OCL representation and is automatically generated by a tool that implements a well-defined set of mapping rules between REDOM constructs and their formal OCL equivalents.

We discussed earlier that, specific problem domain experts apply Domain Analysis to create the airline object-model as an instance of the generic meta-model. The need of a tool that improves the ability of the rule manager to express in an easier way the rules of the problem was imperative. We developed a prototype system, called Visual REDOM which we use to visually instantiate the DAYSY meta-model in the specific application domain with the following objectives:

- be an easy tool that automates the process of stating the rules
- be able to manage activities, properties and constraints and their associations
- give the ability to express constraints in both REDOM and OCL
- be able of converting the expressions from REDOM to OCL according to a set of well defined transformation rules
- be able to automatically generate code for insertion in the RH system (see figure 3).

As an example we give the following rule known as Breaks rule, extracted from Lufthansa's rule-set. According to this rule: "A rest period between the legs of a shift, greater than 3 but less than 11 hours is considered as a 'break'. … There can be no more than one break in a shift. In addition, the duty time before and after the break must not exceed 10 hours". The following expressions belong to activity *shift* REDOM specification and are part of the statement of the above rule.

dutyStart = (departure OF FIRST leg) – briefing
dutyEnd = (arrival OF LAST leg) + debriefing
breakStart=(arrival OF FIRST leg)
where(transitTime>=min_break_time)
maxTransitTime = (max transitTime OVER leg)

In their coresponding OCL expressions which are given below, we must point out that an operation constraint can also be read as a definition of the operation, where the right-hand side of the equal sign determines the value the operation will return.

DutyStart()=self.leg.departure→asSequence→First-
briefing
dutyEnd()=self.leg.Arrival→asSequence →Last
+debriefing
breakStart()= self.leg.Arrival→select(transit_time>=
min_break_time)→asSequence→First
maxTransitTime()= self.leg.transit_time→iterate
(elem;acc|acc.max(elem))

The next step was the production of a low-level representation of these originated expressions. We have already initiated a work for the transformation of REDOM specifications to CHIP code in order to be used by the CHIP rescheduling subsystem [1]. As it was

proven from this work REDOM regulations could easily be transformed and solved in any CLP(FD) target language, like CHIP, which is less natural for constraint representation but computationally more efficient. Regarding the OCL representation since there is no tool that generates code from OCL and implements constraints in runtime we have chosen CHIP++ as the target language for transformation. Making the low-level representation in another form would lead us to the creation of such an engine from scratch. However since there is no relative experience, certain precautions must be taken. The evaluation order of the OCL expressions is of great interest as well as that an OCL constraint must be always treated as an atomic expression. No changes of value of any object in the system can take place during evaluation of the constraint. Another problem arises when we come to the point of checking a constraint. When coding a constraint, we must decide when to check it. Invariants, for example, are true all the time and we must decide when they should be evaluated. The same cautions apply for pre- and post- conditions. OCL does not define these concepts and we need to make our own judgments.

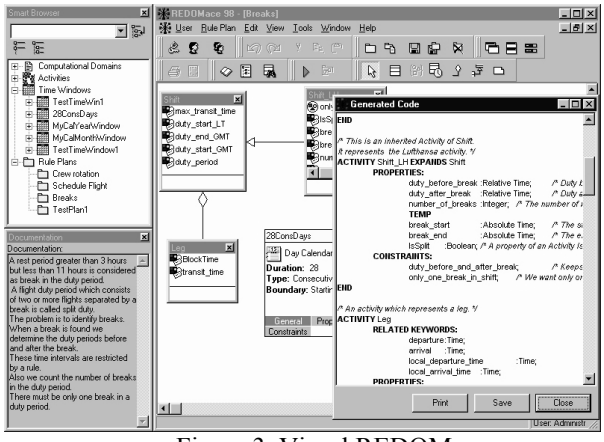

Figure 3. Visual REDOM

OCL proved not so hard for the average modeler to use it. With its well-defined syntax and its predefined types and operations, the usage was effortless. Also, in spite of that OCL is a simple language and does not contain many of the features defined in other specification languages, it was possible to extend the language to our needs, since many operations that we needed were not defined in OCL. Advantages, such as: OCL will be used extensively in the future as a standard formal language for expressing constraints, that it will offer interoperability and portability among the tools for our model, that it needs a small amount of time to learn and that future tools (CASE tools, code generators) will support it, made the language a valuable tool for us. On the other side though, we found most of the REDOM expressions much simpler than their equivalents in OCL and we encountered REDOM much closer in the domain of resource rescheduling

CONCLUSIONS

We have already developed, based on REDOM language, a prototype Regulations Handling System that was tested and evaluated in crewmember rules of Lufthansa German airlines. The demands for natural problem description have been met to a large extent. In particular, the user acceptance of the system, and especially of the REDOM expressive facilities, is very encouraging. To improve and formulate the meta-model's instantiation process as well as the statement of regulations we developed a prototype system for the visual statement of the problem. Visual REDOM provides the end-user a valuable tool with which the problem's domain object model is graphically constructed and it is automatically checked for its compatibility with the REDOM meta-model. The tool supports the subsequent statement of regulations in terms of visual object programming. It also provides for the integration of user friendly REDOM expressions with the more formal and powerful OCL constructs.

We have also initiated work in the direction of transformation of OCL specification to CHIP code for use with the CHIP rescheduling subsystem. To our estimation OCL regulations can be transformed and solved in any CLP(FD) target language, like CHIP. The degree of complexity of stated rules is really impressive, a major role of which is due to the intermediate OCL layer. Some complicated regulations may now be represented more straightforward in this layer. The efficiency of the resulting scheduling application, derived from the combination of CHIP and our system, is fairly reasonable, although it is expected to improve drastically when the transformation work is completed.

REFERENCES

[1] Diamantopoulos,N.,Thrampoulidis,K., Housos,E. "Integrating Object and Constraint Technologies for Stating and Solving Resource Scheduling Problems", 6th Hellenic Conference on Informatics, Athens 1997, p. 238-248.

[2] Dincbas,M., Van Hentenryck,P. Simonis,H. Aggoun,A. Graf,T. and Berthier,F. (1988) "The Constraint Logic Programming Language CHIP" In Proceedings on the international Conference on Fifth Generation Computer Systems FGCS-88, Tokyo, Japan.

[3] Thrampoulidis,K. Goumopoulos C., Housos E. (1997) "Rule Handling in the day-to-day Resource Management problem: an Object-Oriented approach." Information and Software Technology, v. 39, n. 3, p.185-193.

[4] Thrampoulidis K., N.Diamantopoulos, E. Housos, (1997) "REDOM : An Object-Oriented Language to Define and On-line Manipulate Regulations in the Resource (Re)Scheduling Problem" Software - Practice & Experience, , vol. 27(10), 1135-1161.

[5] "Unified Modeling Language - UML Semantics - Object Constraint Language Specification " OMG, version 1.1, September 97.

[6] Warmer Jos, Kleppe Anneke, "The Object Constraint Language-Precise Modeling with UML", Addison-Wesley October 1998.