MODELING A REAL APPLICATION AS A PLANNING PROBLEM BY USING UML.P

TIAGO S. VAQUERO[1], FLAVIO TONIDANDEL[2], LELIANE N. BARROS[3], JOSÉ REINALDO SILVA[1]

[1]*Escola Politécnica – Universidade de São Paulo*
Design Lab. – PMR – Dep de Engenharia Mecatrônica e Sistemas Mecânicos - São Paulo, Brazil

[2]*Centro Universitário da FEI*
IAAA Intelligence Applied in Automation Lab - São Bernardo do Campo, Brazil

[3]*Instituto de Matemática e Estatística – Universidade de São Paulo*
Departamento de Ciência da Computação - São Paulo, Brazil

E-mails: tiago.vaquero@poli.usp.br,flaviot@fei.edu.br,
leliane@ime.usp.br, reinaldo@usp.br

Abstract— There is a great interest in the automated planning community to apply all developments already achieved in the area to real applications. Such scenario makes the community to focus on Knowledge Engineering (KE) applied in modeling of planning problems and domains. The first ICKEPS (International Competition on Knowledge Engineering for Planning and Scheduling) showed powerful tools, such as itSIMPLE, that can help designers of planning domains to better understand, specify, verify and validate their models. The itSIMPLE tool proposes the use of UML to Planning Approach, denominated UML.P, during planning domain modeling process. This paper reports on the exposure of UML.P to a real application, e.g., the problem of sequencing cars in an assembly line. This modeling experience, using a classical manufacturing problem, provides some insights and considerations that can contribute to a general KE process for planning.

Keywords— Automated Planning, Knowledge Engineering, Domain Modeling

Resumo— Existe um grande interesse da comunidade cientifica de planejamento automático em aplicar todos os desenvolvimentos alcançados na área recentemente em situações reais. Tal cenário indica um foco de pesquisa em Engenharia do Conhecimento (EC) aplicada a modelagem de problemas e domínios de planejamento. O primeiro ICKEPS (International Competition on Knowledge Engineering for Planning and Scheduling International Competition on Knowledge Engineering) mostrou ferramentas poderosas, como o itSIMPLE, que auxilia os modeladores de domínios de planejamento a entender, especificar, verificar e validar seus modelos. A ferramenta itSIMPLE propõe, para o processo de modelagem, o uso da UML for planning approach, denominada UML.P. Este artigo apresenta o uso da UML.P em uma aplicação real, e.g., o problema do sequenciamento de carros em linhas de montagem. Esta experiência de modelagem, usando um problema clássico de manufatura, fornece considerações importantes que podem contribuir para o processo geral de EC para planejamento.

Palavras-chave— Planejamento Automático, Engenharia do Conhecimento, modelagem de domínio

1. Introduction

The recent efficiency improvement and the rising demand for planning systems have become a great motivation to try applying all developments already achieved by the planning community in real and complex applications. In this scenario, Knowledge Engineering methodologies become more and more important since modeling actions is considered to be the bottleneck of practical planning systems development. This has been addressed in several initiatives, such as the first *International Competition on Knowledge Engineering for Planning and Scheduling* - ICKEPS 2005. This competition brought extremely important modeling issues and showed powerful tools, such as itSIMPLE [Vaquero et al 2005] and GIPO [Simpson 2005], that can help designers to better understand, specify, verify and validate their planning models. The itSIMPLE tool performs analysis and verification of requirements in a planning domain model, uses UML.P, which is a special planning approach from UML [OMG 2001], as an object-oriented model specification and visualization which can be used to unify different modeling perspectives from planning experts, domain experts and stakeholders.

This paper is an extended version of [Vaquero et al 2006] that describes an experience on the exposure of UML.P to a real application such as the Car Sequencing problem in an assembly line. This is a classical manufacturing problem which encompasses many challenging features such as planning with resources, sequencing, scheduling, optimization, and others. The Car Sequencing problem is based on the daily job of factories where a production day to each ordered vehicle must be assign according to production lines capabilities and delivery dates. This interesting domain was extracted from the fourth ROADEF Challenge 2005 [Nguyen 2003] where researchers explored the requirements and scheduling difficulties encountered in industrial applications.

Following, the paper presents the basic concepts of modeling with UML using the planning approach (UML.P). Next, it presents the requirements for the problem and an overview of the itSIMPLE modeling method followed by the generated UML.P model. It then presents the translations from UML.P to PDDL [Fox and Long, 2003] followed by the verification of the PDDL model with results. Finally, the paper ends with some modeling considerations, suggestions to future works and conclusions.

2. Modeling with UML – The Planning Approach

The UML – Unified Modeling Language is one of the most used languages to model a great variety of applications [D´Souza and Wills 1999]. Besides, the UML has flexibility to attend many kind of models in an object-oriented fashion.

Among all diagrams of UML, the class diagram is the most known diagram for representing and modeling the static structure of object-oriented systems. However, all details of the structure cannot be easily represented and expressed only in the class diagram. For those additional representations, there are others diagrams and a formal constraint specification language called Object Constraint Language – OCL [OMG 2003]. In OCL, constraints are Boolean expressions composed with logical connectives as in predicate calculus. This constraint language can describe invariants and derivated rules on classes, pre and post conditions on actions and it supports universal and existential quantifications.

Since UML is a general purpose modeling language, some specification features are intrinsically related to planning domains. For that reason, the UML.P (UML in a Planning Approach) was firstly defined at [Vaquero et al 2005] as a way of using the general UML for the planning matters where the automated planning concepts are specified and modeled. The UML.P approach has been improved and refined in this work.

This approach first considers relationships between planners, domains and planning problems. In a planning context, the modeling process follows the principles that: domains have their own description and specification (including static structure, dynamic behavior, etc); problems are associated to domains and they have their own constraints initial condition description and goal description; planners plan over associated problems and domain descriptions. The following descriptions of UML diagrams show how designers can specify and better understand their planning domains.

The *Use Case diagram* models the domain in the highest abstraction level where the domain scope is firstly defined. This diagram facilitates the unification of the viewpoints from domain experts, stakeholders and planning experts.

In UML, Use Case specifications are usually described in natural language in the desired abstraction level, but UML.P makes it different. Since natural language specification can create ambiguities and redundancies, a proposal of using a structured Use Case specification contributes to minimize these problems [Silva and Santos 2004].

Other important diagram in UML for planning is the *Class diagram*. The class diagram is a representation of the planning domain static structure and concepts showing the existing entities, their relationships, their features, methods (actions) and constraints. Classes, Class attributes and associations between classes give a visual notion of the semantic.

In order to specify the dynamic behavior of actions, the *StateChart diagram* is necessary where it is possible to define their pre and pos conditions. This diagram is very useful to represent entities that perform dynamic behavior. Usually all actions defined in the class diagram are better specify in this diagram.

Any class in Class diagram has its own StateChart diagram specially those that perform actions. Each diagram does not intend to specify all changes caused by an action, instead, it shows only the changes that it causes in an object of the StateChart diagram's class. The constraints on the Class diagram and all the pre and post conditions on the StateChart diagram are specified using the language OCL.

A problem statement in a planning domain is characterized by a situation where only two points are known: the initial and goal state. The diagram used to describe these states is called Object Diagram or Snapshots [D´Souza and Wills 1999].

A snapshot is a picture of a specific time and an instantiation of the domain structure. Such instantiation represents features such as: how many objects are in the problem; what are their classes; what are the values of each object attribute and how they are related with each other. In fact, a planning problem is composed by two *Object Diagrams*, one describing an initial state and another describing a partial or entire goal state. Additional constraints related to the problem can be specified also using OCL.

3. Car Sequencing as a Planning and Scheduling Problem

In the Car Sequencing planning and scheduling process, customer orders are sent to car factories in real-time. The factories have to assign daily a production goal to each ordered car according to the production line capabilities, constraints and delivery dates. Then, factories have to schedule the order of the vehicles to be put on the line for each production day, while satisfying a set of complex requirements and constraints of the plant shops. A car will be manufactured in the following order: Body, Paint and Assembly.

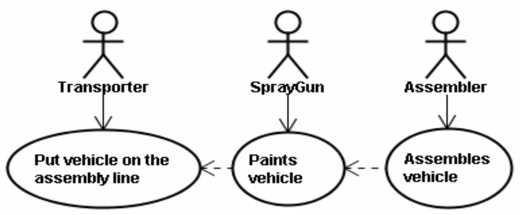


Figure 1: Use Case diagram for Car Sequencing domain

This challenging planning and scheduling manufacturing problem encompasses interesting features such as planning with resources, sequencing, job-shop, scheduling, optimization, cost minimization, flexibility and others that make the problem even more complex when combined. All this aspects make this planning/scheduling domain an excellent challenge for a planning driven modeling process such as the proposed UML.P.

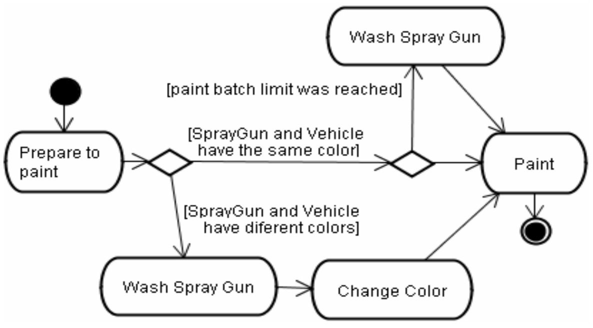
The Car Sequencing problem requirements that will be used as the running example throughout the paper was extracted from an important system competition called ROADEF Challenge where researchers explored the requirements and difficulties encountered in real industrial applications. The fourth edition, called ROADEF Challenge 2005 brought the car sequencing problem provided by RENAULT Co. which will be described in the following by the given requirements presented by [Nguyen 2003].

3.1. Domain and Problem Requirements

The considered real sequencing problem focuses on the requirements of the paint shop and the assembly line, since body shop does not constraint the daily schedule. The order of the scheduled vehicles can not be changed during painting and assembling for a production day. Generally, each vehicle receives identification before getting into the paint shop containing: its *identifier*; its *sequence rank* in the production day given by the planning/scheduling system; the production *date* of the vehicle; its *paint color* and what *special features* the vehicle will receive at assembly. Following, an overview of the paint shop requirements and assembly line is showed.

Paint shop requirements. This part of the plant has to consider the minimization of paint solvent which is used to wash spray guns each time the paint color is changed between two consecutive cars. Implicitly there is a requirement to group vehicles together by paint color. This is a clear necessity to attempt to minimize the spray gun washes. In other words, a necessity of schedule the longest paint color batches possible [Nguyen 2003].

Assembly line requirements. The most important requirement in the assembly line is to smooth the workload. Cars that need special features (for instance, air conditioning, sunroof, etc.), requiring extra assembly operations, have to be evenly distributed throughout the vehicle sequence in order to avoid assembly line overloads.



Wash Spray Gun
[paint batch limit was reached]
Prepare to paint
[SprayGun and Vehicle have the same color]
Paint
[SprayGun and Vehicle have diferent colors]
Wash Spray Gun
Change Color

Figure 2: Activity diagram for painting a vehicle

The vehicles may not exceed a given quota over any sequence of vehicle. This requirement is associated to a ratio constraint **N/P** for each special feature which means that at most **N** vehicles in each consecutive sequence of **P** vehicles has this special feature. The violation of each special feature ratio N/P must be minimized by the planner. As described in [Nguyen 2003], this assembly line requirement is a soft constraint.

4. UML.P Model Representation

The UML.P representation for the running example will be presented first by the Domain modeling and then by the Problem modeling.

4.1. Domain Modeling

From the analysis and discussion on the car sequencing problem requirements described previously, it was possible to define the Domain scope using the Use Case diagram in a high level of abstraction. Figure 1 shows the resulting use case diagram where there are three main agents and three use cases. These three agents (or actors) are the only entities that can act over the domain: *Transporter* puts the vehicles on the line; *SprayGun* paints the vehicles in a line; *Assembler* assembles the vehicles in an assembly line. The entity Transporter was included in the domain for a better matching between model and real application.

In order to better clarify what really happens at the “Paint Vehicle” use case it was used the Activity Diagram for a visual explanation. Figure 2 shows such diagram. The flow at Figure 2 starts at the left black circle and it ends at the right black circle. This activity diagram summarizes the role and capabilities of the *SprayGun* in the domain scope.

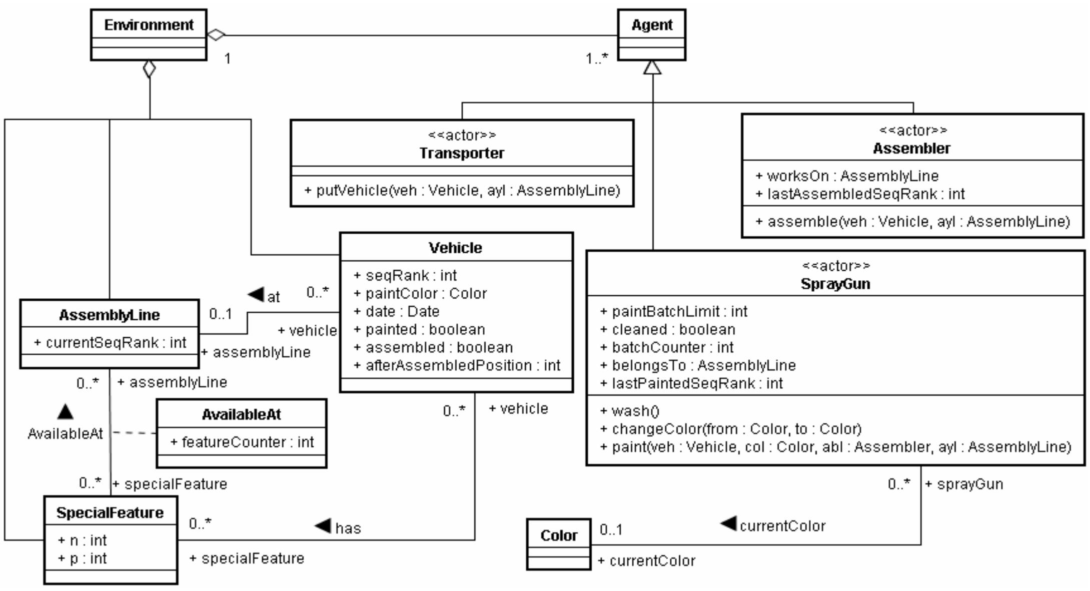
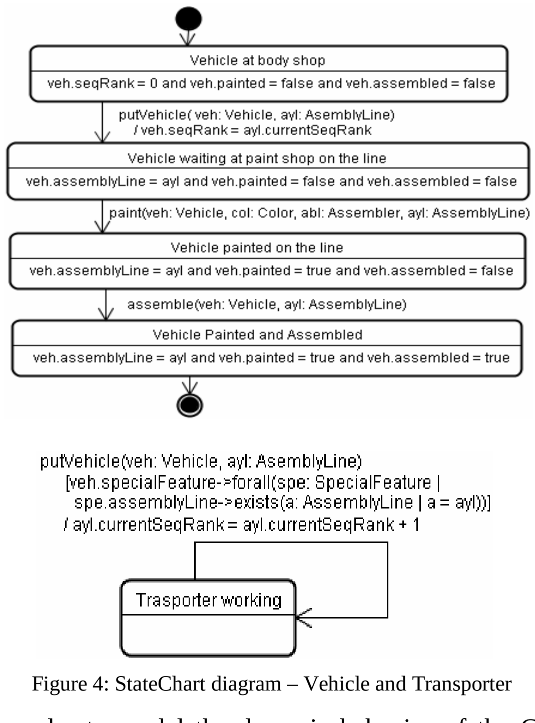


Figure 3: Class diagram for the Car Sequencing Domain

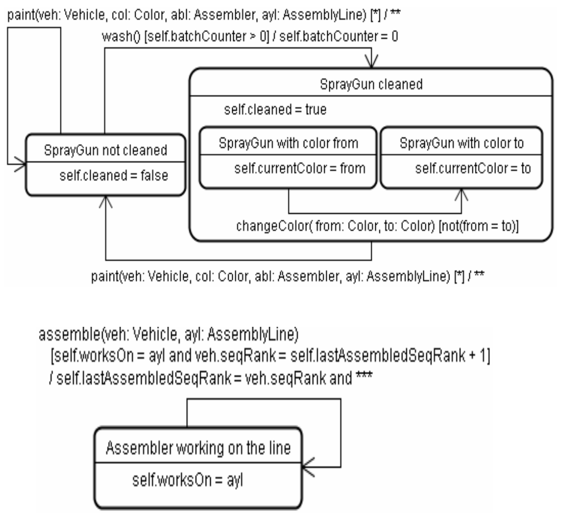
In order to structure all the static concepts of the domain with an object-oriented approach, the Car Sequencing Class diagram was built. Figure 3 shows the Class diagram. Observe that, in Figure 3, the class *Vehicle* has an attribute *paintColor* of type *Color* and that *SprayGun* Class has an association 1 to 1 with Color Class. In fact, we can say that in UML an association from X to Y with multiplicity 1 to 1 is similar to an attribute of the class X.



Vehicle at body shop
veh.seqRank = 0 and veh.painted = false and veh.assembled = false
putVehicle(veh: Vehicle, ayl: AsemblyLine)
/ veh.seqRank = ayl.currentSeqRank
Vehicle waiting at paint shop on the line
veh.assemblyLine = ayl and veh.painted = false and veh.assembled = false
paint(veh: Vehicle, col: Color, abl: Assembler, ayl: AssemblyLine)
Vehicle painted on the line
veh.assemblyLine = ayl and veh.painted = true and veh.assembled = false
assemble(veh: Vehicle, ayl: AssemblyLine)
Vehicle Painted and Assembled
veh.assemblyLine = ayl and veh.painted = true and veh.assembled = true
putVehicle(veh: Vehicle, ayl: AsemblyLine)
[veh.specialFeature->forall(spe: SpecialFeature |
spe.assemblyLine->exists(a: AssemblyLine | a = ayl))]
/ ayl.currentSeqRank = ayl.currentSeqRank + 1
Trasporter working

Figure 4: StateChart diagram – Vehicle and Transporter

In order to model the dynamic behavior of the Car Sequencing domain it was necessary to use the StateChart diagram. Following the UML.P, the classes *Vehicle*, *Tranporter*, *SprayGun* and *Assembler* require a StateChart diagram. Each diagram is specified by using an object-focused specification. For example, when building the *Vehicle* StateChart diagram, the elements that appear at this diagram only concern to the vehicles as a single object. This object-focused specification helps to separate context and model what is important to the object. It is in this diagram where the use of OCL becomes very important and essential. The figures 4 and 5 show each one of these four diagrams.



paint(veh: Vehicle, col: Color, abl: Assembler, ayl: AssemblyLine) [*] / **
wash() [self.batchCounter > 0] / self.batchCounter = 0
SprayGun cleaned
self.cleaned = true
SprayGun not cleaned
self.cleaned = false
SprayGun with color from
self.currentColor = from
SprayGun with color to
self.currentColor = to
changeColor(from: Color, to: Color) [not(from = to)]
paint(veh: Vehicle, col: Color, abl: Assembler, ayl: AssemblyLine) [*] / **
assemble(veh: Vehicle, ayl: AssemblyLine)
[self.worksOn = ayl and veh.seqRank = self.lastAssembledSeqRank + 1]
/ self.lastAssembledSeqRank = veh.seqRank and ***
Assembler working on the line
self.worksOn = ayl

Figure 5: StateChart diagram – SprayGun and Assembler

Unifying all the StateChart diagrams through pre and post conditions of each action specified we have the whole OCL action specification. Following, some action examples in OCL will be given with some design solutions

In the action of figure 4, there is an association between two classes, *Vehicle* and *SpecialFeatures*, called *has* that goes from *Vehicle* to *SpecialFeatures* with a multiplicity of 0..* for both connections. This association can have role names that identify each connection. However, if there is no role name, we can use the name of the class to identify that connection. In the action, therefore, we can call the set of *SpecialFeatures* that is related to an object *Veh1* of *Vehicle* only by call *Veh1.specialFeatures*, for instance.

Therefore, with role names and multiplicity, we can operate sets of objects of classes connected by an association and expression like:

```
veh.specialFeature → forall(spe: SpecialFeature|
spe.assemblyLine → exists(a: AssemblyLine|a = ayl))
```

It can be semantically translated as: "*there is an assembly line with all special features of a vehicle veh*". Another example is the use of the *lastPaintedSeqRank* and *lastAssembledSeqRank* attributes to synchronize the action if paint and the action of assemble. This is done by making the spray gun paint only under the condition that the last car painted was already assembled, i.e., *lastPaintedSeqRank* = *lastAssembledSeqRank*. This fact avoids creating vehicle buffers between paint shop and assembly. The resulting action is:

```
context SprayGun::paint(veh: Vehicle, col: Color, abl:
Assembler, ayl: AssemblyLine)
pre:   self.belongsTo = ayl and self.currentColor = col
and veh.paintColor = col and veh.seqRank =
self.lastPaintedSeqRank + 1 and self.batchCounter <
self.paintBatchLimit and abl.worksOn = ayl and
self.lastPaintedSeqRank = abl.lastAssembledSeqRank
and veh.assemblyLine = ayl and veh.painted = false
and veh.assembled = false
post:   self.batchCounter = self.batchCounter + 1 and
self.lastPaintedSeqRank = veh.seqRank    and if
(self.cleaned = true) then self.cleaned = false
    endif and veh.painted = true
```

4.2. Problem Modeling

Optimization aspects are extremely important in the car sequencing domain. Since these aspects concern to the problem in the form of solution constraints, it was preferable to model optimization after having the whole domain model. In our current example the critical actions for optimization are the *changeColor* (context *SprayGun*) and *assemble* since we need to reduce the paint solvent and also to penalize the sequence chosen by the planner every time each value of *featureCouter* exceed the attribute value **N,** in a sequence of **P** car, of the respective *SpecialFeature*. For these restrictions, two variables are declared: *numberPaintColorChanges* and *numberViolations*. These two additional specifications are presented in OCL the respective actions.

Since a Planning Problem requires an initial state and a goal description, UML.P uses the object diagram in order to describe these two states. This process is done by only instancing the class diagram in an object diagram where the attributes of the objects receive values. For example, instantiating vehicles *V1, V2, V3* and *V4* requiring color *Red, Red, Blue* and *Black* respectively and so on.

5. Translating UML.P Model to PDDL

Since the Sequence Car domain was already modeled in UML.P, it was easier to specify the PDDL model than if we try to model this domain in PDDL from scratch. During translation process it was clear that PDDL has some limitation such as: it is not possible to use subsequent conditional effects (when) inside a universal quantification which results in a code with some repetition.

The most complicated OCL expressions translated were those that use universal and existential quantifications, but it was completely feasible, since they are semantically the same. During the translation process no domain characteristics were left behind, every expression was possible to be translated, but many constraints such as multiplicities were lost. Some of the multiplicity constraints can be expressed using PDDL 3.0 [Gerevini and Long 2005] which will be left for future work.

OCL does not express optimization functions such as :metric in PDDL, for instance (:metric minimize (+ (numberViolations) (numberPaintColorChanges)), but it can model something similar using value invariants.

6. Verification of PDDL Model

To verify the model described in PDDL, generated by our modeling approach, we choose the Metric-FF [Hoffmann 2003]. Metric-FF is a forward heuristic planner which uses a relaxed plan graph to provide heuristic estimates of the distance of the current state to the goal state.

Our initial purpose is to analyze the expressiveness and soundness of our model. Therefore, we are interested in the quality of the solutions that can be found by the Metric-FF planner despite of time for processing it.

In order to make this verification, it was generated two main scenarios in PDDL. The first scenario (s1) represents the set of problems that focus only on the optimization of the paint solvent, i.e., the metric function includes only the minimization of the number of paint color changes. The second scenario (s2) focuses on the optimization of the workload on an assembly line, i.e. the minimization of sequence violations.

Since an industrial application manages the planning/scheduling process using Simulated Annealing or Constraint Satisfaction Programming (CSP) [Brucker 2004], the Metric-FF results for these scenarios were compared with some of these systems by using a solution-checking tool to check the validity and quality of the planning/scheduling solutions, provided by the ROADEF competition organizers. This solution-checking tool penalizes each time there is a constraint violation in a solution. Table 1 shows the results.

Table 1: Results of the solution quality comparing Metric-FF, two Simulated Annealing systems (A1 and A2) and one CSP (C1). Score 0 means an excellent solution. Spf means special features.

Scenario	Problem	Metric-FF	A1	A2	C1
s1	6 cars 2 colors	10000	10000	10000	10000
	8 cars 2 colors	30000	10000	40000	10000
	8 cars 3 colors	50000	20000	30000	20000
s2	6 cars 1 spf	0	0	0	0
	8 cars 1 spf	0	0	10000	0
	6 cars 2 spf	0	0	0	0
	8 cars 2 spf	2000	0	60000	0
	8 cars 3 spf	2000	0	90000	0

The Metric-FF solutions (output) have similar qualities when compared with the others. By analysis the results we can say that planning domain model, created by UML.P methodology, is correctly leading the planner to find high quality solutions, very similar to those generated by dedicated scheduling techniques. The analysis also encourages the use of planning system to real domains as well as its improvements.

7. Future Works and Conclusions

This paper has showed a simplified real application, the Car Sequence problem, extracted from the ROADEF Challenge 2005, modeled in UML.P. The model, initially described in UML.P, was translated to PDDL and verified by Metric-FF planner [Hoffmann 2003].

Many difficulties that a designer can find when modeling a domain from scratch using PDDL can be reduced and sometimes completely overcame when using UML.P and OCL. First, object-oriented approaches are more intuitive than declarative and action driven languages like PDDL. Second, the UML has diagrams that can lead the designer to discover the essences of the model and the correct semantic of the entire application separating domain and problem concerns. Third, the UML.P permits non-planning experts to model their domains without the need of a PDDL expert.

For the future, we intend to implement the new features incorporated by the ROADEF domain into the itSIMPLE tool. Since the concept of the itSIMPLE tool is to be compatible with PDDL, the tool will be improved to deal with new versions of PDDL like version 3.0 (Gerevini and Long 2005). The itSIMPLE tool will also incorporate a complete translation of models to and from PDDL in order to let the designer a great flexibility to choose the way to export and use their models.

References

Brucker, P. 2004. Scheduling Algorithms, 3rd edition. Springer-Verlag New York, Inc.

D'Souza, F. D. and Wills, A. C. 1999. *Object, Components, and Frameworks with UML – The Catalysis Approach*. Addison-Wesley. USA and Canada.

Fox, M. and Long, D. 2003. PDDL 2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research* 20:61-124.

Gerevini, A. and Long, D. 2005. *Plan Constraints and Preferences in PDDL3 – The Language of Fifth International Planning Competition*. Technical Report, Department of Eletronics for Automation, University of Brescia, Italy, August 2005.

Hoffmann, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*. Accepted for special issue on the 3rd International Planning Competition.

Nguyen, A. 2003. *Challenge ROADEF'2005: Car Sequencing Problem.* Renault - Information System and Technologies - Advanced Studies. France.

OMG - Object Management Group, 2001. *Unified modeling language specification: version 1.4.*

OMG - Object Management Group, 2003. *OCL 2.0 – Object Constraint Language*.

Simpson, R. M. 2005. *GIPO Graphical Interface for Planning with Objcets.* ICAPS 2005 Competition on Knowledge Engineering for Planning and Scheduling, Monterey, California, USA.

Vaquero, T.S.; Tonidandel, F.; Silva J. R. 2005. *The itSIMPLE tool for Modeling Planning Domains*. ICAPS 2005 Competition on Knowledge Engineering for Planning and Scheduling, Monterey, California, USA.

Vaquero, T.S.; Tonidandel, F.; Barros, L. N.; Silva J. R.. 2006. *On the use of UML.P for Modeling a Real application as a Planning Problem*. Proceedings of ICAPS 2006 (Short paper). Cumbria, UK.

Silva, R. and Santos, E. A . 2004. Applying petri nets to requirements validation In: *IFAC Symposium on Information Control Problems in Manufacturing. Salvador, 2004*. INCOM'04 : Abstracts.Salvador : IFAC, p. 1.