REQUIREMENTS ANALYSIS OF AUTOMATED PROJECTS USING UML/PETRI NETS

Pedro M. González del Foyo, pedro.foyo@ufpe.br
Dept. of Mechanical Engineering, Federal University of Pernambuco, Brazil
Arianna Olivera Salmon, arianna@usp.br
José Reinaldo Silva, reinaldo@usp.br
Dept. of Mechatronic Engineering and Mechanical Systems, University of São Paulo, Brazil

Abstract. *It is well known that automated systems have to be carefully evaluated to result in good specifications that lead to useful systems. Therefore, that implies that the early design phase, composed by requirement elicitation and requirements analysis should be very precise to spot contradictions, misleading conditions, lack of constraints, and other typical failing requisites which will certainly appear in the modeling and design phase.*
A proper way to deal with this problem is to capture the requirements using UML representation and find a schema to perform process analysis. In other words that means transferring semi-formal requirements in a formal schema representation. In this work we propose to use UML and Petri Net to perform requirements analysis, particularly process analysis. We will discuss how to couple information derived from different UML diagrams and how do deal with multiple instances of a specific class of element that contribute to the same process. Folding and unfolding theorems of Petri Nets are used to generate a High Level representation of the same set of diagrams and generate a further verification procedure.
Applications are very broad for this schema and a simple but realistic case study will be presented to illustrate the main ideas and procedures.

Keywords: *schema, Petri nets, requirements analysis, formal verification*

1. INTRODUCTION

After several years from its creation, UML has became the standard for the analysis and design of object-oriented systems. In 2004 Allan Zeichick (Zeichick, 2004) publish the result of an enquire among developers shown that about 2/3 of software development organizations were using UML, with 82% predicting they would use it in future (totally or partially). According Gartner Inc., UML is now used by more than 10 million IT professionals. The existence of a standard notation set has released pent-up demands and created an industry (Watson, 2008).

UML itself is a family of 13 closely-related diagramming notations [1], each one tailored to a different aspect of application design, and all defined using the Meta-Object Framework, an OMG's standard for defining modeling languages. But, UML is not only used in software analysis and design. There are also a growing family of successful Domain-Specific Languages for diverse application domains. SysML for example is a UML profile for Systems Engineering, and there are more.

UML provides a set of diagrams to model every aspect of an object-oriented application design in sufficient detail, but lacks any mechanism to rigorously check consistency between models, specially for dynamic semantics checking related to the system behavior (Engels *et al.*, 2002). Therefore, more effort could be applied nowadays on creating accurate and consistent UML models rather than implementing the design. UML class, sequence, and statechart diagrams are used in most of the existing consistency checking techniques (Usman *et al.*, 2008), but a common feature of existing dynamic semantic consistency checking techniques is to transformation original models into more formal intermediate representations for further analysis (Yao and Shatz, 2006).

Automatic code generation from UML models has emerged as a promising area in recent years. The accuracy of generated code in some ways depends on UML models consistency. Petri Nets had been also used to perform consistency checking in UML dynamic semantics (Latella *et al.*, 1999), (Saldhana and Shatz, 2000), (Baresi and Pezze, 2001), (Döll *et al.*, 2004), (Usman *et al.*, 2008). As will be shown in the remaining of this article, the use of schemas as Petri Nets could be very interesting not only to check consistency but to also be able to integrate several aspects represented by different diagrams. For the designing of automated systems, specially the class that can be treated by a problem solving paradigm, an state/transition approach would be advisable, specially if it could reflect object orientation.

Besides the growing adoption of UML, some authors also detected complains from developers basically concerned with the management of large projects. The possibility to divide main projects in components is generally claimed as an important issue to manage several development teams, what is not clearly done in UML. Also, the dynamic aspects are not fully represented by state diagrams since restriction in the evolution of states are not included in the representation. In the overall scenario, it seems that the multiple aspects represented separately by diagrams are very suitable in the very beginning, that is, just after requirements elicitation. However, but to evolve to modeling and design phases, integration

[1] There are 14 in the version 2.3

will grow more and more important. That could justify the partial use of UML which becomes even more popular among developers instead of a full use, in the entire development process.

Also, besides filling the gap between requirements analysis and modeling, Petri Nets could also be used in the validation of these same requirements. Its formal basis allow to accelerate the formalization of the design process just like SysML for a wider class of (systems) design. Therefore it could represent a depart from a simple functional approach by Use Case, which is not very suitable for mechatronic design (Doging *et al.*, 2010).

In this paper we present the basis for a new design process using UML as requirements specification language to mechatronic systems, and using Petri Nets to make requirements analysis. That implies in using property analysis, mainly reachability and Invariant Analysis to validate requirements until getting a specification. The advantage would be to use a support environment that can deal with UML modeling and also with an object-oriented hierarchical Petri Net. The folding process will be delineated, just enough to see how it could affect the analysis by using a general definition of Basic High Level Net and the theorem that guarantees that it is always possible to go from a generic classical net to a high level model and back. A case study for a deliver system taken from (Baresi and Pezze, 2001) will be presented.

2. TRANSFORMING UML SEMANTICS DIAGRAMS IN PETRI NETS

There are several proposals to deal with UML using Petri nets extensions as a formal intermediate model. In (Zhao *et al.*, 2004) three layers representing the relationship among UML diagrams were identified: the relationship among different contextual instances of the same UML diagram, the relationship among different diagrams from the same view of a system, and the relationship among various diagrams from different views of a system.

Prospective approaches in (Yao and Shatz, 2006), (Döll *et al.*, 2004), (Latella *et al.*, 1999), (Saldhana and Shatz, 2000), (Guerra and de Lara, 2003) belong to the first layer. All of them used the information coded in *Sequence*, *Activity*, and *Collaboration Diagrams* to do consistency checking.

The approach in (Baresi and Pezze, 2001) can be included in the second layer since it includes information in static diagrams in the transformation process. These kind of proposals are concerned with capturing the correct behavior in specific scenarios.

According (Zhao *et al.*, 2004), the third layer of relationship is rarely considered in the research on verification and transformation of UML models. In this paper we are concerned with the correctness of the system model based on its functional specification and in the clear need to integrate different diagram views. Therefore, our goal is to contribute in the third layer of relationship.

We illustrate the proposed approach throughout an example. Figure 1 shows the class diagram for the *Gas Station Problem* as stated in (Baresi and Pezze, 2001).

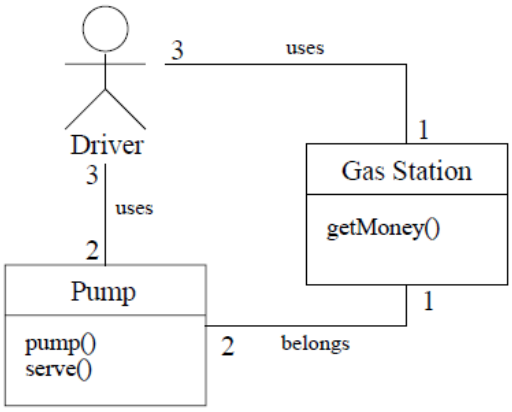


Figure 1. Class Diagram for the Gas Station Problem

Applying graph transformations (Baresi and Pezze, 2001) a Petri net model can be built from UML dynamic diagrams. Figure 2 shows the Petri net model for the use case "fueling a car". The model was edited using the **GHENeSys** Environment[2] (Silva *et al.*, 2009).

Note that Petri net in Fig. 2 is a short representation of the process of serving an unique driver, disposing of a pump. The multiple resource (and multiple demand) problem should be modeled by splitting the post-set of the place $getMoney$ to represent the use of the two different pumps. However, the process of project implied in this paper focus on the functionality of the process, followed by the proper folding in a High Level system, as it will be showed in the

[2]The GHENeSys (General Hierarchical Enhanced Net System) is a general environment proposed to unify classic, high level and extensions of Petri Nets, including object-oriented nets.

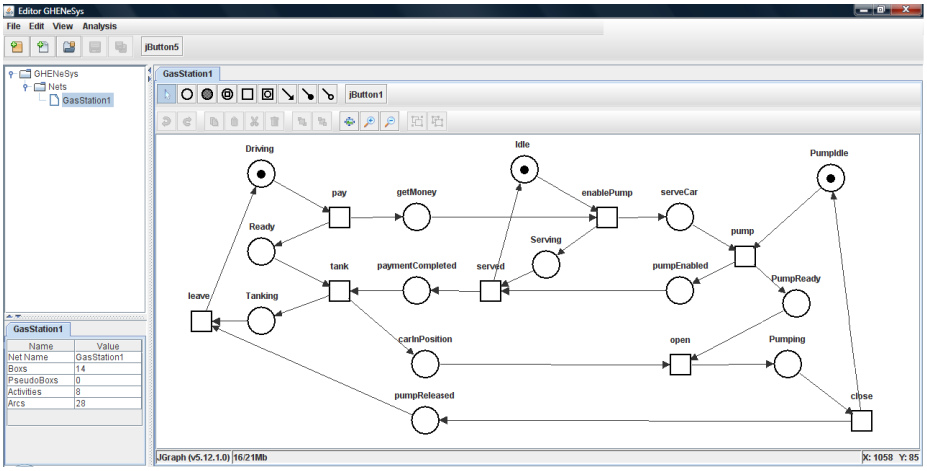


Figure 2. Petri Net model of the use case "fueling a car" for the Gas Station Problem

development. On the other hand, even with this very abstract analysis a few verification steps could be introduced, concerning the consistence of the process. The key point is that to perform the analysis, even abstract aspects represented in several UML diagrams must be collapsed in a unique representation.

Even for this simple example it is possible to notice that the Petri Net representation, synthesized from UML diagrams, can deal with the integration of activity, state, sequence, and also collaboration (not very clear in this example) diagrams. The key point is that this kind of analysis is very important to validation, and essential to the verification of system design models. In complex systems, which depend on different viewpoints (and with also a huge number of use cases), the convergence of different views to a unique system representation is also a key issue. Unfortunately this important issue could not be presented by using simple examples. Thus, we must direct such presentation to its formal aspects, starting with the transformation of informal (or semiformal) requirements on a formal representation using Petri Nets and Temporal Logic.

3. REQUIREMENTS ANALYSIS USING PETRI NET INVARIANTS AND TEMPORAL LOGICS

To be effective, system specifications must be stated as clearly and concise as possible. Temporal logics have already been used successfully in requirement elicitation (Kugler *et al.*, 2005), (Drusinsky, 2008) open a possibility to perform semantic analysis. On the other hand, Petri nets invariants has also been used in system specification (Yamalidou *et al.*, 1996) to express dynamic aspects, with the advantage of being a schematic representation, that is, suitable to a set of different artifacts and applications.

In practice, specifications are usually captured in a natural language and then formalized in some way. In this section we will try to formalize the Gas Station Problem specification using Petri nets invariants and CTL (*Computation Tree Logic*) (Clarke *et al.*, 1986), assuming all elicitation were already done in UML. Therefore, there exist specifications for each different actor in the Gas Station Problem: the drivers, the gas station and the pumps. Those specifications are quite obvious and will not be explained in detail here. Thus, we will concentrate on the requirements for the use case "fueling a car", the most appealing process, to which we had developed the formal model.

The system must ensure that a driver's car is fueled only when payment was completed. In terms of Petri net invariants such requirement can not be clearly established. When dealing with sequences, the obvious choice is Transition invariants.

Transition invariants tell us which transitions and how many times each one must be fired to complete a cycle - if there is a cycle in the net (Murata, 1989). If transitions *pay* and *open* are in all systems solutions there is a possibility that such requirement was fulfilled in the model but no one can be sure about that since nothing can be said about the partial order in which those transitions are fired to complete a cycle.

Using CTL, a branching time temporal logic, we can formalize such requirement as:

$$getMoney \longrightarrow \forall\Diamond Pumping \tag{1}$$

$$\forall\Box(getMoney \longrightarrow \forall\Diamond Pumping) \tag{2}$$

Temporal formula 1 will hold in a graph where all sequences starting with states in which atom *getMoney* is valid lead to an state in which atom *Pumping* is valid. Temporal formula 2 will hold in a graph in which sequences accepted by temporal formula 1 are always reachable from the initial state.

This is an example where a semantic statement - which could not be expressed by a schemata can be combined in a schema representation - Petri Nets - to provide a basis to perform formal verification. In the next section we analyze

specifically the verification process.

4. THE VERIFICATION PROCESS

Using the **GHENeSys** environment, state and transition invariants can be computed. Figure 3 shows the transition invariants for the Gas Station Problem which Petri Net model was depicted in Fig. 2.

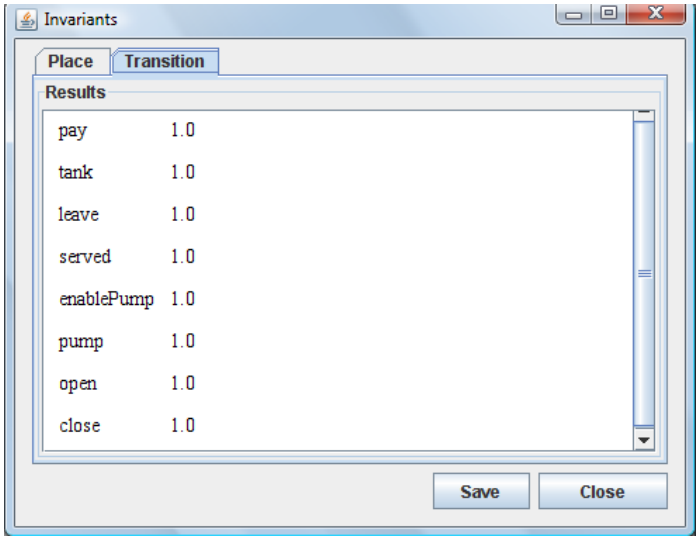


Figure 3. Transitions invariants for Petri Net model in Fig. 2

The transitions invariant vector solution indicates that all system transitions must be executed once to complete a cycle. Transitions pay and $open$ will fire in every cycle - each time an user initiate a fueling process - but formally, nothing can be said about its partial order. Since it is a simple net, an intuitive analysis would lead to the conclusion that pay will always be fired first than $open$ but such a method can not be used with large nets.

The verification process can be done with any model-checker that uses CTL as specification language and Petri nets to perform integrated and dynamical model analysis. In this paper, all results come from a model-checker developed to integrate the GHENeSys system which is a trademark of the Design Lab (D-Lab), Polytechnic School-USP (del Foyo, 2009). Figure 4 shows the results of the verification process for the formula (2) mentioned above. The green sign in the picture means that the formula were satisfied in the model.

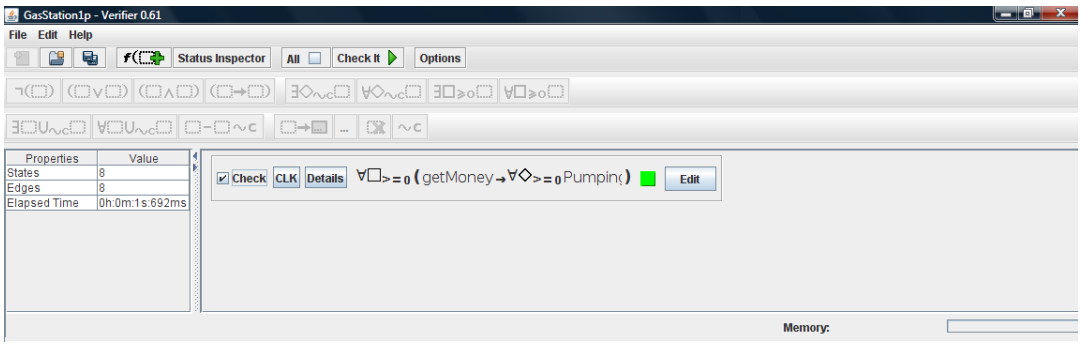


Figure 4. Verification results of the use case “fueling a car” for the Gas Station Problem

However, even when those verifications are useful, they could provide answers only about the consistency among several UML diagrams or about specific object behaviors, but not about system process correctness. Also, system behavior could depend of its interpretation, that is, on the instantiation of its parameters and internal data. Simple and direct examples for these dependence can be found in planning or scheduling domain (Vaquero *et al.*, 2007).

Back to the example in Fig. 1, we consider now three drivers, one gas station and two pumps. In order to investigate the system behavior in such instantiation a Colored Petri Nets (Jensen, 1994) were introduced [3] to deal with such problem (Baresi and Pezze, 2001).

[3]The GHENeSys system were conceived to deal with Petri Net standard, with the extensions and High Level Nets, according the ISO/IEC 15909. However, since the High Level representation is not fully implemented this part of the work were developed using CPN Tools.

Figure 5 shows the high level Petri net model for the gas station problem.

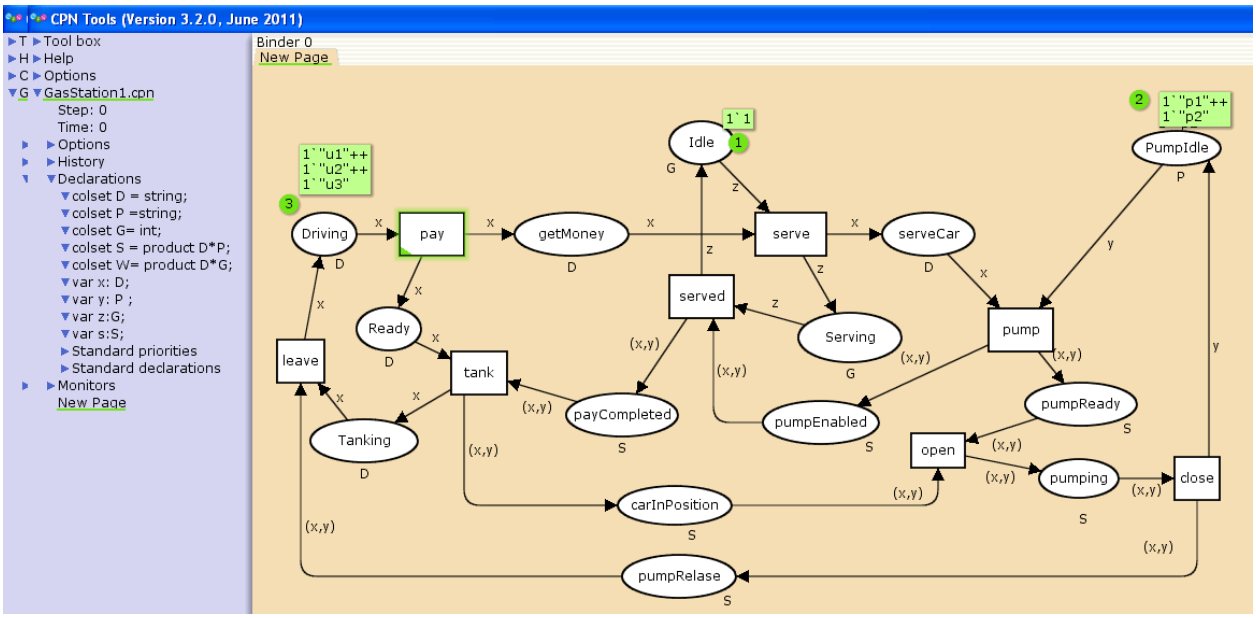


Figure 5. High Level Petri Net model for the Gas Station Problem

More sophisticate verifications can be done for this new problem. For instance, we can verify if drivers can pump a different amount of gas with respect to what they pay for. Such property was verified in (Baresi and Pezze, 2001) using simulation. However, even if simulation can detect some design errors, it is not guaranteed that this errors can be always detected, since that technique generally do not cover the entire state space. On the other hand, it is possible to pursue with the process analysis proposed here once we get close to the real system by introducing the folding of the net presented earlier to fit the multiple demand of drivers and the multiple resource of pumps.

4.1 Extending the analysis using folding and unfolding operations

There are theorems that give support to algorithms that transform Petri nets in high level nets (Jensen and Rozenberg, 1991). Therefore, it is possible to use an exhaustive method of synthesis, first generating a classic Place/Transition net (P/T net) and then explore its symmetry to - based on the mentioned algorithm - generate a high level net or the direct functional analysis proposed here. That would be very suitable for small and medium size nets. Even for large problems this approach could still be feasible, different from other methods based on the generation of a P/T net. A proper procedure for large problems should be also based on functionality, even if not so straightforward as the one presented here. Such procedure is not in the scope of the present work [4].

Colored Petri Nets (CP-nets)(Jensen, 1994) are a high level net formalism and there exists algorithms for the folding and unfolding operations to convert CP-nets into classic Petri nets (Smith, 1998). Figure 6 shows the result of the unfolding process for the high level net shown in Fig. 5. The unfolding process of the high level net in Fig. 6 yields a GHENeSys net with 47 Boxes, 42 Activities and 147 arcs.

Now the specification can be completed by using CTL formulas that verify if drivers can pumps different amounts of gas with respect to what they pay for as suggested in (Baresi and Pezze, 2001).

The following CTL formulas formalize such requirement:

$$\forall\Box(getMoney1 \longrightarrow \forall\Diamond(pumpEnabled1 \vee pumpEnabled2)) \quad (3)$$

$$\forall\Box(pumpEnabled1 \longrightarrow (\forall\Diamond Pumping1\_1 \wedge \neg\forall\Diamond Pumping1\_2)) \quad (4)$$

$$\forall\Box(pumpEnabled2 \longrightarrow (\forall\Diamond Pumping1\_2 \wedge \neg\forall\Diamond Pumping1\_1)) \quad (5)$$

Temporal formulas 3, 4 and 5 are concerned only to *Driver1*. Similar formulas must be verified for *Driver2* and *Driver3*. Figure 7 shows the results of the verification process.

[4]The difference from the current proposal to one involving large systems is that the last would demand a more intensive use of hierarchy and object-orientation, which are also features included in the GHENeSys environment.

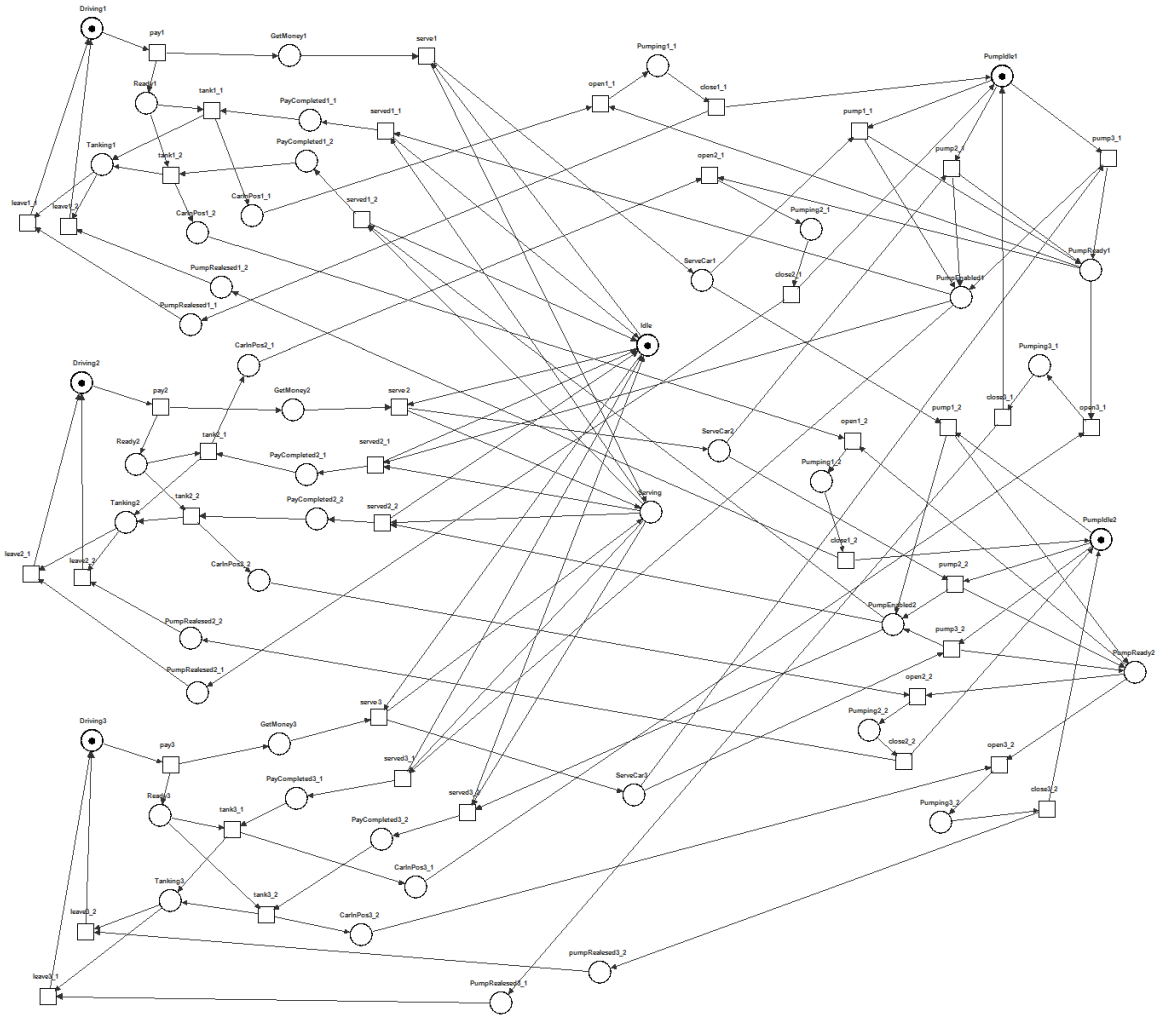


Figure 6. Petri Net model of the High Level net shown in fig. 5

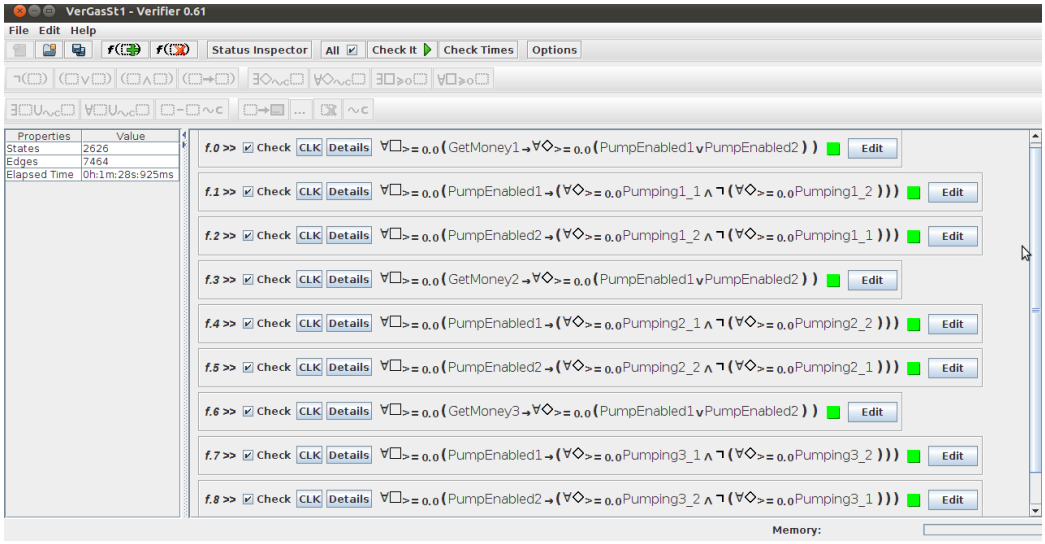


Figure 7. Verification results with the new system specification

5. TOWARDS A MORE DETAILED ANALYSIS

Despite of the enhanced specification and verification power of such approach, there are still some issues to solve in consistency checking and formal verification of systems specified by UML. So far, informations coded in diagrams of the

static structure view has not been included in the graph transformations.

Following the transformation rules proposed in (Zhao *et al.*, 2004) attributes and operations defined in the *Class Diagram* can be represented in the Petri net. However, it must be stated that, for the verifications process to be feasible, the system state space must be finite, which implies that the number of elements of the Petri net model must be finite and the net must be bounded.

Because of that, only relevant states for the verification purposes can be represented in the Petri net model. For example, real numbers are infinite then this kind of property can only be represented in sets, that can be built in such a way that those sets are relevant to the verification process.

Back to our example, the *pay* activity involves certain amount of money. That amount is irrelevant to the verification process that needs just to acknowledged for a succeeded or failed payment operation.

The high level net model that includes the payment result is shown in Fig. 8.

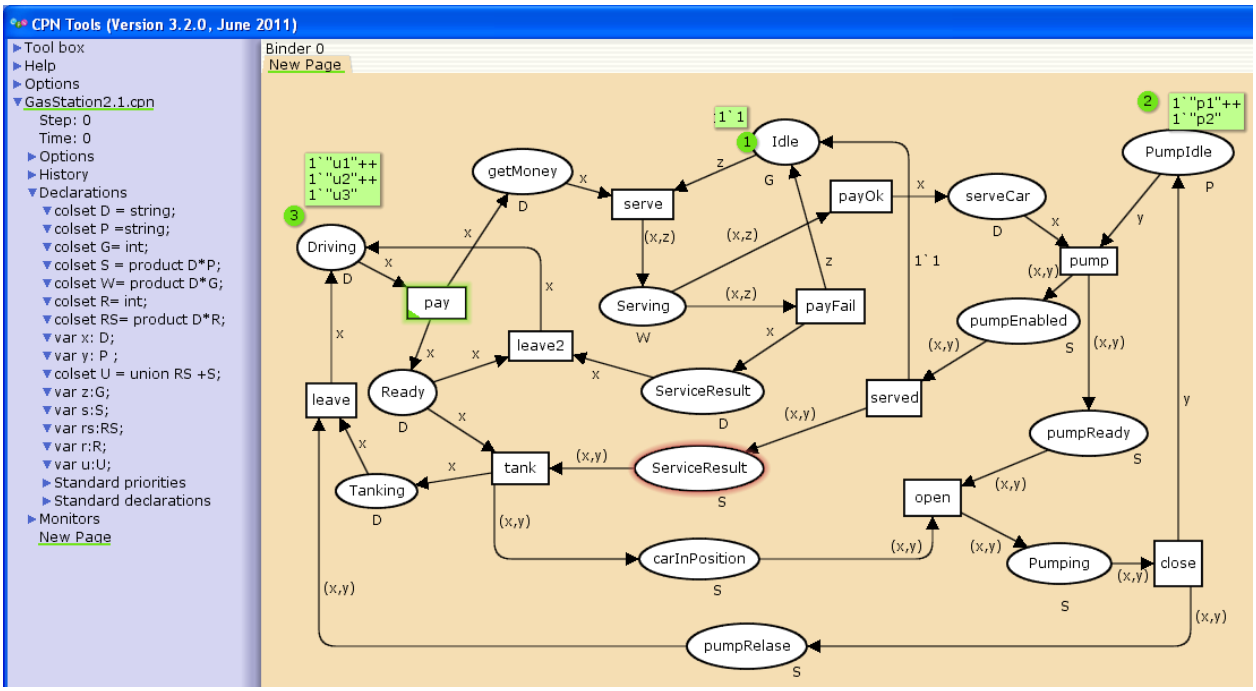


Figure 8. High level net model including payment results

Places $ServiceResult$ can contain a token of two possible types: $x \in D$ or $s \in S$. A token of type D means that the payment transaction fails for driver identified by variable x and a token of type S means that the payment transaction succeeded for driver x and an available pump y was assigned to tank the driver's car[5].

The unfolding process of the high level net in Fig. 8 yields a GHENeSys net with 50 Boxes, 51 Activities and 162 arcs. Note that the unfolding of the $ServiceResult$ box yields 3 places for each driver[6] and one of them we called $NoPaymentn$ where n is the driver id number.

Now more specifications can be checked like, if the payment fails no pump will be enabled for fueling that car. The following CTL formulas formalize such specifications:

$$\forall\Box(ServiceResult1\_1 \longrightarrow \forall\Diamond Pumping1\_1) \tag{6}$$

$$\forall\Box(ServiceResult1\_2 \longrightarrow \forall\Diamond Pumping1\_2) \tag{7}$$

$$\forall\Box(NoPayment1 \longrightarrow \neg\forall\Diamond Tanking1) \tag{8}$$

Figure 9 shows the verification results for the more detailed model.

Formulas 6 and 7 states that $Driver1$ will refuel his car in the pump enabled for him once the payment was cleared. Formula 8 states that if the payment fails this driver will not be able to refuel his/her car. Similar formulas can be stated for $Driver2$ and $Driver3$.

Regarding the statecharts of the classes in the Gas Station problem presented in (Baresi and Pezze, 2001) (see Fig. 10) our detailed model modify the driver and the Gas statecharts.

[5]Note that $s \in S$ e $S = D * P$, where $x \in D$ represent the driver and $y \in P$ represent the pump assigned to it

[6]two due to token type S (for each pump) respectively $ServiceResultn\_1$ e $ServiceResultn\_2$ plus one for token type D that we renamed as $NoPayment\_n$

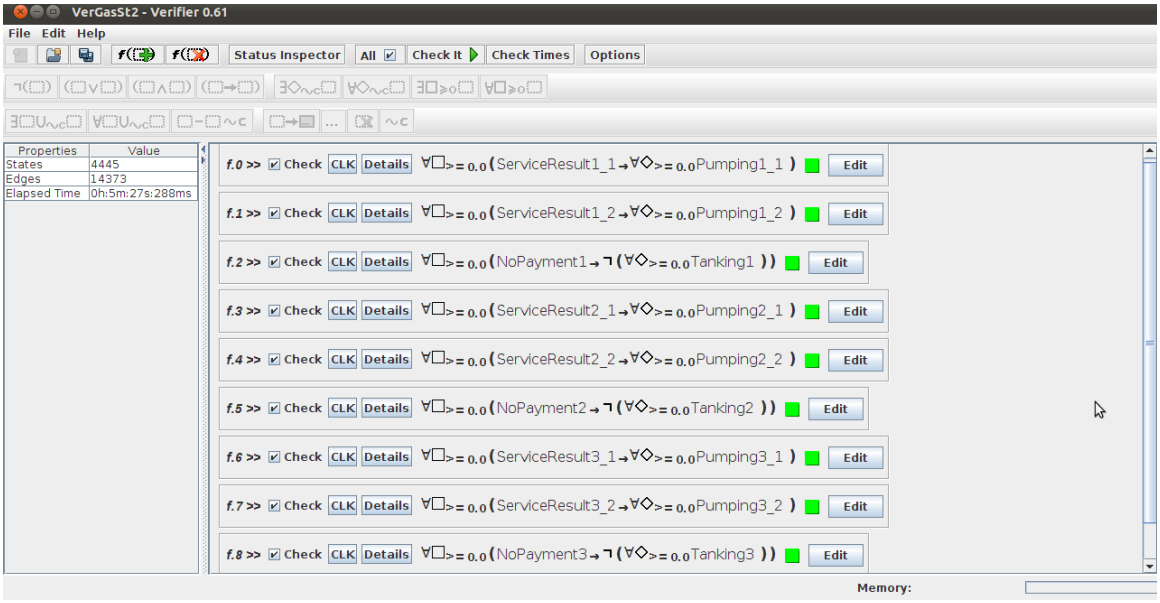
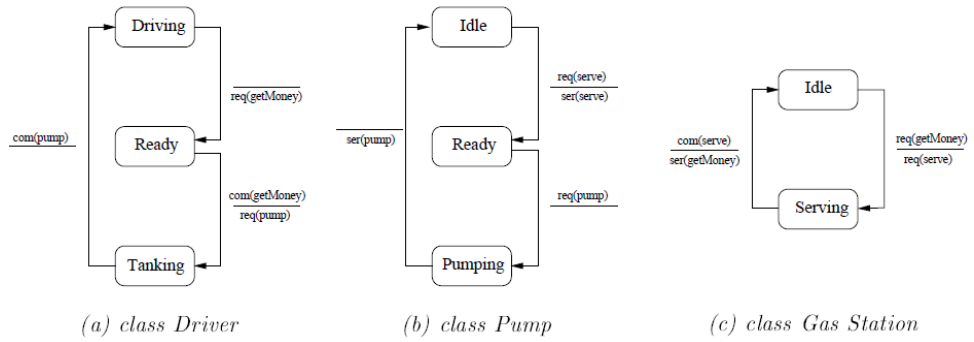


Figure 9. Verification results for the more detailed model

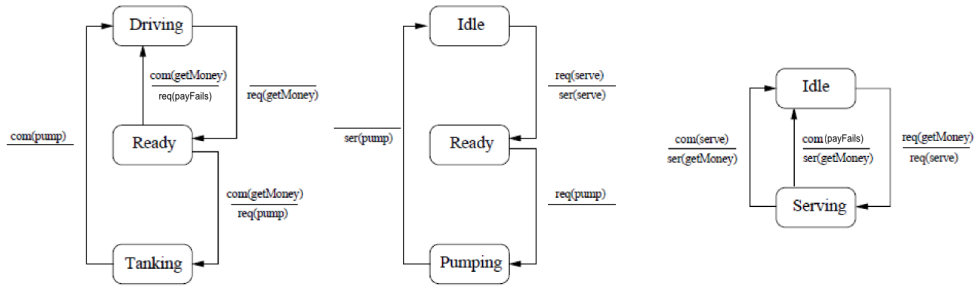


(a) class Driver *(b) class Pump* *(c) class Gas Station*

Figure 10. Statecharts for the Gas Station problem in (Baresi and Pezze, 2001)

According to the introduced modifications, the state *Ready* of the driver statechart now has two outgoing arcs, one to state *Tanking* and other to state *Driving* since the driver could leave the Gas Station without refueling.

The statechart of Gas Station class also was modified since state *Serving* has two outgoing arcs: both to state *Idle* but one requests that the class Pump the instantiation of an available pump, and the other send a message to the driver saying that the payment transaction fails and that there is no pump enabled for refueling his/her car. Figure 11 shows the modified class statecharts for the Gas Station problem.



(a) class Driver *(b) class Pump* *(c) class Gas Station*

Figure 11. Statecharts for the Gas Station problem proposed once concluded the verification process

6. CONCLUSION

This paper has shown a way to use UML and model transformations to derive an analysis model from a UML functional description. Through a simple experiment, we demonstrated suitability of the proposed technique to verify UML diagrams by transforming them into Petri nets and writing the specifications using temporal logic.

We also showed how the expressiveness of the specification language can be enhanced by a more detailed model, first including information in the *Object Diagram* - which contain the problem instantiation - and second by including relevant information of the static structure of the system.

As other approaches, the proposed technique builts the system model using a high level net formalism. Then, an unfolding operation must be executed to generate a basic Petri net that can be verified using a model-checker developed in our Lab. Since the unfolding algorithm can be included in a tool, we can ensure the correct transformation and verification of the system. The high level net construction is done by the user which implies that some errors can be introduced at this stage.

The conducted experiment shows that a more detailed model lead to an increase in the complexity of the verification process. That be observed in the size of the state space computed by the model-checker which was increased from 8, to 2626 and finally with 4445 states for the more detailed model. We claim that the system detail level will depend on the specification process necessities.

As far as we know, there are not yet an automatic transformation graph technique for constructing the system model. Those transformations still depend on the users skills, which is a problem to extend the use of the current proposal. Therefore, our attention is now directed to find facilitators and tools to enhance the synthesis of models from requirements.

7. ACKNOWLEDGEMENTS

One of the authors, Arianna Z. Olivera Salmon is partially supported by CAPES.

8. REFERENCES

Baresi, L. and Pezze, M., 2001. "Improving UML with petri nets". In *UNIGRA 2001, Uniform Approaches to Graphical Process Specification Techniques (a Satellite Event of ETAPS 2001)*. Elsevier, Vol. 44 of *Electronic Notes in Theoretical Computer Science*, pp. 107–119.

Clarke, E.M., Emerson, E.A. and Sistla, A.P., 1986. "Automatic verification of finite state concurrent systems using temporal logic specifications." *ACM transactions on Programming Languages and Systems*, Vol. 8, No. 2, pp. 244–263.

del Foyo, P.M.G., 2009. *Verificação Formal de Sistemas Discretos Distribuídos*. Ph.D. thesis, Escola Politécnica da USP.

Döll, L.M., de Souza, J.U.F. and Stadzisz, P.C., 2004. "Verificação e validação de sistemas orientados a objetos usando redes de petri". In *I Workshop de Computação - WORKCOMP-SUL*. Palhoça, SC, Brasil.

Doging, B., Evermann, J. and Parsons, J., 2010. "Understanding the adoption of use case narratives in unified modeling language". In *Proc. of the First Int. Conference on Information Systems*. St. Louis, USA.

Drusinsky, D., 2008. "From uml activity diagrams to specification requirements". In *Conference on System of Systems Engineering, 2008.* IEEE, Singapore.

Engels, G., Hausmann, J., Heckel, R. and Sauer, S., 2002. "Testing the consistency of dynamic uml diagrams". *Integrated Design and Process Technology*.

Guerra, E. and de Lara, J., 2003. "A framework for the verification of uml models. examples using petri nets". In *Proceedings of JISBD'03*.

Jensen, K., 1994. *Coloured Petri Nets: basic concepts, analysis methods and practical use*, Vol. 2: Analysis Methods of *Monographs in Theoretical Computer Science*. Springer, Berlin.

Jensen, K. and Rozenberg, G., eds., 1991. *High-level Petri Nets. Theory and Application*. Springer-Verlag.

Kugler, H., Harel, D., Pnueli, A., Lu, Y. and Bontemps, Y., 2005. "Temporal logic for scenario-based specifications". In N. Halbwachs and L.D. Zuck, eds., *Lecture Notes in Computer Science*. Springer, Vol. 3440, p. 445Ű460.

Latella, D., Majzik, I. and Massink, M., 1999. "Automatic verification of a behavioural subset of uml statechart diagrams using the spin model-checker". *Formal Aspects of Computing*, Vol. 11, No. 6, Springer-Verlag.

Murata, T., 1989. "Petri nets: Properties, analysis and applications". *Proceedings of the IEEE*, Vol. 77, No. 4, pp. 541–580.

Saldhana, J.A. and Shatz, S.M., 2000. "Uml diagrams to object petri net models: An approach for modeling and analysis". In *Twelfth International Conference on Software Engineering and Knowledge Engineering*. Chicago, IL, USA.

Silva, J.R., Miralles, J.A.S.P., Salmon, A.O. and del Foyo, P.M.G., 2009. "Introducing object orientation in unified petri net approach". In ABCM, ed., *Proceedings of COBEM 2009*. ABCM, Gramado, RS, Brazil.

Smith, E., 1998. "Principles of high level net theory". *Lecture Notes in Computer Science*, Springer.

Usman, M., Nadeem, A., hoon Kim, T. and suk Cho, E., 2008. "A survey of consistency checking tech-

niques for uml models". *Advanced Software Engineering and Its Applications*, Vol. 0, pp. 57–62. doi: http://doi.ieeecomputersociety.org/10.1109/ASEA.2008.40.

Vaquero, T.S., Romero, V., Tonidandel, F. and Silva, J.R., 2007. "itsimple 2.0: An integrated tool for designing planning domains". In *18th International Conference on Automated Planning and Scheduling*. Providence, Rhode Island.

Watson, A., 2008. "UML vs. DSLs: A false dichotomy". Technical Report 08-08-03, Object Managemet Group.

Yamalidou, K., Moody, J., Lemmon, M. and Antsaklis, P., 1996. "Feedback control of petri nets based on place invariants". *Automatica*, Vol. 32, pp. 15–28. ISSN 0005-1098.

Yao, S. and Shatz, S.M., 2006. "Consistency checking of UML dynamic models based on petri net techniques." In *Proceedings of the 15th International Conference on Computer*. IEEE Computer Society, IEEE, Washington DC, USA, pp. 289–297.

Zeichick, A., 2004. "UML adoption making strong progress". *SD Times*.

Zhao, Y., Fan, Y., Bai, X., Wang, Y., Cai, H. and Ding, W., 2004. "Towards formal verification of UML diagrams based on graph transformation". In *Proceedings of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business*. IEEE Computer Socity.

9. Responsibility notice

The authors are the only responsible for the printed material included in this paper