REQUIREMENTS ENGINEERING AT A GLANCE: COMPARING GORE AND UML METHODS IN THE DESIGN OF AUTOMATED SYSTEMS

Javier M. Silva*, Arianna Z. O. Salmon†, Pedro M. G. del Foyo†, José Reinaldo Silva*

*D-Lab, Escola Politécnica
Universidade de São Paulo
São Paulo, SP, Brazil

†Depto. de Eng. Mecânica
Universidade Federal de Pernambuco
Recife, PE, Brazil

jsilva@silva.com, azolivera@gmail.com, pedro.foyo@ufpe.br, reinaldo@usp.br

Abstract— The early phase of Requirements Engineering is crucial to design intelligent automated systems, especially if an analytic formal solutions are not achieved. In such cases Requirement analysis is not feasible and the design process will be more suitable do re-work. Planning and scheduling problems can be approached with Artificial Intelligence (AI) and constitute an important area of machine intelligence - together with machine learning. Therefore, it will be an important result to propose good design practices to solve planning problems, also giving special attention to the early requirements phase. Specifically, it would look for a method that lead to formal methods of requirements analysis using Petri Nets (which is also used to direct solutions that do not use AI). In this work it is compared two direct approaches to requirements engineering: A semi-formal modeling using Unified Modeling Language (UML) that transform the diagrams to Petri Nets to perform analysis, and another approach that uses Goal-Oriented Requirement Engineering (GORE) method, represented by KAOS diagrams that could also be converted to Petri Nets. The possibility to use a unified Petri Net environment and introduce some extensions such as hierarchy is also envisaged.

Keywords— Petri Net, Requirements Engineering, Intelligent Automated Planning, GORE, KAOS, UML

Resumo— A fase de Engenharia de Requisitos é crucial no Design de Sistemas automatizados inteligentes, especialmente quando não é possível formular soluções formais analaticamente. Nesses casos, a análise de requisitos não é viável e o processo de design será o mais adequado para o re-uso no processo. Os problemas de planning e scheduling podem ser abordados com técnicas de Inteligência Artificial (IA) e constituem uma área importante da machine intelligence -conjuntamente com o machine learning. Portanto, será um resultado importante na proposta de boas práticas no design para a solução de problemas de planejamento, oferecendo também especial atenção à fase inicial de requisitos. Escificamente, procuraríamos um método que conduzesse até técnicas formais de análise de requisitos usando redes de Petri (que é usado também para direcionar soluções que não usam técnicas de IA). Neste trabalho, comparamos duas abordagens relacionadas com a Engenharia de Requisitos: uma modelagem semi-formal usando Linguagem de Modelagem Unificada (UML) que transforma os diagramas para redes de Petri para realizar análises, com uma outra abordagem que usa método de Engenharia de Requisitos Orientado a Objetivos (GORE) representados por diagramas KAOS que também podem ser convertidos em redes de Petri. A possibilidade de usar um ambiente de Petri Net unificado e introduzir algumas extensões como hierarquia também está prevista.

Palavras-chave— Redes de Petri, Engenharia de Requisitos, Planejamento Automático Inteligente, GORE, KAOS, UML

1 Introduction

Extracting knowledge related to planning domain may become a difficult task, specially in what concerns the solution of real problems. Analyzing requirements is a good challenge to any target for Engineer since it normally deals with the lack of knowledge about the artifact (be it a a product, process or service). In the case of automated process and/or service requirement analysis becomes really hard, which is just the focus of the present work. Besides, there is an obstacle to determine consistency at this stage, and it is needed to use different representations (and to change from one to another) without losing the focus of the problem [1].

Thus, it is attractive to research in Engineering Design to focus on the early phase of design where - specially to innovation process - very few is known about the problem until its complete solution. The target is to seek for a formal representation, which eventually allows a transference to other formal representations in the process that follows requirement analysis. This is the main goal of the Knowledge Engineering tool applied to Automatic Planning called itSIMPLE, now in its version 4.0 (Vaquero et al., 2013).

Recent research show the relevance of initial stages in the design of almost any application (Knauss et al., 2018) (Alreshidi et al., 2018)(Liebel et al., 2018)(Silva et al., 2018). Requirements modeling and analysis is a key phase whose influence on the success of the remaining phases is sufficient

[1] Most of the software tools that support the early design phase are based on "requirements management" which cover the elicitation support, documentation, some link between requirements, and, more recently, traceability tools, but do not include consistency analysis

to lead to a good progress or to the failing of the project, that is, a good requirement specification is necessary (but not sufficient) condition to obtain a good design process.

The main discussion is about improvements in the design of automated systems that could enhance the modeling and analysis of requirements, and can still lead to good results. However, when are treat problems with high complexity or extensions of real problems, it is necessary to know better the characteristics of the problem and the context in which it is inserted.

Consequently, the representation of the initial phase, and the possibility to evolve from a semi-formal version requirements to a formal one is a very important issue. Initial elicited requirements are normally developed in UML, which could take advantage of the recent SysML representation to convert visual diagrams into formal representation (or to Petri Nets) (Salmon et al., 2011). An alternative approach is to use goal oriented requirements engineering and use LTL (or Petri Nets) (Martinez and Silva, 2015). In this article it is considered advantages and disadvantages of each one.

Section 2 focus on the use of formal methods to requirement analysis using GORE approach and KAOS diagrams. Section 3 will show some aspects about Petri Nets as a formalism for analyzing requirements of automated systems. In the next sections, is introduced a case study for a realistic problem concerning the logistic of gas station which is modeled with both *KAOS-Petri Nets* and *UML-Petri Nets* approaches, including invariants calculus in both cases. Finally, is analyzed both process comparing and adding some concluding remarks and contributions to further work.

2 GORE: Goal Oriented Requirements Methods

For a comprehensive understanding of how a system fits requirements, let us first consider that eliciting requirements lead to address both to functional and non-functional requirements. The first is more intuitive and is generally associated with services to be provided to clients while the second is related to the quality, sometimes performance or resources needed by the service and also related to external demands, such as safety requirements, performance, security, legal aspects, etc. Here, a requirement is a necessary condition to achieve a certain goal in a specific application domain (Horkoff et al., 2017).

Therefore, since in the direct approach non-functional requirements are in general neglected or fail to compose a complete set, Goal-Oriented methods are becoming an interesting alternative up to large systems. We will shortly describe the essential notion to this approach and its diagrammatic representation, called KAOS (from Knowledge Engineering Object System, or, more informally, "Keep All Objectives Satisfied") (Dardenne et al., 1993).

2.1 The KAOS Method

The method involves four semantic networks expressing conceptual models linked to objectives, agents, objects, and operations covering expected and target results (objectives), the responsibility to achieve such goals (from agents that could be machines or humans), an static object representation and a dynamic set of operations that change system state. Initial representation is semi-formal as any visual diagram, but it could converge rapidly to a disciplined and finally to a formal representation in linear temporal logic (LTL).

Therefore the modeling process is divided in two steps: i) composing a graphical representation and then synthesizing an internal representation to requirements using LTL(Cailliau and van Lamsweerde, 2015).

However, this approach is suitable only to small or even medium size projects where it is affordable to represent each process as unique sequence disregarding concurrent events. In practice, even to a small system as we will introduce in the example, concurrence is a key issue to automation and all change of state are associated to a combination of events with different pre and post conditions. That is the motivation to look for an alternative representation such as Petri Nets. Before that we will show the basis for KAOS diagrammatic and formal representation.

2.2 KAOS Graphic Representation

The goal diagram is represented by a tree in which all nodes represent goals and the edges represent relations (such as composition, refinement, dependency, restriction, etc.). The main goal (the tree's root), is an abstraction of the system - graphically represented by a parallelogram - related to the main goal. Fig. 1 show the basic elements for KAOS diagrams.

Parallelograms are used for goals, requirements (necessary conditions to achieve a goal), and also expectations (very important to traceability). To stress the difference expectations are always filled with yellow color.

The main difference between a KAOS diagram and UML is just the fact that while one should integrate four diagrams (always) in this representation. The same representation in UML would require a previous of one among thirteen structural diagrams and more twelve behavior diagram to come up with the proper set (maybe re-

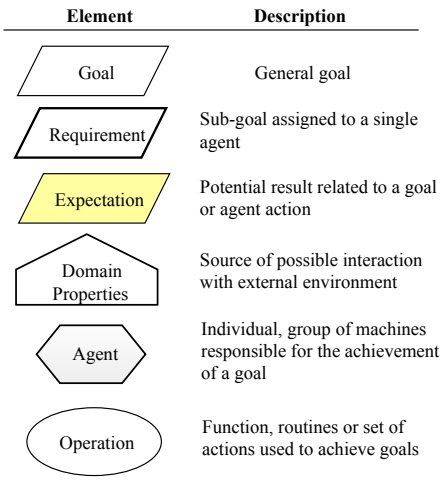


Figure 1: Main elements of Goal Diagram

dundant) to show a diagram with the same semantic.

Concerning the formal approach UML can be converted into SysML which also has a non-empty intersection with the visual diagrams (Freedenthal). In the case of Goal Oriented approach a more straightforward conversion could be applied to KAOS diagrams to convert it to LTL.

2.3 Formal Representation

A goal may be described as a valid final state, derived from the general behavior of the system. Separately each sub-goals can emerge from different course of actions but converge to the main goal. Such behaviors can be represented as paths in a graph or by as a combination of different automata. The formal representation prescribed by KAOS method is based in LTL, but it could also be represented by an state-transition formal representation.

A transition could be represented formally in terms of LTL sentences such as:

$$C \Rightarrow \Theta T$$

where C is a current condition, T is a target condition and Θ is one of the LTL operators represented in Table1.

Table 1: Temporal Logic Operators

Operator	Description
$\bigcirc$	In the next state
$\Diamond$	Eventually in the future
$\Box$	Always in the future
$\mho_d \leqslant$	Hold until d is true

Requirement Engineering tools such as Objectiver can help to formalize requirement specifications that come from KAOS diagrams into LTL formulas. However, such formulas specify each process and it is not so easy to express distributed dynamics. In order to face that several works propose the translation of LTL representation to Petri Nets (Lacerda and Lima, 2011)(Fahland, 2007)(Martinez and Silva, 2015).

3 Using Petri Nets to Analyze Requirements

There are different proposals to use of Petri Nets in requirements analysis, since using direct Place/Transition (P/T) Nets (Fahland, 2007), to the use of extended nets (DeVries, 2013)(Martinez and Silva, 2015), up to coloured nets.

In this work we will use a direct approach based on P/T nets to fit the comparison with a similar approach using UML (Salmon et al., 2011)(Salmon et al., 2014).

To save some space we will refrained from going into basic details of Petri Nets theory, which could be found in known references (Murata, 1989). The use of Petri Nets to formal verification and model checking is also a known topic (Silva and del Foyo, 2012)(Salmon, 2017).

It is worth to mention that in this work it will used an unified Petri Net environment that could synthesize both P/T and High Level Nets, as well as extensions such as hierarchical nets, tagged nets, inhibitor gates and so on. That was encapsulated in an environment called GHENeSys (del Foyo, 2001)(Silva and del Foyo, 2012). However, any other Petri Net environment could be used.

In the following it will presented the target problem under which both methods UML-Petri Nets and KAOS-Petri Nets would be analyzed and compared.

4 The gas station problem

The gas station is a simple problem concerning service and workflow, where the target system is a gas station with two pumps. The furnishing of gas is coordinated by a cabin with an operator (initially supposed a human for simplicity).

A driver who arrives at the station should request and pay for the gas and will have the requested service assigned to a pump with a QR code. The model could be extended to more services (such as oil exchange, washing, changing filters, etc., configuring an automatic or semi-automatic service). If some intelligence planning is inserted, the human operator could also be replaced by an automatic system using a cell phone application and a credit card. For the time being let us deal with the simple system operated by humans.

4.1 Modeling Requirements with KAOS

For this case study the primary objective is: *A customer should be served as soon as possible.* This primary goal depends on subgoals such as: *operator process driver request and payment* and *control admits a new driver*, where this last goal launches the service process. The first objective relies on the expectation - from the stakeholder - that the customer external queue is kept with a minimal number of markings. Of course that depends on the number of pumps and space available to receive drivers.

The assignment of a pump (to an specific driver) is based on the expectation that the customer will run the service of fueling the car for his/her own, using a card with a QR code identifying the driver and the service already paid. The pump can recognize the service and provide the proper amount of oil. Another automatic operation will control customer external queue and will admit a new driver as soon as a "space" is available (by opening a cancel, for instance).

The Goal Diagram for the Gas Station Problem is shown in Figure 2.

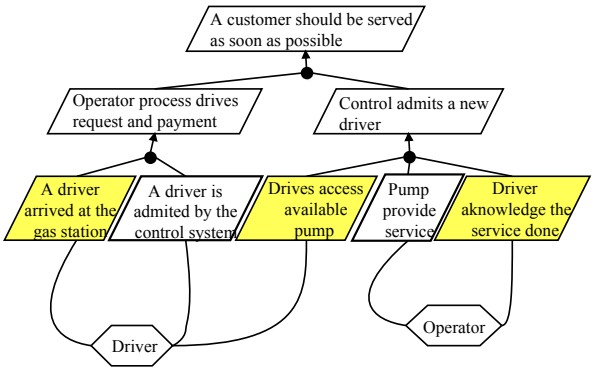


Figure 2: Goal Diagram of Gas Station Problem

Table 2 shows LTL sentences associated with each goal.

Figure 3 shows the P/T net that results from the application of a translation algorithm proposed in (Martinez and Silva, 2015) which synthesizes a Petri Net from the KAOS Diagram.

Normally, for analyses purposes, a place should be added between transitions **leave_GS** and **drv_arrives** making the net cyclic, but this is not necessary in the current model and it has a regular open system where the service itself is already a closed sub-net. Transitions invariants can obtained using conventional algorithms which reveals that once a diver is admitted he/she could be served by one of the available "places" (waiting for the operator, requesting and paying or using one of the pumps). This loop denotes the service provided by the Gas Station.

Place invariants are shown in Table 3. Invariants equation 1 and 3 show that pumps can

Table 2: LTL sentences associated to each goal of Goal Model

Goal	LTL Sentences
A driver is admited by the control system.	$\exists(d$:Driver, gs :GasStation); **arrives**$(d) \wedge$ **IdleVacancy**$\rightarrow$ $\Diamond$ **insert**(d, q);
Operator process driver request and payment.	$\exists(d$:Driver, op :Operator, $c : card$, $s : gasService$); **request**(d,s) $\wedge$ **pay**$(d,op) \rightarrow \Diamond$ **ready**(d,s,c);
Driver access available pump.	$\exists$ $(d : Driver, p : Pump, c : card,$ $s : GasService)$; **ready**$(d, s, c) \wedge$ **PX_free**$(p) \rightarrow \Diamond$ **assigned**(d,p);
Driver and service are received by the pump.	$\exists$ $(d : Driver, p : Pump, c : card,$ $s : GasService)$; **assigned**$(d, p) \rightarrow$ $\Diamond$ **identify**(p,c,s);
Pump provide the service	$\exists(d$: Driver, p:Pump, $s : GasService)$; **identified**$(p,c,s) \rightarrow$ $\Diamond$ [**provided**$(s) \wedge$ **acknowledged**(d) $\wedge$ **IdleVacancy** $\wedge$ **PX_free**(p)];
Driver leaves Gas Station.	$\exists(d$: Driver, gs:GasStation); **acknowledged**$(d) \rightarrow \Box$ **leave**(d, gs);

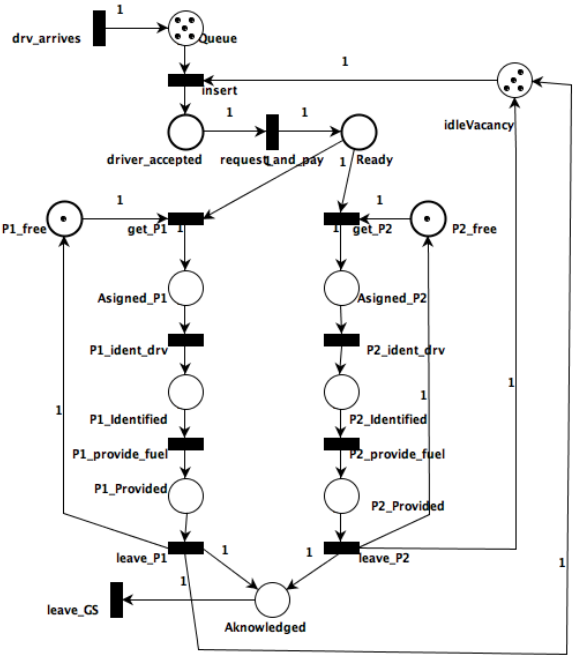


Figure 3: Petri Net of Station Gas Problem from KAOS Method.

be either free or serving an assigned driver. The second equation stands for the Gas station maximum number of accepted drivers from the external queue (M(IdleVacancy)=4) to maximize the number of drivers attended simultaneously.

It is possible to see that each invariant equation is related to some goal (directly or indirectly), which turns the workflow analysis easier. It should also be pointed that in many cases the Petri net representation is derived from LTL equations. In

cases like that, the number of objects (pumps for instance) would not appear easily and conflicts such as the one showed in the net in Fig 3 do not appear explicitly. In this work the Petri net is synthesized from the KAOS diagram (Martinez and Silva, 2015) instead of the LTL (Lacerda and Lima, 2011)(Fahland, 2007), and the object diagram is used to provide the number of objects.

In the following a UML-Petri Net approach is introduced to analyze the same problem.

4.2 Modeling Requirements using UML-Petri Nets

The first step would be to elicit and represent system requirements using UML class and state diagrams. UML has a high power graphic expression, but despite this, restrictions for the systems behavior (specially when related to the domain environment) are not adequately expressed in UML diagrams.Extension mechanisms such as stereotypes, tagged values and predefined constraints are not enough to express such constraints.

For this reason, it will used OCL to formulate system constraints, which would be primarily represented by invariants(Salmon et al., 2014). After the UML modeling the result will be transformed into a classic Petri net. Transformation algorithm (Salmon, 2017) will be an enhancement of the the one proposed by Baresi(Baresi and Pezzè, 2001).

Figure 4 shows the class diagrams for the Gas Station Problem.

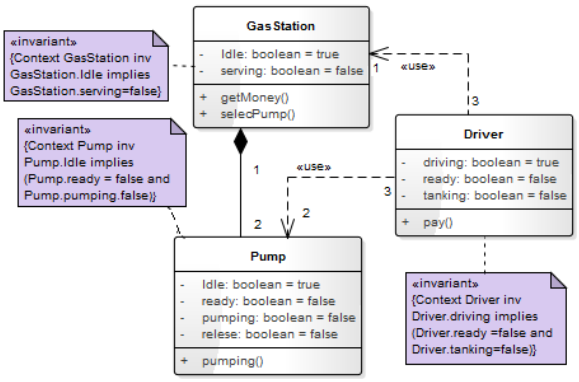


Figure 4: Class diagram of the Gas Station Problem

Table 3: Place invariants derived from the KAOS approach.

Id	**Invariants equations**
1	$M(P2\_free) + M(Asigned\_P2) + M(P2\_Identified) + M(P2\_Provided) = 1$
2	$M(Asigned\_P1) + M(P1\_Identified) + M(idleVacancy) + M(P1\_Provided) + M(driver\_accepted) + M(Ready) + M(Asigned\_P2) + M(P2\_Identified) + M(P2\_Provided) = 4$
3	$M(Asigned\_P1) + M(P1\_Identified) + M(P1\_Provided) + M(P1\_free) = 1$

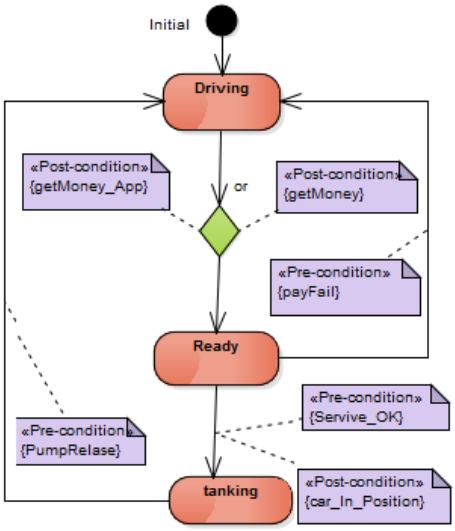


Figure 5: State diagram of the class *Driver*

Based on the class diagram shown in Figure 4, an state diagrams is constructed. Figures 5, 6 and 7 show the state diagrams for the class *Driver*, *Gas Station* and *Pump* respectively.

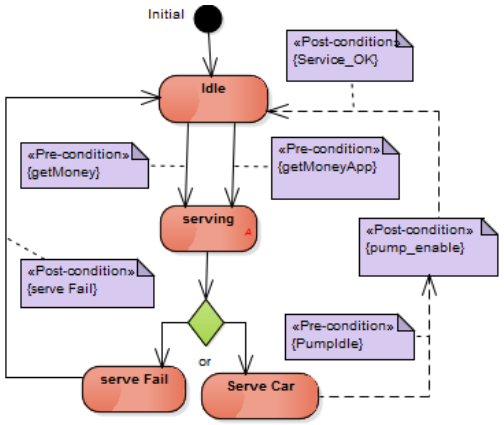


Figure 6: State diagram of the class *Gas Station*

State diagrams are transformed into a classic Petri net (Salmon, 2017). Figure 8 shows the Petri net for the Gas Station Problem.

In the following invariants are used in the verification of the model obtained. Place and transition invariants represent the conservative components of the net. Transition invariants will denote which transitions must fire and how many times each one should fire, so that the initial marking is reproduced. These invariants represent the repetitive components of the net.

Place invariants are used both in the representation and verification of system requirements(Salmon et al., 2014). To do so, the information presented in the class diagram shown in Figure 4 is also considered, besides the specifications described in Ontology Constraint Language - OCL (Kalibatiene and

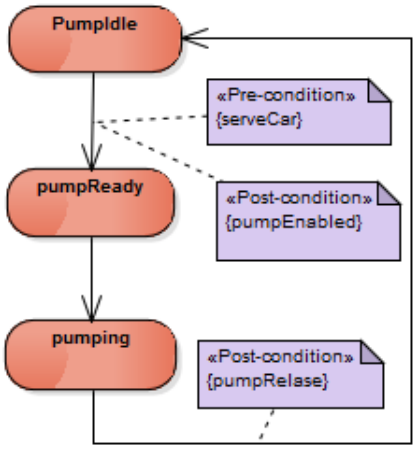


Figure 7: State diagram of the class *Pump*

Vasilecas, 2012). All system requirements can be written as a sum of elements of the marking vector.

Table 4 shows constraints in OCL for the Gas Station Problem, whereas in tables 5 respective invariants are defined.

The accuracy of equations in Table 5 can be verified by first computing the invariants and verifying thereafter whether the set of places of each inequality belongs to some vector in the solution set of Petri net place invariants. Using the generator set obtained it is possible to calculate the invariants shown in Figure 9. This set of invariants coincide with the equations defined in Table 5. This demonstrates that all equations described in these tables are true and therefore the system should meet the desired requirements.

Notice that equation 1 of table 5 is a subset of equations that form the invariant 1 of Figure 9 , and so the rest of the invariants respectively. The invariants places we want to verify are painted yellow in Figure 9.

Figure 10 shows the transition invariants for

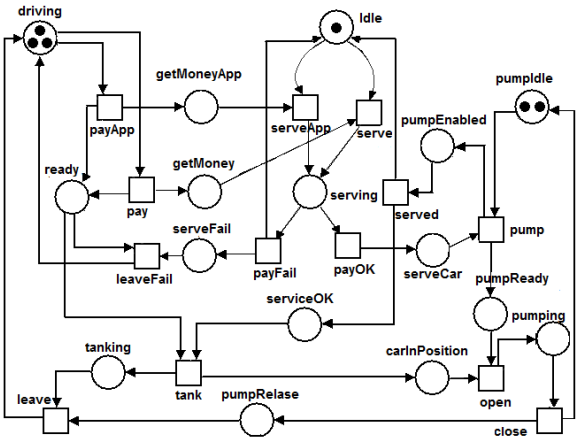


Figure 8: Petri net model of the Gas Station Problem from UML-Petri Nets method

Table 4: OCL specification corresponding to the Gas Station Problem Gas Station Problem

context invariants	
Id	OCL specification
1	(Driver.driving) **or** (Driver.ready) **or** (Driver.tanking)
2	(GasStation.Idle) **or** (GasStation.serving)
3	(Driver.payApp) **or** (Driver.payMoney)
4	(GasStation.serveOk) **or** (GasStation.serveFail)
5	(Pump.Idle) **or** (Pump.ready) **or** (Pump.pumping)
6	(Driver.driving) **or** (Driver.ready) **or** (Driver.carInPosition) **or** (Pump.pumping)**or** (Pump.release)

Table 5: Place invariants for the Gas Station Problem from UML-Petri Nets

Id	**Invariants equations**
1	$M(driving) + M(ready) + M(tanking) = 3$
2	$M(Idle) + M(serving) \leq 1$
3	$M(getMoneyApp) + M(getMoney)) \leq 1$
4	$M(serveOk) + M(serveFail) \leq 1$
5	$M(pumpIdle) + M(pumpReady) + M(pumping) = 2$
6	$M(driving) + M(ready) + (carInPosition) + M(pumping) + M(pumpRelease) \leq 1$

Transition | Place

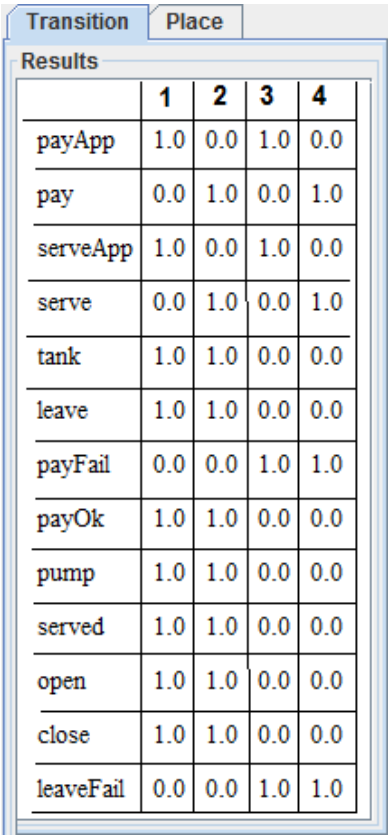
Results

	1	2	3	4	5	6
driving	1.0	0.0	1.0	1.0	0.0	1.0
getMoneyApp	0.0	0.0	1.0	1.0	0.0	0.0
getMoney	0.0	0.0	1.0	1.0	0.0	0.0
Idle	0.0	1.0	0.0	0.0	1.0	0.0
serving	0.0	1.0	1.0	1.0	1.0	0.0
ready	1.0	0.0	0.0	0.0	0.0	1.0
tanking	1.0	0.0	1.0	1.0	0.0	0.0
serveFail	0.0	0.0	1.0	1.0	0.0	0.0
pumpIdle	0.0	0.0	0.0	1.0	1.0	0.0
pumpEnabled	0.0	1.0	1.0	1.0	1.0	0.0
carInPosition	0.0	0.0	0.0	0.0	0.0	1.0
pumpReady	0.0	0.0	0.0	1.0	1.0	0.0
serveCar	0.0	1.0	1.0	1.0	1.0	0.0
serveOK	0.0	0.0	1.0	1.0	0.0	0.0
pumping	0.0	0.0	0.0	1.0	1.0	1.0
pumpRelease	0.0	0.0	0.0	0.0	0.0	1.0

Figure 9: Place Invariants of Petri net corresponding in Figure 8.

the Gas Station Problem which Petri Net model was depicted in Figure 8. The transitions invariant vector solution indicates which transitions must be executed in order to complete a cycle.

Transition invariant 1, shown in Figure 10,



Transition | Place

Results

	1	2	3	4
payApp	1.0	0.0	1.0	0.0
pay	0.0	1.0	0.0	1.0
serveApp	1.0	0.0	1.0	0.0
serve	0.0	1.0	0.0	1.0
tank	1.0	1.0	0.0	0.0
leave	1.0	1.0	0.0	0.0
payFail	0.0	0.0	1.0	1.0
payOk	1.0	1.0	0.0	0.0
pump	1.0	1.0	0.0	0.0
served	1.0	1.0	0.0	0.0
open	1.0	1.0	0.0	0.0
close	1.0	1.0	0.0	0.0
leaveFail	0.0	0.0	1.0	1.0

Figure 10: Transition Invariants of Petri net corresponding to the Gas Station Problem: figure 8.

means that if the driver pays using a mobile application and the payment is confirmed, then the actions corresponding to the pumping service will be performed, and the driver leaves the Gas station. Transition invariant 2 has a similar meaning with a difference that the payment is made using credit card. Transition invariants 3 and 4 represent the situation in which the payment (made with a mobile application -invariant 3, or made with credit card - invariant 4) is not confirmed. In both cases the pumping service cannot be performed, and the driver should leave the gas station.

Therefore, this set of transition invariants allows the verification of four situations that may arise in the system: (1) the driver pays using a mobile application and the payment is not confirmed, then the driver leaves without filling the car; (2) the driver pays using the mobile application and the payment is confirmed, then the driver can filling the car; (3) the driver pays using the credit card and the payment is not confirmed, then the driver leaves without filling the car; (4) the driver pays using the credit card and the payment is confirmed, then the driver can filling the car.

5 Conclusions

This paper shows a very similar problem treated with two different approaches: one based on Goal Oriented Requirements Engineering, where elicited requirements are represented by KAOS diagrams and are translated to classic Petri Nets, and another, based initially in UML (2.1) diagrams and also followed by a translation to classic Petri Nets. In the second case some attributes of automation were inserted such as the way of payment (by a mobile application or by credit card) while those features were omitted in the first approach. However the important result do compare is the workflow of the design process its analysis, as well as possible advantages concerning the Requirements Engineering of automated systems.

Let us start considering as a criterion the facility to synthesize the Petri Net, which is practically the same if class diagrams are considered in the UML based approach (object diagrams are default in KAOS modeling). That will assure the cardinality of objects can be considered in both cases. Second, another important criterion concerning innovation is requirements traceability, where the KAOS approach take some advantage since responsibility diagram explicitly connects goals to agents. On the other hand, the proposed UML approach (Salmon et al., 2014)(Salmon, 2017) takes some advantage by including explicit restrictions raised from the interaction between the target system and its environment. That was not particularly critical in the case study showed here but should be important in critical automated systems. KAOS modeling can also include restrictions using the representation of conflicts between goals or sub-goals, however it is not possible to configure explicitly in KAOS those points where system and environment interact.

The Gas Station problem could be incremented with more automated features (without changing the main goals) and that requirements evolution seams to be easier in KAOs modeling. The target example could evolve to a timed problem taking in count the time a drives waits to be served, the time spent by the operator to receive the order and payment (or automating this process with mobile applications and credit cards, as in the UML approach); including the time necessary to provide the service, and for the drive to leave the Gas Station liberating space. We could use either fixed time intervals or realistic real intervals (Silva and del Foyo, 2012). Classic Petri Nets synthesis could lead to model checking verification in both cases.

Finally, the fact that in KAOS modeling we cannot be concerned with non-functional requirements is again an advantage to KAOS approach. Of course, the introduction of SysML as a formalization language (similarly to the use of LTL in KAOS approach) should enhance the appeal to use the UML approach, since the formalization process should be smoother than the synthesis of LTL equations. On the other hand we should start

with a bigger number of UML diagrams (not just state and class diagrams) and the problem of selecting a proper set should be considered.

Further work points to explore the definition of a minimal set of UML diagrams or to the enhance Petri Nets synthesis from KAOS diagrams, and include model checking in both approaches. That would lead to a more detailed comparison between these two approaches.

References

Alreshidi, E., Mourshed, M. and Rezgui, Y. (2018). Requirements for cloud-based bim governance solutions to facilitate team collaboration in construction projects, *Requirements Engineering* **23**.

Baresi, L. and Pezzè, M. (2001). Improving uml with petri nets, *Electronic Notes in Theoretical Computer Science* **44**(4): 107–119.

Cailliau, A. and van Lamsweerde, A. (2015). Handling knowledge uncertainty in risk-based requirements engineering, *Requirements Engineering Conference (RE), 2015 IEEE 23rd International*, IEEE, pp. 106–115.

Dardenne, A., van Lamsweerde, A. and Fickas, S. (1993). Goal-directed requirements acquisition, *Science of Computer Programming* **20**(1): 3 – 50.

del Foyo, P. M. G. (2001). *Ghenesys: Uma rede estendida orientada a objetos para projeto de sistemas discretos*, Master's thesis, Universidade de São Paulo, São Paulo.

DeVries, B. (2013). Mapping of uml diagrams to extended petri nets for formal verification, *Technical Lybrary.Paper 156.* pp. –69.

Fahland, D. (2007). Synthesizing petri nets from ltl specifications - an engineering approach, *Proc. of Algorithmen und Werkzeuge fur Petrinetze*, pp. 69–74.

Horkoff, J., Aydemir, F. B., Cardoso, E., Li, T., Maté, A., Paja, E., Salnitri, M., Piras, L., Mylopoulos, J. and Giorgini, P. (2017). Goal-oriented requirements engineering: an extended systematic mapping study, *Requirements Engineering* .

Kalibatiene, D. and Vasilecas, O. (2012). Application of the ontology axioms for the development of ocl constraints from pal constraints, *Informatica* **23**(3): 369–390.

Knauss, E., Yussuf, A., Blincoe, K. and Damian, D. (2018). Continuous clarification and emergent requirements flows in open-commercial software ecosystems, *Requirements Engineering* **23**.

Lacerda, B. and Lima, P. (2011). Designing petri net supervisors from ltl specifications, *Proceedings of Robotics: Science and Systems*.

Liebel, G., Tichy, M., Knauss, E. and Ljungkrantz, O. (2018). Organization and communication problems in automotive requiremens engineering, *Requirements Engineering* **23**.

Martinez, J. and Silva, J. (2015). Combining the use of KAOS and GHENeSys in the Requirement Analysis of service manufacturing, *IFAC-PapersOnLine* **48**: 1634–1639.

Murata, T. (1989). Petri nets: properties, analysis and applications, *Proceedings of IEEE* **77**(4): 541–580.

Salmon, A. Z. O. (2017). *Modelagem e análise de requisitos de sistemas automatizados usando UML e Redes de Petri.*, PhD thesis, Universidade de São Paulo.

Salmon, A. Z. O., González del Foyo, P. M., Silva, J. R. et al. (2014). Verification of automated systems using invariants, *Congresso Brasileiro de Automática-CBA. Belo Horizonte*, SBA.

Salmon, A. Z. O., Miralles, J. A., del Foyo, P. M. G. and Silva, J. R. (2011). Towards a unified view of modeling and design with ghenesys., *Proceedings of the 21st International Congress of Mechanical Engineering* .

Silva, J. and del Foyo, P. M. G. (2012). Timed petri nets, *Petri Nets: Manufacturing and Computer Science*, Intech, chapter 16, pp. 359–372.

Silva, J. R., Martinez, J., Pereira, C., Avram, C. and Stan, S. (2018). New trends in residential automation, *in* E. Ottaviano, A. Pellicio and V. Gatulli (eds), *Mechatronics for Cultural Heritage and Civil Engineering*, Springer, chapter 5.

Vaquero, T. S., Silva, J. and Beck, J. C. (2013). Post-design analysis for building and refining ai planning systems, *Engineering Applications of Artificial Intelligence* **26**: 1967–1979.