CNMAI Congresso Nacional de Matemática Aplicada à Indústria 2014
18 a 21 de novembro de 2014, Caldas Novas - Goiás

INDUSTRIAL PRODUCTION LINE MODELING AND REASONING BY PETRI NETS*

Vaston Gonçalves da Costa, vaston@ufg.br [1,4]
Bruno Lopes Vieira, blopesvieira@gmail.com [3]
Marcelo Henrique Stoppa, mhstoppa@gmail.com[2]
Leandro Rodrigues da Silva Souza, leandrorodrigues.s@gmail.com[2]

[1]Universidade Federal de Goiás - Departamento de Ciência da Computação
Av. Dr. Lamartine P. de Avelar, 1120 - Catalão - GO, 75.704-020

[2]Universidade Federal de Goiás - Departamento de Matemática
Av. Dr. Lamartine P. de Avelar, 1120 - Catalão - GO, 75.704-020

[3]Pontifícia Universidade Católica do Rio de Janeiro - Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea, Rio de Janeiro - RJ, 22451-900

[4]CAPES Scholarship Proc.: BEX 11765/13-5

Abstract. **CNMAI2014-0021**

Petri Nets are a widely used formalism to execute and analyze concurrent procedures. It takes advantage of an intrinsic support to parallelism and concurrence as well as of the possibility of intuitive graphical interpretation. It can be used for modeling all the behavior of procedures and tracking resources flow. Petri-PDL is a logic formalism capable of making inferences on Petri Nets even in its concurrent dealings. Besides other works evolving reasoning in concurrent systems or those using Petri Nets, Petri-PDL is a decidable, sound and complete formalism that takes advantage of Petri Nets resources. The inspection method used in a VIN (vehicle identification number) validation does not meet the quality requirements demanded by the market. In some cases reported, the inspection is performed manually, by a visual checking operator, which exposes the company to the risk of commercializing a vehicle with differences between its VIN and the vehicle documentation. Due to the emotional factor and specially fatigue, the inspection should not be performed by a person. It requires a technological tool to assist on the transcription of the validating data to the chassis in order to avoid possible problems. This work presentes a new approach based on Petri Nets using Propositional Dynamic Logics to verify whether the model of a low cost system for validating the data transcribed to a chassis is correct – and therefore efficient.

Keywords: *Industrial Modelling, Petri Nets, Dynamic Logic*

1. INTRODUCTION

The current criterion adopted worldwide for identifying vehicles started to take effect in Brazil in May 1988, through the National Council of Transit's (CONTRAN – BRASIL 1988) Resolution No. 691. Each identification is forged by etching and comprises the VIN (vehicle identification number), the VIS (Vehicle Identification sector), the number of the engine and the nameplate (Cecere 2010). Statistics point out a significant increase in the number of vehicles robbery and theft. Their chassis are then often tampered.

This criminal practice is profitable and relatively easy to do, since the manufacturers still use obsolete methods for printing the identification data, such as puncture. Nowadays we can count on more efficient methods for vehicle identification recording, as, for example, microdots printing, which uses nanotechnology to coding the vehicle. They are not always applied, however.

*The authors gratefully acknowledge the partial financial support by CAPES, CNPq, UFG/RC, UFU, FAPEG, and FAPEMIG

The chassis recording process adopted by many Brazilian manufacturers has not been as much improved as in other countries (Cecere 2010). Many of them make use of antiquated methods of chassis recording, remarkably those guided by human visual conference. As a result, there are significant annual amounts of rework and lawsuits against the car companies.

A natural way to replace the human visual conference is to adopt a software system to perform this task. da Silva Souza et al. (2014) presents a low cost system to assist on the validation of the data transcribed by puncture to the chassis. Although this system has been submitted to a variety of tests, they are not sufficient to evaluate all possible existent scenarios. In cases like that, a formal system applied to the verification process can ensure that another system really does what it is supposed to do.

For a long time there have been intense discussions regarding the verification of software correctness. By correctness we oppose to ad hoc software development process (SDP), as they do not support all the possible decision scenarios.

The simple usage of general purpose abstractions such as UML diagrams, the latter coupled to methods (also of general purpose) that identify (or do not) steps in the SDP, do not further ensure the correct implementation of the system functionality that is required. The fact, for example, that SDP based on UML have been well accepted by the market is strongly influenced by the existence of tools for the derivation of (formally uncertified) executable codes – sometimes already in the final stage of production.

However, it is a common practice to validate through tests or simulations the code produced by the UML abstraction, for example, rather than doing this validation by means of formal analysis techniques on the abstraction level of the UML specification itself. Model checkers have been properly used for validating a behavioral specification in their abstraction level.

It is unnecessary to say that formal analysis techniques have always been used in the validation of specifications/critical systems (time-dependent, based on real-time, fault tolerant etc.), especially when they involve the preservation of human life, when errors may lead to huge financial losses or, finally, when dealing with information security.

In the case of Software Systems and the problems they solve, or the parcel of the world they help to automate, the comparison with what happens with scientific theories (STs) is perfect. In the latter, as in the former, there is no definite way to guarantee infallibility.

The validation process, as stated by Dijkstra – paraphrasing the Hypothetical-Deductive proposal for confirmation of STs – says that "the experiment (test) did not refute the hypothesis designed based on ST (system + world)". Thus, an experiment is most successful when it rejects a hypothesis constructed by the theory. That is why the wider the range of experiments (validation tests) for a system, the better. The aim is to increase the search space to be successful in the task of "finding errors". On the other hand, with what we know about probability, we can state surely that a system little tested/validated that has an infinite number of inputs or different behaviors has the same probability of being definitely correct than a very much tested/validated one.

In order to deal with system validation it is necessary to take into account its most intrinsic aspects through model checkers and theorem provers that may faithfully attest that the implemented system meets the requirements proposed for it. That is what justifies the increasing expansion of the use of various formal systems for checking systems correctness.

A widely used class of logics to reason about systems is the Dynamic Logics (Fischer & Ladner 1979) and Propositional Dynamic Logic (PDL). They are used in the most varied ways due to their being decidable and complete, among other good properties. By "decidable" it is meant that there is an effective method for determining whether a formula/argument can be satisfied or not. "Complete", by its turn, means that every valid property can be verified in the system. PDL can be used for model checking (De Giacomo & Massacci 1998) and there are tools implemented for its reasoning (Schmidt 2004).

Petri Nets is present in the literature as one of the most used formalisms to deal with concurrent and parallel systems. Despite its algebraic formalism, it takes advantage of an intuitive graphical interpretation that can simplify the modeling process.

Unifying these two formalisms, Propositional Dynamic Logic for Petri Nets (Petri-PDL – Lopes, Benevides & Haeusler 2014*b*) is presented as a logic to reason about Petri Nets. A Petri-PDL formula is in the form $\langle s, \pi \rangle \varphi$ denoting that after some running of the Petri Net program π with initial markup s, supposing that π stops, φ holds on (with its necessity correspondence in $[s, \pi]\varphi$, meaning that if π stops then φ holds on after any of its possible runnings). Another advantage of Petri-PDL inherited from Petri Nets is that it can model the resources accumulation and consumption by the markup of the Petri Net program.

Using Petri-PDL to make inferences about a model used to implement a system that validates the data transcribed to a chassis allows the modeler to deal with resources acquisition and consumption, besides using all the other properties of Petri Nets in a decidable, sound and complete logical system.

In this paper we present a new approach based on Petri Net using Petri-PDL for modeling and verifying properties of the system presented by Souza et al. (2015). First we present the theoretical background, then a practical example on how to make inferences using Petri-PDL.

2. BACKGROUND

In this section we present formal method definitions and the Petri Net model used in this work.

2.1. Formal Methods

Formal methods are mathematical techniques, often supported by tools, for developing software and hardware systems. It is very important to find a suitable and comprehensive way to define and describe the underlying problem so that it becomes easier to find a solution.

Formal methods in the decade of 1970 was limited to software and hardware systems (Hoare 1972, Dijkstra 1975). But due to the complexity of the computacional systems used to manage complex systems in different industrial areas, formal methods have became mandatory to increase the reliability of these systems (Gabbar 2006).

Formal methods to ensure the reliability of critical systems become important in the decade of 1990. Rushby (1993) presented what formal methods are and how they can be applied in the development and certification of critical systems. Even NASA published a technical report on the usage of formal methods for specification and verification of software and computer systems (NASA 1998, 1997).

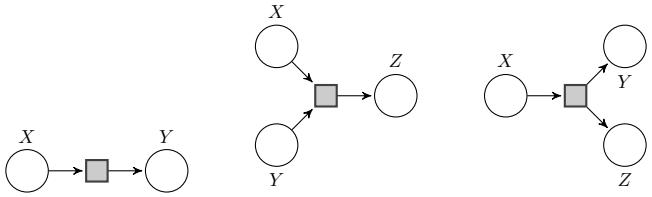
Recently the logical systems, based in type theories and deductive logical systems, have been applied to model checkers and automated deduction/theorem provers, being considered as strong tools in formal methods Gabbar (2006).

Amongst all the applications of logics in formal methods some of the most remarkable are the use of description logic to represent knowledge, the use o lambda calculus to construct powerful theorem provers such as Coq (Bertot & Castéran 2004) and PVS (Owre et al. 1992) and model checkers such as CTL (Alur et al. 1990).

These tools can be adjusted to comply with different logical frameworks to model and verify properties of computacional systems. This is very useful, since dealing with different models requires specific logical systems.

2.2. Petri Nets

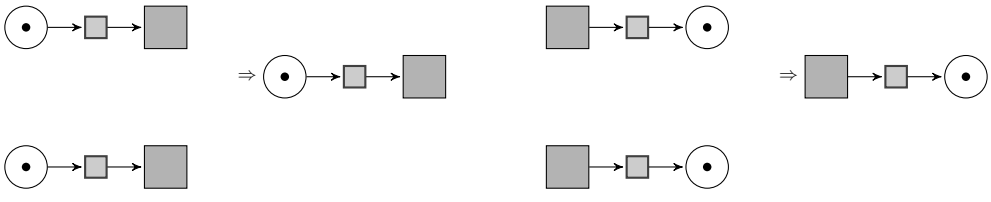
The works presented bellow concern the Marked Petri Net Model defined by de Almeida & Haeusler (1999). In this model there are only three types of transition which define all valid Petri Nets due to its compositions. These basic Petri Nets are as in figure 1.



(a) $Type\ 1:\ t_1$ (b) $Type\ 2:\ t_2$ (c) $Type\ 3:\ t_3$

Figure 1. Basic Petri Nets

A gluing procedure (de Almeida & Haeusler 1999) is used to compose more complex Petri Nets out of these three basic kinds of composition . The operations involved in this process are Conflict on the Left (figure 2(a)), Conflict on the Right (figure 2(b)), two cases of Sequence (figures 3(a) and 3(b)), Joint on the Left (figure 3(c)), Joint on the Right (figure 3(d)) and three cases of Repetition (figures 4(a), 4(b) and 4(c)).

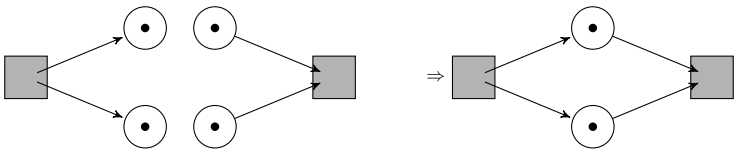


(a) Conlict-L (b) Conlict-R

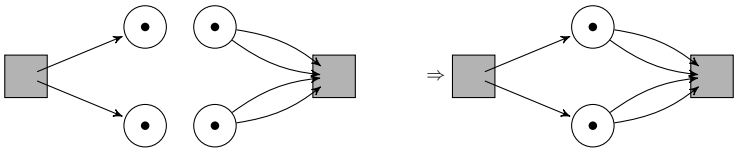
Figure 2. Example of application of Conflict rules, where black boxes represent any valid subnet of a Petri Net

3. PROPOSITIONAL DYNAMIC LOGIC FOR PETRI NETS: PETRI-PDL

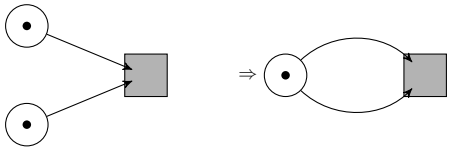
Among many logics to deal with concurrent systems (Abrahamson 1980, Benevides & Schechter 2008), Petri-PDL (Lopes, Benevides & Haeusler 2014*b*, Benevides et al. 2011) is an extension of Propositional Dynamic Logic (PDL – Fischer & Ladner 1979) that benefits from the intuitive graphical representation of Petri Nets and is a decidable, sound and complete formal system. It is a muti-modal logic where each modality is a program π with a markup s, says $\langle s, \pi \rangle$.



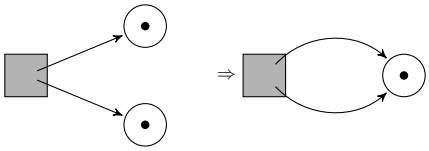
(a) Sequence case 1



(b) Sequence case 2

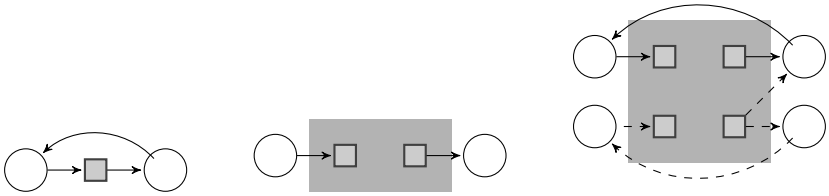


(c) Joint-L



(d) Joint-R

Figure 3. Examples of Sequence and Joint rules applications where black boxes represent any valid subnet of a Petri Net



(a) Case 1 (b) Case 2 (c) Case 3

Figure 4. Examples of the three cases of Repetition rule application

Its language consists of:

Propositional symbols: $p, q \ldots$

Place names: e.g.: $a, b, c, d \ldots$

Transition types: $T_1 : xt_1y, T_2 : xyt_2z$ and $T_3 : xt_3yz$

Petri Net Composition symbol: $\odot$

Sequence of names: $S = \{\epsilon, s_1, s_2, \ldots\}$, where ϵ is the empty sequence. We use the notation $s \prec s'$ to denote that all names occurring in s also occur in s'.

Definition 3.1. Programs:

Basic programs: $\pi_b ::= at_1b \mid at_2bc \mid abt_3c$ where t_i is of type $T_i, i = 1, 2, 3$

Petri Net Programs: $\pi_{PN} ::= \pi_b \mid \pi_{PN} \odot \pi_{PN}$, denoted by $\eta_1, \eta_2, \ldots$

A formula is defined as $\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle s, \pi \rangle \varphi$.

We use the standard abbreviations $\bot \equiv \neg\top$, $\varphi \vee \phi \equiv \neg(\neg\varphi \wedge \neg\phi)$, $\varphi \rightarrow \phi \equiv \neg(\varphi \wedge \neg\phi)$ and $[s, \pi]\varphi \equiv \neg\langle s, \pi \rangle \neg\varphi$.

Definition 3.2. The *firing* of a transition is defined by the function $f : S \times \pi_b \rightarrow S$ as follows:

- $f(s, at_1b) = \begin{cases} s_1bs_2, & if \quad s = s_1as_2 \\ \epsilon, & if \quad a \not\prec s \end{cases}$

- $f(s, abt_2c) = \begin{cases} s_1cs_2s_3, & if \quad s = s_1as_2bs_3 \\ \epsilon, & if \quad a, b \not\prec s \end{cases}$
- $f(s, at_3bc) = \begin{cases} s_1s_2bc, & if \quad s = s_1as_2 \\ \epsilon, & if \quad a \not\prec s \end{cases}$
- $f(\epsilon, \eta) = \epsilon$, for all petri nets programs η.

Definition 3.3. A frame for Petri-PDL is a 3-tuple $\langle W, R_\pi, M \rangle$, where

- W is a non-empty set of states;
- $M\colon W \to S$;
- R_α is a binary relation over W, for each basic program $\alpha \prec \pi_b$, satisfying the following condition. Let $s = M(w)$
  - if $f(s, \alpha) \neq \epsilon$, $wR_\alpha v$ iff $f(s, \alpha) \prec M(v)$
  - if $f(s, \alpha) = \epsilon$, $wR_\alpha v$ iff $w = v$
- we inductively define a binary relation R_η, for each Petri Net program $\eta = \eta_1 \odot \eta_2 \odot \cdots \odot \eta_n$, as $R_\eta = \{(w, v) \mid$ for some $\eta_i, \exists u$ such that $s_i \prec M(u)$ and $wR_{\eta_i}u$ and $uR_\eta v\}$ Where $s_i = f(s, \eta_i)$, for all $1 \leq i \leq n$.

Definition 3.4. A *model* for Petri-PDL is a pair $\mathcal{M} = \langle \mathcal{F}, \mathbf{V} \rangle$, where $\mathcal{F}$ is a PDL frame and $\mathbf{V}$ is a valuation function $\mathbf{V}\colon \Phi \to 2^W$.

The semantical notion of satisfaction for Petri-PDL is defined as follows.

Definition 3.5. Let $\mathcal{M} = (\mathcal{F}, \mathbf{V})$ be a model. The notion of *satisfaction* of a formula φ in a model $\mathcal{M}$ at a state w, notation $\mathcal{M}, w \Vdash \varphi$, can be inductively defined as follows:

- $\mathcal{M}$,$w \Vdash p$ iff $w \in \mathbf{V}(p)$;
- $\mathcal{M}$,$w \Vdash \top$ always;
- $\mathcal{M}$,$w \Vdash \neg\varphi$ iff $\mathcal{M}$,$w \nVdash \varphi$;
- $\mathcal{M}$,$w \Vdash \varphi_1 \wedge \varphi_2$ iff $\mathcal{M}$,$w \Vdash \varphi_1$ and $\mathcal{M}$,$w \Vdash \varphi_2$;
- $\mathcal{M}$,$w \Vdash \langle s, \eta \rangle \varphi$ if there exists $v \in W$, $wR_\eta v$, $s \prec M(w)$ and $\mathcal{M}$,$v \Vdash \varphi$.

If $\mathcal{M}, v \Vdash A$ for every state v, we say that A is *valid in the model* $\mathcal{M}$, notation $\mathcal{M} \Vdash A$. And if A is valid in all $\mathcal{M}$ we say that A is *valid*, notation $\Vdash A$.

3.1. Axiomatic System

The set of axioms for Petri-PDL is as follows, where p and q are proposition symbols, φ and ψ are formulas, $\eta = \eta_1 \odot \eta_2 \odot \cdots \odot \eta_n$ is a Petri Net program and π is a Marked Petri Net program.

(PL) Enough propositional logic tautologies
(K) $[s, \pi](p \to q) \to ([s, \pi]p \to [s, \pi]q)$
(Du) $[s, \pi]p \leftrightarrow \neg\langle s, \pi \rangle \neg p$
(PC) $\langle s, \eta \rangle \varphi \leftrightarrow \langle s, \eta_1 \rangle \langle s_1, \eta \rangle \varphi \vee \langle s, \eta_2 \rangle \langle s_2, \eta \rangle \varphi \vee \cdots \vee \langle s, \eta_n \rangle \langle s_n, \eta \rangle \varphi$,
where $s_i = f(s, \eta_i)$, for all $1 \leq i \leq n$ and π is not a basic program.
($\mathbf{R}_\epsilon$) $\langle s, \eta \rangle \varphi \leftrightarrow \varphi$, if $f(s, \eta) = \epsilon$
(Sub) If $\Vdash \varphi$, then $\Vdash \varphi^\sigma$, where σ uniformly substitutes proposition symbols by arbitrary formulas.
(MP) If $\Vdash \varphi$ and $\Vdash \varphi \to \psi$, then $\Vdash \psi$.
(Gen) If $\Vdash \varphi$, then $\Vdash [s, \pi]\varphi$.

3.1.1. Soundness, decidability and completeness

Petri-PDL is proved to be sound, complete regarding the presented semantics and decidable (Lopes, Benevides & Haeusler 2014*b*,*a*).

To reach a finite model property we restrict a subset of Petri Nets denoted by normalised Petri Net. They are composed by any Petri Net as in section 2.2 that does not contain any place who can accumulate an infinite amount of tokens. Hence we filter the model by a quotient on the Fischer-Ladner closure (Goldblatt 1992).

The first step is to define the Fischer-Ladner closure (FL). It is inductively defined as follows, where $FL(\varphi)$ denotes the smallest set containing φ which is closed under sub formulae.

Then, given a Petri-PDL formula φ and a Petri-PDL model $\mathcal{K} = \langle W, R_\eta, M, \mathbf{V} \rangle$, we define a new model $\mathcal{K}^\varphi = \langle W^\varphi, R_\eta^\varphi, M^\varphi, \mathbf{V}^\varphi \rangle$, the filtration of $\mathcal{K}$ by $FL(\varphi)$, as follows.

The relation $\equiv$ over the worlds of $\mathcal{K}$ is defined as

$$u \equiv v \leftrightarrow \forall \phi \in FL(\varphi), M,u \Vdash \phi \text{ iff } M,v \Vdash \phi$$

and the relation R_η^φ is defined as

$$[u]R_\eta^\varphi[v] \leftrightarrow (\exists u' \in [u] \wedge \exists v' \in [v] \wedge u' R_\eta v').$$

(a) $[u] = \{v \mid v \equiv u\}$
(b) $W^\varphi = \{[u] \mid u \in W\}$
(c) $[u] \in \mathbf{V}^\varphi(p)$ iff $u \in V(p)$
(d) $M^\varphi([u]) = \langle s_1, s_2, \dots \rangle$ where for all $j \geq 1, v_j \in [u]$ iff $M(v_j) = s_j$

The detailed proof is presented by Lopes, Benevides & Haeusler (2014*b*).

4. THE MODEL OF THE SYSTEM

da Silva Souza et al. (2014) presents a low cost system to assist on the validation of the data transcribed by puncture to a vehicle chassis.

The numerical sequence inserted in a chassis is composed of 17 digits, each digit making reference to a certain piece of information, such as country of manufacture, year, model, among other characteristics and purposes (BRASIL 1988).

The process of recording a chassis makes use of a machine to etching the numerical sequence. The information to be etched is inserted via keyboard or via bar code reader interfaces, allowing the inclusion of information that will be used for data manipulation or the insertion of a code to be written.

However, in the process to pick up the information, several flaws may appear, such as poorly formatted bar codes that can generate errors in reading and consequently inconsistent records, besides the possibility of a wrong typing by the operator. In this context, it is important to make a final validation of the work performed by the machine, and in case of inconsistence not to allow that the recorded chassis goes ahead along to the assembly process.

For an automated inspection it is proposed the implementation of a software that uses a low cost camera to perform data entry, providing information to the system in its sequence of tasks, going through the steps of capturing, processing and identification of the regions considered. Once this is done, then comes the optical character recognition (OCR) of the character recorded on the chassis.

The block diagram below describes the flow implemented.

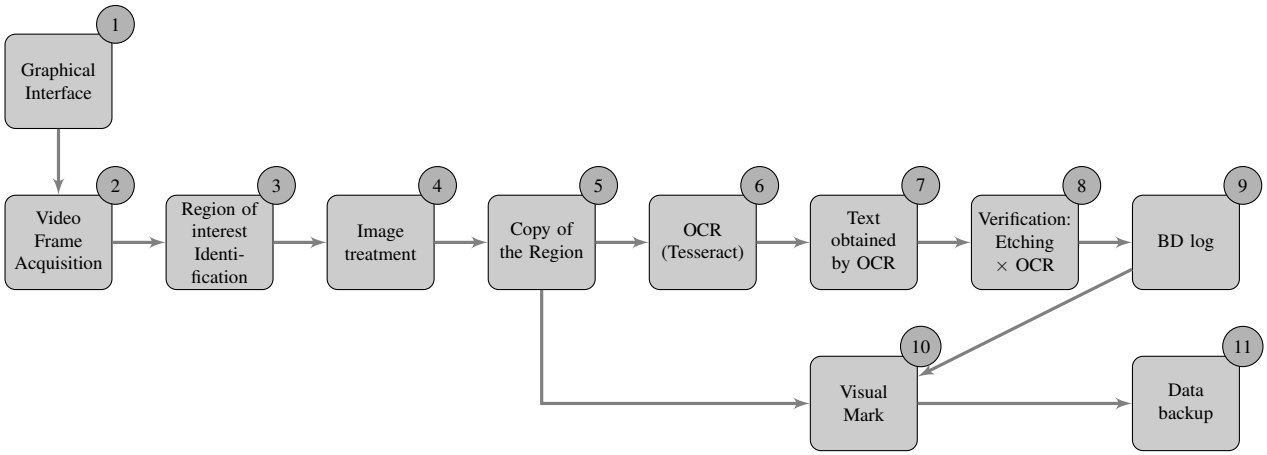


Figure 5. Model of System

The image process starts focusing on the region of interest, since the other parts of the image do not contain the VIN's information. So, the processing time of the algorithms is optimized, by recognizing and working only on the numbers on the chassis. Then a contrast correction over the image is performed, since diffused lighting was not applied.

As the chassis are usually black, a correction is also applied to distribute a gray-scale on the image. The well known Canny algorithm (Hou & Koh 2003) is applied to detecting the edges of the image. In order to achieve a better location of the VIN, the Canny algorithm is applied twice, first in gray-scale and after in color scale.

This region of interest is sent to the OCR for the implementation of image analysis algorithms. The process of recognition of characters contained in the digital image is performed by Tesseract Engine, which transcripts the text contained in the digital image as a plaintext.

After this, the plaintext is stored to be used to ensure that the information transcribed in the chassis is correct. If the characters transcribed match with the information required, a check mark ($\surd$) is added to the image stored. Otherwise the mark $\times$ is used.

The result obtained in the OCR analysis process is recorded in the database, which can be accessed at any time. Concerning the veracity of the information contained in the image, an authentication is included in the digital document, so that the encoding inserted in the images ensures reliability and makes future checkins easier (Rey & Dugelay 2002).

Finally, the original information and the images containing the validation tags – obtained via processing – are stored together for future audit.

We can model this process in the Petri Net as shown in Figure 6, where the numbers (labels of places) correspond to the steps described in Figure 5.

Notice that this is a high-level modelling wherewith further work will detail each step and provide other useful information to construct inferences to verify interesting properties.

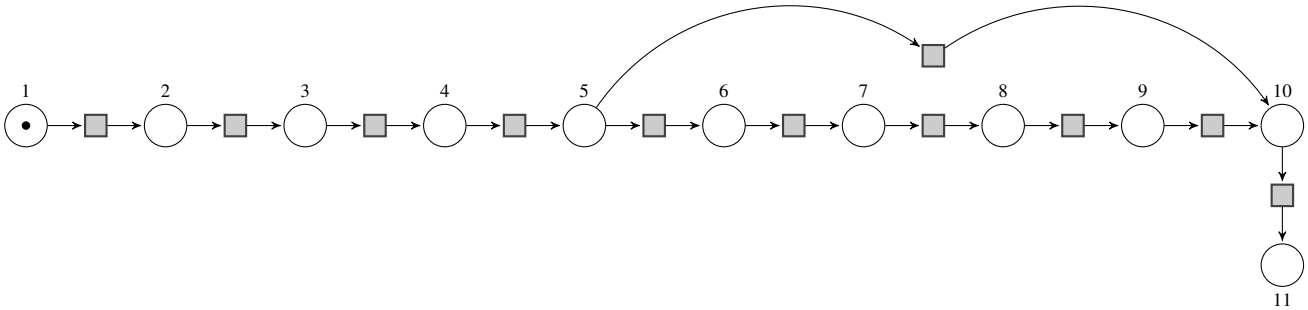


Figure 6. Process modelled as a Petri Net

Using Petri-PDL, we can model this Petri Net into the formula $\langle(1), 1t_1 2 \odot 2t_1 3 \odot 3t_1 4 \odot 4t_1 5 \odot 5t_1 6 \odot 6t_1 7 \odot 7t_1 8 \odot 8t_1 9 \odot 9t_1 10 \odot 10t_1 11 \odot 5t_1 10\rangle A$ where A is a proposition meaning that the process was finished.

As a usage example of Petri-PDL, we can verify if it is possible that some image may be stored before the OCR is used.

Namely, we want to verify if, from the initial stage, it is possible that the basic program $10t_1 11$ runs without the basic program $5t_1 6$ runs.

It is equivalent to verify if the formula $\langle(1), 1t_1 2 \odot 2t_1 3 \odot 3t_1 4 \odot 4t_1 5 \odot 5t_1 6 \odot 6t_1 7 \odot 7t_1 8 \odot 8t_1 9 \odot 9t_1 10 \odot 10t_1 11 \odot 5t_1 10\rangle A \rightarrow \langle s, 1t_1 2 \odot 2t_1 3 \odot 3t_1 4 \odot 4t_1 5 \odot 6t_1 7 \odot 7t_1 8 \odot 8t_1 9 \odot 9t_1 10 \odot 10t_1 11 \odot 5t_1 10\rangle A$ where $10 \prec s$ holds.

To verify if both images will be stored, that is, if the initial stage we will achieve two tokens in place 11, we just need to verify if the formula $\langle(1), 1t_1 2 \odot 2t_1 3 \odot 3t_1 4 \odot 4t_1 5 \odot 5t_1 6 \odot 6t_1 7 \odot 7t_1 8 \odot 8t_1 9 \odot 9t_1 10 \odot 10t_1 11 \odot 5t_1 10\rangle A \rightarrow \langle(11, 11), 1t_1 2 \odot 2t_1 3 \odot 3t_1 4 \odot 4t_1 5 \odot 5t_1 6 \odot 6t_1 7 \odot 7t_1 8 \odot 8t_1 9 \odot 9t_1 10 \odot 10t_1 11 \odot 5t_1 10\rangle A$ is true.

Notice that this proof is straightforward from the usage of the firing function definition and axiom PC.

The resolution-based calculus for Petri-PDL Lopes, Nalon, Hermann & Dowek (2014) can be used to check a formula for (un)satisfiability.

5. CONCLUSIONS AND FURTHER WORK

This work describes a methodology for using Petri-PDL to reason and verify system properties that validate the information casting in the chassis of vehicles. As Petri-PDL is a decidable, sound and complete formalism who inherits the advantages of Petri Nets, its use enables the usage of a "memory" where resources are accumulated and may be consumed progressively during the program execution, which well suitable for industrial modeling.

This methodology can be applied to any kind of software. As the regular Petri Nets, it can deal with all aspects of concurrent systems and lets the user model the system graphically.

A next step to complete the formalization of the system built is to validate the code generated. This is also a future work, that will be done with the help of the automatic proofing and model checking.

REFERENCES

Abrahamson, K. R. (1980), Decidability and Expressiveness of Logics of Processes, PhD thesis, Department of Computer Science, University of Washington.

Alur, R., Courcoubetis, C. & Dill, D. (1990), Model-checking for real-time systems, *in* 'Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on e', pp. 414–425.

Benevides, M., Haeusler, E. & Lopes, B. (2011), Propositional dynamic logic for petri nets, *in* 'Annals of the 6th Workshop on Logical and Semantic Frameworks, with Applications'.

Benevides, M. R. F. & Schechter, L. M. (2008), A propositional dynamic logic for CCS programs, *in* 'Proceedings of the XV Workshop on Logic, Language', Vol. 5110, pp. 83–97.

Bertot, Y. & Castéran, P. (2004), *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*, Texts in Theoretical Computer Science. An EATCS Series, Springer.

BRASIL (1988), 'Resolução 691, de 05 de abril de 1988: identificação de veículos rodoviários'.

Cecere, A. V. (2010), Estudo de medidas para a melhora da identificação veicular no Brasil, Master's thesis, Escola Politécnica da Universidade de São Paulo, São Paulo.

da Silva Souza, L. R., Oliveira, R. & Stoppa, M. H. (2014), Proposta de inspeção por câmera no processo de validação da gravação de chassi, *in* 'Anais VIII CONEM - Congresso Nacional de Engrenharia Mecânica', Uberlandia-MG - Brasil. To appear in august 2014.

de Almeida, E. S. & Haeusler, E. H. (1999), 'Proving properties in ordinary Petri Nets using LoRes logical language', *Petri Net Newsletter* **57**, 23–36.

De Giacomo, G. & Massacci, F. (1998), 'Combining deduction and model checking into tableaux and algorithms for Converse-PDL', *Information and Computation* **160**, 2000.

Dijkstra, E. W. (1975), 'Guarded commands, nondeterminacy and formal derivation of programs', *Commun. ACM* **18**(8), 453–457.
URL: *http://doi.acm.org/10.1145/360933.360975*

Fischer, M. J. & Ladner, R. E. (1979), 'Propositional dynamic logic of regular programs', *Journal of Computer and System Sciences* **18**, 194–211.
URL: *http://www.sciencedirect.com/science/article/pii/0022000079900461*

Gabbar, H. (2006), *Modern Formal Methods and Applications*, Springer, New York.

Goldblatt, R. (1992), 'Parallel action: Concurrent dynamic logic with independent modalities', *Studia Logica* **51**, 551–558.

Hoare, C. (1972), 'Proof of correctness of data representations', *Acta Informatica* **1**(4), 271–281.
URL: *http://dx.doi.org/10.1007/BF00289507*

Hou, Z. & Koh, T. (2003), 'Robust edge detection', *Pattern Recognition* **36**(9), 2083 – 2091. Kernel and Subspace Methods for Computer Vision.
URL: *http://www.sciencedirect.com/science/article/pii/S0031320303000463*

Lopes, B., Benevides, M. & Haeusler, E. H. (2014*a*), 'Extending Propositional Dynamic Logic for petri nets', *Electronic Notes in Theoretical Computer Science* **305**, 67–83.

Lopes, B., Benevides, M. & Haeusler, E. H. (2014*b*), 'Propositional dynamic logic for petri nets', *Logic Journal of the IGPL* .

Lopes, B., Nalon, C., Hermann, E. & Dowek, G. (2014), A calculus for automatic verification of petri nets based on resolution and dynamic logics, *in* '17th Brazilian Logic Conference', Laboratorio Nacional de Computação Científica (LNCC)- Petrópolis- Brazil.

NASA (1997), Formal methods, specification and verification guidebook for the verification of software and computer systems. vol ii: A practitioner's companion, Technical Report NASA-GB-001, NASA, Washington, DC.

NASA (1998), Formal methods, specification and verification guidebook for the verification of software and computer systems. vol i: Planning and technology insertion, Technical Report NASA/TP-98-208193, NASA, Washington, DC.

Owre, S., Rushby, J. & Shankar, N. (1992), Pvs: A prototype verification system, *in* D. Kapur, ed., 'Automated Deduction—CADE-11', Vol. 607 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 748–752.

Rey, C. & Dugelay, J.-L. (2002), 'A survey of watermarking algorithms for image authentication', *EURASIP J. Appl. Signal Process.* **2002**(1), 613–621.
URL: *http://dl.acm.org/citation.cfm?id=1283100.1283165*

Rushby, J. (1993), Formal methods and the certification of critical systems, Technical Report CSL-93-7, Computer Science Laboratory SRI International, Menlo Park, CA.

Schmidt, R. A. (2004), 'PDL-Tableau'. Accessed in February, 2013.
URL: *http://www.cs.man.ac.uk/ schmidt/pdl-tableau/*

Souza, L., Oliveira, R. & Stoppa, M. (2015), Proposal of Automated Inspection Using Camera in Process of VIN Validation, *in* M. Ceccarelli & E. E. Hernández Martinez, eds, 'Multibody Mechatronic Systems', Vol. 25 of *Mechanisms and Machine Science*, Springer International Publishing, pp. 285–293.
URL: `http://dx.doi.org/10.1007/978-3-319-09858-6_27`

AUTHORSHIP RESPONSIBILITY

"The authors are solely responsible for the content of this work".