VERIFICATION OF AUTOMATED SYSTEMS USING INVARIANTS

Arianna Z. Olivera Salmon*, Pedro M. G. del Foyo†, José R. Silva*

*Design Lab, Department of Mechatronics and Mechanical Systems Engineering, University of São Paulo
Av. Professor Mello Moraes, 2231, CEP 05508-970
São Paulo/SP, Brazil

†Mechanical Engineering Department, Federal University of Pernambuco,
Av. Acadêmico Helio Ramos s/n, CEP 50670-901
Recife/PE, Brazil

Emails: arianna@usp.br, pedro.foyo@ufpe.br, reinaldo@usp.br

Abstract— Nowadays, Petri net and its extensions have been used for modeling and verification of complex systems, used as a sound description language. Algorithms derived from this modeling framework can facilitate the analysis and verification of properties. Methods of verification based in invariants are among the most computationally efficient and allow the verification of other important properties. This work proposes the use of invariants for the verification of desirable properties for an automatic system in the early design phase. Therefore invariants are used to validate requirements assuming they are elicited using UML Diagrams and modeled also in Petri Nets.

Keywords— Invariants, Verification, Petri net.

1 Introduction

The raising demand for complex systems inspired recent developments in formal methods and computer tools to support applications in this domain. However, the challenge to treat large and complex systems still persist as a need to improve some analytical approaches based on property analysis and verification. Discrete systems approach is a promising path for the development and application of such methods.

A well known formal representation for discrete system is the schematic approach based on Petri Nets. One of the reasons for the success of Petri Nets is the expressive and simple way it treats (simple or complex) problems and the formal approach to concurrency, besides a sound property analysis, based on graph theory. One of these properties is invariant analysis, which is present in several knowledge fields, ranging from software engineering to physics or biology. In this work we will explore the vision of invariants close to software engineering.

In fact, invariants are properties that appear in the whole design process. If a project of a discrete target admit a dynamic behavior, it could as well be described by a discrete representation such as Petri Nets and associated to invariants. Therefore it is a challenge and also a goal to introduce invariant analysis in the early phases of the design process to provide correctness and also traceability. Specially when dealing with automated systems this approach would be advisable, since it is imperative to guarantee correctness of behavior to such systems since the beginning of design.

Those are only some of the reasons why property analysis of discrete systems - specially invariant analysis - has received much attention in the last three decades with the appearance of general proposals (Murata, 1989), (Silva, 1998), disregarding its insertion in the design process. The computation complexity the invariant method has attracted more attention due to its efficiency and lead to proposals relaying on schematic approach and providing algorithms to a next design phase with a formal modeling already done. Thus, invariant analysis is normally attached to the design phase, that is, it is not present in the requirement analysis and specification phase.

The main goal of the approach presented in this work is just to provide support for the early phase of project development, mainly the require-
ments analysis. We assume that requirements are already elicited and represented in a classic language notation as UML (Unified Modeling Language). The relationship between the system and its context environment would be described in Object Constraint Language (OCL), which is very important to detach invariants. Prospective results are illustrated by a case study based on a evaporation system (Machado et al., 2007).

The paper is organized in the following way: Section 2 describe some formal definitions concerning invariants in Petri Nets representation. Section 3 presents the main analysis methods existing in the literature based on classics Petri Nets (Place/Transition). Sections 4 and 5 present the proposal of the authors and a case study. Section 6 has some final considerations and oint to further work.

2 Invariants in Petri nets: Definitions

Invariants are one of the structural properties of Petri nets that depend only on its topological structure and not on the net’s initial marking. There are two kinds of invariants: place invariants and transition invariants.

Definição 1 Place invariants are sets of places whose token distribution remains always constant. These invariants represent a conservative component of the net. They are represented by an n-column vector $x$, where n is the number of places of the Petri net. The non-zero entries correspond to the places that belong to the particular place invariant and the zeros to the remaining places. Place invariants are a non-negative integer solutions of the homogeneous equation:

$$A^T x = 0 \quad (1)$$

Considering the state equation of a net system, that means to have an integer vector solution $x$ that satisfies the equation:

$$M^T x = M_0^T x \quad (2)$$

Where $M_0$ is an initial marking and $M$ belongs to $R(M_0)$.

Equation 2 means that the possibly weighted sum of the tokens in the places of the invariant remains constant in all markings and this sum is determined by the initial marking of the Petri net.

Definição 2 Transition invariants denote a sequence of transitions which firing can reproduce the initial marking in the sequence. These invariants represent the repetitive components of the net and are represented by an m-column vector $y$ (where $m$ is the number of transitions) that contains integers in the positions corresponding to the transitions belonging to the transition invariant and zeros everywhere else. The integers denote how many times the corresponding transition must fire in order to go back to the initial marking. They can be derived from the state equation as:

$$Ay = 0 \quad (3)$$

Transition invariant can be physically interpreted as a firing sequence of transitions that do not modify the marking of the net. Therefore, the existence of transition invariants in the Petri net denotes some cyclic behaviour.

As with place invariants, any linear combination of transition invariants is also a transition invariant for the Petri net.

3 Analysis of invariants in Petri nets

Invariants are fundamental algebraic characteristics of Petri nets, and are used in various situations, such as checking (the necessary condition of) liveness, boundedness, the presence of loops and so on (Murata, 1989). There are sets of places and transitions which behaviour do not change during execution. The identification and interpretation of each of these sets is important, because they reflect certain properties of the net that might be of interest to the modeling system.

Place and transition invariants are important issues for analysing Petri nets since they allow for the net’s structure to be investigated independently of any dynamic process (Lautenbach, 1987). Another advantage of the invariants is that its analysis can be performed on local subnets without considering the whole system, i.e., the analysis of invariants can furthermore be inserted into a hierarchical structured net \(^2\). Therefore invariants can be calculated in any abstraction level. Some works consider that invariants are not much affected by refinements on the net. Besides, the calculation of invariants is consistent in the whole net, except by the dimension of the vector, which is proportional to the number of places or transitions.

Invariants are also used for model validation and verification. In addition, the invariant gives us a mathematical tool for analysing other properties of the net. Currently, the calculation of invariants has low computational cost, compared with other methods of analysis, like the method of reachability tree.

Two formal presentations have been extensively used for structural analysis: graph theory and linear algebra. Clearly, the main goal is to develop techniques that can be easily implemented

\(^2\) The project ISO/IEC 15.900 is developing a unification of Petri nets, where the basic nets are the Place/Transition (P/T), coloured nets, and asymmetric nets and where hierarchical nets are considered extensions.
on a computer. This has motivated the development of methods and techniques to optimize the analysis of the structural properties of Petri nets. Techniques that have been considered efficient are those based on linear-algebra, because of their simplicity to obtaining invariants as an initial step for studying the structural properties of Petri nets.

For instance, many works have been developed to analyse properties of Petri nets using Linear Programming techniques. One of the main advantages of using this method is that the computational complexity of linear programming problems is polynomial (Silva et al., 1998).

There are other works, such as (Bouyekhf and Moudni, 2005), that considers some structural aspects of general Petri nets and tries to improve the link between Petri nets and linear algebraic techniques.

Regardless the method used for analysis, Petri nets has been used on many phases of software development (Denaro and Pezzè, 2004), due to its suitability to represent and understand the behaviour of systems. Properties of Petri nets can be used to the representation and verification of requirements. Specifically, invariants are the properties of Petri nets most used on the elicitation and analysis of requirements, which allows to analyse and validate the system in the early phase of project. Invariants represent properties of the systems that are satisfied in all reachable states of the system, enabling the verification from several dynamic properties.

Verification using invariants is very similar to the use of invariants in programming verification: the designer must find a set of equations that denotes the desired properties and test if they hold in any reachable state.

4 Modeling systems using UML and Petri nets

UML is a suitable language for modeling systems, which has been successfully used in different projects of system design. Recent research shows that UML has became the standard for the analysis and design of object-oriented systems: in 2004 Allan Zeichick (Zeichick, 2004) published the result of an inquire among developers showing that about 2/3 of software development organizations were using UML, with 82% predicting they would use it in future (totally or partially). According to Gartner Inc., UML is now used by more than 10 million IT professionals. The existence of a standard notation has released pent-up demands and created an industry (Watson, 2008).

UML provides a broad 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 related to the system behaviour (Engels et al., 2002). Petri nets can be attached to the analysis and verification object modeling systems in a very suitable way.

There are several proposals to deal with UML using Petri nets extensions as a formal intermediate model. In (Zhao et al., 2004) it is presented some technical transformation of graphs, which can be used to translate UML diagrams into Petri nets. Other approaches (Yao and Shatz, 2006), (Doll et al., 2004), are using the information of some diagrams such as the sequence diagrams and activity diagrams to transform them on a Petri net and thereafter do a consistency checking.

The various diagrams which form the UML model are rather correlated, and some relationships among diagrams reflect the grammar rules and semantics of UML itself. Therefore, when transforming a UML model into a Petri net model, not only the static structure and dynamic semantic of every single diagram need to be transformed, but also the relationships among them. 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. This third layer describes the relationship between the diagrams of static structure view and the diagrams of dynamic behaviour view. According to Zhao et al. (Zhao et al., 2004), the third layer of relationship is rarely considered in the available works on verification and transformation of UML models. In this paper our goal is verify models that integrate diagrams with static and dynamic views, thus contributing to the third relationship level.

The first step of our approach will be to elicit and represent the requirements of a system using UML class and statechart diagrams (or any other state diagram). UML has a high power graphic expression, but despite this, there are properties and restrictions of systems that are very complex or impossible to be adequately expressed in a diagram. Even though UML has extension mechanisms such as stereotypes, tagged values and predefined constraints, those could be enough. For this reason, we will use OCL to formulate some of the system requirements, primarily those representing invariants.

When the UML modellling is done, the resulting models will be transformed into Petri net. The transformation will be based on a method proposed by (Baresi and Pezzè, 2001). Finally, a Petri net will be built using the GHENeSys environment. We illustrate the proposed approach through a
case study.

4.1 Describing the case study

Figure 1 shows a Evaporator System which consists of two tanks (one of them is heated and mixed), a condenser, level sensors and on/off valves, as stated in (Machado et al., 2007). In the normal operation mode the system works as follows.

- Tank1 is filled with two solutions by opening valves V1 and V2.
- The mixer starts working in order to promote the dilution.
- After two time units, the heated device is switched on for 20 time units to increase temperature solution. During this period part of the liquid is evaporated and cooled by the condenser. At that point the required liquid concentration has been reached and the heater is switched off.
- When Tank1 is full valves V1 e V2 are closed.
- The remaining liquid is drained to tank2 by opening valve V3.
- The mixing device is switched off when tank1 is empty and V3 is closed.
- The solution stays in tank2 for post-processing, to stay liquid, for 32 time units and then valve V4 is open to empty tank2.

Throughout normal operation mode, the system may malfunction. During evaporation, the condenser may fail: the steam cannot be cooled and the pressure inside the condenser rises. Therefore, the heater must be switched off to avoid the condenser explosion. By doing so, the temperature of tank1 decreases and the solution may become solid and cannot be drained in tank2. Hence, valve V3 must be opened early enough, but after opening first valve V4, for preventing tank2 overflow.

4.2 UML diagrams

The plant and the controller of the system will be modeled separately. Figures 2 and 3 show the class diagrams of the plant and the controller respectively.

Figures 4 and 5 show the statechart diagrams of the plant and the controller respectively.

According to (Baresi and Pezzè, 2001) the statechart diagrams were transformed into a high level and extensions of PetriNets, including object-oriented nets.
GHENeSys net (del Foyo, 2009). For this work the time constraints of the system were disregarded. Some states of statechart diagrams and their corresponding places into the GHENeSys net are summarized on table 1.

<table>
<thead>
<tr>
<th>Input signals of the controller</th>
<th>Place</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tank1 Full/Tank2 Full</td>
<td>T1F/T2F</td>
</tr>
<tr>
<td>Tank1 Empty/Tank2 Empty</td>
<td>T1E/T2E</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output signals of the controller</th>
<th>Place</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnHeater/OffHeater</td>
<td>HOn/HOff</td>
</tr>
<tr>
<td>OnMixer/OffMixer</td>
<td>MOn/MOff</td>
</tr>
<tr>
<td>openV1V2</td>
<td>V1V2opened</td>
</tr>
<tr>
<td>closeV1V2</td>
<td>V1V2closed</td>
</tr>
<tr>
<td>openV3/V4</td>
<td>V3/V4opened</td>
</tr>
<tr>
<td>closeV3/V4</td>
<td>V3/V4closed</td>
</tr>
<tr>
<td>Malf</td>
<td>malf</td>
</tr>
</tbody>
</table>

Table 1: Correspondence between the states of statechart diagrams and their respective places in the Petri nets.

Figures 6 and 7 show the nets of the plant and the controller respectively. The model was edited using the GHENeSys environment (Salmon et al., 2011).

5 Verifying the system requirements

Representation and verification of requirements using invariants is an approach that has been widely discussed. The verification method based on invariants has some advantages compared with other methods of verification like the model checking.

Model checking is a technique for verifying systems which automatically check the validity of a property of a modeled system. It has a number of advantages over traditional approaches based on simulation, testing, and deductive reasoning. However, model checking is limited by the state explosion problem, which occurs in large systems. In such cases, as state variables in the system increases, the size of its state space grows exponentially. The main challenge in model checking is therefore to deal with the state space explosion problem during the state space generation.

Invariants are used for the analysis of some structural properties of Petri nets, as well as for the verification of system requirements. Unlike
model checking, in the invariant-based verification there is no need to compute all reachable states of the system, which makes the method computationally efficient. However the verification using invariants is limited by the amount of properties that can be verified. On the other hand, despite the fact that using invariant make it possible to check several structural properties of the net, invariant analysis allows only to decide partially about some properties, like as deadlock, i.e., it finds only necessary or just sufficient conditions.

On the other hand, invariants can also be 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 artefacts and applications. So, the analysis and verification of requirements using invariants can be done before the system is modeled (in design phase), with a low computational cost. This is the main advantage the invariant-based verification in this early phase.

If the properties that need to be checked cannot be represented by invariants, then verification using model checking is more suitable.

This section shows the use of invariants in the property verification of systems modeled in Petri nets. The invariants are used both in the representation and verification of system requirements as presented in (Salmon and Silva, 2012).

In (Salmon and Silva, 2012) invariants were defined before the construction of a Petri net model. In fact they were used in the construction of the Petri net model. Thus, the existence of the desired invariants were ensured in the whole design process. In this method, the amount of requirements that can be verified is limited. Therefore, in this paper we propose a variation on the method presented in (Salmon and Silva, 2012): Invariants continue to be defined before net modeling, as was shown in class diagrams of figures 2 and 3, but it will not be regarded in the synthesis of the Petri net. Instead, the requirements to be verified will be represented as formal rules, transformed into a set of inequalities which represent the invariants place.

To verify the existence of desired invariants we use an algorithm to calculate the invariant, which is based on linear algebra methods. The proposed algorithm obtain the set of vectors that represent the basic solution. This means that any linear combination of these vectors is also a solution of the homogeneous system shown in equation 1. Thus we can obtain all possible invariants.

Following we will verify some of the desirable requirements in the evaporation system shown in figure 1.

5.1 Setting the invariant

In the following we define the requirement of the Evaporator systems, shown in figure 1, which will be verified using Petri net invariants. To do so, we consider the information presented in the class diagrams shown in Figures 2 and 3, besides some additional specifications described in OCL. System specifications can be written as a sum of elements of the marking vector:

\[
\sum_{i=1}^{r} p_i \leq k
\]

Equation 4 means that the sum of tokens in the places \( p_1,\ldots, p_r \) of the Petri net can never exceed the number \( k \). This number \( k \) will depend on the initial marking of the net \( (M_0) \) with \( k \leq m(M_0) \).

In this paper it was just verified those requirements considered the most important for the system, however these properties are not the only one that could be verified using the proposed method.

Table 2 shows specifications in OCL concerning the Plant, whereas in table 3 the invariants of the Plant are defined, corresponding to each specification described in table 2.

<table>
<thead>
<tr>
<th>context</th>
<th>CPaint inv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>OCL specification</td>
</tr>
<tr>
<td>1</td>
<td>(Condenser.SteamProd) implies not (Condenser.Error) and (Condenser.NoSteamProd) or (Tank1.Empty) or (Tank1.Filling) or (Tank1.Overflow)</td>
</tr>
<tr>
<td>2</td>
<td>(Tank2.Empty) or (Tank2.Overflow)</td>
</tr>
</tbody>
</table>

Table 2: OCL specification corresponding to the Plant

<table>
<thead>
<tr>
<th>Id</th>
<th>Invariants</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(M(SteamProd) + M(NoSteamProd) + M(Error) \leq 1)</td>
</tr>
<tr>
<td>2</td>
<td>(M(T1F) + M(T1E) + M(T1Filling) + M(T1Overflow) \leq 1)</td>
</tr>
<tr>
<td>3</td>
<td>(M(T2F) + M(T2E) + M(T2Filling) + M(T2Overflow) \leq 1)</td>
</tr>
</tbody>
</table>

Table 3: Definition of the inequalities that represent invariants of place for the Plant

Table 4 shows specifications in OCL for the Controller, and in table 5 the invariants of the Controller are defined, corresponding to each specification described in table 4.

5.2 Computing the invariants

To verify the accuracy of the equations described in tables 3 and 5 we first compute the invariants, thereafter we verify that the sets of places of each inequality belong to some vector in the solution set of Petri net place invariants.
In this work we have used requirements represented by class and state diagrams characteristic of UML as a start point to insert property analysis - and specifically invariant analysis - in the design process of automated systems, where the holding of the invariant property is assured since the very beginning of the process up to the end. The introduction of requirements is made using OCL as
a complement of the UML representation.

The proposed method is promising and can lead to a consistent way to design (discrete) automated systems. A comparison is made with model checking showing that if timed system is the target the proposed approach can still provide good guidance with a good computational performance. However, model checking allow the verification of a bigger and diversified set of properties specially to real time systems. Thus, the idea here is not to compete with model checking but just show that for a class of system good results could be obtained with less effort.

However, the great advantage or the proposal is to be use invariants since the very beginning of the process, introducing it in requirements analysis. In this work we used the canonical representation based on UML complemented by OCL. In further work it would be a good idea to explore object-oriented representations with a structured semantic such as the one used in the KAOS system. In such case the synthesis of a Petri net could be not so easy, but on the other hand the use of invariants could fit even better in the design process.

Acknowledgments

The authors Arianna Z. Olivera Salmon and P. M. G. del Foyo, would like to thank the Brazilian research funding agencies CAPES and CNPq which partially support this research through scholarship programs.

References


