A Straightforward Introduction to Formal Methods Using Coloured Petri Nets

Franciny Medeiros Barreto, Joslaine Cristina Jeske de Freitas,
Michel S. Soares and Stéphane Julia
Universidade Federal de Uberlândia - Faculdade de Computação, Uberlândia, Minas Gerais, Brazil

Keywords: Formal Methods, Coloured Petri Nets, Modelling.

Abstract: Coloured Petri Nets (CPN) arose from the need to model very large and complex systems, which are found in real industrial applications. The idea behind CPN is to unite the ability to represent synchronization and competition for resources of Petri nets with the expressive power of programming languages, data types and diverse abstraction levels. Through this union, systems which study was previously impractical have become amenable to study. The objective of this paper is to present a formal modeling of the Health Watcher System applying the concepts of CPN using CPN Tools. Using a graphical language such as CPN often proves to be a helpful didactic method for introducing formal methods. This paper presents a brief introduction to Coloured Petri Nets, and illustrates how the construction, simulation, and verification are supported through the use of CPN Tools.

1 INTRODUCTION

With the explosive involvement of technology in the lives of humans, it is extremely important to design software systems which are free from errors and are able to satisfy their users in terms of performance, efficiency, correctness, and easy of use. If due to certain reasons, these systems are designed and implemented incorrectly the cost of correcting these errors becomes enormous. Thus, it is very important to have a mechanism where design errors can be discovered and corrected early (OMG, 2011).

Many modelling languages were proposed in past years to design software systems. Currently UML is one of the most used languages for software and systems modelling. UML (Unified Modelling Language) (Booch et al., 2005) is a graphical language for visualizing, specifying, constructing, and documenting information about software-intensive systems. The language gives us a standard way to write a system's view, covering conceptual aspects such as business processes and system functions, as well as elements such as classes to be implemented in a specific programming language, database schemas, and reusable software components.

UML is a semi-formal modelling language, making it difficult to analyze semantics and to verify correctness of a system. Therefore, it is necessary that a formal approach is applied. Various authors have argued the advantages of such formal modelling languages: they reduce the vagueness and ambiguity of informal descriptions, they allow for validation of completeness and consistency through formal proofs, and they bridge the gap between the informal model and the design of a system (Sommerville, 2010), (Thayer et al., 2002). However, other authors believe that formal languages suffer from problems which severely limit their practical usefulness: they are often not expressive enough to deal with real world applications, formal models are complex and hard to read, and constructing a formal model is a difficult, error prone and expensive process (Hall, 1990).

A Petri net is a formal language that allows the modelling of systems, using as a foundation a strong mathematical background. The language has the particularity to enable modelling parallel, concurrent, asynchronous and non-deterministic characteristics of systems (Murata, 1989). Like other industry standards such as UML Activity Diagrams or BPMN, Petri nets offer a graphical notation for stepwise processes that include choice, iteration, and concurrent execution. Unlike these standards, Petri nets have an exact mathematical definition of their execution semantics, with a well-developed mathematical theory for process analysis using different methods. However, when ordinary Petri nets are used for the mod-

Medeiros Barreto F., Cristina Jeske de Freitas J., S. Soares M. and Julia S..
A Straightforward Introduction to Formal Methods Using Coloured Petri Nets.
DOI: 10.5220/0004861901450152
In *Proceedings of the 16th International Conference on Enterprise Information Systems* (ICEIS-2014), pages 145-152
ISBN: 978-989-758-028-4

eling process, the size of very large and complex systems became an issue of major complication. Then, the concept of CPN arose from the need to represent these complex systems, which are found in real industrial applications.

The idea of CPN is to unite the ability to represent synchronization and competition of Petri nets with the expressive power of programming languages with their data types. Through this union, systems which study was previously impractical have become amenable to study (Jensen and Kristensen, 2009). CPN is a language for the modelling and validation of systems in which concurrency, communication, and synchronization play a major role (Aalst et al., 2013). CPN is a discrete-event modelling language combining Petri nets with the functional programming language Standard ML (Milner et al., 1997).

Recent examples of applications using CPN can be found in (Noguera et al., 2010), in which the authors provide a mapping between UML and Web Ontology Language (OWL), through a set of mapping rules. This mapping, which results in a formalization of collaborative processes, also sets a basis for subsequent construction of executable models using the CPN formalism. For this purpose, the authors also provide appropriate mappings from OWL-based ontological elements into CPN elements. In (Kolr and Kvetonov, 2012), the paper presents how the whole public transport system can be described by a Petri net model, in particular by CPN Tools. In (Weidlich et al., 2013), the authors take up the challenge of modelling event processing networks using CPN. They outline how this type of system is modelled and illustrate the formalization with the widely used showcase of the Fast Flower Delivery Application (FFDA). Finally, they show how the net of the FFDA is employed for analysis with CPN Tools.

The importance of validating software requirements has been widely discussed (Boehm, 1984), (Queralt and Teniente, 2012), (Michael et al., 2011). The advantages of providing formal validation for software has been extensively explored (Berard et al., 2010), (Goknil et al., 2010). In recent research, the authors show how to validate UML models using CPN Model. For example, in (Laxman, 2013), the author defends the thesis that it is possible to validate UML models for interactive systems with CPN and the SPIN model checker. In (Ribeiro and Fernandes, 2009), the validation of scenario-based business requirements with CPN is presented.

It is possible to realize that CPN is a general purpose modelling language, i.e., it is not focused on modelling a specific class of systems, but aimed towards a very broad class of systems that can be characterized as concurrent systems. The objective of this paper is to present a formal modelling of the Health Watcher System (Soares et al., 2002) applying the concepts of CPN using the CPN Tools. Using a graphical language such as CPN often proves to be a helpful didactic method for introducing formal methods.

The requirements document for the Health Watcher (HW) System was specified in 2002 (Soares et al., 2002). Since then, the requirements, design and implementation for HW have been used in several studies, but none of them using a formal method. For example, in (Cavalcante et al., 2012) the system was the case study with focus on cloud computing. Another example is presented in (Dyer et al., 2012), in which the authors show an empirical study using quantitative metrics to evaluate the Health Watcher System. Additional examples of the use of Health Watcher System can be found in (d'Amorim and Borba, 2010), (Preece, 2010), (Dai, 2009) and (Siy et al., 2007).

This paper presents a brief introduction to CPN, and illustrates how the construction, simulation, and state space analysis are supported by the use of CPN Tools. The main contribution of this paper is to show that it is possible to use CPN Tools for modelling and validation of information systems without initially modelling using UML and then validate the model in CPN. Therefore, CPN are used not only for the verification and validation activities, but also for the modeling activities of a software system.

The reminder of this paper is as follows. Section 2 introduces the concepts of CPN. Section 3 illustrates the steps to create a model of CPN from the requirements of the Health Watcher System presented in (Soares et al., 2002) using CPN Tools. Section 4 shows how the simulation of the Health Watcher model is supported. Section 5 shows results obtained from the monitoring. Section 6 presents the formal verification of the model. Finally, section 7 concludes the paper and provides references for additional works concerning the modelling language for CPN, practical examples, and use of tools for Petri nets.

2 CONCEPTS OF CPN

CPN is a graphical modelling language (Jensen and Kristensen, 2009), which combines the strengths of Petri nets (Wolfang Reisig, 2013) and of functional programming languages (Milner et al., 1997). The formalism of Petri nets is well suited for describing concurrent and synchronizing actions in distributed systems. Programming languages can be used to de-

fine data types and manipulation of data. An introduction to the practical use of CPN can be found at (Kristensen et al., 2004).

The CPN are designed to reduce the size of the model, allowing individualization of tokens, using colours assigned to them, so different processes or resources can be represented in the same network. Colours do not mean just colours or patterns. They can represent complex data types (Jensen and Kristensen, 2009).

CPN models are formal, in the sense that the CPN modelling language has a mathematical definition of its syntax and semantics. This means that they can be used to verify system properties, i.e., prove that certain desired properties are fulfilled or that certain undesired properties are guaranteed to be absent.

Large and complex models can be built using hierarchical CPN in which modules, which are called pages in the CPN terminology, are related to each other in a well-defined way. Without the hierarchical structuring mechanism, it would be difficult to create understandable CPN models of real-world systems (Jensen and Kristensen, 2009).

2.1 Formal Definition of Coloured Petri Net

The formal definition of a CPN is as follows (Jensen and Kristensen, 2009):
A Coloured Petri Net is a tuple CPN = $(\sum, P, T, A, N, C, G, E, I)$ satisfying the following requirements

(i.) $\sum$ is a finite set of non-empty types, called colour sets.

(ii.) P is a finite set of places.

(iii.) T T is a finite set of transitions.

(iv.) A is a finite set of arcs such that:

- $P \cap T = P \cap A = T \cap A = \emptyset$.

(v.) N is a node function. It is defined from A into $PXT \cup TXP$.

(vi.) C is a colour function. It is defined from P into $\sum$.

(vii.) G is a guard function. It is defined from T into expressions such that:

- $\forall t \in T: \quad [Type(G(t)) = Bool \wedge Type(Var(G(t))) \subseteq \sum]$

(viii.) E is an arc expression function. It is defined from A into expressions such that:

- $\forall a \in A: \quad [Type(E(a)) = C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \sum]$
  where p(a) is a place of N(a).

(ix.) I is an initialization function. It is derived from P into closed expressions such that:

- $\forall p \in P : [Type(I(p)) = C(p)_{MS}]$.

2.2 Hierarchical CPN

One of the problems presented in Petri nets is associated with the fact that, as the size of the system grows, it becomes increasingly difficult to maintain the clarity of the model.

CPN models can be structured into a number of related modules. This is particularly important when dealing with CPN models of large systems. The module concept of CPN is based on a hierarchical structuring mechanism, which supports bottom-up as well as top-down working style. New modules can be created from existing modules, and modules can be reused in several parts of the CPN model. By means of the structuring mechanism it is possible to capture different abstraction levels of the modeled system in the same CPN model.

2.3 CPN Tools

The practical application of CPN modelling and analysis heavily relies on the existence of computer tools supporting the creation and manipulation of models. CPN Tools is a tool suite for editing, simulating, providing state space analysis, and providing performance analysis of CPN models. It is currently licensed to more than 4000 users in more than 100 different countries and is available both for MS-Windows and Linux platforms. The user of CPN Tools works directly on the graphical representation of the CPN model. The graphical user interface (GUI) of CPN Tools has no conventional menu bars or pull-down menus, but is based on interaction techniques, such as tool palettes and marking menus. A license for CPN Tools can be obtained free of charge via the CPN Tools web pages http://www.cpntools.org.

3 CPN MODEL FOR HEALTH WATCHER

The purpose of the system is to collect and then manage public health related complaints and notifications. The system is also used to notify people about important information regarding the Health System.

System users are employees of the Department of Health and any citizen who wants to interact with the system. A citizen can access the system through the

Internet and make their complaint or request information about the different health service sectors available. In the event of a complaint being made, it will be registered on the system and addressed to a specific department. This department will be able to handle the complaint in an appropriate manner and give a suitable response to the client once the complaint has been dealt with. This response will be registered on the system and available to be queried. The system should provide access to 20 users simultaneously.

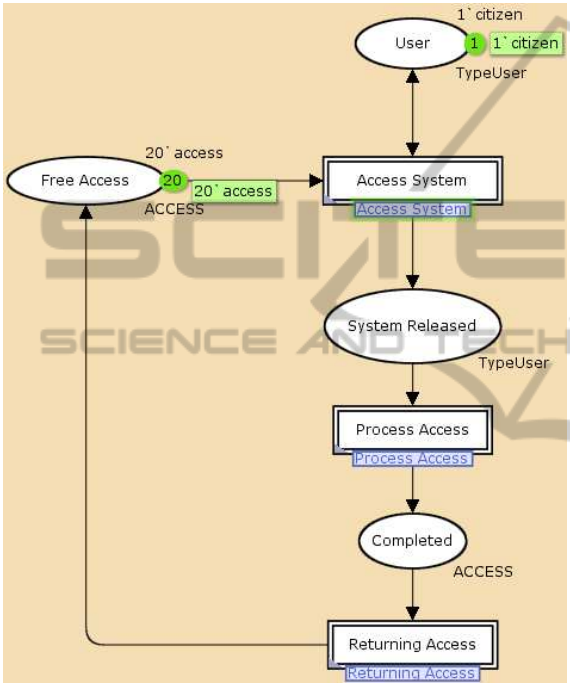

Figure 1: The most abstract level of modelling for Health Watcher.

The modelling approach is hierarchical, where each transition can be replaced by a module with details of the activities associated with the transition. The most abstract level of the modelling of the Health Watcher system is depicted in figure 1.

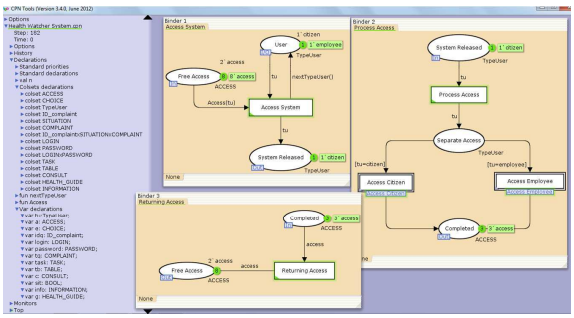

Figure 2: Module: Access System, Process Access and Returning Access.

As depicted in figure 1, the colset *ACCESS* is a set of colours that contains only 20 tokens of the *access* type as the system should only allow 20 concurrent accesses. The colset *UserType* is a set of colours that contains tokens of types *citizen* and *employee*. The transition *Access System* will be ready for binding when there is at least one free access and one user to use the system.

Figure 2 depicts, in details, three modules *Access System*, *Process Access* and *Returning Access*.

The transition *Access System* is enabled when there is a token type *access* and another token type *UserType*. When the transition is binding, then the two tokens are consumed and then they generate a new token with function *NextTypeUser* in place *User*, and other token in place *System Released*.

When there is a token in place *System Released*, the transition *Process Access* is enabled. When the transition *Process Access* is binding, the token is consumed and a token is produced in place *Separate Access*. In this case, the transition to be enabled depends on the type of token produced. The guard condition *tu = citizen* or *tu = employee* enables the correct transition. Besides, when the task ends - place *Completed* - the token is returned to the place *Free Access*.

There are several scenarios defined by the specification of the requirements of the Health Watcher System and we can not show them all due to space limitations. Therefore, the authors chose the scenario in which the citizen accesses the system to make a complaint.

3.1 Complaint Specification

According to the work presented in (Soares et al., 2002), the main flow of events for the complaint specification is:

1. The citizen logs into the system; (figure 3)
2. The citizen chooses the option *Specify Complaint*; (figure 4)
3. The citizen informs the system of the type of complaint being made and reports data; (figure 5)
4. The system records the complaint; (figure 6)
5. Access is released. (figure 7)

The sequence of figures 3, 4, 5, 6, 7, show the modelling of the scenario when the citizen decides to make a complaint.

4 SIMULATION

A CPN model of a system describes the states of the system and the events (transitions) that can cause the

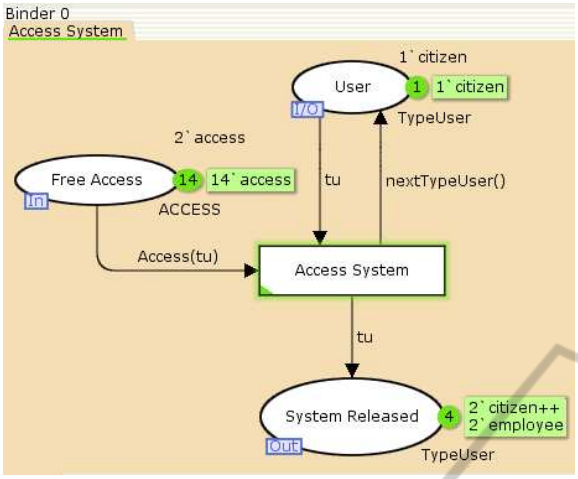

Figure 3: The citizen logs into the system.

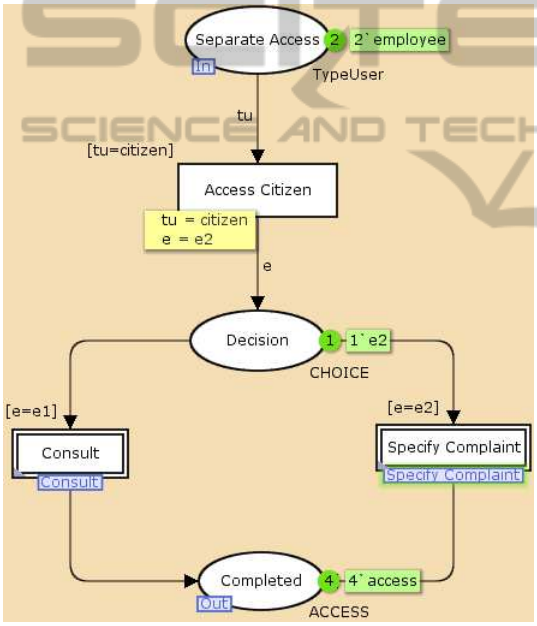

Figure 4: The citizen chooses the option Specify Complaint.

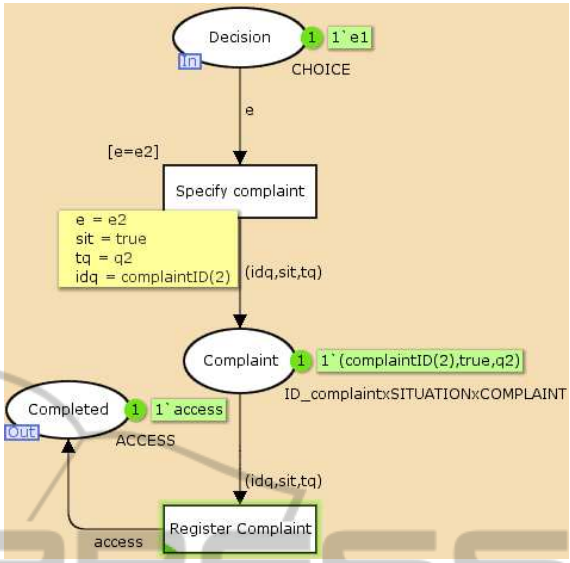

Figure 5: The citizen informs the system of the type of complaint being made and reports data.

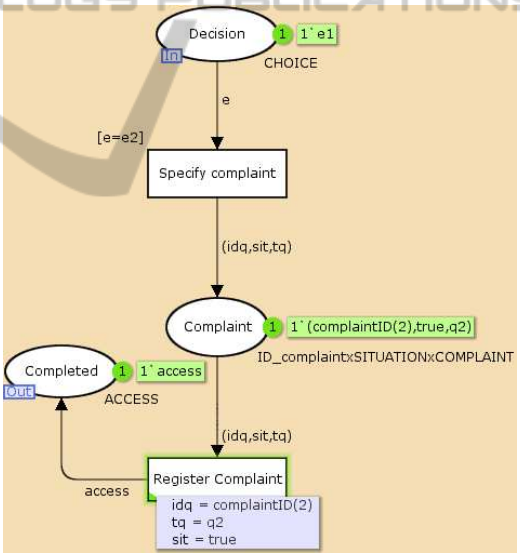

Figure 6: The system records the complaint.

system to change state. By making simulations of the CPN model, it is possible to investigate different scenarios and to explore the behaviors of the system. Very often, the goal of simulation is to debug and to investigate the system design. CPN nets can be simulated interactively or automatically. An interactive simulation is similar to single step debugging. It provides a way to "walk through" a CPN model, investigating different scenarios in detail and checking whether the model works as expected. During an interactive simulation, the modeler is in charge and determines the next step by selecting between the enabled events in the current state.

Before and after an automatic simulation, the current marking and the enabled transitions are displayed as described for the interactive mode. However, the token game is not displayed during automatic simulations. Of course, this typically constitutes less information than desired. A straightforward possibility to obtain information about "what happened" is to use the simulation report, which is a textual file containing detailed information about all the bindings of transitions that occurred. Figure 8 shows a simulation report of the first 8 steps from an automatic simulation of the Health Watcher System.

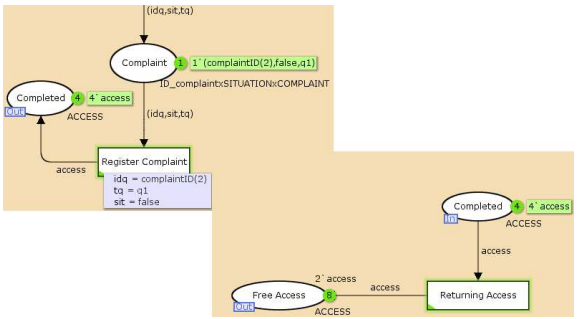

Figure 7: Access is released.

```
------------------------------------------------------------------
1     0    Access_System   @ (1:Access_System)    - tu = citizen
2     0    Process_Access  @ (1:Process_Access)   - tu = citizen
3     0    Access_System   @ (1:Access_System)    - tu = employee
4     0    Access_System   @ (1:Access_System)    - tu = citizen
5     0    Access_System   @ (1:Access_System)    - tu = employee
6     0    Access_Citizen  @ (1:Access_Citizen)   - tu = citizen
                                                    - e = e2
7     0    Access_System   @ (1:Access_System)    - tu = employee
8     0    Access_System   @ (1:Access_System)    - tu = citizen
```

Figure 8: Partial sample simulation report.

5 MONITORING

In CPN Tools, monitors can be used to examine the binding elements that occur and the markings that are reached during a simulation. Different kinds of monitors can be used for different purposes, and breakpoint monitors can be used to stop simulations when specific conditions are fulfilled. A transition enabled monitor is a standard breakpoint monitor that can be associated with a transition, and the monitor will stop a simulation when the transition is enabled (or disabled, as determined by an option for the monitor).

Note that these statistics have been calculated for data that is not necessarily independent or identically distributed.

Untimed statistics				
Name	Count	Sum	Avrg	Max
Count_trans_occur_Access_Citizen'Access_Citizen_1	535	535	1.000000	1
Count_trans_occur_Access_Employee'Access_Employee_1	508	508	1.000000	1
Count_trans_occur_Access_System'Access_System_1	1046	1046	1.000000	1
Count_trans_occur_Different_Information'Consult_Different_Information_1	147	147	1.000000	1
Count_trans_occur_Health_Guide'Consult_Health_Guide_1	116	116	1.000000	1
Count_trans_occur_Process_Access'Proccess_Access_1	1044	1044	1.000000	1
Count_trans_occur_Register_Table'Register_Table_1	497	497	1.000000	1
Count_trans_occur_Specify_Complaint'Specify_complaint_1	272	272	1.000000	1
Count_trans_occur_Update_Complaint'Update_Complaint_1	485	485	1.000000	1

Simulation steps executed: 10000
Model time: 0

Figure 9: Monitoring Report.

The monitoring results are obtained during the running of the simulation. After 10000 simulation steps, a result of the monitoring is obtained and can be seen in figure 9.

The result shows that the system was accessed 1046 times during the simulation, which was made up of 508 employees and 535 citizens. However, monitoring shows that access allows employees to perform several different tasks. For example, there were 497 activities related to the Register table, 485 activities opinion of complaints, totaling 982 tasks associated to the employee, and only 535 citizen - 116 activities related to consult health guide, 147 activities related to consult different information and 272 activities related to specify the complaint.

6 STATE SPACE ANALYSIS

CPN models are formal, in the sense that the CPN modelling language has a mathematical definition to its syntax and semantics. This means that they can be used to verify system properties, i.e., prove that certain desired properties are fulfilled or that certain undesired properties are guaranteed to be absent. Verification of system properties is supported by a set of state space methods. The basic underlying idea of state spaces is to compute all reachable states and state changes of the CPN model and represent these as a directed graph where nodes represent states and arcs represent occurring events. State spaces can be constructed entirely automatic. From a constructed state space it is possible to answer a large set of verification questions concerning the behavior of the system such as absence of deadlocks, the possibility of always being able to reach a given state, and the guaranteed delivery of a given service.

```
Statistics
------------------------------------------------------------------
State Space                                 Scc Graph
     Nodes:  17089                             Nodes:  13131
     Arcs:   58108                             Arcs:   44343
     Secs:   103436                            Secs:   20
     Status: Partial
```

Figure 10: State space report: statistics.

For the CPN model in Figure 1, the state space report is the same as that shown in Figures 10 - 13. First we have some state space statistics (see Fig.10) telling how large the state space is. For the Health Watcher System we have 17089 nodes and 58108 arcs. Statistics about the SCC-graph are also received. It has 13131 nodes and 44343 arcs.

The next parts of the state space report contain information about the boundedness properties. The boundedness properties give information as to how many (and which) tokens a place may hold, when all reachable markings are considered.

Figure 11 specifies the best upper and lower integer bounds. The best upper integer bound of a place specifies the maximal number of tokens that can re-

```
Boundedness Properties
------------------------------------------------------------------
  Best Integer Bounds
                                              Upper        Lower
     Access_Employee'Decision_Task 1              1            0
     Access_Employee'Login 1                      2            0
     Access_citizen'Decision 1                    4            0
     Register_Table'Record 1                      1            0
     Register_Table'Record_Made 1                 1            0
     Consult'Consult_Type 1                       3            0
     Consult_Health_Guide'Health_Guide 1          2            0
     Consult_Different_Information'
              Different_Information 1             2            0
     Specify_Complaint'Complaint 1                3            0
     Update_Complaint'Record_Made 1               1            0
     Update_Complaint'Search_Complaint 1          1            0
     Proccess_Access'Separate_Access 1            4            0
     Top'Completed 1                              2            0
     Top'Free_Access 1                            4            0
     Top'System_Released 1                        4            0
     Top'User 1                                   1            1
```

Figure 11: State space report: integer bounds.

side on a place in any reachable marking. The best upper integer bound of place Top'User is 1 which means that there is at most one token in place Top'User, and there exists reachable markings where there is one token in Top'User.

The best lower integer bounds for a place specifies the minimal number of tokens that can reside in the place on any reachable marking. The place Top'User has a best lower integer bound of 1 which means that there is always at least one token in this place. When the best upper and lower integer bound are equal it implies that the place always contains the same number of tokens.

```
Home Properties
------------------------------------------------
  Home Markings
     None
```

Figure 12: State space report: home properties.

Figure 12 shows the part of the state space report that specifies the home properties. The home properties tell us that there does not exist a single home marking.

```
Liveness Properties
------------------------------------------
  Dead Markings
     None
  Dead Transition Instances
     None
  Live Transition Instances
     None
```

Figure 13: State space report: liveness properties.

The liveness properties in Figure 13 specify that there is not a single dead marking. A dead marking is a marking in which no binding elements are enabled.

Figure 13 specifies that there are no live transitions. A transition is live if from any reachable marking we can always find an occurrence sequence containing the transition. In other words, we cannot do things which will make it impossible for the transition to occur afterwards.

Figure 13 also specifies that there are no dead transitions. A transition is dead if there are no reachable markings in which it has been enabled. There are no dead transitions means that each transition in the Health Watcher System has the possibility to occur at least once. If a model has dead transitions then they correspond to parts of the model that can never be activated. Hence, we can remove dead transitions from the model without changing its behavior.

7 CONCLUSION

Coloured Petri nets are a formal method in which models depicting the exact functionality of the system are designed, simulated and analyzed. This is a technique with a lot of research done to prove the correctness of a vast variety of systems.

The focus of this paper was the modelling and simulation of Coloured Petri Networks using the CPN Tools. Normally, critical systems are modelled through the use of formal methods. However, the modelling of an information system, such as the Health Watcher System, provides a different perspective to formal modelling. The CPN Tools unite all the properties of a CPN, which makes its use extremely practical and powerful, leaving the user with only the job of how to create the semantic model. Another factor which favors the use of the tool is the ease by which it can be obtained. The CPN Tools is distributed free of charge, unlike other simulation software.

As an important approach to modelling, UML has been successfully applied in many fields of software engineering. However, because of the semantic gap, UML is hard to be checked, which can cause disastrous consequences for the system. CPN has a precise mathematical semantics and automatic verification tools. Using CPN Tools, it is possible to investigate the behaviour of the modelled system using simulation, to verify properties by means of state space methods and model checking, and to conduct simulation-based performance analysis. Therefore, CPN is an interesting modelling approach because it is capable of describing the software detects errors and obtaining greater confidence of the correctness of the model.

REFERENCES

Aalst, W. M., Stahl, C., and Westergaard, M. (2013). Strategies for modeling complex processes using colored petri nets. *Transactions on Petri Nets and Other Models of Concurrency*, 7:6–55.

Berard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., and Schnoebelen, P. (2010). *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer Publishing Company, Incorporated, 1st edition.

Boehm, B. W. (1984). Verifying and validating software requirements and design specifications. *IEEE Softw.*, 1(1):75–88.

Booch, G., Rumbaugh, J., and Jacobson, I. (2005). *Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional.

Cavalcante, E., Almeida, A., Batista, T., Cacho, N., Lopes, F., Delicato, F. C., Sena, T., and Pires, P. F. (2012). Exploiting software product lines to develop cloud computing applications. In *Proceedings of the 16th International Software Product Line Conference - Volume 2*, SPLC '12, pages 179–187, New York, NY, USA. ACM.

Dai, L. (2009). Security variability design and analysis in an aspect oriented software architecture. In *Proceedings of the 2009 Third IEEE International Conference on Secure Software Integration and Reliability Improvement*, SSIRI '09, pages 275–280, Washington, DC, USA. IEEE Computer Society.

d'Amorim, F. and Borba, P. (2010). Modularity analysis of use case implementations. In *Proceedings of the 2010 Fourth Brazilian Symposium on Software Components, Architectures and Reuse*, SBCARS '10, pages 11–20, Washington, DC, USA. IEEE Computer Society.

Dyer, R., Rajan, H., and Cai, Y. (2012). An exploratory study of the design impact of language features for aspect-oriented interfaces. In *Proceedings of the 11th annual international conference on Aspect-oriented Software Development*, AOSD '12, pages 143–154, New York, NY, USA. ACM.

Goknil, A., Kurtev, I., and van den Berg, K. (2010). Tool support for generation and validation of traces between requirements and architecture. In *Proceedings of the 6th ECMFA Traceability Workshop*, ECMFA-TW '10, pages 39–46, New York, NY, USA. ACM.

Hall, A. (1990). Seven myths of formal methods. *IEEE Softw.*, 7(5):11–19.

Jensen, K. and Kristensen, L. (2009). *Coloured Petri Nets*. Springer.

Kolr, D. and Kvetonov, S. (2012). People transfer in city transport modeled via cpn. In *Proceedings of the 13th international conference on Computer Aided Systems Theory - Volume Part I*, EUROCAST'11, pages 192–199, Berlin, Heidelberg. Springer-Verlag.

Kristensen, L. M., Jrgensen, J. B., and Jensen, K. (2004). Application of coloured petri nets in system development. In *In Lecture on Concurrency and Petri Nets, Jorg Desel, Wolfgang Reisig and Grezegorz Rozenberg (Eds.), Springer, LNCS 3089*, pages 626–685. Springer-Verlag.

Laxman, P. B. (2013). *Validation of UML Models for Interactive Systems with CPN and SPIN*. PhD thesis, Department of Computer Science and Engineering - National Institute of Technology Rourkela.

Michael, J. B., Drusinsky, D., Otani, T. W., and Shing, M.-T. (2011). Verification and validation for trustworthy software systems. *IEEE Softw.*, 28(6):86–92.

Milner, R., Tofte, M., and Macqueen, D. (1997). *The Definition of Standard ML*. MIT Press, Cambridge, MA, USA.

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

Noguera, M., Hurtado, M. V., Rodríguez, M. L., Chung, L., and Garrido, J. L. (2010). Ontology-driven analysis of uml-based collaborative processes using owl-dl and cpn. *Sci. Comput. Program.*, 75(8):726–760.

OMG (2011). *OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1*. Object Management Group.

Preece, J. J. (2010). I persuade, they persuade, it persuades! In *Proceedings of the 5th international conference on Persuasive Technology*, PERSUASIVE'10, pages 2–3, Berlin, Heidelberg. Springer-Verlag.

Queralt, A. and Teniente, E. (2012). Verification and validation of uml conceptual schemas with ocl constraints. *ACM Trans. Softw. Eng. Methodol.*, 21(2):13:1–13:41.

Ribeiro, s. R. and Fernandes, J. M. (2009). Validation of scenario-based business requirements with coloured petri nets. In Boness, K., Fernandes, J. M., Hall, J. G., Machado, R. J., and Oberhauser, R., editors, *ICSEA*, pages 250–255. IEEE Computer Society.

Siy, H., Aryal, P., Winter, V., and Zand, M. (2007). Aspectual support for specifying requirements in software product lines. In *Proceedings of the Early Aspects at ICSE: Workshops in Aspect-Oriented Requirements Engineering and Architecture Design*, EARLYASPECTS '07, pages 2–, Washington, DC, USA. IEEE Computer Society.

Soares, S., Laureano, E., and Borba, P. (2002). Implementing distribution and persistence aspects with aspectj. *SIGPLAN Not.*, 37(11):174–190.

Sommerville, I. (2010). *Software Engineering*. Addison-Wesley, Harlow, England, 9 edition.

Thayer, R., Dorfman, M., and Hunter, R. (2002). *Software Engineering Member Package (4 Volume Set)*. Practitioners. Wiley.

Weidlich, M., Mendling, J., and Gal, A. (2013). Net-based analysis of event processing networks: the fast flower delivery case. In *Proceedings of the 34th international conference on Application and Theory of Petri Nets and Concurrency*, PETRI NETS'13, pages 270–290, Berlin, Heidelberg. Springer-Verlag.

Wolfang Reisig (2013). *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Springer. 230 pages; ISBN 978-3-642-33277-7.