Prosega/CPN: An Extension of CPN Tools for Automata-based Analysis and System Verification

[1]*J.C. Carrasquel <julio.carrasquel@yahoo.com>*
[2]*A. Morales <ana.morales@ciens.ucv.ve>*
[3]*M.E. Villapol <maria.villapol@aut.ac.nz>*
[1]*La Sapienza University of Rome, Department of Computer, Control, and Management Engineering, Via Ariosto 25, Rome, 00185, Italy*
[2]*Central University of Venezuela, School of Computer Science, Av. Paseo Los Ilustres, Caracas, 1040, Venezuela*
[3]*Auckland University of Technology, School of Engineering, Computer and Mathematical Sciences, 55 Wellesley Street East, Auckland, 1010, New Zealand*

Abstract. The verification and analysis of distributed systems is a task of utmost importance, especially in today's world where many critical services are completely supported by different computer systems. Among the solutions for system modelling and verification, it is particularly useful to combine the usage of different analysis techniques. This also allows the application of the best formalism or technique to different components of a system. The combination of Colored Petri Nets (CPNs) and Automata Theory has proved to be a successful formal technique in the modelling and verification of different distributed systems. In this context, this paper presents Prosega/CPN (*Protocol Sequence Generator and Analyzer*), an extension of CPN Tools for supporting automata-based analysis and verification. The tool implements several operations such as the generation of a minimized deterministic finite-state automaton (FSA) from a CPN's occurrence graph, language generation, and FSA comparison. The solution is supported by the Simulator Extensions feature whose development has been driven by the need of integrating CPN with other formal methods. Prosega/CPN is intended to support a formal verification methodology of communication protocols; however, it may be used in the verification of other systems whose analysis involves the comparison of models at different levels of abstraction. For example, business strategy and business processes. An insightful use case is provided where Prosega/CPN has been used to analyze part of the IEEE 802.16 MAC connection management service specification.

Keywords: formal methods; coloured Petri nets; CPN tools; finite-state automata; protocol verification.

DOI: 10.15514/ISPRAS-2018-30(4)-7

For citation: Carrasquel J.C., Morales A., Villapol M.E. Prosega/CPN: An extension of CPN Tools for Automata-based Analysis and System Verification. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 4, 2018. pp. 107-128. DOI: 10.15514/ISPRAS-2018-30(4)-7

1. Introduction

The verification of distributed systems and the assurance of their correctness is a task of utmost importance; specially in today's world where many critical services are completely supported by computer technologies. Among the solutions for system modelling and verification, Petri Nets [1] play a major role since its capability of graphically visualize systems, and for maintaining the formal rigor, so it allows to perform a convenient analysis of the behavioral properties of a system. Thus, the formalism of Petri Nets has been extended to other models in order to enrich their expressiveness and practicability. Particularly, we consider Coloured Petri Nets (CPNs) [2] where data types (*colors*) may be associated to net elements. CPN Tools [3] is a consolidated software tool for editing, simulating, and analyzing CPN models.

However, when dealing with a higher complexity of the system, it may be useful to combine the usage of different analysis techniques. This also allows the application of the best formalism or technique to different components of a system. In the context of Colored Petri Nets, the last version of CPN Tools includes the Simulator Extensions whose development has been driven by the need of integrating CPN with other formal methods [4]. In particular, we consider the integration of CPNs and Finite-state Automata (FSA) which has been proved to be useful for the validation of different protocols and communication systems [5] [6] [7].

For instance, given a CPN's occurrence graph (OG), the arcs through a path in the OG may be seen as the sequence of service primitives that a user (i.e. another system entity in a higher layer) invokes in order to request some action by a service provider. The nodes in the OG may be considered as changes of state in the system due to the services invocations. Finally, some nodes of the OG may represent halt states, meaning the termination of a specific process. Hence, the OG can be seen as a FSA, which can be analyzed using well-known algorithms and theorems.

There are several tools for building, combining, optimizing, and searching Finite-state Automata. However, in order to apply them for analyzing CPNs and occurrence graphs, these ones must be converted into FSA specific formats (i.e. see [5] [6]). Using several tools may complicate the verification process.

Thereby, we developed a solution called Prosega/CPN (*Protocol Sequence Generator and Analyzer*). The tool aims to bridge conveniently the formalism of CPNs with Finite-state Automata, taking advantage of the Simulator Extensions feature in CPN Tools. Thus, the software provides a mechanism for transforming a CPN's occurrence graph into a minimized deterministic FSA as well as other operations for language generation and FSA comparison. Prosega/CPN has been conceived to support the protocol verification methodology proposed by Billington [8]. However, the tool may be useful to support the verification of other systems whose strategy may involve the usage of FSAs, or the comparison of models at different levels of abstraction; for example, business strategy and business processes.

The remainder of this paper is structured as follows. Section 2 introduces the literature related to our work. Section 3 presents some formal definitions for understanding the models managed by Prosega/CPN. Sections 4 and 5 describe the tool functionalities and architecture respectively. Section 6 describes a use case where the tool has been used to analyze part of the IEEE 802.16 MAC connection management service specification. Finally, Section 7 presents the conclusions.

2. Related work

Prosega/CPN has been developed within the context of system verification through the formalism of Coloured Petri Nets (CPNs) and Finite-state Automata (FSA). The tool has been conveniently developed as an extension of CPN Tools [3] since it performs several operations on FSAs generated from a CPN model. i.e. the reduction of a CPN's occurrence graph into a FSA. Hence, through the development of Prosega/CPN we have been focused in three topics within the literature:

- Works dealing with the development of extensions for CPN Tools [4] [9] [10] [11].
- Tools and other solutions for the analysis and manipulation of FSA [12] [13] [14] [15] [16].
- Works proposing a system verification methodology using CPNs and FSA, and the use cases in which it has been applied [5] [6] [7] [8] [17], and other scenarios where both formalisms have been used together [18] [19] [20].

CPN tools has a history for communicating with external solutions; its architecture provides a set of communication primitives for connecting external software to the CPN simulator engine. As an initial effort, it was developed Comms/CPN [9], a library for Java and C/C++ which makes it possible for CPN Tools to communicate based on TCP/IP with external application and processes. The BRITNeY Suite [10] is other solution which provides model visualizations in an external tool, and more recently Access/CPN [11] that provides a channel to interact with the CPN Tools simulator engine from external Java programs. However, while these previous tools have made it easy to *interact* with CPN Tools, they have not made it possible to *extend* the software. Thereby, it was developed the Simulator Extensions [4] feature included in the last version of CPN Tools. This component provides a mechanism for adding new functionalities within the CPN Tools Graphical User Interface (GUI), thereby allowing integrating other related formalisms with CPN models; as a result, it has been possible to handle other models in the tool such as low-level Place/Transition nets, Declare models, and drawing message sequence charts from model executions [4].

On the other hand, Finite-state Automata (FSA) have been used in a much wider spectrum of fields than CPNs; as an important tool for FSA manipulation we highlight the FSM Library from AT&T Labs [12] which is a collection of Unix software tools for creating and manipulating finite-state machines. Despite the library is quite general purpose, it was designed for speech processing applications such as speech recognition/synthesis; FSM Library was used as well in previous works regarding the verification of communication systems based on CPNs and automata [5] [6]. Some of the researchers of the AT&T FSM project developed later an enhanced version called OpenFST [13], which is an open-source alternative that also allows to construct finite-state transducers, and it provides a C++ template library. Within the range of tool solutions for FSA manipulation, we may also find Foma [14], the FAdo project [15] and the specialized pedagogical tool JFLAP [16] among many others.

Bridging CPNs and FSA may be useful for verification of systems of very high complexity. In particular, Billington [8] proposed a CPN and FSA approach for the verification communication systems that has proven to be successful; namely, in the verification of the Resource Reservation Protocol (RSVP) [5], the Wireless Application Protocol (WAP) [6], the Transmission Control Protocol (TCP) [7], and the Internet Open Trading Protocol (IOTP) [17], among other cases. Between other domains in which both formalisms have been applied together we may find the verification of web-services composition [19] [20] or vehicular traffic control systems [18], just to mention a few.

3. Formal Definitions

This section presents some formal definitions of the models and data structures that are manipulated through the functionalities of CPN Tools and Prosega/CPN. In particular, it is formulated how it can be derived an occurrence graph (OG) from a CPN model, and afterwards is explained how can it be generated a Finite-state Automaton (FSA) from a CPN's occurrence graph. The following formulations are based in the work done in [8]. Albeit CPNs are managed in this work; for the formal definition it has been rather convenient to generalize the type into a High-level Petri Net (i.e. for proving further theorems regarding the relationship between an OG and a FSA as described in [8]). Hence, we firstly take the definition of a High-level Petri net (HLPN) [21].

Definition 1. *A High-level Petri Net is a structure of the form* $HLPN = (P, T, D; Type, Pre, Post, m_0)$ *where:*

- P*is a finite set of Places;*
- T*is a finite set of Transitions such that*$P \cap T = \emptyset$
- D*is a non-empty finite set of non-empty domains where each element of* D *is called a type.*
- $Type: P \cup T \rightarrow D$*is a function used to assign types to places, and to determine transition modes.*
- $m_0 \in \mu PLACE$*is a multi-set called the initial marking of the net such that* $\mu PLACE$*is a set of all possible multi-sets of* $PLACE$
- $Pre, Post: TM \rightarrow \mu PLACE$*are the pre and post mappings with*
    - $TM = \{(t, m) | t \in T, m \in Type(t)\}$*the set of transition modes.*
    - $PLACE = \{(p, g) | p \in P, g \in Type(p)\}$*the set of elementary places.*

For the analysis of a High-level Petri net it is generated an occurrence graph (OG). We consider that an OG can be defined as a labelled and rooted directed graph, where the nodes of the graph represent *markings* of the Petri Net, and the directed arcs represent the *transition modes* (or binding elements [2]) that can *occur* in all executions from the initial marking. On the other hand, the root of the graph refers to a node, which is considered as the initial state. In addition, the arcs of an OG may be labelled by the transition modes. Thus, we start by defining a labelled and rooted directed graph, and then we give the definition of an OG associated to a HLPN.

Definition 2. *A labelled directed graph, with* v_0*as the root node, is a triple* $G = (V, L, E)$ *where:*

- V*is a finite set of vertices or nodes;*$v_0 \in V$*represents the root or initial node.*
- L*is a set of labels;*
- $E \subseteq V \times V$*is a set of labelled directed edges.*

Definition 3. *An occurrence graph of a HLPN with an initial marking*v_0*is a labelled and rooted directed graph*$OG = (V, TM, A)$*where*

- V*is a finite set of vertices or nodes reachable from*m_0*(the reachability set);* $m_0 \in V$*represents the initial marking (root node);*
- TM*is the set of transition modes of the HLPN;*
- $A = \left\{(m, tm, m') \in V \times TM \times V' | m \overset{tm}{\rightarrow} m'\right\}$*is the set of arcs (directed edges) labelled by transition modes.*

Remark. $m \overset{tm}{\rightarrow} m'$*indicates the ocurrence of a transition mode tm* $\in$ *TM in a marking*m*which results in a new marking*m'

However, when we are only interested in the transition names, then the arcs of the OG are just labelled with such transitions names rather than the transition modes (binding elements). For example this is useful when it is just required to understand which user observable events (service primitives) may lead from a state of the system to another one; instead of transition modes which involve the parameters binded to such events. In addition, when we are also interested in the identification of the markings for the nodes of the OG, rather than the marking details, we introduce an injection $I: [m_0] \rightarrow \mathbb{N}$such that this function maps the set of reachable markings from m_0(denoted as$[m_0]$) into the set of natural numbers. Giving the described abstractions for transitions and markings, we consider the definition of an abstract OG.

Definition 4. *An abstract OG of a HLPN with an initial marking* m_0 *is a labelled and rooted directed graph*$OG = (V, T, A)$*where*

- $V = \{I(m) | m \in [m_0]\}$ *is the set of nodes;*
- $I(m_0) \in V$ *represents the root or initial node.*
- T*is the set of transitions of the HLPN ;*
- $A = \left\{\left(I(m), t, I(m')\right) \in V \times T \times V' | m \overset{(t,m) \in TM}{\rightarrow} m'\right\}$ *is the set of arcs labelled by transition.*

We point out that the abstract occurrence graph OG defined above is *finite*. i.e. It has a finite number of states. Indeed this is an important fact when dealing with real scenarios. This means that the corresponding Petri Net must be a bounded net [1], and hence a preliminary boundedness analysis on the Petri Net is performed. Finally, it is presented a mapping from an abstract OG (Definition 4) into a Finite-state Automaton FSA. We define a function *Prim: T* → *SP* ∪ {ε} that maps each transition of the HLPN to either an identifier name (i.e. an user observable event or service primitive name), or to an *epsilon* (i.e. an empty move); *SP* is the set of identifiers (for the user observable events or service primitive names) for the system that we are describing.

Definition 5. *A Given an abstract occurrence graph*$OG = (V, T, A)$*it is derived the corresponding Finite-state Automaton*$FSA = (V, SP, A_{SP}, v_0, F)$*where*

- V*is the set of nodes of the abstract OG (the states of the FSA);*
- SP*is the set of identifiers (for the user observable events or service primitive names) of the system of interest (the alphabet of FSA);*
- $A_{SP} = \{(v, Prim(t), v') | (v, t, v') \in A\}$*is the set of transitions labelled by elements of SP or epsilons (the transition relation of the FSA);*
- v_0*corresponds to the abstract initial marking (initial state of the FSA).*
- $F \subseteq V$*the set of final (acceptance) states.*

Prosega/CPN performs the conversion of an OG as described in Definition 4 into a FSA as described in Definition 5. Moreover, this mapping between OG and the FSA allows the tool conveniently manage the generation of the language and the comparison between other FSAs.

4. Functionalities

Prosega/CPN is an extension in CPN Tools. Thus, the user interacts with the application using a Graphical User Interface (GUI) through a tool palette added to CPN Tools (see fig. 1) - available under the Tool box entry [3]. The tool supports the generation of a minimized deterministic Finite-state Automaton (FSA) derived from the CPN's occurrence graph, the language generation, and the comparison between two different FSAs. We proceed to explain these functionalities in detail.

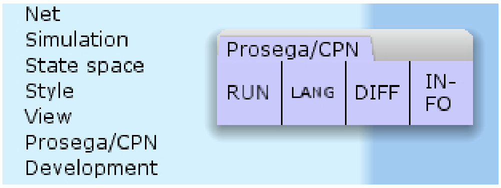
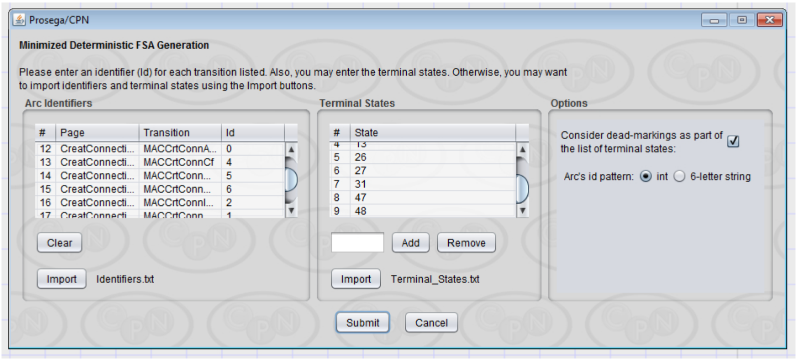


Fig. 1. Tool palette of Prosega/CPN

4.1 FSA Generation

Once the occurrence graph (OG) from a CPN model is generated using the CPN Tools simulator [3], its associated Finite-state Automaton (FSA) can be generated and reduced using the RUN tool (see Fig. 1). To this aim, the following steps are performed: getting the transitions, and dead markings of the OG, assigning identifiers to transitions (i.e. constructing the mapping *Prim* defined in Section 3), reducing the FSA, and displaying the results. Here, we consider the structure of an abstract OG where the nodes are identified by numbers, which represent the markings and the arcs are just labelled with the transitions rather than the binding elements (see Definition 4).

Firstly, the tool communicates with the CPN Tools simulator in order to obtain all the transitions and the dead markings (see Section 5). The user interacts with the Prosega/CPN GUI to assign identifiers (corresponding to user observable events or service primitive names) to the model transitions (i.e. mapping elements from a set *SP*). The character 0 is considered as an epsilon (ε). Hence, any transition assigned with 0 is considered an epsilon transition (or empty move). Then, the user chooses the set of terminal states *F* for the FSA. which may include nodes representing the dead markings or other nodes in the OG. Thereby, it is obtained a FSA in line with Definition 5.


Prosega/CPN

Minimized Deterministic FSA Generation

Please enter an identifier (Id) for each transition listed. Also, you may enter the terminal states. Otherwise, you may want to import identifiers and terminal states using the Import buttons.

Arc Identifiers

#	Page	Transition	Id
12	CreatConnecti...	MACCrtConnA...	0
13	CreatConnecti...	MACCrtConnCf	4
14	CreatConnecti...	MACCrtConn...	5
15	CreatConnecti...	MACCrtConn...	6
16	CreatConnecti...	MACCrtConnI...	2
17	CreatConnecti	MACCrtConn	1

Identifiers.txt

Terminal States

#	State
4	13
5	26
6	27
7	31
8	47
9	48

Terminal_States.txt

Options

Consider dead-markings as part of the list of terminal states: ✓

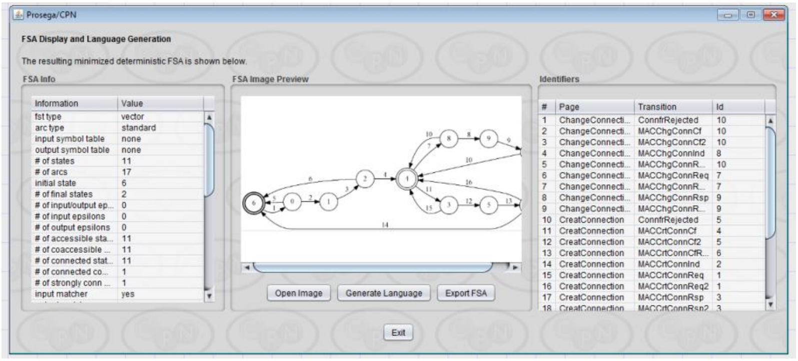
Arc's id pattern: [✓ int] 6-letter string

Fig. 2. Intial Prosega/CPN in terface where the user can assign Ids to transitions and enter terminal states

For instance, fig. 2 shows the Prosega/CPN interface which supports the described operation. In particular, it is defined a FSA given a CPN's occurrence graph extracted from the use case in Section 6. The user assigns identifiers for the CPN transitions. For example, the identifier 1 to the transition *MACCrtConnReq*, which is in the CPN model page *CreatConnection*. Later, the user chooses the following nodes of the OG as terminal states: 1, 7, 8, 13, 26, 27, 31, 48 (some are not displayed in the figure due to window size limitation). Afterwards, the modelled FSA is reduced by following the algorithm described in [22], which consists in performing the following operations over a FSA:

- removal of *epsilon* transitions (remove empties);
- removal of non-determinism (determinization);
- reduction by identifying and merging equivalent states (minimization).

The algorithm produces as output a reduced deterministic FSA with a minimal number of states that is equivalent to the input automata. Finally, an interface showing the results of the FSA reduction is displayed to the user as shown in fig. 3.



The resulting minimized deterministic FSA is shown below.

Information	Value
fst type	vector
arc type	standard
input symbol table	none
output symbol table	none
# of states	11
# of arcs	17
initial state	6
# of final states	2
# of input/output ep...	0
# of input epsilons	0
# of output epsilons	0
# of accessible sta...	11
# of coaccessible ...	11
# of connected stat...	11
# of connected co...	1
# of strongly conn ...	1
input matcher	yes

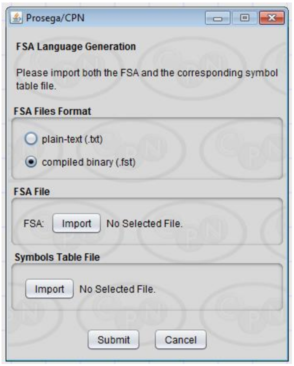
#	Page	Transition	Id
1	ChangeConnecti...	ConnfrRejected	10
2	ChangeConnecti...	MACChgConnCf	10
3	ChangeConnecti...	MACChgConnCf2	10
4	ChangeConnecti...	MACChgConnInd	8
5	ChangeConnecti...	MACChgConnR...	10
6	ChangeConnecti...	MACChgConnReq	7
7	ChangeConnecti...	MACChgConnR...	7
8	ChangeConnecti...	MACChgConnRsp	9
9	ChangeConnecti...	MACChgConnR...	9
10	CreatConnection	ConnfrRejected	5
11	CreatConnection	MACCrtConnCf	4
12	CreatConnection	MACCrtConnCf2	5
13	CreatConnection	MACCrtConnCfR...	6
14	CreatConnection	MACCrtConnInd	2
15	CreatConnection	MACCrtConnReq	1
16	CreatConnection	MACCrtConnReq2	1
17	CreatConnection	MACCrtConnRsp	3
18	CreatConnection	MACCrtConnRsp2	3

Fig. 3. Interface showing the results of the FSA reduction process

The interface shows general information about the reduced FSA (FSA Info), such as initial state and number of arcs, which may be relevant for the FSA analysis. It also includes a graphical representation of the FSA (FSA Image Preview), and the established mapping between the identification numbers/names assigned by the user and the transition names, which may be useful for debugging and verification of the model.

4.2 Language generation

The language accepted by a FSA can be generated by using either the LANG tool in Fig. 1 or the Generate Language button in Fig. 3. The interface shown in Fig. 4 is displayed to the user after it clicks on the LANG tool. Then the user can choose both the FSA, in plain text or in the compiled format [13], for which the language will be generated and the corresponding symbol table file—for mapping the arc inscriptions with the symbols selected by the user.


Prosega/CPN

FSA Language Generation

Please import both the FSA and the corresponding symbol table file.

FSA Files Format

plain-text (.txt) [✓ compiled binary (.fst)]

FSA File

FSA: Import No Selected File.

Symbols Table File

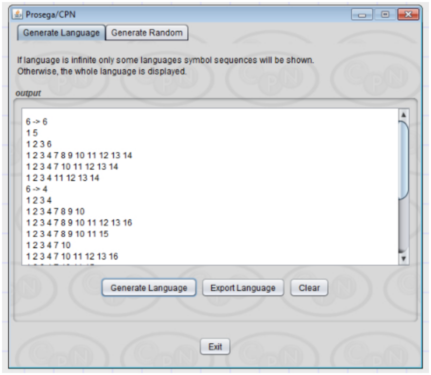
Import No Selected File.

Submit Cancel

Fig. 4. Language generation interface

The language generator module generates the language *L* of the FSA by extension; if *L* is finite, the whole sequences are printed; otherwise a subset of the language, *L'* ⊆ *L* is generated, as illustrated in fig. 5. In particular, *L'* is a set of symbol sequences whose symbols belong to different arcs in the FSA. Notice that some arcs of the FSA may be labelled with the same symbol. However, in the generation of each sequence, each arc of the FSA is visited just once.

Indeed, for generating each sequence accepted by the automaton it was developed an algorithm based on iterative Depth-first Search (DFS), which was implemented in the language generator component of Prosega/CPN (as mentioned in Section 5). This component performs DFS between the initial state of the FSA, to each of the halt states. Hence, the symbols of the arcs visited through the path from the initial state to a specific halt state are printed, thereby representing a sequence accepted by the automaton. In addition, this module supports a generator of random sequences of the language symbols, as shown in Fig. 6, which may be useful when the language is infinite. For example, in Fig. 5 and 6, we can see the following sequence of language symbols: 1, 5 which corresponds to the sequence of actions (transitions): *MACCrtConnReq*, *MACCrtConnCf2* (as shown in the interface in fig. 4, where the user assigned an Id (language symbol) to each transition).

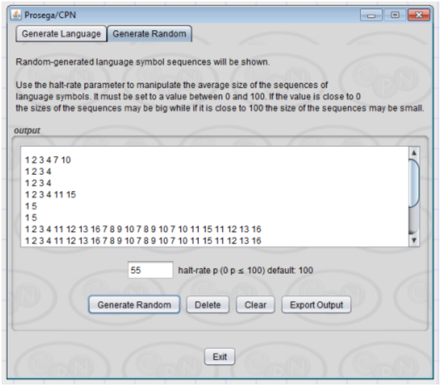


If language is infinite only some languages symbol sequences will be shown.
Otherwise, the whole language is displayed.

```
6 -> 6
1 5
1 2 3 6
1 2 3 4 7 8 9 10 11 12 13 14
1 2 3 4 7 10 11 12 13 14
1 2 3 4 11 12 13 14
6 -> 4
1 2 3 4
1 2 3 4 7 8 9 10
1 2 3 4 7 8 9 10 11 12 13 16
1 2 3 4 7 8 9 10 11 15
1 2 3 4 7 10
1 2 3 4 7 10 11 12 13 16
```

Fig. 5. Interface showing part of the language accepted by the FSA of Fig. 3

In particular, for generating each random sequence it is computed a random walk in the FSA from the initial state to any of the halt states. Whenever a halt state is visited, the walk will be terminated with a probability $p/100$ s.t $0 < p \leq 100$, and the sequence of symbols, which were collected throughout the visited path will be printed. Thus, in the Generate Random interface (fig. 6), the user can manipulate the average size of the randomly generated sequences of language symbols by entering the halt-rate parameter value p. Therefore, if the value p is close to 0, the number of language symbols in each sequence may be big, while if p is close to 100, then the number of language symbols in each sequence may be small, thereby determining the length of each sequence. i.e. since the halt-rate parameter value in fig. 6 is 55, in that case the sizes of the sequences are medium.



Random-generated language symbol sequences will be shown.

Use the halt-rate parameter to manipulate the average size of the sequences of
language symbols. It must be set to a value between 0 and 100. If the value is close to 0
the sizes of the sequences may be big while if it is close to 100 the size of the sequences may be small.

```
1 2 3 4 7 10
1 2 3 4
1 2 3 4
1 2 3 4 11 15
1 5
1 5
1 2 3 4 11 12 13 16 7 8 9 10 7 8 9 10 7 10 11 15 11 12 13 16
1 2 3 4 11 12 13 16 7 8 9 10 7 8 9 10 7 10 11 15 11 12 13 16
```

55 halt-rate p (0 p ≤ 100) default: 100

Fig. 6. Interface showing some randomly generated sequences of language symbols

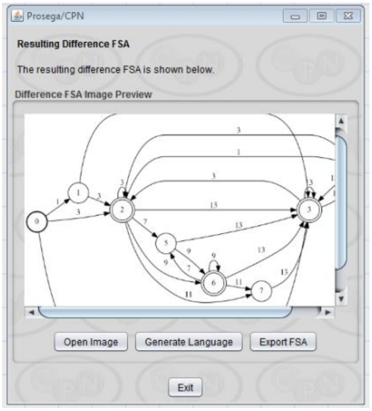


Fig 7. The interface shows the resulting difference FSA given two automata as parameters

4.3 FSA Difference

The user can use the DIFF tool to calculate the difference between two automata, F A and F B. This functionality, whose output interface is illustrated in fig. 7, generates a new automaton F C which only accepts the sequence of symbols accepted by the first automaton F A , and that are not accepted by the second one F B . In particular, F B must be an epsilon-free, deterministic finite automaton. This is useful to understand the sequences of languages symbols in which may differ two models; in this sense, as seen in fig. 7, this functionality allows to generating the language of F C for getting such sequences in which may differ two models.

5. Architecture

Prosega/CPN is implemented in Java programing language, so we use the new feature in CPN Tools 4 called Simulator Extensions [4] to add the software functionalities. Fig. 8 shows the software architecture, which illustrates the relation ship among all the components of our tool, CPN Tools and the third-party components. Communication between the CPN Tools GUI and the simulator, and between the simulator and the Simulator Extensions is supported by the BIS (Boolean - Integer - String) protocol. Each protocol message is encoded using a number of booleans, integers, and strings as explained in [23]. In order to facilitate the development of Prosega/CPN we use some third-party libraries, which implement many of the functions to manage and display the automata.

In particular, we utilize OpenFST [13] [24] for FSA reduction and FSA difference, and Graphviz [25] for drawing the automata. On the other hand, we wrote the code for language generation (fsm2language) in C programming language [26]. The fsm2language implements the procedures for language generation and the computation of random sequences accepted by a FSA that were described in Section 4. The bridge between the fsm2language component and the Prosega/CPN tool is supported by JNI (Java Native Interface), which enables a Java program to call native libraries written in C/C++ programming language.

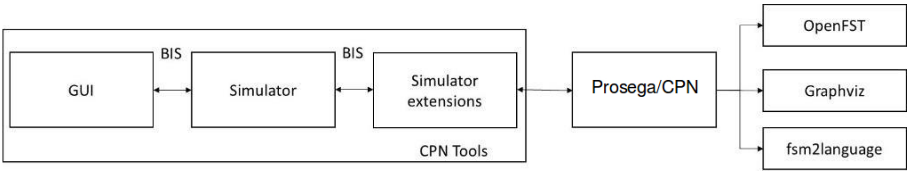


Fig. 8. Prosega/CPN Architecture

6. Use Case

The IEEE 802.16 standard [27] is responsible for specifying and describing the air interface of Broadband Wireless Access Systems (BWA), and point-multipoint fixed/mobile wireless metropolitan area network. The standard is limited to the description of the Medium Access Control (MAC) and physical (PHY) layers. In overall, IEEE 802.16 provides great benefits for providing mass broadband wireless connectivity, allowing user mobility, mesh-mode network support, and even has been thought as an alternative for Internet-of-Things deployments. However, due to its inherent complexity, there are several parts of the specification that turn out to be ambiguous, difficult to understand and imprecise. In this context, Morales et al. [28] [29] has contributed establishing a formal model for a module of IEEE 802.16. In particular, it developed a formal verification of the MAC connection management service specification. To this aim, the Prosega/CPN tool has been used in conjunction with the Billington’s protocol verification methodology [8]. Fig. 9 illustrates the steps of the methodology; we proceed to explain such steps, and how they have been applied within our use case using CPN Tools and Prosega/CPN.

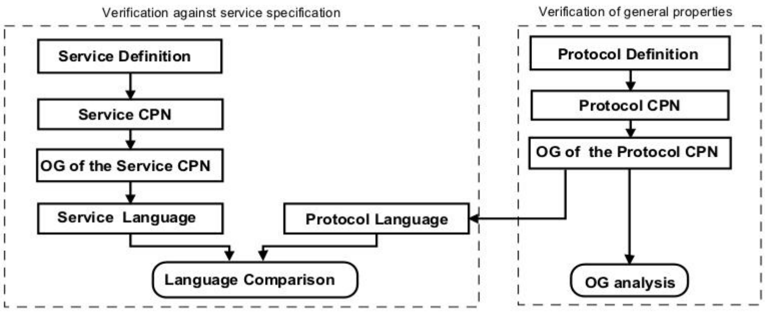


Fig. 9. Steps within the protocol verification methodology proposed in [8].

6.1 Service Definition

In fig. 9, the dashed box in the left represents the first step which consists in modelling the service specification of the system, and to define the services that it aims to provide (either to a higher layer or to another system entity). In the scenario of the IEEE 802.16 MAC layer, the service specification consists in a set of service primitives that the MAC sub-layer, responsible for connection management procedures, provides to the sub-layer on top of it. Each of these primitives correspond to one of the following procedures: The establishment of a connection between communication peers, the connection maintenance (i.e. management of the dynamic network resources) and the termination of the connection by any of the communication peers.

6.2 Service CPN and OG

Using CPN Tools, it is created the CPN model of the service specification. fig. 10 presents the CPN main page which shows a top view of the model [2]. This top module is linked with the pages that model the service primitives that correspond to the establishment, maintenance, and termination of a connection through the transitions *CreatConnection*, *ChangeConnection*, and *TerminateConnection* respectively. Each of these pages of the model can be checked in [28]. Afterwards, it is generated the CPN's occurrence graph (OG), shown in fig. 11, which is the input for the FSA reduction feature of Prosega/CPN.

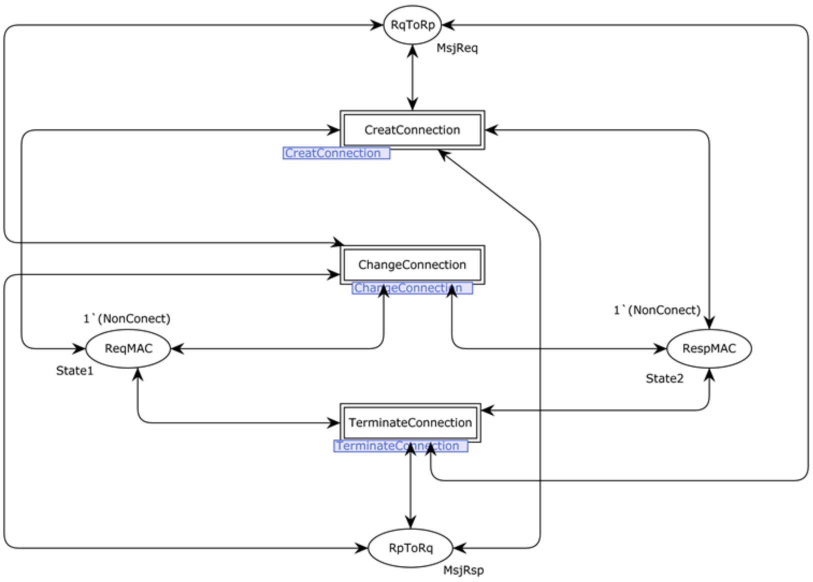


Fig. 10. CPN model representing the hierarchical view for the processes of creation, change and termination of connections between peer MAC entities in the IEEE 802.16 service specification

6.3 FSA Reduction

Once the service OG is generated, it is modelled as a FSA in line with Definitions 4 and 5. To this aim, it is used the RUN tool of Prosega/CPN for converting the OG into a FSA (as presented in fig. 2). For each transition of the CPN model, it is assigned a number value which represents the associated service primitive identifier (Id) (resembling the function *Prim* described in Section 3). Transitions that are considered as empty moves (or internal events) are labelled with 0 (*epsilon* transitions). Later, there are assigned the terminal states. The assignation performed between all the model transitions and the service primitive identifiers as well as the decision of the terminal states can be fully checked in [28]. Afterwards, the FSA is minimized following the procedure explained in Section 4. Fig. 12 presents the minimized deterministic FSA (exported from the output/analysis interface of the RUN tool previously presented in Fig. 3.

Table. 1. Service primitivies on the IEEE 802.16 MAC Layer and their corresponding identification number [22]

Service Primitive	Id
MAC_CREAT_CONNECTION.Request	1
MAC_CREAT_CONNECTION.Indication	2
MAC_CREAT_CONNECTION.Response	3
MAC_CREAT_CONNECTION.Confirmation	4, 5, 6
MAC_CHANGE_CONNECTION.Request	7
MAC_CHANGE_CONNECTION.Indication	8
MAC_CHANGE_CONNECTION.Response	9
MAC_CHANGE_CONNECTION.Confirmation	10
MAC_TERMINATE_CONNECTION.Request	11
MAC_TERMINATE_CONNECTION.Indication	12
MAC_TERMINATE_CONNECTION.Response	13
MAC_TERMINATE_CONNECTION.Confirmation	14, 15, 16

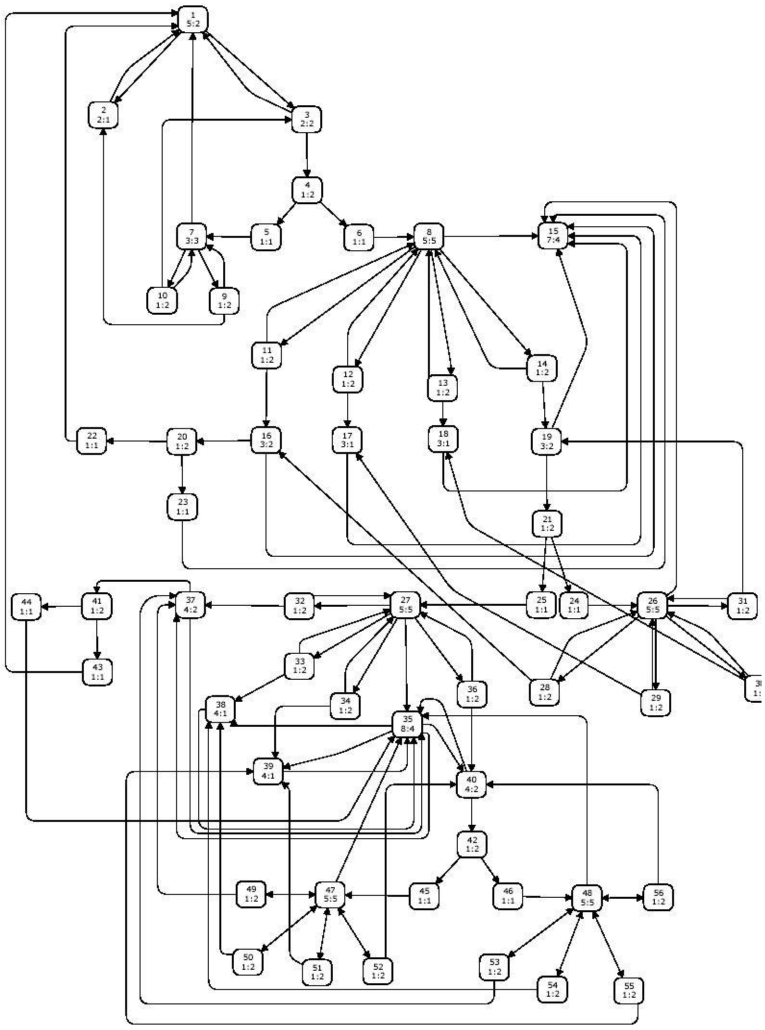


Fig. 11. OG of the CPN model representing the IEEE 802.16 MAC connection management service specification.

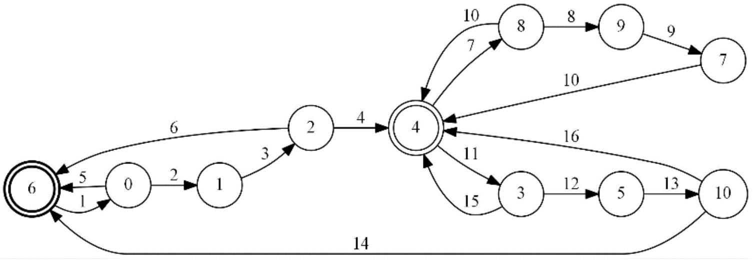


Fig. 12. Minimized deterministic FSA generated from the OG illustrated in fig. 11

6.4 Language Generation

The service language (the set of sequences of service primitives) is generated using Prosega/CPN as explained in Section 4 —utilizing FSA minimization (RUN tool) and FSA language generation (LANG tool). Fig. 5 presented some sequences that are accepted by the FSA. In addition, Table. 1 shows the identifier selected for each primitive service [28]. For example, the sequence of language symbols 1, 2, 3, 4, 7, 8, 9, 10 represent the service primitives invoked by the protocol entity in top of the MAC for the successful establishment and maintenance (change of a communication resource) of the connection. In overall, the minimized FSA generated by Prosega/CPN provides a compact description of the possible sequences of service primitives, and allows to remove complexity from the model, which allows the language to present a clear specification of the service that the system provides.

6.5 Further Steps

The second part of the methodology (dashed box in the right of Fig. 9) concerns to the modelling of the protocol, and its comparison against the service specification through language equivalence. These further steps are still in progress within the research work [28]. The modelling of the protocol consists in constructing the CPN model, which describes the protocol procedures which are performed when a service primitive is invoked by a higher entity of the system. Later, it is generated the OG associated to this CPN model. On the one hand, behavioral properties of the protocol may be analyzed through the OG. On the other hand, the OG may be reduced into a minimized deterministic FSA. i.e. using again the RUN tool of Prosega/CPN.

Then, the FSA of the service specification may be compared with the FSA of the protocol. i.e. using the DIFF function of Prosega/CPN – see fig. 7. Finally, the language of the difference FSA may be generated in order to determine language equivalence between the service and the protocol. Thus, we can determine the sequences of service primitives, which are in the protocol specification but are not in the service specification. It is important to know if the service specification meets the protocol specification, since it is not desirable to have a service requirement from the service user which cannot be met by the protocol. In addition, it may not be wanted a service provided by the protocol which actually it is never required by the user.

7. Conclusion

This work has presented Prosega/CPN. The tool is an extension of CPN Tools for supporting several operations for FSA-based analysis and system verification. The tool provides a feature for generating a minimized deterministic Finite-state Automaton (FSA) from a CPN's occurrence graph (OG). It includes as well operations for language generation, and for automata comparison. These functionalities are supported taking advantage of consolidated third-party components such as OpenFST and Graphviz. In addition, we developed a module for language generation.

Prosega/CPN has been integrated within the CPN Tools GUI using the Simulation Extensions (new feature in the last version of CPN Tools) component whose development has been driven by the demand of many research works to suitably integrate Colored Petri Nets with other formalisms [4]. In particular, the integration between CPNs and FSA was not existing within CPN Tools, and the application of this multi-formalism strategy has shown its merits in many published papers, specially from the domain of protocol verification.

Furthermore, other works may be benefited from this FSA-based verification; for example, as presented in our use case, the analysis of an equivalent reduced FSA provides a compact and clear description of the possible user observable events (service primitive calls) rather than to deal with the analysis of the OG, thereby allowing to reduce the time complexity when it may be required to check the behavioral properties of the system through the FSA.

As future work, the tool will keep providing support within the further steps of the formal verification work of the IEEE 802.16 standard, regarding to the MAC connection management procedures. On the other hand, as another further direction for the tool enhancement, the tool has been thought to be tested in other domains; indeed, as it has been stated, Prosega/CPN can be used in other cases where FSA may be required, and within the verification of other systems whose analysis may involve the comparison of models at different levels of abstraction.

This future work on other use cases will be able to keep maturing the tool. i.e. integrating new operations/features for automata manipulation, and testing the tool performance in terms of scalability, among other key facts. In addition, it has been considered to keep exploiting more capabilities offered by the Simulator Extensions channel; for example, to be able draw and manually edit a FSA in the CPN Tools canvas, instead of only using the Graphviz support for automata drawing.

References

[1]. T. Murata. Petri nets: Properties, analysis and applications. Proceedings of the IEEE, vol. 77, no. 4, April 1989, pp. 541–580

[2]. K. Jensen and L. M. Kristensen. Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Berlin, Heidelberg: Springer-Verlag, 2009

[3]. CPN Tools – A tool for editing, simulating, and analyzing Coloured Petri Nets. Available at: http://www.cpntools.org/, accessed: 20.06.2018

[4]. M. Westergaard. CPN Tools 4: Multi-formalism and Extensibility. In Application and Theory of Petri Nets and Concurrency. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 400–409

[5]. M. E. Villapol. Modelling and Analysis of the Resource Reservation Protocol Using Coloured Petri Nets. Ph.D. dissertation, University of South Australia, Australia, December 2003

[6]. S. Gordon, L. M. Kristensen, and J. Billington. Verification of a Revised WAP Wireless Transaction Protocol, In Application and Theory of Petri Nets and Concurrency. Berlin, Heidelberg: Springer-Verlag, 2002, pp. 182–202

[7]. B. Han. Formal Specification of the TCP Service and Verification of TCP Connection Management. Ph.D. dissertation, University of South Australia, Australia, April 2004

[8]. J. Billington, G. E. Gallasch, and B. Han. A Coloured Petri Net Approach to Protocol Verification. Berlin, Heidelberg: Springer-Verlag, 2004, pp. 210–290.

[9]. G. Gallasch and L. M. Kristensen. Comms/CPN: A Communication Infrastructure for External Communication with Design/CPN. In Proc. of the Third Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, DAIMI PB-554, pages 75–91

[10]. M. Westergaard and K. B. Lassen. The BRITNeY Suite Animation Tool. In Applications and Theory of Petri Nets and Concurrency. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 431–440

[11]. M. Westergaard. Access/CPN 2.0: A High-Level Interface to Coloured Petri Net Models. In Application and Theory of Petri Nets and Con- currency. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 328–337

[12]. AT&T Researchers – Inventing the Science Behind the Service. Available at: http: //www.research.att.com/evergreen/portfolio/, accessed: 20.06.2018

[13]. OpenFST Library. Available at: http://www.openfst.org/twiki/bin/view/FST/WebHome, accessed: 20.06.2018

[14]. M. Hulden. Foma: A Finite-state Compiler and Library. In Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics: Demonstrations Session. Stroudsburg, PA, USA: Association for Computational Linguistics, 2009, pp. 29–32

[15]. A. Almeida, M. Almeida, J. Alves, N. Moreira, and R. Reis. FAdo and GUItar: Tools for Automata Manipulation and Visualization. In Implementation and Application of Automata. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 65–74

[16]. S. H. Rodger. JFLAP: An Interactive Formal Languages and Automata Package. USA: Jones and Bartlett Publishers, Inc., 2006

[17]. C. Ouyang and J. Billington. Formal Analysis of the Internet Open Trading Protocol. In Applying Formal Methods: Testing, Performance, and M/E-Commerce. Berlin, Heidelberg: Springer-Verlag, 2004, pp. 1–15

[18]. S. Barzegar, M. Davoudpour, M. R. Meybodi, A. Sadeghian, and M. Tirandazian. Traffic Signal Control with Adaptive Fuzzy Coloured Petri Net Based on Learning

Automata. In Annual Meeting of the North American Fuzzy Information Processing Society, July 2010, pp. 1–8

[19]. N. Danapaquiame, E. Ilavarasan, N. Kumar, and S. K. Dwivedi. Ratification strategy for web service composition using CPN: A survey. In Proc. of the IEEE International Conference on Computational Intelligence and Computing Research, December 2013, pp. 1–4

[20]. J. Zhu, K. Zhang, and G. Zhang. Verifying Web Services Composition based on LTL and colored Petri Net. In Proc. of the 6th International Conference on Computer Science Education, August 2011, pp. 1127–1130

[21]. ISO/IEC. High-level Petri Nets – Part 1: Concepts, Definitions and Graphical Notation. Software and Systems Engineering, ISO/IEC FDIS 15909-1. Final Draft International.

[22]. W. A. Barrett and J. D. Couch. Compiler Construction: Theory and Practice. Chicago, Illinois: Science Research Associates Inc., 1979

[23]. M. Westergaard. CPN Tools 4 Extensions: Part 4: Advanced Communication and Debugging. Available at: https://westergaard.eu/2013/11/cpn-tools-4-extensions-part-4-advanced-communication-and-debugging/, November 2013, Blog entry/, accessed: 20.06.2018

[24]. C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri. OpenFst: A General and Efficient Weighted Finite-State Transducer Library. In Implementation and Application of Automata. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 11–23.

[25]. Graphviz – Graph Visualization Software. Available at: http://www.graphviz.org//, accessed: 20.06.2018

[26]. J. C. Carrasquel. Java/PROSEGA: An extension in CPN Tools for generating languages accepted by FSA and minimized deterministic FSA from a state space. Central University of Venezuela, Caracas, Venezuela, Tech. Rep., October 2015.

[27]. IEEE 802.16 Working Group on Broadband Wireless Access Standards. IEEE Std. 802.16e-2005. Local and Metropolitan Area Network. Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems

[28]. A. V. Morales and M. E. Villapol. Towards Formal Specification of the Service in the IEEE 802.16 MAC Layer for Connection Management. In Proceedings of the 9th WSEAS International Conference on Computational Intelligence, Man-machine Systems and Cybernetics. World Scientific and Engineering Academy and Society (WSEAS), 2010, pp. 140–146

[29]. A. V. Morales and M. E. Villapol. Reviewing the Service Specification of the IEEE 802.16 MAC Layer Connection Management: A Formal Approach. CLEI Electronic Journal, vol. 16, August 2013, pp. 1– 12

Prosega/CPN: расширение CPN Tools для автоматного анализа и верификации систем

[1]*Х. С. Карраскель <julio.carrasquel@yahoo.com>*
[2]*А. Моралес <ana.morales@ciens.ucv.ve>*
[3]*М. Е. Виллаполь <maria.villapol@aut.ac.nz>*
[1]*Римский университет Ла Сапиенца, отдел компьютерной, контрольной и управленческой инженерии, ул. Ариосто 25, г. Рим, 00185, Италия*
[2]*Центральный университет Венесуэлы, Школа компьютерных наук, просп. Пасео Лос-Илюстрес, г. Каракас, 1040, Венесуэла*
[3]*Оклендский технологический университет, Школа инженерии, компьютерных и математических наук, ул. Уэллсли-стрит-восток 55, г. Окленд, 1010, Новая Зеландия*

Аннотация. Верификация и анализ распределенных систем являются чрезвычайно важными задачами, особенно сейчас, когда многие компьютерные системы реализуют критически важные сервисы. Для моделирования и верификации систем полезно сочетать разные методы анализа. В частности, это позволяет применять тот формализм и ту технику анализа, которые лучше подходят для того или иного компонента системы. Комбинация из раскрашенных сетей Петри (CPN, Coloured Petri Nets) и конечных автоматов представляет собой успешную формальную методику моделирования и верификации распределенных систем. В связи с этим в данной статье рассматривается инструмент Prosega/CPN (Protocol Sequence Generator and Analyzer), расширение CPN Tools для поддержки автоматного анализа и верификации. Инструмент реализует несколько функций, таких как генерация минимизированного детерминированного конечного автомата из графа достижимости (occurrence graph) раскрашенной сети Петри, генерация языка и сопоставление конечных автоматов. Это решение поддерживается функцией Simulator Extensions, развитие которой обусловлено необходимостью интеграции раскрашенных сетей Петри с другими формализмами. Инструмент предназначен для поддержки формальной методологии верификации коммуникационных протоколов; однако он может использоваться для верификации других систем, анализ которых включает сравнение моделей на разных уровнях абстракции, например, бизнес-стратегий и бизнес-процессов. В статье приведен подробный пример, в котором инструмент Prosega/CPN используется для анализа части спецификации службы управления соединениями MAC IEEE 802.16.

Ключевые слова: формальные методы; раскрашенные сети Петри; CPN Tools; конечные автоматы; верификация протоколов.

DOI: 10.15514/ISPRAS-2018-30(4)-7

Для цитирования: Карраскель Х. С., Моралес А., Виллаполь М. Е. Prosega/CPN: расширение CPN Tools для автоматного анализа и верификации систем. Труды ИСП РАН, том 30, вып. 4, 2018 г., стр. 107-128 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(4)-7

Список литературы

[1]. T. Murata. Petri nets: Properties, analysis and applications. Proceedings of the IEEE, vol. 77, no. 4, April 1989, pp. 541–580

[2]. K. Jensen and L. M. Kristensen. Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Berlin, Heidelberg: Springer-Verlag, 2009

[3]. CPN Tools – A tool for editing, simulating, and analyzing Coloured Petri Nets. Доступно по ссылке: http://www.cpntools.org/, дата обращения: 20.06.2018

[4]. M. Westergaard. CPN Tools 4: Multi-formalism and Extensibility. In Application and Theory of Petri Nets and Concurrency. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 400–409

[5]. M. E. Villapol. Modelling and Analysis of the Resource Reservation Protocol Using Coloured Petri Nets. Ph.D. dissertation, University of South Australia, Australia, December 2003

[6]. S. Gordon, L. M. Kristensen, and J. Billington. Verification of a Revised WAP Wireless Transaction Protocol, In Application and Theory of Petri Nets and Concurrency. Berlin, Heidelberg: Springer-Verlag, 2002, pp. 182–202

[7]. B. Han. Formal Specification of the TCP Service and Verification of TCP Connection Management. Ph.D. dissertation, University of South Australia, Australia, April 2004

[8]. J. Billington, G. E. Gallasch, and B. Han. A Coloured Petri Net Approach to Protocol Verification. Berlin, Heidelberg: Springer-Verlag, 2004, pp. 210–290.

[9]. G. Gallasch and L. M. Kristensen. Comms/CPN: A Communication Infrastructure for External Communication with Design/CPN. In Proc. of the Third Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, DAIMI PB-554, pages 75–91

[10]. M. Westergaard and K. B. Lassen. The BRITNeY Suite Animation Tool. In Applications and Theory of Petri Nets and Concurrency. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 431–440

[11]. M. Westergaard. Access/CPN 2.0: A High-Level Interface to Coloured Petri Net Models. In Application and Theory of Petri Nets and Con- currency. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 328–337

[12]. AT&T Researchers – Inventing the Science Behind the Service. Доступно по ссылке: http: //www.research.att.com/evergreen/portfolio/, дата обращения: 20.06.2018

[13]. OpenFST Library. Доступно по ссылке: http://www.openfst.org/twiki/bin/view/FST/WebHome, дата обращения: 20.06.2018

[14]. M. Hulden. Foma: A Finite-state Compiler and Library. In Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics: Demonstrations Session. Stroudsburg, PA, USA: Association for Computational Linguistics, 2009, pp. 29–32

[15]. A. Almeida, M. Almeida, J. Alves, N. Moreira, and R. Reis. FAdo and GUItar: Tools for Automata Manipulation and Visualization. In Implementation and Application of Automata. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 65–74

[16]. S. H. Rodger. JFLAP: An Interactive Formal Languages and Automata Package. USA: Jones and Bartlett Publishers, Inc., 2006

[17]. C. Ouyang and J. Billington. Formal Analysis of the Internet Open Trading Protocol. In Applying Formal Methods: Testing, Performance, and M/E-Commerce. Berlin, Heidelberg: Springer-Verlag, 2004, pp. 1–15

[18]. S. Barzegar, M. Davoudpour, M. R. Meybodi, A. Sadeghian, and M. Tirandazian. Traffic Signal Control with Adaptive Fuzzy Coloured Petri Net Based on Learning Automata. In Annual Meeting of the North American Fuzzy Information Processing Society, July 2010, pp. 1–8
[19]. N. Danapaquiame, E. Ilavarasan, N. Kumar, and S. K. Dwivedi. Ratification strategy for web service composition using CPN: A survey. In Proc. of the IEEE International Conference on Computational Intelligence and Computing Research, December 2013, pp. 1–4
[20]. J. Zhu, K. Zhang, and G. Zhang. Verifying Web Services Composition based on LTL and colored Petri Net. In Proc. of the 6th International Conference on Computer Science Education, August 2011, pp. 1127–1130
[21]. ISO/IEC. High-level Petri Nets – Part 1: Concepts, Definitions and Graphical Notation. Software and Systems Engineering, ISO/IEC FDIS 15909-1. Final Draft International.
[22]. W. A. Barrett and J. D. Couch. Compiler Construction: Theory and Practice. Chicago, Illinois: Science Research Associates Inc., 1979
[23]. M. Westergaard. CPN Tools 4 Extensions: Part 4: Advanced Communication and Debugging. Доступно по ссылке: https://westergaard.eu/2013/11/cpn-tools-4-extensions-part-4-advanced-communication-and-debugging/, November 2013, Blog entry/, дата обращения: 20.06.2018
[24]. C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri. OpenFst: A General and Efficient Weighted Finite-State Transducer Library. In Implementation and Application of Automata. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 11–23.
[25]. Graphviz – Graph Visualization Software. Доступно по ссылке: http://www.graphviz.org//, дата обращения: 20.06.2018
[26]. J. C. Carrasquel. Java/PROSEGA: An extension in CPN Tools for generating languages accepted by FSA and minimized deterministic FSA from a state space. Central University of Venezuela, Caracas, Venezuela, Tech. Rep., October 2015.
[27]. IEEE 802.16 Working Group on Broadband Wireless Access Standards. IEEE Std. 802.16e-2005. Local and Metropolitan Area Network. Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems
[28]. A. V. Morales and M. E. Villapol. Towards Formal Specification of the Service in the IEEE 802.16 MAC Layer for Connection Management. In Proceedings of the 9th WSEAS International Conference on Computational Intelligence, Man-machine Systems and Cybernetics. World Scientific and Engineering Academy and Society (WSEAS), 2010, pp. 140–146
[29]. A. V. Morales and M. E. Villapol. Reviewing the Service Specification of the IEEE 802.16 MAC Layer Connection Management: A Formal Approach. CLEI Electronic Journal, vol. 16, August 2013, pp. 1– 12