A Model Driven Analysis of the 802.11 CSMA/CA Protocol through SD2PN

Mohamed Ariff Ameedeen
Faculty of Computer Systems & Software Engineering
Universiti Malaysia Pahang
Pahang, Malaysia
mohamedariff@ump.edu.my

Behzad Bordbar
School of Computer Science
University of Birmingham
Birmingham, United Kingdom
b.bordbar@cs.bham.ac.uk

***Abstract*— Unified Modelling Language (UML) has been conferred as the *de facto* standard in modeling by majority in the software system development community. Among the various types of diagrams that exist under the umbrella of UML is Sequence Diagram. Sequence Diagrams are capable of modeling interactional behaviours as well as dynamic happenings in a system, and as such are generally used in the modeling of complex software systems. However in this paper, Sequence Diagrams are used in the modeling of the IEEE 802.11 Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol. The Sequence Diagram representing this protocol will then be used for formal, mathematical analysis by first transforming the Sequence Diagram through the MDA model transformation tool called SD2PN, and performing analysis such as liveness analysis, boundedness analysis and reachability analysis of the resulting Petri Net.**

Keywords - sequence diagrams, Petri Nets; modelling, IEEE 802.11.

I. INTRODUCTION

Modelling is becoming more and more significant nowadays, be it for software system designs or even for more fundamental architectures such as protocols. Protocols such as the IEEE 802.11 CSMA/CA [1] could also be represented as visual models in the form of UML diagrams [2]. This would not only benefit the protocol from being a textual description to being depicted in an easier to understand, semi-formal notation of the UML diagrams, but it will also allow it to take advantage of the various analytical tools available.

There are also Model-driven Architecture (MDA) [3] based tools such as SD2PN [4] and UML2Alloy [5] that transforms UML diagrams into formal languages such as Petri Nets [6] and Alloy [7]. These tools could also be used to perform mathematical analysis of protocols such as the aforementioned IEEE802.11 CSMA/CA protocol.

In this paper, the IEEE 802.11 CSMA/CA protocol will first be represented as a UML Sequence Diagram. The Sequence Diagram will then be transformed into Petri Nets using the MDA model transformation called SD2PN. The resulting Petri Net would then be analyzed for its liveness, boundedness and reachability.

In order to facilitate the understanding of this paper, the Foundation chapter briefly introduces Sequence Diagrams, Petri Nets, MDA, SD2PN and the IEEE 802.11 CSMA/CA protocol. The following chapter presents how SD2PN is used in the analysis of the IEEE802.11 CSMA/CA protocol, and finally the Discussion and Conclusion chapter discusses the outcome that can be concluded from this paper.

The preliminary version of this paper was presented at the 2012 Fourth International Conference on Computational Intelligence, Modelling and Simulation

II. FOUNDATION

In this chapter, the foundation behind this paper is established where preliminary knowledge of all the underlying architecture and framework used in this paper is provided.

A. Sequence Diagrams

Sequence Diagrams is UML 2.0 version of Message Sequence Charts [2] and are widely used in Software Engineering [8]. Sequence Diagrams can be used in modeling complex Enterprise Systems as they provide a sequential listing of events and are also able to model parallelism and alternatives. They are also effective in modeling behaviour and concurrency, and as such more than adequate to model complex protocols such as the IEEE802.11 CSMA/CA.

Fig. 1 represents a subset of UML 2.0 Sequence Diagrams metamodel used in this paper, comprising of important constructs used for depicting models with complex behavior. The main fragments of the Sequence Diagram are represented by model elements *Message* and *CombinedFragments*. The model element *Message* represents the interaction between the instances of objects in the system while *CombinedFragments* are high level addition to Sequence Diagrams and consist of Interaction Operators *alternative*, *option*, *break* and *parallel*. These model elements will be referred to as fragments of Sequence Diagrams throughout this paper.

The model element *EventOccurrence* and *GeneralOrdering* denotes the sequencing of events in the diagram. *EventOccurrence* is a specialization of *MessageEnd* where each *message* is given a specific order in reference to the previous and subsequent *messages*.

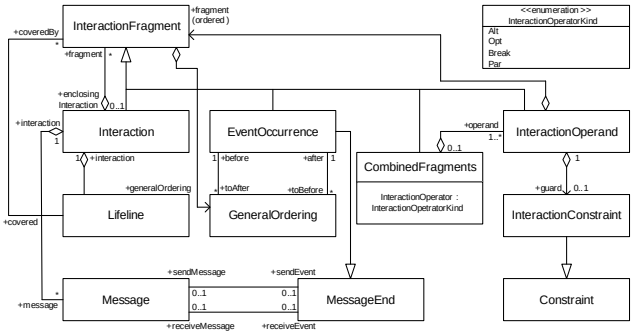

Fig. 1 : Sequence Diagram Metamodel.

B. *Petri Nets*

Petri Nets are a graphical and mathematical modelling language applicable to Enterprise Systems. Petri Nets can be parallel, asynchronous, concurrent, distributed and stochastic as well as being dynamic [6]. Similar to flow-charts, a Petri Net can model the flow of events in a system graphically [9]. Petri nets can be formalized as follows:

Definition 1: A Petri Net is a triple $N = (S, T, F)$ where S is a finite set of places and T is a set of transitions where $S \cap T = \varnothing$. F is a relations on $S \cup T$ where $F \cap (S \times S) = F \cap (T \times T) = \varnothing$. A marking of N is a function $\mathbf{m}:S \rightarrow\{0,1,2,3, \ldots\}$, where each place $s \in S$ is assigned the number of tokens. M_0 is used to show the *initial marking*, the number of tokens in each place at the beginning of execution.

Graphical representations of Petri nets depict each place as a circles and each transition as a square. Places are generally used to represent states where as transitions are used to represent actions or events.

Fig. 2 depicts the metamodel of Petri Net used in this paper. Referring to Definition 1, S and T are represented by the instances *Place* and *Transition* while *InputArc* and *OutputArc* represents the relationship F. The instance *Marking* refers to the function $\mathbf{m}:S$. Every place may have a *Mark* which represents the number of tokens that belongs to a place. This instance is represented by an integer, i.e. 0, 1, 2 and so on.

C. *Model Driven Architecture*

Model Driven Architecture (MDA) [10] aims to promote the role of modelling in software development. Models in the context of MDA are captured in machine-readable representations, using languages which are widely adopted by the industry [7]. Hence it is possible to communicate such models to various parties and reuse them. This results in lower production cost and shorter development cycles. For the purpose of this paper, MDD is used in the seamless transition of models between two languages; Sequence Diagrams and Petri Nets. As such, the IEEE 802.11 CSMA/CA protocol is designed in Sequence Diagrams and analyzed using Petri Net, a more formal and mathematical language.

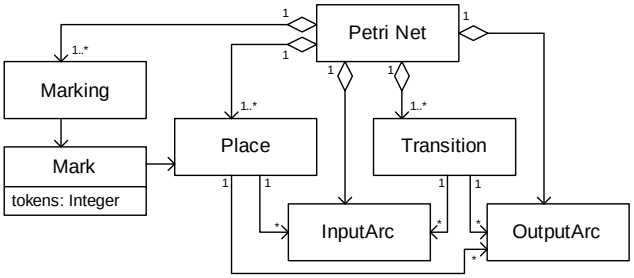

Fig. 2 : Petri Net Metamodel.

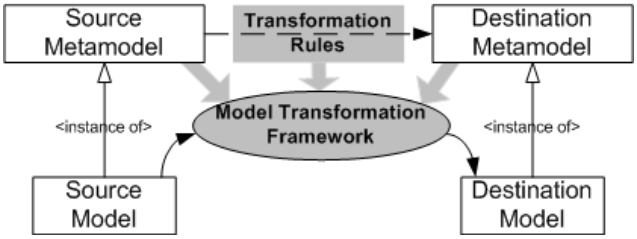

Fig. 3 : Model Driven Architecture.

MDA outlines the concept of model transformation which is central to the work presented in this paper. Fig. 3 depicts an sketch of MDA model transformation as outlined by MDA [3] and the metamodels that comply to the Meta-Object Facility (MOF) [11]. A number of *transformation rules* are used to define how various elements of one metamodel (*source metamodel*) are mapped into the elements of another metamodel (*destination metamodel*). The process of model transformation is carried out automatically via the software tools which are commonly referred to as *model transformation frameworks* [12-14]. A typical model transformation framework requires three inputs: source metamodel, destination metamodel and Transformation Rules. For any instance of the source metamodel, a *transformation engine* executes the rules to create an instance of the destination metamodel.

III. SD2PN: An MDD Model Transformation from Sequence Diagrams to Petri Nets

SD2PN is a rule-based MDD model transformation that transforms any Sequence Diagrams that conforms to the metamodel in Figure 1 into Petri Nets. It was first introduced in [4] and subsequently updated through [15-17]. The model transformation process is hereby described in three stages:

Stage 1: Decomposition
The Sequence Diagram inputted into SD2PN is decomposed into multiple small fragments based on the Sequence Diagram metamodel.

Stage 2: Transformation
Each Sequence Diagram fragment from Stage 1 is transformed into a Petri Net block based on a set of model transformation rules.

Stage 3: Composition
The Petri Net blocks from Stage 2 are put together using two local functions; *morph* and *substitute*.

Each instance of the model transformation goes through the three stages in order to successfully transform Sequence Diagrams into Petri Nets. The three stages are explained in depth in Sub-sections A, B and C below.

A. Decomposition

The process of decomposition of a Sequence Diagram is carried out on the concrete syntax representation and involves identification of various model elements and their relationships. The metamodel in Fig. 1 depicts the model elements used in a Sequence Diagram.

The main model element chosen from the metamodel is *message*. *Message* refers to the events, or the flow of information between objects in the Sequence Diagrams. Each *message* consists of two *MessageEnds*, as its sending and receiving events. These *MessageEnds* are instances of *EventOccurrence*; where the causality of the events is determined by *GeneralOrdering*. In Sequence Diagrams, this causality ordering is identical to a top-down visual ordering. For the purpose of SD2PN, a *message* is considered to be a Sequence Diagram fragment.

CombinedFragments are high level additions to Sequence Diagrams. They are instances of *InteractionFragment* that consists of *InteractionOperators*. *CombinedFragments* may include multiple *InteractionFragments*; which means it could consist of other *CombinedFragments*. As a consequence, *CombinedFragments* may have a hierarchical structure. This hierarchical structure is also sometimes referred to as *nested CombinedFragments*. The nesting of *CombinedFragments* may also occur between different *InteractionOperatorKinds*. There are four *InteractionOperatorKind* used in this thesis as depicted in Figure 1; *alternative*, *option*, *break* and *parallel*. Since each of the four *InteractionOperatorKind* changes the flow of events in a different way, they each are designated as a fragment type.

Overall, there are five types of Sequence Diagram fragments; *message*, *alternative*, *option*, *break* and *parallel*. Each Sequence Diagram inputted into SD2PN is decomposed based on these five fragment types. The decomposition however preserves the causality of the *messages* or the hierarchical structure of the *CombinedFragments*. In the next section, these fragments are transformed into an equivalent Petri Net block.

B. Transformation

Before the transformation rules are presented, a destination metamodel has to be introduced. The Petri Net metamodel depicted in Fig. 2 corresponds to the description of standard Petri Nets presented in the Foundation section. However for the purpose of this section of this paper, a temporary, necessary extension of Petri Nets is introduced through two new concepts; *placeholders* and *Petri Net blocks*. As such the metamodel of Fig. 2 is extended for the use of SD2PN, as depicted in Fig. 4.

Definition 2: *Placeholders* are temporary nodes that mimic the structure of a *place* in Petri Nets and are depicted as dashed rectangles.

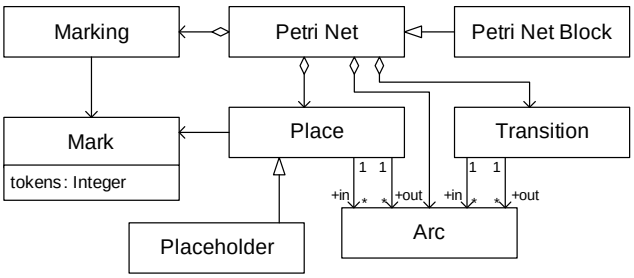

Fig. 4 : Extended Petri Net Metamodel for SD2PN.

Definition 3: Petri Net blocks are blocks of Petri Nets that have unique input and output *places*, which are referred to as *precondition* and *postcondition* respectively. A more formal definition of Petri Net blocks is as follows.

A Petri Net block is a four tuple $B = (S, T, P, F)$ where S is a finite set of *places*, T is a finite set of *transitions*, and P is a finite set of *placeholders*. $F \subseteq ((S \cup P) \times T) \cup (T \times (S \cup P))$ is a set of *arcs*. $In(B)$, $Out(B) \in S$ are *unique* places (*precondition* and *postcondition* respectively) such that $In(B)$ has no incoming arcs and $Out(B)$ has no outgoing arcs. They represent the start and end places in the Petri Net blocks respectively. As such, a Petri Net block can also be textually represented as the sum of all its components.

Using the extended Petri Net metamodel as depicted in Fig. 4, as well as the definitions of *placeholders* and Petri Net blocks, five model transformation rules are outlined to transform every single Sequence Diagram fragment into a corresponding Petri Net block. The rules are as follows:

Rule 1: For every *message* fragment, a corresponding Petri Net block is generated with the following structure.

$$B = (\{s1,s2\},\{m\},\{\ \},\{(s1,m),(m,s2)\})$$

Rule 2: For every *CombinedFragment* with the *InteractionOperatorKind alternative*, an equivqlent Petri Net block is created. The structure of this block is $\square\square$ = $\square\square,\square\square,\square\square,\square\square$ where

$$S = (s1,s2)$$
$$T = (t1,t2,t3,t4)$$
$$P = (ph1, ph2)$$
$$F = \{(s1,t1), (s1,t2), (t1,ph1), (t2,ph2), (ph1,t3), (ph2,t4), (t3,s2), (t4,s2)\}$$

Rule 3: The Petri Net block that is generated for the *option* *InteractionOperatorKind* is $\square\square$ = $\square\square,\square\square,\square\square,\square\square$ where

$$S = (s1,s2)$$
$$T = (t1,t2,t3,t4)$$
$$P = (ph1, ph2)$$

F = {(s1,t1), (s1,t2), (t1,ph1), (t2,ph2), (ph1,t3), (ph2,t4), (t3,s2), (t4,s2)}

Rule 4: For the *break CombinedFragment*, a Petri Net block is created such that $B = (S,T,P,F)$ where

S = (s1,s2,X)
T = (t1,t2,t3)
P = (ph)
F = {(s1,t1), (s1,t2), (t1,ph), (t2,X), (ph,t3), (t3,s2)}

Rule 5: The final rule of SD2PN is a rule that transforms every *CombinedFragment* with *InteractionOperatorKind parallel* into a Petri Net block such that $B = (S,T,P,F)$ where

S = (s1,s2)
T = (t1,t2)
P = (ph1, ph2)
F = {(s1,t1), (t1,ph1), (t1,ph2), (ph1,t2), (ph2,t2), (t2,s2)}

For a more graphical reprrsentation of the rules, please refer to [15].

C. Composition

Following the mapping of each Sequence Diagram fragment into a corresponding Petri Net block, an integrated Petri Net that corresponds to the original Sequence Diagram needs to be produced by composing the Petri Net blocks.

Examined closely, there is a commonality between all the Petri Net blocks generated via SD2PN; each have a single input and output *place*, or as previously introduced, *precondition* and *postcondition*. This is deliberate to allow a uniform method of putting the Petri Net blocks together. There are two local functions used for this purpose: *morph* and *substitute.*

Definition 4: The function *morph* puts together two causal Petri Net blocks. Suppose

$B_1 = (S_1, T_1, P_1, F_1)$ and $B_2 = (S_2, T_2, P_2, F_2)$

are two Petri Net Blocks. The *morphing* of B_1 and B_2, denoted by $B_1 \otimes B_2$ results in a Petri Net Block $B = (S, T, P, F)$ such that

$$T = T_1 \cup T_2$$
$$P = P_1 \cup P_2$$
$$S = (S_1 \cup S_2) \setminus \{Out(B_1)\}$$
$$In(B) = In(B_1) \text{ and } Out(B) = Out(B_2)$$
$$F = ((F_1 \cup F_2) \setminus \{(x, y) \mid y = Out(B_1)\} \cup \{(x, In(B_2) \mid (x, Out(B_1) \in F_1\}$$

Definition 5: The function *substitute* is used for composing hierarchical behaviour between Petri Net blocks. *Substitute* can only be used to replace a *placeholder* with a Petri Net block. Suppose

$B_1 = (S_1, T_1, P_1, F_1)$ and $B_2 = (S_2, T_2, P_2, F_2)$

are two Petri Net Blocks. Let *ph1* be a *placeholder* in B_2. *Substituting* the Petri Net Block, B_1 into *ph1*, denoted by $B_2[B_1/ph1]$ results in a Petri Net Block, $B = (S, T, P, F)$, where

$$S = S_1 \cup S_2$$
$$T = T_1 \cup T_2$$
$$P = (P_1 \cup P_2) \setminus \{ph1\}$$
$$In(B) = In(B_2)$$
$$Out(B) = Out(B_2)$$
$$F = (F_1 \cup F_2 \setminus \{(x, y) \mid x = ph1 \text{ or } y = ph1\}) \cup \{(x, In(B_1)) \mid (x, ph1) \in F_1\} \cup \{(Out(B_1), y) \mid (ph1, y) \in F_1\}$$

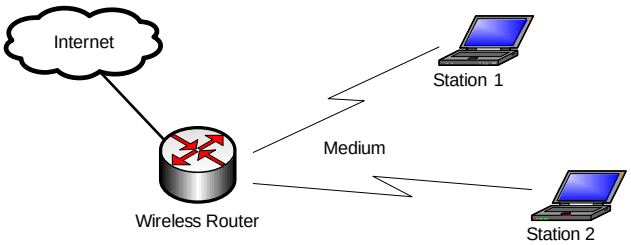

Fig. 5 : Overview of a Personal Area Network.

As with the Transformation rules in the previous section, a more graphical representation of the *morph* and *substitute* functions is available in [15].

IV. Analysis of IEEE 802.11 CSMA/CD via SD2PN

In a typical scenario of a Personal Area Network (PAN), there exist a number of stations and a router. However, an unseen element in the PAN is the medium between the stations and the router. In order to send a packet to the router, the stations in the PAN would have to compete to gain access to the medium. Thus, the more stations there are in a PAN, the larger the maximum waiting time is for a single station to gain access to the medium. To deal with this, various protocols have been introduced within the IEEE 802.11 standard. However in this example, a specific protocol within the IEEE 802.11 standard is modelled using Sequence Diagrams and transformed into a Petri Net for analysis.

Fig. 5 presents a simplified PAN that has two stations and a Wireless Router that serves as an access point to the Internet. In the router, the basic IEEE 802.11 Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol is used [1]. The router in this example uses a basic IEEE 802.11 CSMA/CA protocol. CSMA/CA assigns different *waiting time* to packets in order to manage the access of the stations to the medium. There are three different waiting times for various types of packets. The shortest waiting time for medium access is called *Short inter-frame spacing* (SIFS) which is used for short control messages or polling responses. The waiting time for time-bounded service such as a poll from the access point is considered *PCF inter-frame spacing* (PIFS) and the longest waiting time and lowest priority, *DCF inter-frame spacing* (DIFS) is used for asynchronous data services. There is a mechanism called *contention window* (CW), which is introduced in order to facilitate collision *avoidance*. The contention window makes use of an integer value that starts with $CW_{min} = 7$ and doubles every time a collision occurs.

Every time a station tries to gain access to the medium, a random number is generated between 0 and CW and is added to the waiting time. This ensures that the stations do not send their packets at the same time. CW is doubled for every collision that occurs to accommodate a larger number of stations vying for the access of the medium. Readers are referred to [1] for more information.

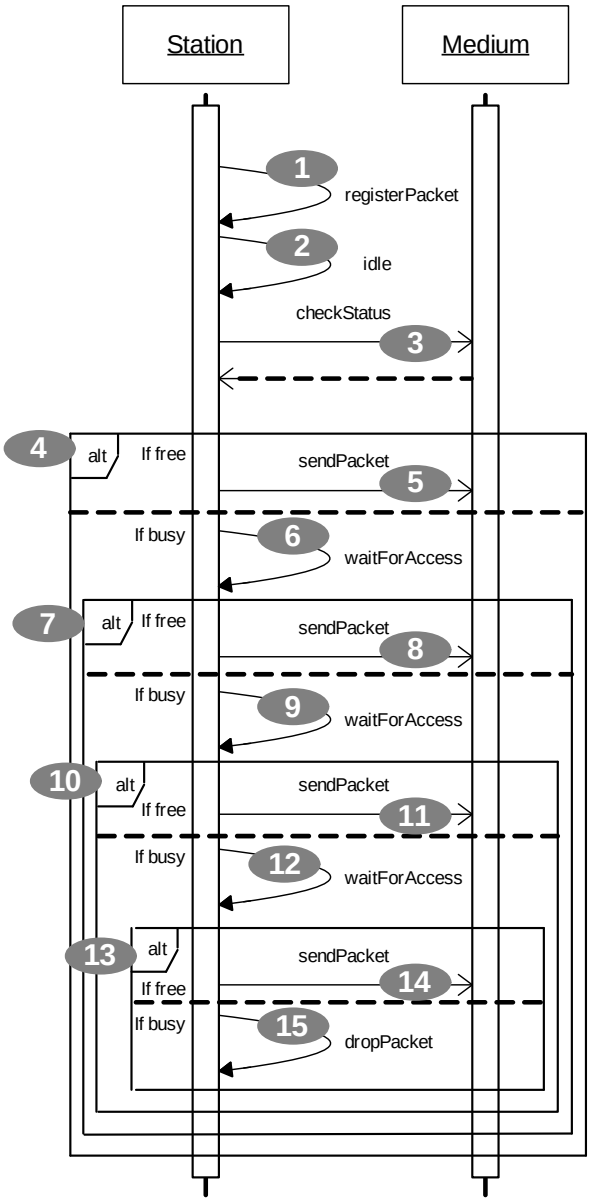

Fig. 6 : Sequence Diagram of the IEEE 802.11 CSMA/CA Protocol.

Several assumptions were made in this example for the sake of clarity and to provide a better understanding of the tool. Firstly, the waiting time for all packets is constant and all packets are categorized as DIFS. Secondly, the CW is constant and does not increase, and since there are only two stations, the CW would be minimum, i.e. $CW_{min} = 7$. Thirdly, the packets are dropped after the unsuccessful tries from the station and each station sends only one packet. These assumptions do not invalidate the results of the analysis by any means; they only limit the scope of this example.

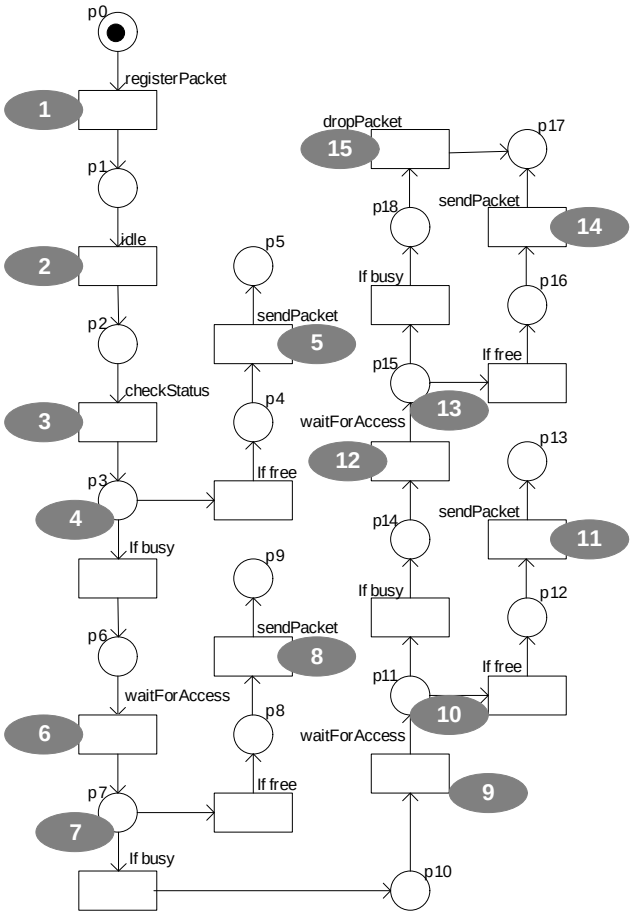

Fig. 7 : Petri Net of the IEEE 802.11 CSMA/CA Protocol Generated by SD2PN.

This scenario is modelled using a Sequence Diagram, where a single station attempts to gain access to the medium in order to send a packet to the router, based on the CSMA/CA protocol.

The Sequence Diagram in Fig. 6 gives an overview of how a station sends a packet to the medium in the IEEE 802.11 protocol. The medium access control (MAC) layer of the station receives a packet from an application and registers it.

It then idles before checking the status of the medium. If the medium is free the station is able to send the packet across to the medium. However, if the medium is busy the station has to wait until the medium is free before idling again. The MAC then checks the status of the medium again before either sending the packet across or waiting again.

Each of the events in this scenario has multiple sub-events that occur in the background. The diagram is however simplified for the sake of clarity.To perform analysis on the protocol, the Sequence Diagram first needs to be transformed into a Petri Net using SD2PN. The intricate details of the transformation are omitted due to page constraints, however it could be found in [15]. The Petri Net that is generated by SD2PN is as depicted in Fig. 7.

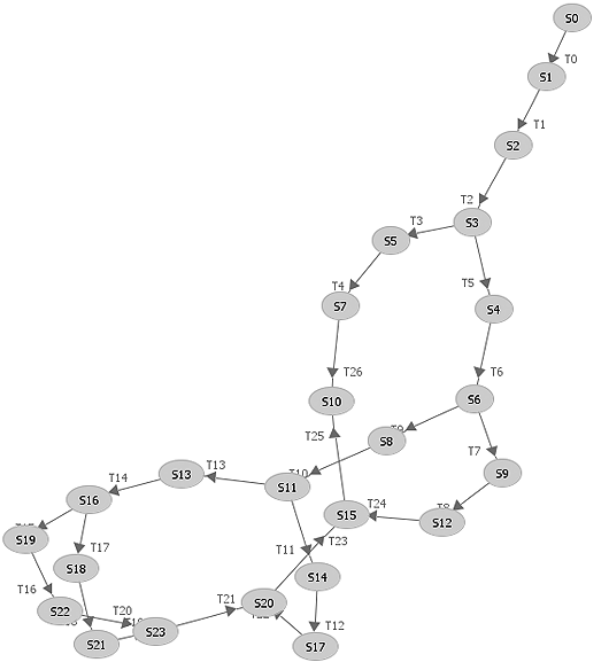

Fig. 8 : Reachability Graph Generated using PIPE.

There are various analysis methods associated with Petri Nets, however as mentioned in the introduction, only the liveness analysis, boundedness analysis and the reachability analysis will be performed for the purpose of this paper using a well-known Petri Net tool, PIPE [18].

The liveness and boundedness of the Petri Net are both calculated through State Space Analysis in PIPE where the liveness is determined through the absence of deadlocks in the Petri Net while boundedness is computed through a P-invariant calculation. The result of the analysis confirms that the Petri Net is both live and bounded. Through the P-invariant calculation, it is also revealed that the Petri Net in Fig. 7 is safe (bounded with the value of 1).

Subsequently, a reachability analysis is performed on the Petri Net, resulting in a Reachability Graph as presented in Fig. 8. As a result, the reachability analysis reveals that every state in the Petri Net is reachable through a series of event. Table 1 prints a summary of the basic analysis results for the analyses performed on the protocol.

V. Discussion and Conclusion

The purpose of this paper is to illustrate the capability of UML Sequence Diagram to model more than just software systems. As such, the modelling of the IEEE 802.11 CSMA/CA protocol proves that modeling is more globally beneficial and thus, everything that can be modeled in Sequence Diagrams can be transformed into Petri Nets and analyzed mathematically.

The correctness of the model transformation performed by SD2PN has also been proven mathematically using a common semantics domain in Labelled-Event Structures [19] by using the semantic mapping introduced in [20, 21].

TABLE I. Summary of Analysis Results

ANALYSIS	RESULT
Liveness	Yes
Boundedness (Value)	Yes (Value 1)
Safeness	Yes
Reachability	All states are reachable

As such, the validity of the analysis results obtained via SD2PN should be regarded as accurate.

Petri Nets could also perform time related analysis such as performance analysis and Quality of Service (QoS) analysis. This flavor of Petri Nets can be obtained using the enhancement of SD2PN that is described in [16].

References

[1] Schiller, J.H., Mobile Communications. 2003: Pearson Education.

[2] UML, UML Superstructure 2.0, Object Management Group, available at www.omg.org. 2003.

[3] MDA, Model Driven Architecture, Object Management Group www.omg.org/mda/. 2005.

[4] Ameedeen, M.A. and B. Bordbar, A Model Driven Approach to Represent Sequence Diagrams as Free Choice Petri Nets, in 12th International IEEE Enterprise Distributed Object Computing Conference (EDOC). 2008: München, Germany. p. 213 - 221.

[5] Anastasakis, K., et al. UML2Alloy: a Challenging Model Transformation. in ACM/IEEE 10th international conference on Model Driven Engineering Languages and Systems. 2007.

[6] Murata, T., Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE, 1989. 77(4): p. 541-580.

[7] AlloyAnalyzer, Alloy Analyzer Website, http://alloy.mit.edu/beta/ 2005.

[8] Campos, J. and J. Merseguer. On the Integration of UML and Petri Nets in Software Development. in 27th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency. 2006. Turku, Finland: Springer.

[9] Butler, J., R. Hubbly, and W. Melo. An MOF-based repository for enterprise architecture models. in avaialble at www-128.ibm.com/developerworks/rational/library/mar05/melo/index.html . 2005.

[10] Stahl, T. and M. Volter, Model Driven Software Development; technology engineering management. 2006: Wiley.

[11] MOF. Meta Object Facility (MOF) 2.0 Core Specification, Object Management Group, available at www.omg.org,. 2004; Available from: http://www.omg.org.

[12] ATLAS, ATLAS, Université de Nantes, http://www.sciences.univ-nantes.fr/lina/atl/. 2005.

[13] kermeta, Triskell Metamodelling Kernel, www.kermeta.org. 2005.

[14] Akehurst, D.H., et al. SiTra: Simple Transformations in Java. in ACM/IEEE 9TH International Conference on Model Driven Engineering Languages and Systems. 2006.

[15] Ameedeen, M.A., A Model-Driven Approach for Analysis and Synthesis of Sequene Diagrams, in School of Computer Science. 2011, University of Birmingham: Birmingham.

[16] Ameedeen, M.A., B. Bordbar, and R. Anane, A Model Driven Approach to Analysis of Timeliness Properties, in Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA 2009) (to appear). 2009.

[17] Ameedeen, M.A., B. Bordbar, and R. Anane, Model Interoperability via Model Driven Development accepted for publication in Journal of Computer and System Sciences, 2010.

[18] Bonet, P., et al., PIPE v2.5: a Petri Net Tool for Performance Modeling, in XXXIii Conferencia Latinoaméricana de Informática. 2007.

[19] Winskel, G., An introduction to event structures, in Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop. 1989, Springer-Verlag.

[20] Küster-Filipe, J., Modelling concurrent interactions. Theoretical Computer Science, 2006. 351(2): p. 203-220.

[21] McMillan, K.L., A technique of state space search based on unfolding. Form. Methods Syst. Des., 1995. 6(1): p. 45-65.