A Model Driven Approach to the Analysis of Timeliness Properties

Mohamed A. Ameedeen[1], Behzad Bordbar[1], Rachid Anane[2]

[1]University of Birmingham, Birmingham, UK
{M.A.Ameedeen, B.Bordbar}@cs.bham.ac.uk
[2]Coventry University, Coventry, UK
R.Anane@coventry.ac.uk

Abstract. The need for a design language that is rigorous but accessible and intuitive is often at odds with the formal and mathematical nature of languages used for analysis. UML and Petri Nets are a good example of this dichotomy. UML is a widely accepted modelling language capable of modelling the structural and behavioural aspects of a system. However UML lacks the mathematical foundation that is required for rigorous analysis. Petri Nets on the other hand have a strong mathematical base that is well suited for analysis of a system but lacks the appeal and ease-of-use of UML. Design in UML languages such as Sequence Diagrams and analysis in Petri Nets require on one hand some expertise in potentially two incompatible systems and their tools, and on the other a seamless transition from one system to the other. One way of addressing this impediment is to focus the software development mainly on the design language system and to facilitate the transition to the formal analysis by means of a combination of automation and tool support. The aim of this paper is to present a transformation system, which takes UML Sequence Diagrams augmented with time constraints and generates semantically equivalent Petri Nets that preserve the timing requirements. A case study on a small network is used in order to illustrate the proposed approach and in particular the design, the transformation and the analysis processes.

1 Introduction

One of the most pressing tasks facing software developers in general and software engineers in particular is the development of software tools that support an integrated approach to the design and the analysis of software systems. The design of a system may be considered as an essentially cognitive activity with a focus on clarity, while its analysis is usually firmly grounded on mathematics and relies often on formal representations and formal processing. The tension that results from this dichotomy has presented a serious challenge for the deployment of existing software tools in both areas. This difficulty is further compounded by the lack of interoperability between the tools associated with each phase. There is a clear need for the development of

tools and frameworks that can reconcile the goals of the design and the analysis processes [1-3].

On the design side the Unified Modelling Language (UML) has been a focal point of activity in the software design community. Its rich constructs have conferred to UML a privileged role in designing software systems in a variety of domains such as network, business modelling and security [4]. One shortcoming of UML is however its inability to support the model analysis process.

The requirements for the formal analysis of software systems have been met by the introduction of a wide range of formal languages which are well suited for analysis, among them Alloy [5], Z [6] and Petri Nets (PN) [7]. In providing support for design and analysis, one common approach is to create the design in UML languages and transform it into a formal representation for analysis. For example, UML2Alloy makes use of Alloy for the analysis of a model which is captured in UML class diagrams and OCL [3]. The analysis performed in the Alloy framework involves solving logical constraints on the model. Although Alloy is useful for the analysis of the static aspects of a design but it is not particularly suitable for behavioural modelling [3]. This limitation is one of the reasons that led developers to rely on formal languages such as Petri Nets. Petri Nets are suitable for analysis of behavioural aspects of models such as deadlock detection, liveness and reachability. Their usefulness has also been enhanced by the availability of tools such as PIPE [8] and CPNTools [9].

The gap between the design and analysis in this respect can be bridged by using Model Driven Development (MDD) model transformation as outlined in [10]. The proposed model transformation addresses this issue by combining the strengths of the two languages: the specification of the behaviour of a system is formulated in UML Sequence Diagrams and the analysis is performed on Petri Nets. The transition from UML to Petri Nets is achieved via a transformation process, which takes a model of Sequence Diagrams and automatically generates the equivalent Petri Net model. The model can subsequently be analysed with Petri Net tools. Figure 1 gives a high-level description of this process. The transformation has already been covered in a previous work by the development of a model driven development (MDD) model transformation tool, SD2PN [10]. As an initial stage in the development of the framework it did not include timing capabilities.

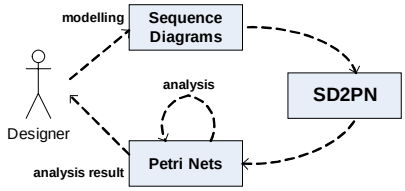

Fig. 1. Overview of the Model Transformation

The contribution of this paper is extending the model transformation in [10] with timing constraints, as part of a framework for software development. The main advantage of this extension is the application of the tool to the analysis of time sensitive systems, such as the modelling of Quality of Service (QoS) and its

validation. The proposed framework is supported by a case study on a QoS specification and analysis in a Personal Area Network (PAN). UML Sequence Diagrams are used to model parts of the IEEE 802.11 protocol. The model is then transformed into Petri Nets and analysed using relevant Petri Net tools for calculating the maximum waiting time for a station in the PAN.

The remainder of the paper is organized as follows. Section 2 provides some background information on MDD, UML and Petri Nets. Section 3 presents a summary of previous work [10] and its extension. Section 4 describes the proposed tool. Section 5 deals with a case study based on the PAN and its analysis in Petri Nets. Section 6 provides a discussion and outlines some further work. Section 7 concludes the paper.

2 Preliminaries

This section introduces some preliminary material regarding Unified Modelling Language, Petri Nets, Model Driven Development and their role in the transformation process.

2.1 Unified Modelling Language

The function of a *model* is to capture a view of the system. In software engineering models are abstractions of a physical system which has a specific purpose [11]. Unified Modelling Language (UML) is a family of languages, which is widely accepted as the *de facto* standard for software modelling. UML models can be used to specify the structure of the system, its behaviour and the constraints that the system must adhere to. This includes constraints related to the timing of the occurrence of events.

Models in UML are instances of *metamodels.* A metamodel includes system elements, their relationships and a set of rules to which every model must conform in order to be well defined. In this paper Sequence Diagrams are used as the main modelling language for describing the behaviour of a system.

Sequence Diagrams. Sequence Diagrams are used to define the interactions between objects and the flow of events within a system. They are based on Message Sequence Charts which are extensively used to capture scenarios for distributed telecommunication systems. Figure 2 is a small metamodel for Sequence Diagrams, adapted from [10], and will be used throughout this paper for explanation purposes. The metamodel of Figure 2 extends the metamodel used in our previous work [10]. However, while [10] presented a metamodel for general Sequence Diagrams, this paper presents an extension that enables the Sequence Diagrams to be augmented with timing constraints while still adhering to the UML 2.1 standards. The shaded boxes in the metamodel in Figure 2 depict these time related extensions.

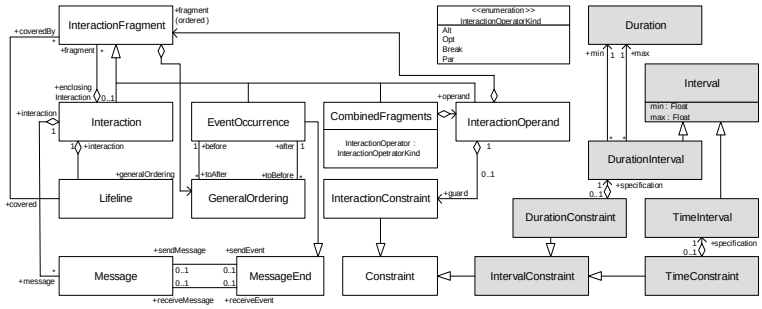

Fig. 2. Sequence Diagram Metamodel

Time Properties. The shaded elements in the metamodel of Figure 2 depict the previously mentioned extension that signifies the addition of time properties into Sequence Diagrams. These elements are adapted from "Common Behaviors", chapter 13 of the UML 2.1 Superstructure [11]. IntervalConstraint, TimeConstraint and DurationConstraints are all specifications of the class Constraint and they are used to define particular types of constraints. TimeConstraint and DurationConstraint refer to TimeInterval and DurationInterval respectively. An Interval is used to specify the range between two ValueSpecifications through a maximum and minimum value. These values are inferred as float instead of ValueSpecification as to keep the metamodel to a minimum. While Intervals have a maximum and minimum value, Duration "defines a value specification that specifies the temporal distance between two time instants" as described in page 437 of [11]. This is also evident in [13] where Douglass interprets that an Interval is not a length of time, but rather a start and end point of a time frame while Duration is a relative time measure that has a scalar value independent of the start time. Douglass further noted that time constraints are syntactically represented textually inside curly brackets, which is also evident in page 59 of [11]

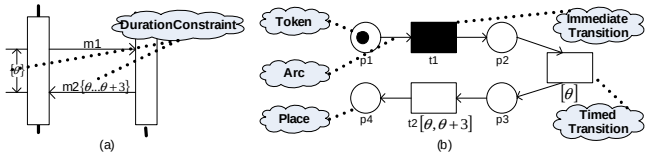

Fig. 3. Examples of a (a) Sequence Diagram and (b) its equivalent Petri Net

Störrle [14] interprets the concept of time in UML 2.0 Sequence Diagrams as divided to two types; the first of which is preserving the state of the system for a certain duration or a time interval while the second represents the duration for a single event to occur. Both these types can be represented by intervals between pairs of event occurrences. This is consistent with UML 2.1 since page 482 of [11] also describes Duration to be "always between occurrences".

Figure 3 (a) shows an example of a Sequence Diagram that features both types of time constraints. The interval between *m1* and *m2* denoted by θ shows that the state

after *m1* is completed is preserved for the duration of θ while the occurrence of message *m2* takes between θ and θ *+3* to occur, where θ is a constant.

2.2 Petri Nets

Petri Nets are a mathematical and graphical modelling language, which can be applied to complex systems. Petri Nets can be used to model a diverse set of behaviour including parallel, asynchronous, concurrent, hierachical and stochastic as well as dynamic behaviours [7]. Similarly to Sequence Diagrams, a Petri Net can model the flow of events in a system graphically. The formal and mathematical nature of Petri Nets can be used to overcome the limitations of Sequence Diagrams with respect to analysis.

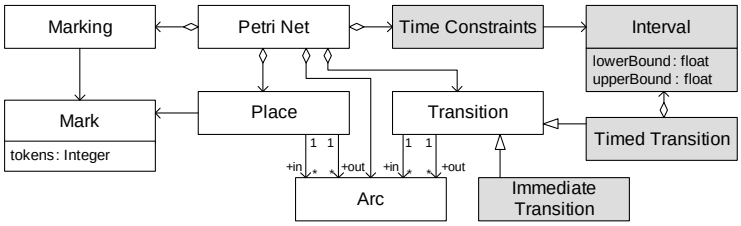

Fig. 4. Petri Net Metamodel

Figure 4 shows the metamodel of a Petri Net, which will be used throughout the paper. This metamodel is adapted from [10] and enhanced with timing properties (shaded elements). The model elements will be explained in terms of an example specified in Figure 3 (b).

The example depicts a Petri Net that models the behaviour captured in the Sequence Diagram of Figure 3 (a). This Petri Net consists of 4 *places* and 3 *transitions* that are all connected by *arcs.* An *arc* in a Petri Net serves as a connector between *places* and *transitions* and may not connect 2 *places* or 2 *transitions*.

A *transition* in the Petri Net has *input places* and *output places*, which are *places* that have *arcs* in and out of the *transition* respectively. A *transition* is *enabled* and ready to *fire* when all of its *input places* have at least a *token* each. *Tokens*, as presented in Figure 3 (b), are depicted as filled circles contained inside *places*. When a *transition fires*, a *token* will be removed from each of the *input places* and added into one of the *output places*. For further information on Petri nets see [7]. The shaded parts of Figure 4 are extension of the conventional Petri Net metamodel in [10] by including time properties, which are explained as follows.

Timed Petri Nets. Timed Petri Nets are extensions to the conventional Petri Nets by the inclusion of timing information such as the time associated to the firing of transitions. There are different flavours of Timed Petri Nets. In this paper, the Timed Petri Net with closed intervals as outlined in [15] are used. The timing information in the metamodel are inferred from the Petri Net tools where [8] shows the existence of two distinct types of transitions and [16] states that each time marking is modelled via

closed intervals. These *intervals* are defined via specific upper and lower bounds attached to a *transition*. For a *transition* to *fire*, firstly it must be *enabled*. Secondly, from the moment it gets *enabled*, a clock starts; the *transition* can *fire* when the value of the clock is within the interval. An example of a timed transition is shown in Figure 3 (b) where the transition *t2* has a time constraint with the closed interval [θ, θ+3]. The transition *t2* can only fire under two conditions; it must be enabled and the clock must be between θ and θ+3. For more information regarding Timed Petri Nets, readers are referred to [15].

The graphical representation of Timed Petri Net also differs slightly from the Petri Net used in [10]. In this paper, the *immediate transitions* or *transitions* without time constraints are depicted as black rectangles while the *timed transitions* are depicted as white rectangles; both of which are shown in Figure 3 (b) This is to provide a contrast between the two types of transition although semantically, an immediate transition is equivalent to a timed transition with an interval of [0, 0]. For *timed transitions*, the *interval* is shown in a bracket by the label of the *transitions*, with a comma separating the upper and lower bound. In a scenario such that the upper and lower bounds are the same i.e. [50, 50]; it is abbreviated as [50]. Each of the upper and lower bound must be of type *float* as inferred from the Sequence Diagrams.

The inclusion of time constraints in Petri Nets enhances their capability for modelling time-sensitive systems. Moreover, with the benefit of using existing Timed Petri Net tools such as CPNTools [9] and PIPE [8], time analysis could take place, thus making it an ideal destination model in an MDD model transformation.

Petri Net Analysis. The mathematical nature of Petri Nets creates a strong base for various types of analysis. Murata [7] outlines a number of analysis methods how they relate to the problems in designing an enterprise system. Among others, *Reachability* analysis is used to study the dynamic properties of a system i.e. how taking one action may effect the chances of an event happening in the future. A *Boundedness* analysis is used to check the effect of the system to the buffers and registers for storing intermediate data while a *Liveness* analysis checks the system for deadlocks. All these analysis and more can be performed on general Petri Nets and are supported by tools such as CPNTools [9], PIPE [8] and various other tools.

While the analysis capabilities of general Petri Nets focus on the structural and behavioural properties of a system, the addition of time properties to the Petri Nets allows for performance analysis as well. A Cycle-time analysis could be used to determine the duration for a complete sequence of action in the system while a tool such as CPNTools [9] can be used for computing the amount of time that separates two events, i.e. time between requesting access to a resource and getting the resource. Various Petri Net tools also provide a platform for other performance analysis such as average time, standard deviations, confidence intervals and throughput analysis as described in [8, 16].

2.3 Model Driven Development

One of the aims of Model Driven Development [17] is to promote the role of *modelling* in software development. Central to the MDD is the process of Model Transformation, which automatically generates a new model from an existing one. Figure 5 depicts an outline of MDD and the process of *Model Transformation.* 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* [18-20].

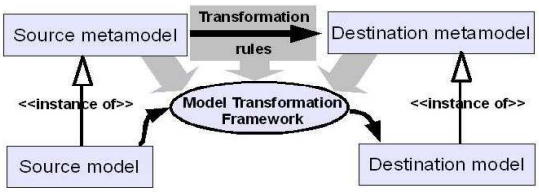

Fig. 5. Model Driven Development

A typical Model Transformation Framework requires three inputs: source metamodel, destination metamodel and transformation rules. For any instance of the source metamodel, the Model Transformation Framework executes the rules to create an instance of the destination metamodel. One way to express such rules is through Query/View/Transformation [21]. QVT is a standard for expressing MDD model transformations governed by the Object Management Group (OMG). Further reading on QVT could be found in [21].

3 Model Transformation

This section recalls the model transformation in [10] and discusses the extensions made to it to enable the analysis of timeliness properties using MDD. SD2PN [10] used a rule-based approach to map Sequence Diagrams into conventional Petri Nets. This model transformation had three stages; Decomposing Sequence Diagrams into fragments, transforming each fragment into Petri Net blocks and putting together the blocks of Petri Nets. A brief outline of the five transformation rules in SD2PN is given below.

Figure 6 shows the transformation rules used in SD2PN. Rule 1 describes the transformation of a *message* fragment into a block of Petri Net as shown in Figure 6. Rule 2 in refers to the CombinedFragment with the InteractionOperatorKind alternative while Rule 3 refers to option. Page 468 of [11] describes an option with with a sole operand to be semantically equivalent to an alternative where the second operand is empty, which will be the default for Rule 3. The guards for these CombinedFragments can be directly transformed from the sequence diagram fragment

and incorporated as guards on the respective transitions. However, Timed Petri Net tools are not equipped to consider the guards as constraints and this limitation is a course for future research. Rules 4 and 5 refer to the *InteractionOperatorKind break* where the node *X* signifies the terminal node and *parallel* fragment, respectively.

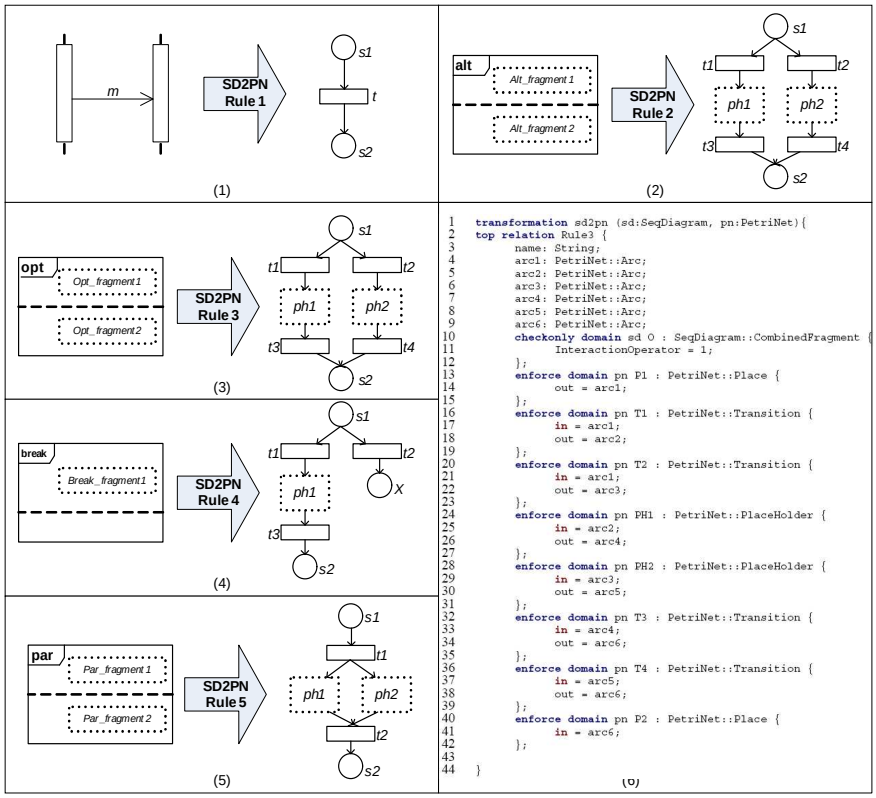

Fig. 6. Five Rules of SD2PN and example of QVT

The snippet of QVT given for Rule 3 is an example of how this model transformation could be carried out. In line 11, the type of the InteractionOperator is checked, and the enumeration for InteractionOperatorKind *option* is 1. The Petri Net is then built according to the diagram of Rule 3.

In this paper, the five rules of SD2PN are refined with time properties to make them compliant with the new metamodels. Referring to section 1 of Figure 6, Rule 1 is used to transform every message in a Sequence Diagram into a Petri Net block consisting of two *places*, *s1* and *s2*, and a *transition, t.* By adding a time constraint to this rule, the *transition t* is given an Interval constraint with a maximum and minimum value acting as its upper and lower bound. There are three possible scenarios that could provide different outcomes to the rule. If a message has an interval with different maximum and minimum values associated to it i.e. {10...30}, the *transition t* in the resulting Petri Net block will be designated as a *Timed*

Transition with a closed interval [10, 30]. Similarly, if a message has a duration associated to it i.e. {20}, the *transition t* in the resulting Petri Net block will be designated as a *Timed Transition* with a closed interval [20, 20] or abbreviated as [20]. However, if a message does not have any time properties attached to it, the *transition t* in the resulting Petri Net block will be designated as an *Immediate Transition*.

Rules 2 through 5, depicted by sections 2 through 5 in Figure 6 respectively, refer to the transformation of each InteractionOperators into a Petri Net block. However, since there are no intervals or durations that are attached to InteractionOperators, every *transition* in the resulting Petri Net block are designated as *Immediate Transitions*.

This paper also introduces a new rule for SD2PN, Rule 6 as illustrated in Figure 7, to map time properties in Sequence Diagrams that are not attached to a message into a Petri Net block. This rule, although resulting in a Petri Net block similar to Rule 1, only has two possible scenarios. In cases where exists an interval in the Sequence Diagrams i.e. {10...30}, the *transition t* in the resulting Petri Net will have an interval of [10, 30] where else if there exist a duration in the Sequence Diagram i.e. {20}, the *transition t* will have an interval of [20].

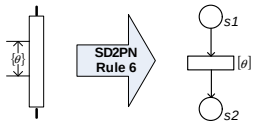

Fig. 7. Rule 6 of SD2PN

Once all the fragments are transformed into Petri Net blocks, they are amalgamated using two operations, which are referred to as *morph* and *substitute* defined in [10]. *Morph* is used for aggregating two Petri Net blocks to create larger Petri Nets. It can be seen that each Petri Net block has a single input and output *place*. Invoking *morph* with two Petri Net blocks merges the former's output *place* with the latter's input *place*. *Substitute* is used to replace a *placeholder* in the Petri Net blocks, such as *ph1*and *ph2* in sections 2 through 5 of Figure 6 with a different Petri Net block. This will be done repeatedly until there are no longer any *placeholders* left in the block.

It is shown in Theorem 1 of [10] that the model transformation generates only Free Choice Petri Nets that are predominantly used for effective and efficient analysis in enterprise system [2]. This result is preserved in this paper since the same Petri Net blocks are used. More information on SD2PN and the proof that it generates Free Choice Petri Net are available in [10].

4 Transformation Tool

Figure 8 depicts the architecture of the tool by showing the stages involved in the execution of a transformation [22]. The tool makes use of the XMI [23] representations of Sequence Diagrams which is provided by all mainstream UML tools such as [24, 25]. Using an XMI parser, the tool creates Java objects based on the

Sequence Diagram metamodel. By utilizing SiTra [26], the tool transforms the Sequence Diagram objects into Petri Net objects based on the transformation rules.

Fig. 8. SD2PN Transformer

Finally, the functions *morph* and *substitute* introduced in [10] are used to aggregate the Petri Net objects, and thus create the Petri Net which corresponds to the original Sequence Diagram. The resulting Petri Net model can then be analysed by using a chosen Petri Net tool. The XML writer for the tool can be customized to correspond with a specific tool. This allows the designer to create a system completely in Sequence Diagram while still taking advantage of the analytical capabilities of Petri Net.

5 Case Study

This section presents a case study, which involves the specification and analysis of the Quality of Service (QoS) of a Personal Area Network (PAN) via SD2PN. The case study demonstrates the transformation of Sequence Diagrams into Timed Petri Nets and the use of the created Petri Nets to analyse QoS properties such as maximum delay.

5.1 Case Description

Figure 9 depicts 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 [27]. Due to space constraints only the elements of the protocol that are relevant to QoS will be considered.

CSMA/CD 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 [27] for more information.

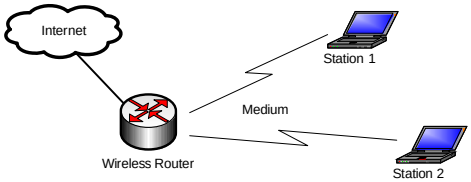

Fig. 9. Personal Area Network (PAN)

Several assumptions were made in this case in order to simplify matters and 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 just limit the scope of this case study.

5.2 Interaction Sequence Diagram

The Sequence Diagram in Figure 10 (a) gives an overview of how a station sends a packet to the medium in the IEEE 802.11 protocol. This Sequence Diagram also features the time properties in regards to the events that occur. The medium access control (MAC) layer of the station receives a packet from an application and registers it. It is then idle for the duration of the waiting time, which in the case of DIFS is 50μs. After idling, the MAC checks 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 will have to wait until the medium is free before idling again for 50μs. This is followed by a random time slot generated based on the CW, which in this case is between 1 and 7. Since a standard time slot is 20μs, this means that an additional waiting time of between 0 to 140μs referred to as the elapsed backoff time (bo_e) for a total waiting time of between 120μs and 240μs as shown in the Sequence Diagram. The MAC then checks the status of the medium again before either sending the packet across or waiting again. If the medium becomes busy while the station is still counting down the bo_e, then the counter stops and the remaining time is called residual backoff time (bo_r). As a result, the next waiting time will only be incremented by the value of bo_r and this increases the probability of a successful attempt from the stations' point of view. Note that the maximum waiting time in Figure 10 (a) is reduced with every attempt.

The diagram is a simplified overview of the events that take place. In reality, each of the events has multiple sub-events that occur in the background. For example, the details for the calculation of bo_e and bo_r are not shown in the Sequence Diagram and are all grouped under the event *waitForAccess*.

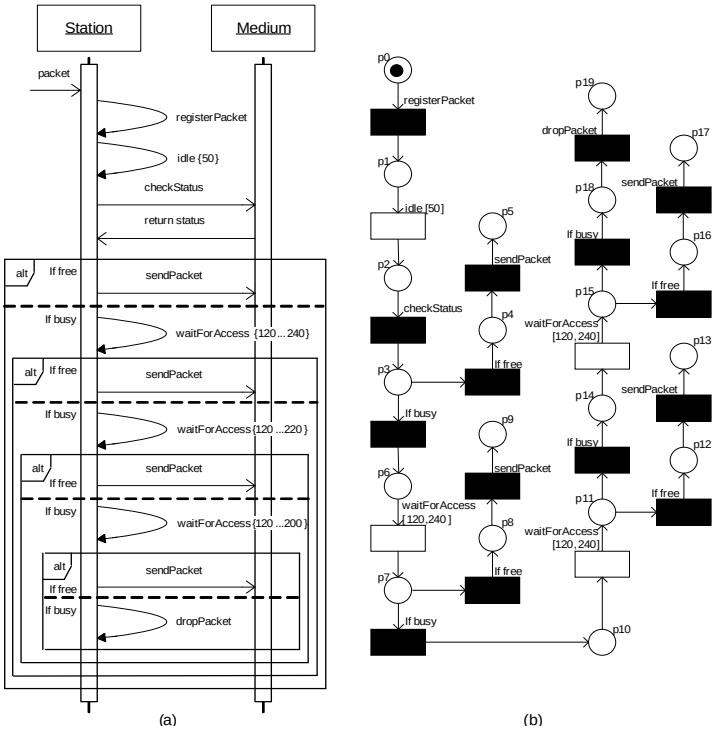

Fig. 10. Model of (a) Sequence Diagram for a station in PAN and (b) its equivalent Petri Net

5.3 Model Transformation

A set of Petri Nets was generated by taking the Sequence Diagrams in Figure 10 (a) as the source model in SD2PN; they represent the process of sending packets in IEEE 802.11. Figure 10 (b) gives the overview of the result of the transformation process as a mapping from the Sequence Diagram in Figure 10 (a). The Petri Nets preserve the time constraints specified in the sequence diagrams and will allow for various forms of QoS analysis to be performed.

The Petri Net in Figure 10 (b) models the behaviour of one station trying to gain access to the medium to send a packet. In cases where more than one station are trying to access the medium, the Petri Net in Figure 10 (b) is duplicated for each station and is synthesized i.e. merged using a bottom-up synthesis technique [28]. Although the tool does not currently support *synthesis*, we are actively working towards its integration into the system.

5.4 Model Analysis

The Petri Net that results from the transformation lends itself to various types of analysis such as deadlock detection, liveness and safeness analysis [29]. Time sensitive analysis such as tangible states analysis and throughput analysis are also

possible [8]. In this case study, throughput analysis will be used to analyse maximum delay as one aspect of QoS.

The maximum delay is calculated based on how long it takes for a station to gain access to the medium *(sendPacket).* For the case of a single station shown in Figure 10 (a), the maximum delay will be 50μs since there is no contention with other stations. However, in the case where there are two stations competing for access to the medium, the maximum waiting time for a station is 290μs as shown in Figure 11. This calculation is based on the flow of events in the Petri Net. Since there are two stations, the Petri Net in Figure 10 (b) is duplicated to model the second station. After registering the packet (firing of *registerPacket* transition), in Figure 10 (b), both stations will face a mandatory idle time of 50μs (firing of *idle* transition) before checking the status of the medium. Following that, only one station will be able to gain access to the medium while the other will have to wait between 120μs and 240μs (firing of *waitForAccess* transition), thus a maximum waiting time of 290μs (= 240μs + 50μs).

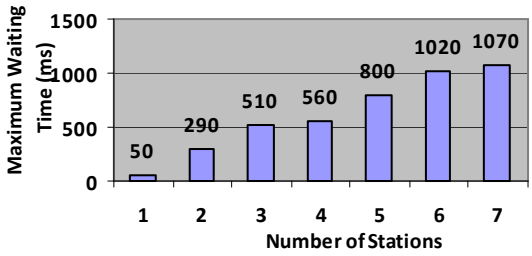

Fig. 11. Maximum Waiting Time analysis result

The graph in Figure 11 indicates the maximum delay that a station may face before gaining access to the medium to send a packet. The number of stations is limited to 7 to ensure there are no collisions; this is based on the previous assumption that the CW does not increase. Such information can be used to analyse the choice of protocols for interaction within the system.

6 Discussion

Many researchers have highlighted the trade-off between the ease of use of UML and its lack of precision. Recent work in this area has been marked by a concerted effort aimed at enhancing UML by incorporating formal methods techniques [30-34]. Formalisation offers many advantages including the ability to analyse a model via techniques such as model checking and theorem proving in order to ensure correct specification. The introduction of logical and timing constraints into a model, in particular, facilitates the investigation of non-functional aspects of the system such as QoS and security. It has been noted however that formalisation is often achieved at the expense of simplicity and that the main challenge is to strike a balance between precision and ease of use.

Formalising UML is an active area of research. For example, Evans *et al* [30] propose the use of Z as the underlying semantics for Class Diagrams to deal with the static aspects of models. Küster-Filipe [35] presents a semantics for Sequence Diagrams based on Labelled Event Structures which are used to prove the correctness of SD2PN [10]. The approach adopted in this paper, although it promotes the use of formal methods, differs from the latter in a significant way. It relies on model transformation and formal method analysis tools to facilitate *automated* analysis.

Model transformation has also received considerable attention. Kim [32] proposes transforming both Class Diagrams and State Machines into Object-Z using MDA technology. To the best of our knowledge, this transformation has not been implemented yet. A similar approach is adopted in [34] and [33] which transform Class Diagrams and OCL Constraints into the formal language B [36]. In particular, [33] proposes a UML profile for B called UML-B and the automation of the transformation with a tool called U2B. A major feature of of this approach is that it makes use of B provers to check the conformance of the operations' pre and post conditions to the invariants of the model. The main difficulty with provers, as underlined in [33], is that even semi-automatic provers assume a substantial amount of knowledge from the user. In contrast the approach presented in this paper aims to limit the reliance on formal method expertise such that a designer may model a system conveniently in Sequence Diagram and still manage to use the analysis capabilities of Petri Nets.

The proposed framework transforms the UML Sequence Diagrams into Timed Petri Nets and takes advantage of their suitability for formal analysis. The transformation produces Free Choice Petri Nets, which support the investigation of various properties such as liveness, safeness and deadlocks detection [29]. It is possible to integrate existing Petri Net tools into the tool set, so that for a created UML Sequence Diagram, through a chain of tools, the user can *automatically* receive feedback on, among others, the liveness, safeness and deadlock freeness of the model. This combination of formalisms, tools and model transformations is bound to reduce the cognitive load on users since a thorough understanding of the underlying formal structure of the model is no longer required. Moreover, Free Choice Petri Nets are also proving to be particularly suitable for the analysis of large-scale systems [1, 2], an important feature that widens the scope of the application of the proposed framework to encompass similar systems. In addition to the structural and behavioural analysis, the time properties included in this paper will also enable performance analysis to be conducted on the system. Analysis like maximum throughput, density probability, interval, cycle time [8, 16] and many other time related analysis can be carried out.

It is possible to augmenting Sequence Diagrams with logical constraints as pre and post conditions for each execution of events. Such constraints can be expressed in languages such as OCL [12] and mapped by extending the model transformation presented in the paper. This would result in Coloured Petri Nets [7] which are an extension of Petri Nets. Coloured Petri Nets have been investigated extensively and various tools, such as CPNTools [9], have been developed for their analysis.

7 Conclusion

This paper has presented a framework of applying Model Driven Development for transforming time augmented Sequence Diagrams into Timed Petri Nets. This model transformation serves to bridge the gap between the design and analysis phases of a system, thus enabling a designer to conveniently design a system in UML Sequence Diagram while taking advantage of Petri Nets' strong mathemathical foundations to analyse the model. Furthermore, the addition of time properties into the model transformation allows for performance analysis such as execution time computation and throughput analysis on top of the established structural and behavioral analysis capabilities of Petri Nets. The presented approach has been evaluated successfully with the help of an example of a Personal Area Network.

Acknowledgement. The authors wish to thank Behrang Saroui for his part in the tool development.

8 References

1. van der Aalst, W.M.P., *The Application of Petri Nets for Workflow Management.* The Journal of Circuits, Systems and Computers, 1998. **8**(1): p. 21-66.
2. Vanhatalo, J., H. Volzer, and F. Leymann, *Faster and More Focussed Control-Flow Analysis for Business Process Models Through SESE Decomposition*, in *Fifth International Conference on Service Oriented Computing*. 2007, Springer: Vienna, Austria. p. 43-55.
3. Anastasakis, K., et al. *UML2Alloy: a Challenging Model Transformation*. in *ACM/IEEE 10th international confernece on Model Driven Engineering Languages and Systems*. 2007.
4. Juerjens, J., *Secure Systems Development With UML*. 2004: Springer.
5. Jackson, D., *Software Abstractions Logic, Language, and Analysis*. 2006: MIT press.
6. Spivey, J.M., *The Z Notation: a reference manual*. 2001: Prentice Hall (out of print, available at http://spivey.oriel.ox.ac.uk/~mike/zrm/).
7. Murata, T., *Petri Nets: Properties, Analysis and Applications.* Proceedings of the IEEE, 1989. **77**(4): p. 541-580.
8. Bonet, P., et al., *PIPE v2.5: a Petri Net Tool for Performance Modeling*, in *XXXIii Conferencia Latinoaméricana de Informática*. 2007.
9. CPNTools, *Computer Tool for Coloured Petri Nets, http://wiki.daimi.au.dk/cpntools/.*
10. 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.
11. OMG, *OMG Unified Modelling Language (UML) Superstructure 2.1, available at www.omg.org*. 2007.
12. OMG, *UML 2.0 OCL 2nd revised submission, available at www.omg.org.* 2003.
13. Douglass, B.P., *Doing Hard Time: Developing Real-time Systems with UML, Objects, Frameworks and Patterns*. Object Technology Series. 1999: Addison Wesley.
14. Störrle, H., *Trace Semantics of UML 2.0 Interactions*. 2004, University of Munich.
15. Wang, J., *Timed Petri Nets: Theory and Application*. 1998: Springer.

16. Jensen, K., L.M. Kristensen, and L. Wells, *Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems.* International Journal on Software Tools for Technology Transfer (STTT), 2007.
17. Stahl, T. and M. Volter, *Model Driven Software Development; technology engineering management*. 2006: Wiley.
18. Jouault, F., Kurtev, I. *Transforming Models with ATL*. Model Transformations in Practice Workshop at MoDELS 2005. 2005.
19. P.A. Muller, F. Fleurey, and J. M. J´ez´equel. Weaving Executability into Object-Oriented Meta-languages. In MoDELS'05: 8th Int. Conf. on Model Driven Engineering Languages and Systems, Montego Bay, Jamaica, Oct 2005. Springer.
20. Akehurst, D.H., et al. *SiTra: Simple Transformations in Java*. in *ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (formerly the UML series of conferences)*. 2006. Genova, Italy.
21. OMG, *MOF 2.0 Query/View/Transformation (QVT) Specification, available at www.omg.org*. 2008.
22. Saroui, B.S., *Model Transformation from Sequence Diagrams to Petri Nets*. 2008, University of Birmingham: Birmingham, UK.
23. XMI, *XML Metadata Interchange (XMI), v2.1, available at www.omg.org*. 2005.
24. ArgoUML, *ArgoUML web site, sourceforge.net/projects/argouml*. 2005.
25. Poseidon. *Poseidon for UML, from Gentleware, www.gentleware.com/*. 2006.
26. Akehurst, D.H., et al. *SiTra: Simple Transformations in Java*. in *ACM/IEEE 9TH International Conference on Model Driven Engineering Languages and Systems*. 2006.
27. Schiller, J.H., *Mobile Communications*. 2003: Pearson Education.
28. Agerwala, T. and Y.-C. Choed-Amphai, *A synthesis rule for concurrent systems*, in *ACM IEEE Design Automation Conference*. 1978.
29. Desel, J. and J. Esparza, *Free Choice Petri Nets*. 1995: Cambridge University Press.
30. Evans, A.F., Robert & Grant, Emanuel. *Towards Formal Reasoning with UML Models*. in *Proceedings of the OOPSLA'99 Workshop on Behavioral Semantics*. 1999.
31. Kim, D., et al. *A UML-Based Metamodeling Language to Specify Design Patterns*. 2003
32. Kim, S.-K., *A Metamodel-based Approach to Integrate Object-Oriented Graphical and Formal Specification Techniques*. 2002, University of Queensland: Brisbane, Australia.
33. Snook, C. and M. Butler, *UML-B: Formal modelling and design aided by UML*, in *ACM Transactions on Software Engineering and Methodology*. 2006.
34. Marcano, R. and N. Lévy, *Transformation Rules of OCL Constraints into B Formal Expressions*, in *5th International Conference on the Unified Modeling Language*. 2002: Dresden, Germany.
35. Küster-Filipe, J., *Modelling concurrent interactions.* Theoretical Computer Science, 2006. **351**(2): p. 203-220.
36. Abrial, J.-R., *The B-book: Assigning Programs to Meanings*. 1996: Cambridge University Press.