Journal of Computer and System Sciences 77 (2011) 332–347

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Model interoperability via Model Driven Development

Mohamed A. Ameedeen [a,*], Behzad Bordbar [a], Rachid Anane [b]

[a] University of Birmingham, Birmingham, UK
[b] Coventry University, Coventry, UK

ARTICLE INFO

Article history:
Received 1 June 2009
Received in revised form 15 October 2009
Available online 1 February 2010

Keywords:
Model Driven Development
Model interoperability
Software development
UML
Petri Nets

ABSTRACT

Among the factors that contribute to the inherent complexity of the software development process is the gap between the design and the formal analysis domains. Software design is often considered a human oriented task while the analysis phase draws on formal representation and mathematical foundations. An example of this dichotomy is the use of UML for the software design phase and Petri Nets for the analysis; a separation of concerns that leads to the creation of heterogeneous models. Although UML is widely accepted as a language that can be used to model the structural and behavioural aspects of a system, its lack of mathematical foundations is seen as a serious impediment to rigorous analysis. Petri Nets on the other hand have a strong mathematical basis that are well suited for formal analysis; they lack however the appeal and the ease-of-use of UML. A pressing concern for software developers is how to bridge the gap between these domains and allow for model interoperability and the integration of different toolsets across them, and thus reduce the complexity of the software development process. The aim of this paper is to present a Model Driven Development (MDD) model transformation which supports a seamless transition between UML and Petri Nets. This is achieved by model interoperability from UML Sequence Diagrams to Petri Nets and supported by tool integration. The model transformation framework allows a software system to be designed in terms of UML Sequence Diagrams and subjected to formal analysis by taking advantage of the strong mathematical framework of Petri Nets. The behaviour of a Personal Area Network will be used to illustrate the proposed approach and to highlight model interoperability and tool integration through the design, the transformation and the analysis phases.

1. Introduction

The complexity of the software development process has presented researchers with a significant challenge. This complexity is due to various factors including the variety of application domains, the variety of software platforms and the variety of the methods and the tools that support the software development process. This complexity is further compounded by the requirement for a software system to satisfy a set of specific properties, such as fault-tolerance and security. Many approaches and methods have been proposed as a way of addressing and reconciling these issues [1]. Of particular significance in the generation of a software system is the need to facilitate a smooth transition from one phase of the development process to the next. The transition from an informal design to formal analysis is often critical, especially as it often involves incompatible domains of discourse.

This dichotomy between design and analysis manifests itself in the multiplicity of formalisms, languages and software tools that are required for each phase. The limited scope of these tools and their tight coupling with specific domains is

* Corresponding author.
E-mail addresses: M.A.Ameedeen@cs.bham.ac.uk (M.A. Ameedeen), B.Bordbar@cs.bham.ac.uk (B. Bordbar), R.Anane@coventry.ac.uk (R. Anane).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2010.01.011

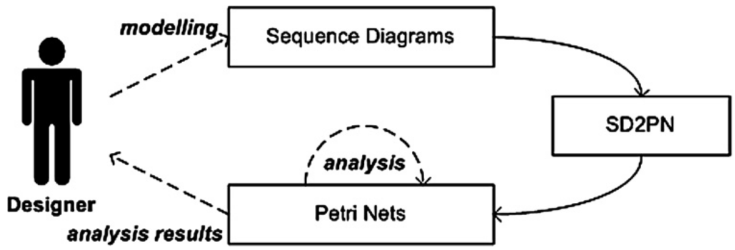

Fig. 1. SD2PN model transformation framework.

one of the main sources of heterogeneity between the models created in the design phase and the models required for analysis [2]. One of the main concerns of software developers is how to bridge the gap between the different underlying domains and allow for a seamless transition between them [3]. More specifically, the main issue is how to facilitate the interoperability between the models that pertain to design and those required by the analysis phase; another requirement is how to integrate the corresponding software tools. In this respect, model interoperability and tool integration are considered as critical factors in reducing complexity in software development.

Various languages and formalisms were introduced in order to support the software development in general and the software design and analysis in particular. Software design has been eased by the introduction of Unified Modelling Language (UML) [4]. Its rich constructs have conferred to UML a privileged role in the design of software systems in a variety of domains including networks, business modelling and security. The choice of UML for software design is also facilitated by the widely available UML software tools such as ArgoUML [5] and Poseidon [6]. One shortcoming of UML however is its lack of support for formal analysis [7]; this characteristic has led software developers to rely more and more on formal languages such as Petri Nets. Petri Nets are well suited to structural analysis such as *liveness* and *deadlock* detection as well as behavioural analysis such as *reachability* [8]. Their relevance and usefulness have also been enhanced by the availability of tools such as PIPE [9] and CPNTools [10].

This paper is concerned with addressing model interoperability [11,12] between Sequence Diagrams and Petri Nets, through a model driven approach. This is achieved through a model transformation framework, SD2PN, which supports a seamless transition between these heterogeneous models and allows for the integration of different toolsets as shown in Fig. 1. A designer creates a model of a system as Sequence Diagrams using UML tools, and performs the required analysis in Petri Nets using Petri Net tools. This combination of model interoperability and tool integration results in a significant reduction of complexity in software development in general and model analysis in particular. This is achieved by the automated transformation that allows complex analysis to be performed by using Petri Net tools without an extensive knowledge of Petri Nets themselves. The system design phase is facilitated by a combination of a user-friendly interface of UML tools and the rigorous analytical framework of the Petri Net tools. The behaviour of a Personal Area Network will be used throughout the paper to illustrate the transition between Sequence Diagrams and Petri Nets and the analysis that can be applied.

The remainder of this paper is organised as follows. Section 2 provides a review of Sequence Diagrams, Petri Nets and Model Driven Development as well as a brief introduction to the case study, which will be used throughout this paper. Section 3 describes the methods and the SD2PN framework for transforming Sequence Diagrams into Petri Nets; the transformation process is illustrated with an example from the case study. Section 4 deals with the extension of SD2PN with timeliness properties and its implications. Section 5 raises some issues for discussion and Section 6 offers some conclusions.

2. Preliminaries

This section provides a brief introduction to UML Sequence Diagrams, Petri Nets and Model Driven Development as well as the behaviour of a Personal Area Network. The behaviour of the Personal Area Network is used in the subsequent sections of this paper to illustrate the transition from Sequence Diagrams to Petri Nets.

2.1. UML Sequence Diagrams

Unified Modelling Language (UML) [4] 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 a system, its behaviour and the constraints that the system must adhere to. 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 modelling language for describing the behaviour of a system.

Sequence Diagrams are a UML 2.1 version of Message Sequence Charts [13] and they are widely used in Software Engineering [14]. Sequence Diagrams can be used in modelling complex Enterprise Systems as they provide a sequential listing of events and are also able to model parallelism and conflicts. As such, Sequence Diagrams are well suited in modelling behaviour and interactions.

Fig. 2 represents a subset of UML 2.1 Sequence Diagrams metamodel used in this paper; it includes important constructs used for specifying models with complex behaviour. The main fragments of the Sequence Diagram are represented by model

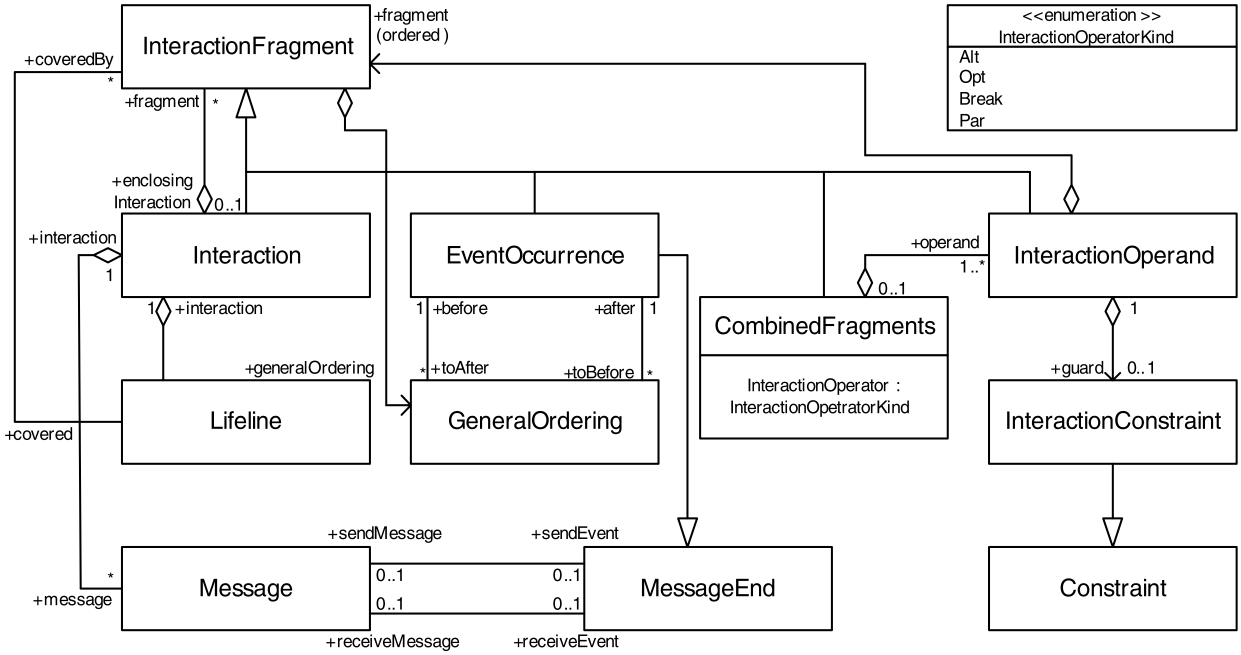

Fig. 2. Sequence Diagram metamodel.

elements *Message* and *CombinedFragments*. The model element *Message* represents the interaction between the instances of objects in the system while *CombinedFragments* is a high level addition to Sequence Diagrams and consists 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 specialisation of *MessageEnd* where each *message* is given a specific order in reference to the previous and subsequent *messages*.

From the metamodel in Fig. 2, it is evident that Sequence Diagrams have a comprehensive construct that enables the accurate representation of behaviour as well as relationship between events such as causality, concurrency and conflict. However, Sequence Diagrams and UML in general have a limitation with regards to analysis, especially when compared with more formal languages such as Petri Nets.

2.2. Petri Nets

A Petri Net is a mathematical and graphical modelling language that can be used to model a diverse set of behaviours including parallel, asynchronous, concurrent, hierarchical and stochastic as well as dynamic behaviours [8]. Similarly for Sequence Diagrams, Petri Net models graphically the flow of events in a system. The formal and mathematical nature of Petri Nets makes them ideal candidates for complementing Sequence Diagrams in addressing their shortcomings with regards to analysis.

Fig. 3 presents a metamodel for Petri Nets that will be used in this paper. A Petri Net consists of a set of *places*, a set of *transitions* and a set of *arcs* that connect *places* and *transitions*. A Petri Net also consists of a set of *markings* that assign a number of tokens to each *place*. Graphical representation of the Petri Net elements depicts *places* as circles and *transitions* as rectangles. *Arcs* are shown as directed arrows while tokens are represented by dots inside *places*.

A *transition* in a 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. When a *transition fires*, a *token* will be removed from each of the *input places* and added into one of the *output places*. A more comprehensive introduction to Petri Nets could be obtained from [8].

2.3. Model Driven Development

Model Driven Development [15] aims to promote the role of *modelling* in software development. Models in the context of MDD are captured in machine-readable representations, using languages which are widely adopted by the software industry [4]. Hence it is possible to communicate such models to various parties and reuse them. This results in lower software production cost and shorter development cycles. In this paper, MDD is further used to develop a method to benefit from advantages of using two representations of a system, Sequence Diagrams and Petri Nets.

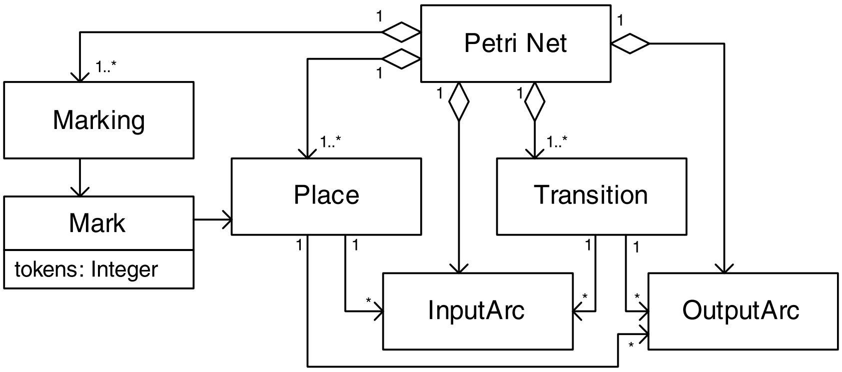

Fig. 3. Petri Net metamodel.

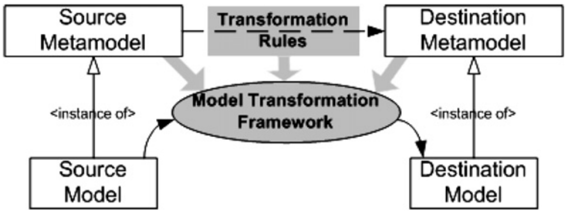

Fig. 4. Model Driven Development.

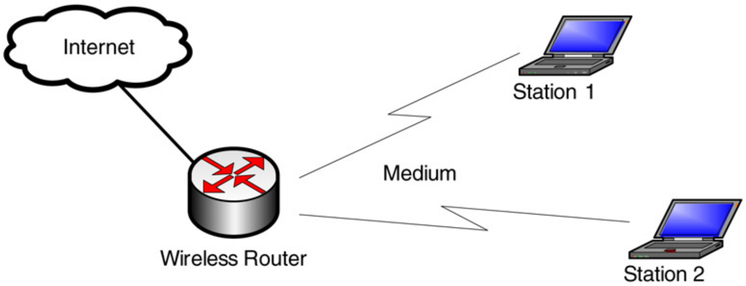

Fig. 5. Personal Area Network (PAN).

In order to allow for the integration of existing modelling software tools through the proposed approach, the standards set by Model Driven Architecture (MDA) [16], a flavour of MDD initiated by the Object Management Group (OMG), is used. Meta Object Facility (MOF) [17] is one standard for describing *metamodels*. Metamodels are themselves *models*, from which models of systems are instantiated. MOF can be compared to EBNF, which is used for defining the grammar of programming languages. MOF is a blueprint from which *MOF Compliant metamodels* are created.

Fig. 4 gives an outline of MDA and the process of *Model Transformation*. A number of *Transformation Rules* are used to specify 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]. 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.

2.4. Case study: Personal Area Network

In order to demonstrate the role of Model Driven Development in facilitating transitions from Sequence Diagrams to Petri Nets while insulating the user from the underlying complexity, a case study featuring the behaviour of a Personal Area Network (PAN) is utilised throughout this paper. This section provides a brief introduction to the Personal Area Network.

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 [21]. As the example is only meant to illustrate the capabilities of the model transformation, deeper technical details are omitted from this description.

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 [21] for more information.

Several assumptions were made in this case 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 categorised 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 case study.

3. SD2PN Model Transformation and analysis

The transition from Sequence Diagrams to Petri Nets (SD2PN) allows model interoperability between the user-friendly interface of UML and the formal mathematical nature of Petri Nets. This transition is achieved by the model transformation tool SD2PN [22]. The transformation involves three distinct phases:

Phase 1: Decomposition of Sequence Diagrams into fragments.
Phase 2: Transformation of each fragment into a Petri Net block.
Phase 3: Composition of the Petri Net blocks using *morph* and *substitute*.

These phases will be detailed in Sections 3.1, 3.2 and 3.3 and their deployment will be illustrated using a Sequence Diagram describing the behaviour of a Personal Area Network. Following the completion of the model transformation, the resulting Petri Net will be subjected to structural and behavioural analysis.

3.1. Decomposition of Sequence Diagrams into fragments

The decomposition process involves the identification of the model element in the Sequence Diagram based on the metamodel in Fig. 2. The model elements used in the model transformation process are *message* and four types of *CombinedFragments*: *alternative*, *option*, *break* and *parallel*. These model elements are hereafter referred to as *fragments*. Each type of fragment would be assigned a transformation rule in the next section to map the fragment into a Petri Net block. Examples of these fragments are illustrated in Fig. 6 where a Sequence Diagram is decomposed into 15 numbered fragments based on the model elements.

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.

Throughout the decomposition process, the causality order between these fragments is preserved in the metamodel element *GeneralOrdering*. In the Sequence Diagrams, this ordering is the same as top–down visual ordering. The hierarchical order between elements is also preserved in the metamodel as indicated by the relationship between *CombinedFragment* and *InteractionFragment*. As a result, the behaviour of the original Sequence Diagram could be incorporated into the resulting Petri Net to ensure that they are semantically equivalent.

3.2. Transformation of fragment into Petri Net block

This section describes how each Sequence Diagram fragment generated in Phase 1 is transformed into a corresponding Petri Net block using a series of five transformation rules; one rule for each type of fragment.

Rule 1 – Message: A *message* is either a call for the execution of an *operation* or sending and receiving a signal [4]. The execution of a *message* m in a Sequence Diagram fragment is depicted as the *firing* of a *transition* t in the corresponding Petri Net block. Places s_1 and s_2 model a pre-condition and a post-condition for the firing of the transition as shown in Fig. 7(a). These places will be used to create correct causality of events within the sequence diagram. As a further condition to this rule, if m is the first *message* in the Sequence Diagram, then s_1 in the corresponding block of Petri Net must be given a *token* to signify the start of the Petri Net and to allow the transitions to *fire*.

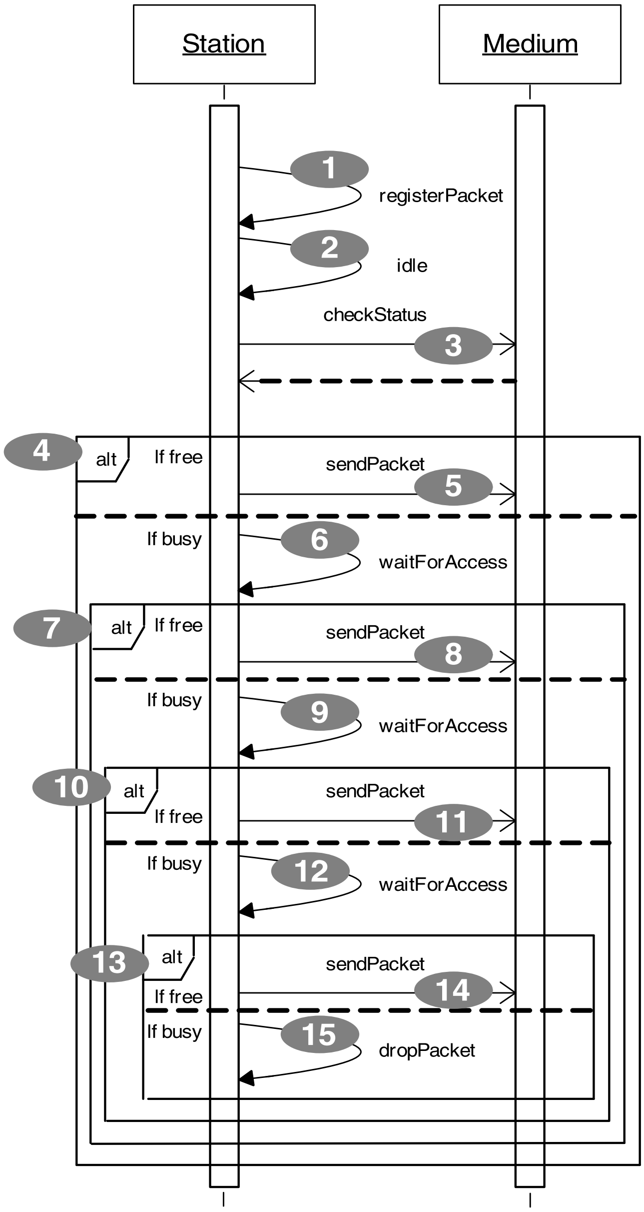

Fig. 6. Sequence Diagram for a station in PAN.

Rule 2 – Alternative: The Interaction operator *alternative* specifies a different set of events that may occur based on the conditions in the fragment [4]. In order to preserve the semantics, this fragment is represented as a Petri Net block that starts with a place s_1 that splits into two transitions t_1 and t_2. These two transitions denote the different alternative scenarios in the Sequence Diagrams and will each map into a *placeholder* block ph_1 and ph_2 respectively, which represent *alt_fragment1* and *alt_fragment2*. These placeholders will later be *substituted* with the actual events inside the fragment. They will then map into transitions t_1 and t_2 to signal the end of the *alternative* fragments and will terminate at place s_2 as shown in Fig. 7(b).

Rule 3 – Option: Interaction operator *option* can be treated in the same way as an *alternative* fragment because of the similarity of their constructs. The same block of Petri Net in Fig. 7(b) is used, with the exception of ph_1 and ph_2 representing *opt_fragment1* and *opt_fragment2* instead.

Rule 4 – Break: *Break* consists of a *guard* (condition) such that when it is satisfied, the operation *breaks* (i.e. terminates) [4]. This is modelled with the help of two transitions: t_1 for the case where the guard fails and t_2 for when the guard is satisfied. Transition t_1 connects to ph_1, which represents *break_fragment1*, which is the set of events that happen if the break condition is not satisfied while t_2 leads to place X, which is the terminal node. The placeholder ph_1 is then connected to a transition t_3 as shown in Fig. 7(c) to mark the termination of the block at s_2.

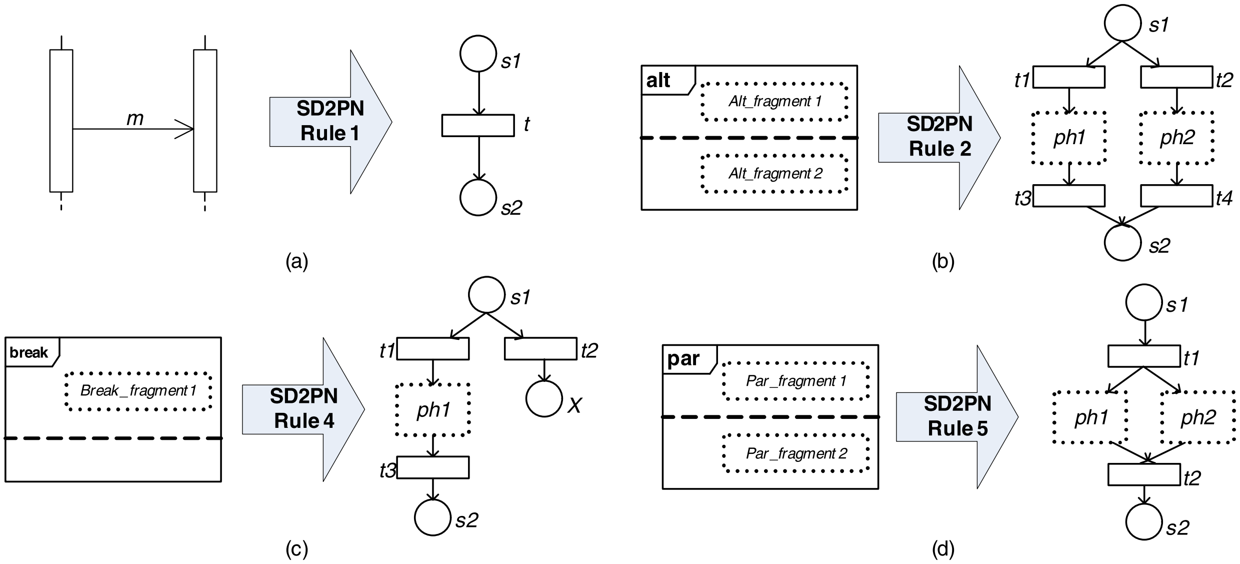

Fig. 7. SD2PN Model Transformation rules.

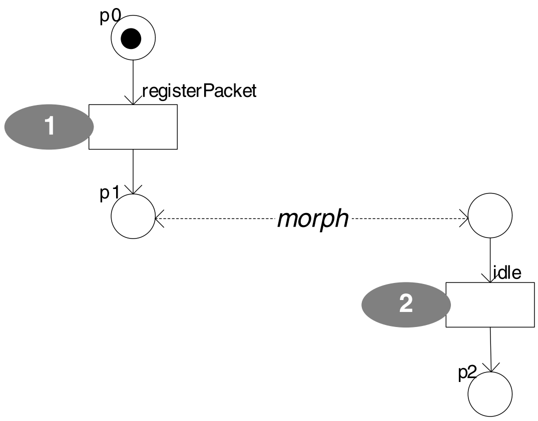

Fig. 8. Example of *morph* in SD2PN.

Rule 5 – Parallel: A *parallel* operator specifies that two sets of events should occur concurrently without any pre-defined set of conditions [4]. As depicted in Fig. 7(d), the corresponding block of Petri Nets must ensure the parallel execution of *par_fragment1* and *par_fragment2*.

3.3. Composition of the Petri Net blocks using morph and substitute

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. A closer examination of the five transformation rules from Phase 2 reveals that each rule produces a Petri Net block with a single input place and a single output place. This allows the composition of the Petri Net blocks to be conducted using *morph* and *substitute*.

Morph is used to compose causality relationship between Petri Net blocks. Calling a *morph* function with two Petri Net blocks results in the post-condition of the first block being morphed with the pre-condition of the second block, as shown in Fig. 8.

The function *substitute* is used for composing hierarchical behaviour between Petri Net blocks. *Substitution* is used only for replacing a *placeholder* with a complete Petri Net block as shown in Fig. 9.

The process of composing the Petri net blocks starts with the mapping of the causal relationships. This mapping requires calling the *morph* function recursively for each causal relationship in the original Sequence Diagram. Once all the causal relationships are mapped, the hierarchical relationships between the Petri Net blocks are considered. The hierarchical

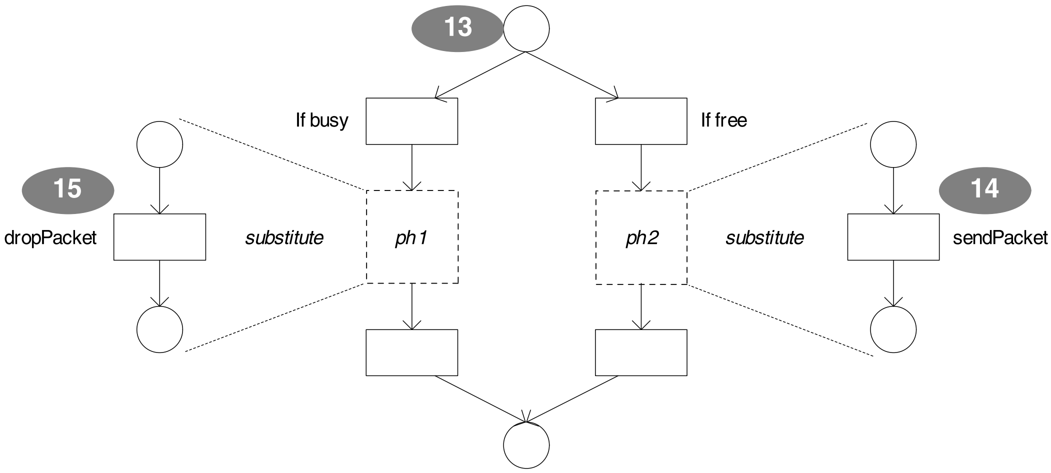

Fig. 9. Example of *substitute* in SD2PN.

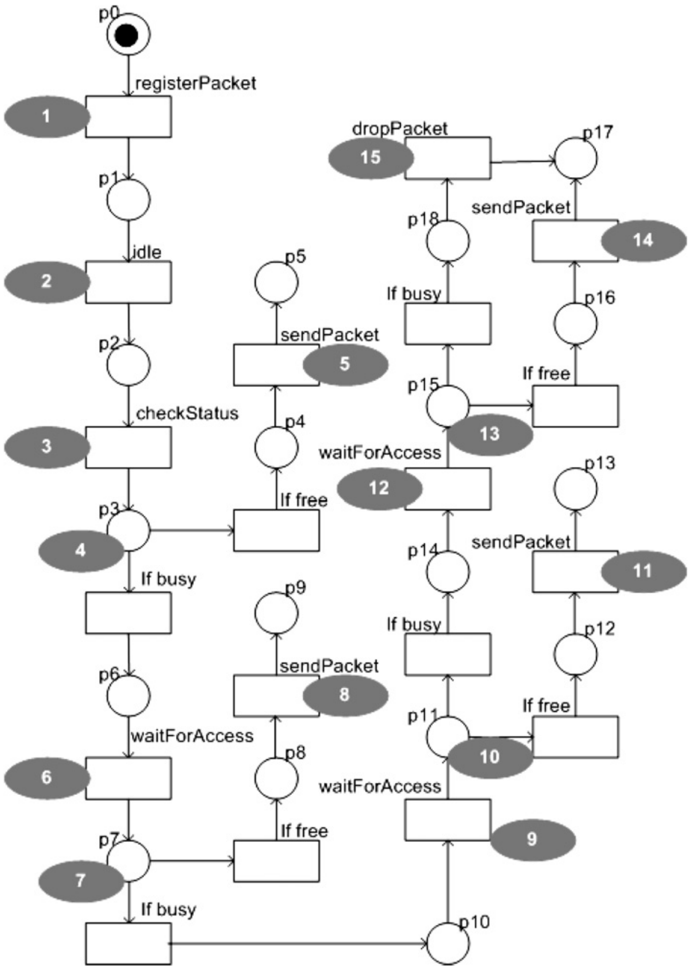

Fig. 10. Petri Net for a station in PAN.

relationships are mapped by recursively applying the *substitute* function for every *placeholder* that exists in the Petri Net blocks.

The Petri Net in Fig. 10 is the result of applying the SD2PN model transformation to the Sequence Diagram in Fig. 6. Each numbered Petri Net block corresponds to the original numbered Sequence Diagram fragment and the order of events from

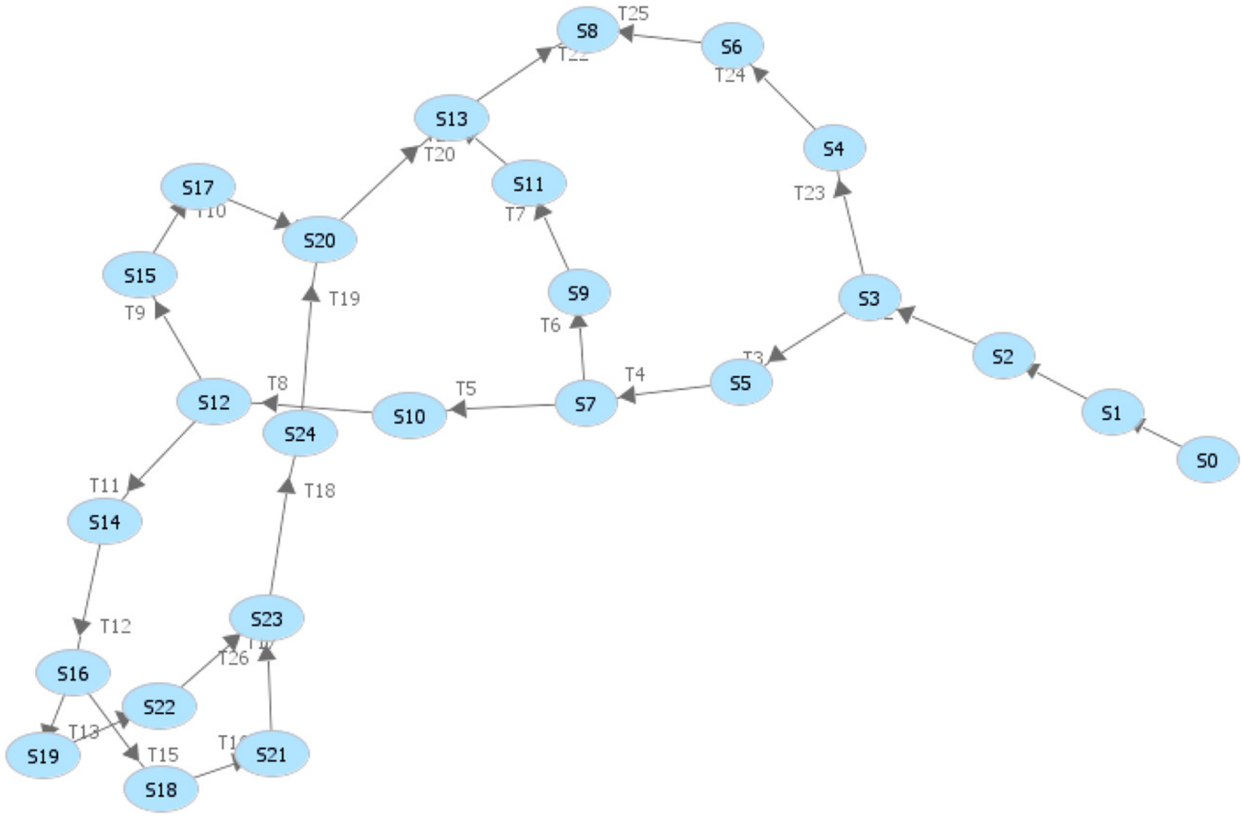

Fig. 11. Reachability Graph generated using PIPE.

the original Sequence Diagram is preserved through the execution of *morph* and *substitute*. Thus the Petri Net in Fig. 10 is considered semantically equivalent[1] to the Sequence Diagram in Fig. 6. This allows the Petri Net to be analysed using widely available Petri Net tools such as CPNTools [10] and PIPE [9]. This would be further elaborated in the next section.

3.4. Analysis of the resulting Petri Net

The mathematical nature of Petri Nets creates a strong base for various types of analysis. Murata [8] outlines a number of analysis methods and indicates how they relate to the problems in designing an enterprise system including structural analysis methods such as *liveness* and *boundedness* as well as behavioural analysis methods such as *reachability* analysis. A liveness analysis checks the system for deadlocks while a boundedness analysis is used to check the effect of the system on the buffers and registers when storing intermediate data. On the other hand, a reachability analysis is used to study the dynamic properties of a system e.g. how one action may affect the chances of an event happening in the future.

In the case of the Petri Net in Fig. 10, PIPE [9] was used to perform a structural analysis on the system. The liveness and boundedness of the system were computed through State Space Analysis where 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 confirmed that the Petri Net was not only live and bounded; it was also safe (bounded with a value of 1).

Subsequently, a behavioural analysis was conducted in the form of a Reachability Graph generated using the Petri Net tool PIPE as shown in Fig. 11. The Reachability Graph identifies all the different states of the Petri Net and determines whether each state is reachable from the initial marking of the Petri Net. The graph in Fig. 11 shows that every state in the Petri Net is reachable through a series of event.

The structural and behavioural analysis performed earlier highlights the critical nature of Petri Nets in determining the usability of a system. A system with deadlocks does not terminate while a system that is not bounded will overflow the buffers and registers of a host machine. The reachability analysis on the other hand allows the system designers to analyse all possible aspects of a system that may be affected by a user-initiated action.

By providing for a seamless transition from Sequence Diagrams to Petri Nets, SD2PN allows the analysis capabilities of Petri Nets to be applied to Sequence Diagrams while masking the complexity behind the model transformation. One

[1] The semantic equivalence between every Sequence Diagram and its corresponding Petri Net generated via SD2PN has been previously established in [7] using a common semantics domain.

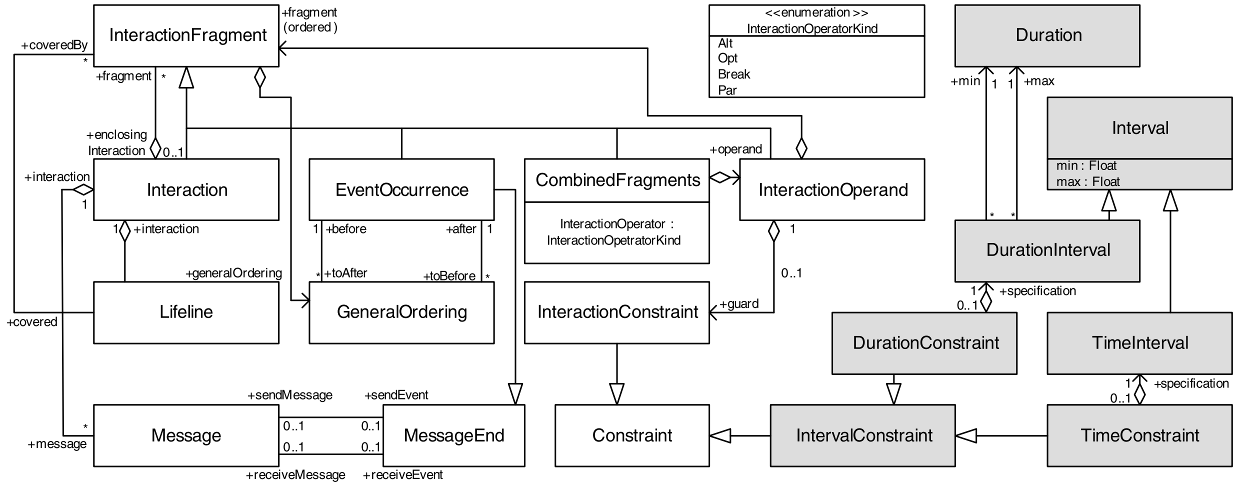

Fig. 12. Sequence Diagram metamodel augmented with timeliness properties.

type of analysis, performance analysis, could not be performed with conventional Petri Nets. This is due to the inability of conventional Petri Nets to support time constraints. This shortcoming is the motivation behind extending SD2PN with timeliness properties.

4. Extension of SD2PN with timeliness properties

It was established that SD2PN allows a model level interoperability between Sequence Diagrams and Petri Nets in such a way that a system could be designed using Sequence Diagrams and analysed as a Petri Net without any remodelling. However, it was also determined that the scope of the analysis in conventional Petri Nets did not extend to performance analysis, which is a critical factor in real-time systems. Thus, an enhancement to the model transformation was introduced in [23] where SD2PN is augmented with timeliness properties. This section details the enhancement and illustrates the significance of integrating time as a component in the model transformation.

The process of extending SD2PN with timeliness properties is conducted by enhancing both the metamodels of the Sequence Diagrams and the Petri Nets with time constraints. This is followed by the enhancement of the transformation rules to include the new time constraints specified in the metamodels. Both these enhancements are detailed in Sections 4.1 and 4.2 respectively. The composition of the Petri Net blocks using *morph* and *substitute*, is not affected thanks to the structural consistency of the transformation rules.

4.1. Metamodel enhancement

The extension of SD2PN with timeliness properties requires the enhancement of the metamodels of Sequence Diagrams in Fig. 2 and Petri Nets in Fig. 3.

4.1.1. Sequence Diagrams

To allow time constraints to be present in Sequence Diagrams, the Sequence Diagram metamodel in Fig. 2 is enhanced with time constraints. Fig. 12 presents an enhanced metamodel for Sequence Diagrams where the shaded elements in the metamodel represent the extensions that signifies the addition of time properties into Sequence Diagrams. The shaded elements are adapted from “Common Behaviours”, Chapter 13 of the UML 2.1 Superstructure [4].

Interval and *Duration* are the two types of time-related constraints added into the metamodel. Interval represents a time frame with a maximum and minimum value where the occurrence of a specific event must be within the maximum and minimum value [4,24]. A Duration is defined as the temporal distance between two time instances [4,24]. A Duration consists of only one value and an event associated with a particular Duration could only occur on the exact time specified by the Duration. Both Interval and Duration are syntactically represented textually inside curly brackets as specified in [4,24] and each value is expressed as *float* instead of *Value Specification* in order to manage the constraints more accurately and to keep the metamodel to a minimum.

Fig. 13 shows an example of a Sequence Diagram that features both types of time constraint, Interval and Duration. The Interval between the sending and receiving events of $m2$ indicates that the completion (sending and receiving) of $m2$ takes between θ and $\theta + 3$ to occur, where θ is a constant. The Duration between $m1$ and $m2$ on the other hand indicates that after $m1$ is completed, the state is preserved for the duration of θ before $m2$ could be sent.

The presence of Interval and Duration in the Sequence Diagram could present a unique case that is not represented in the previously defined fragments. The example in Fig. 13 shows the presence of a Duration that is not attached to a *message*.

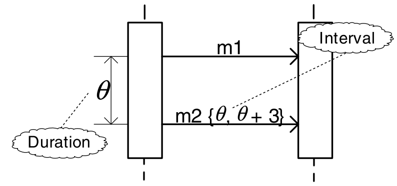

Fig. 13. Example of a Sequence Diagram with time constraints.

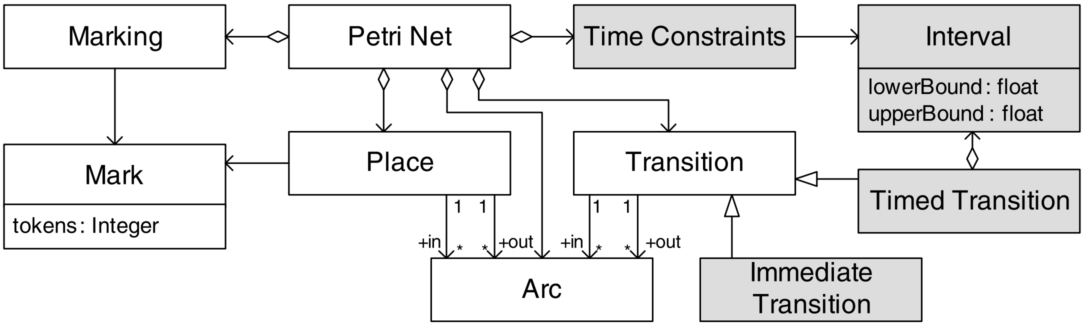

Fig. 14. Petri Net metamodel with timeliness properties.

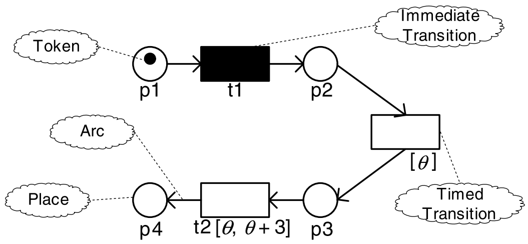

Fig. 15. Example of a Timed Petri Net.

This warrants the inclusion of an additional fragment type and an additional transformation rule that will be addressed in Section 4.2.

4.1.2. Petri Nets

The enhancement of the Sequence Diagram metamodel with time constraints introduces an inconsistency between the source and the destination metamodels of the model transformation. To allow the Sequence Diagrams to be accurately mapped into Petri Nets, the Petri Net metamodel has to be enhanced with time constraints as well.

The addition of constraints to an ordinary Petri Net results in a type of Petri Net called Timed Petri Net [25]. Fig. 14 represents the metamodel of Timed Petri Net where the shaded elements refer to the extension of the metamodel in Fig. 3 with time properties.

The shaded elements in the metamodel in Fig. 14 include Interval and two specialisations of *transition*; *immediate transition* and *timed transition*. The Intervals are expressed as closed intervals [25] and consist of an upper and lower bound of type *float*, to be consistent with Sequence Diagrams. Intervals are connected to *transitions*. For a *transition* to *fire*, it must be *enabled* and once *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 Fig. 15 where the transition $t2$ has a time constraint with the closed interval $[\theta, \theta + 3]$. The transition $t2$ can only fire under two conditions: it must be enabled and the clock must be between θ and $\theta + 3$.

Two types of *transition* are identified in Fig. 15, *immediate transitions* and *timed transitions*. *Immediate transitions*, which are *transitions* without time constraints, are depicted as black rectangles while the *timed transitions* are depicted as white rectangles. An *immediate transition* may be considered as 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 bounds. If the upper and lower bounds of the interval are the same, as in [50, 50], it is abbreviated as [50].

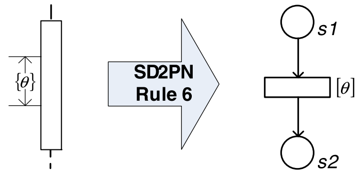

Fig. 16. Rule 6 of SD2PN.

4.2. Transformation rules enhancement

An MDD model transformation consists of three main components; a source metamodel, a destination metamodel, and a set of transformation rules. Both the source and the destination metamodels have been enhanced to include timeliness properties; this requires the transformation rules to be enhanced as well.

In this section, Rule 1 is modified to accommodate the existence of the two types of *transition* while Rules 2 through 5 remain unchanged since there are no intervals or durations that are attached to CombinedFragments. Every transition in Rules 2 through 5 is therefore designated as *immediate transitions*.

Rule 1 from Section 3.2 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 cases for the execution of this rule:

Case 1: If a message has an interval associated with it e.g. $\{10\ldots30\}$, the *transition* t in the resulting Petri Net block is designated as a *Timed Transition* with a closed interval $[10, 30]$.

Case 2: If a message has a duration associated to it e.g. $\{20\}$, the *transition* t in the resulting Petri Net block is designated as a *Timed Transition* with a closed interval $[20, 20]$ or abbreviated as $[20]$.

Case 3: If a message does not have any time properties attached to it, the *transition* t in the resulting Petri Net block is designated as a *transition* with a closed interval $[0, 0]$ or an *Immediate Transition*.

Rule 6 – Duration: To accommodate the new type of fragment defined in Section 4.1.1, an additional rule is introduced to SD2PN. Rule 6, as illustrated in Fig. 16 maps time properties that are not attached to any particular message into a Petri Net block. This results in a Petri Net similar to Rule 1. However, there are only two possible execution cases for Rule 6:

Case 1: If a time constraint has an interval associated to it e.g. $\{10\ldots30\}$, the *transition* t in the resulting Petri Net block is designated as a *Timed Transition* with a closed interval $[10, 30]$.

Case 2: If a time constraint has a duration associated to it i.e. $\{20\}$, the *transition* t in the resulting Petri Net block is designated as a *Timed Transition* with a closed interval $[20, 20]$ or abbreviated as $[20]$.

4.3. Enhanced SD2PN model transformation

The metamodel and transformation rules enhancement from the previous section result in an enhancement of the SD2PN model transformation. However, the fundamentals of the model transformation process described in Section 3 remain unchanged. The three phases of SD2PN are still valid:

Phase 1: Decomposition of Sequence Diagrams into fragments.
Phase 2: Transformation of each fragment into a Petri Net block.
Phase 3: Composition of the Petri Net blocks using *morph* and *substitute*.

The process of Sequence Diagram decomposition in Phase 1 is enhanced through the introduction of an additional fragment type. In Section 3.1, five fragment types were introduced; *message* and *CombinedFragments* of type *alternative*, *option*, *break* and *parallel*. However, for the purpose of the time enhanced model transformation, an additional fragment type is introduced, as described in Section 4.1.1.

Phase 2 of the model transformation makes use of a set of six transformation rules specified in Section 4.2, one for each fragment type. The rules consist of an enhancement to the set of five transformation rules of Section 3.2 and the addition of Rule 6 in Section 4.2. The composition of the Petri Net blocks in Phase 3 of SD2PN remains unchanged from Section 3.3 since the enhancements made to the transformation rules in Section 4.2 do not affect the structural consistency of the Petri Net blocks i.e. all Petri Net blocks begin and end with a *place*. The application of the three phases results in the transformation of a Sequence Diagram into a semantically equivalent Petri Net.

In Section 3, an example of the transformation process was provided. The Sequence Diagram in Fig. 6, a representation of a Personal Area Network, was transformed via SD2PN into the Petri Net in Fig. 10. To illustrate the introduction of time as an element in the model transformation, the Sequence Diagram in Fig. 6 is augmented with time constraints, resulting in the Sequence Diagram in Fig. 17(a). Using the enhanced SD2PN model transformation, this Sequence Diagram is transformed into the Petri Net depicted in Fig. 17(b).

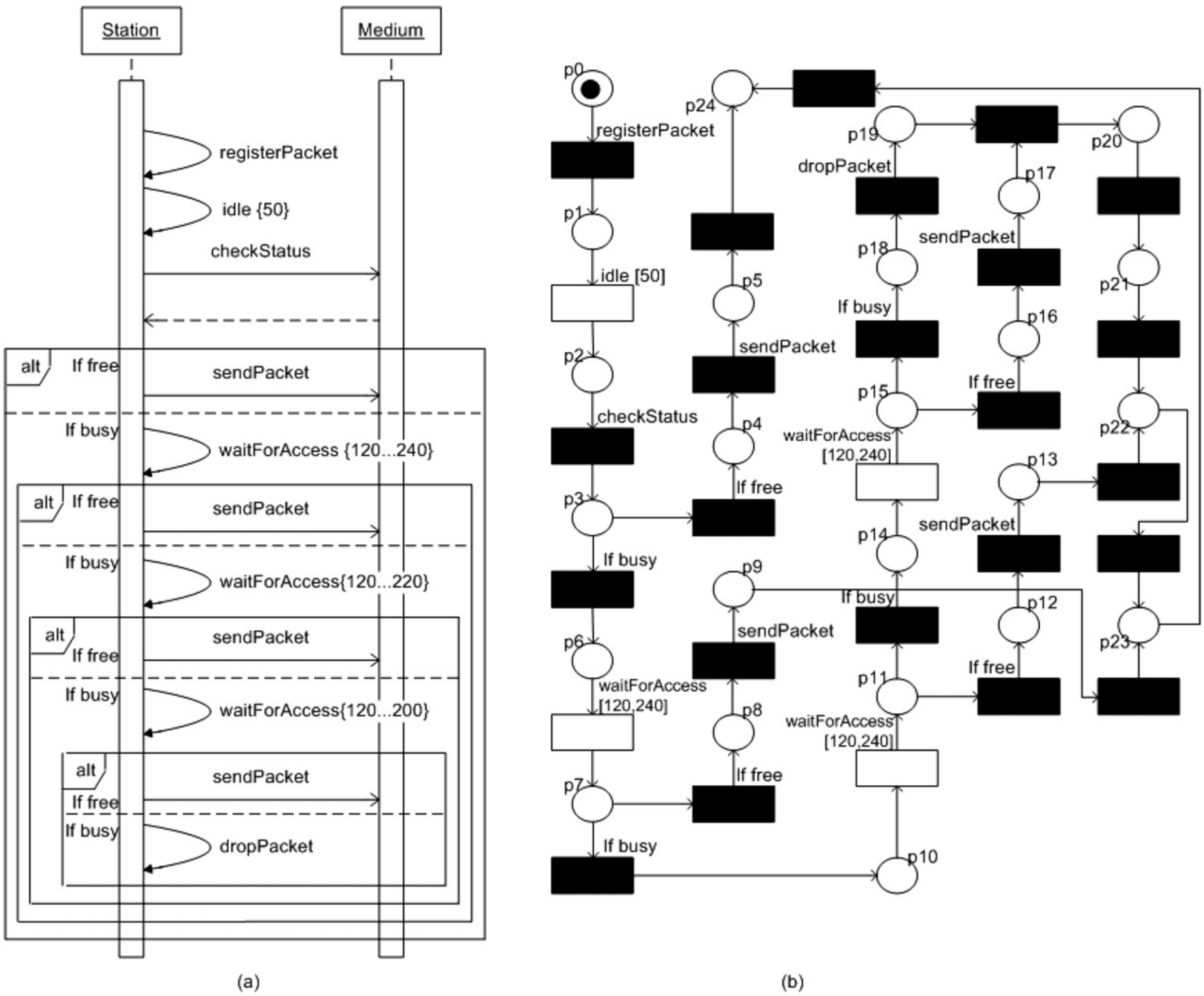

Fig. 17. (a) Sequence Diagram for a station in PAN and (b) its equivalent Timed Petri Net.

The Petri Net generated via the enhanced SD2PN in Fig. 17(b) is structurally equivalent to the Petri Net in Fig. 10; thus indicating the consistency of the model transformation. However, the introduction of timeliness properties into SD2PN vastly expands the scope of analysis that could be performed on the resulting Petri Nets.

4.4. Extended analysis of the resulting Petri Net

The extension of SD2PN with timeliness properties allows performance analysis to be performed in addition to the existing structural and behavioural analysis; time-sensitive analysis such as a cycle-time, average time, standard deviations, confidence intervals and throughput analysis can be performed, as described in Refs. [9,26].

The Petri Net in Fig. 17(b) is still amenable to the structural and behavioural analysis as described in Section 3.4. However, since there is no structural difference between the Petri Nets in Fig. 10 and Fig. 17(b), the results of the structural and behavioural analysis remain the same. The focus of the performance analysis in this case is throughput analysis; this will be used to analyse the maximum delay for a station in the Personal Area Network.

The maximum delay is calculated based on the time it takes for a station to gain access to the medium (*sendPacket*). The factor that contributes to the increase in waiting time is the number of stations. A higher number of stations will increase contention between the stations. This inevitably leads to a longer maximum waiting period. For the case of a single station in the PAN, the Petri Net would be the same as the Petri Net in Fig. 17(b). However, for cases where there is more than one station, the Petri Net in Fig. 17(b) would be replicated for each station. The throughput analysis will determine the maximum waiting time based on the last station to gain access to the medium via the *message* 'sendPacket'. For example, in a case where there are two stations trying to gain access to the medium, after registering the packet (firing of *registerPacket* transition), in Fig. 17(b), both stations will face a mandatory idle time of 50 μs (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

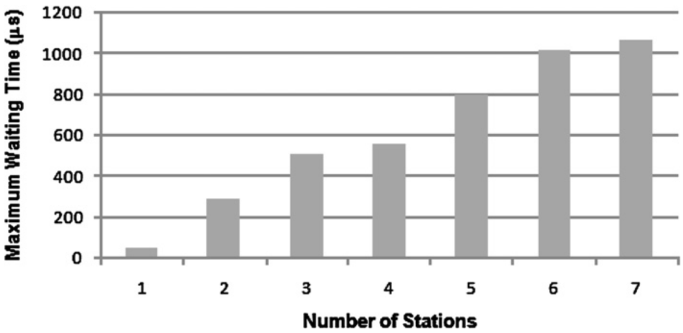

Fig. 18. Maximum waiting time analysis result.

will have to wait between 120 µs and 240 µs (firing of *waitForAccess* transition), thus a maximum waiting time of 290 µs (= 240 µs + 50 µs).

The graph in Fig. 18 indicates the maximum delay that a station may face before gaining access to the medium to send a packet based on the throughput analysis. The number of stations is limited to 7 to ensure there are no collisions; this is based on the previous assumption that the *contention window* (CW) does not increase.

In the example of the Petri Net in Fig. 17(b), the analysis performed could provide a basis to optimise related protocols to ensure a better performance. This provides a domain of interoperability from Sequence Diagrams to Petri Net allowing not only structural and behavioural analysis, but also performance analysis. The performance analysis is not limited only to throughput analysis. Various other performance analyses such as cycle-time analysis, average time, standard deviations, and confidence intervals analysis can also be performed. Various analysis methods are covered in detail in Refs. [9,26].

5. Discussion

The dichotomy between the design and analysis domains in software development exists due to the trade-off between the ease-of-use of UML and its lack of precision. The requirement for analysis using a formal language, as a sequential step to a less formal design phase, results inevitably in the generation of heterogeneous models. One approach to addressing this issue is to enhance the formalism of languages used in the design. Recent work in this area has been marked by a concerted effort aimed at formalising UML by integrating formal methods techniques into the model [27–31]. 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. Integrating formal method techniques with UML is an active area of research. For example, Evans et al. [27] propose the use of Z as the underlying semantics for Class Diagrams to deal with the static aspects of models. Küster-Filipe [32] presents a semantics for Sequence Diagrams based on Labelled Event Structures. However, it has been noted that formalisation increases complexity and is often achieved at the expense of simplicity. The main challenge is to strike a balance between precision and ease of use. This can be achieved by creating a domain for interoperability between UML and a formal language.

The use of model transformation in supporting interoperability between design and analysis models in software engineering is increasingly gaining importance in the software development community. Anastasakis et al. [33] deals with issues related to model transformation from UML to Alloy [34]. They propose UML2Alloy [35] as a tool for the analysis of UML models via the Alloy framework. UML2Alloy allows the analysis of static models which are qualified with OCL constraints [36]. Alloy does not however provide the mechanisms required for capturing complex dynamic behaviour such as parallelism.

The choice of Petri Nets as the formal language for performing behavioural analysis is due to its flexibility, expressiveness and power as well as wide availability of tools. Petri Nets are also a popular choice for representing dynamic models. For example, Van der Aalst [37] makes use of Petri Nets for the analysis of Workflow Management Models. Using the analytical capabilities of Petri Nets, the Workflow Models are analysed for validation, verification, and performance analysis. Vanhatalo et al. [38] decomposed Business Process Models into blocks of Single Entry Single Exit (SESE) models and analysed each blocks independently. This technique makes it possible to analyse the liveness and soundness of a Business Process Model. Moreover, they state that the fastest technique used in the analysis of Workflow Models involves transforming them into Free Choice Petri Nets [39,40]. Free Choice Petri Nets is a subclass of Petri Nets where conflicting behaviour and concurrent behaviour may occur, but not simultaneously. This subclass of Petri Net is predominantly used for effective and efficient analysis of a systems [38]. Free Choice Petri Nets are also proving to be particularly suitable for the analysis of large-scale systems [37,38], an important feature that widens the scope of the application of the proposed framework to encompass similar systems.

In the transformation process SD2PN generates Free Choice Petri Net. This result has been established and proved in [22]. The seamless transition from Sequence Diagrams to Petri Nets takes advantage of their suitability for formal analysis and support for the investigation of various properties such as liveness, safeness and deadlocks detection [39]. It is also possible to integrate existing Petri Net tools into a tool set, so that for a given UML Sequence Diagram, by applying a sequence of tools, the user can *automatically* receive feedback on, among others, the liveness, safeness and deadlock freeness of the model. This complete tool integration 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. This results in *full* model interoperability where the less formal Sequence Diagrams are transformed into the formal expression of Petri Nets, and the analysis of the Petri Nets is returned as feedback to the user in a less formal style. This enables the user to make amendments to the original Sequence Diagram.

It was established in this paper that the model interoperability and tool integration provided by SD2PN could be used to generate Petri Net models from Sequence Diagrams and provides a basis for structural, behavioural and performance analysis using Petri Net tools such as PIPE and CPNTools. The development and deployment of the tool owe much to the abstract approach that MDD promotes. MDD provides a platform for models to be reused across domains; in this case those identified by the design and formal analysis phases. Reusing models across domains result in shorter development cycle and lower production cost, and in turn reduce complexity in the software development process. This is evident in SD2PN where the model created in the software design domain could be reused in the analysis domain, allowing model interoperability between Sequence Diagrams and Petri Nets. The transition between the two models is well supported by tool integration.

SD2PN is till under development and suffers from some limitations; among these is the inability to map the data flow and data constraints into Petri Nets. This limitation can be an impediment to the modelling of some complex systems. Conventional Petri Nets and Timed Petri Nets are unable to handle data types, and as such incapable of modelling data flow or data constraints. This limitation could be addressed by using another flavour of Petri Nets: Coloured Petri Nets (CPN). This will be the focus of future research.

6. Conclusion

This paper has presented a method of model interoperability, which makes use of Model Driven Development in order to bridge the gap between the design and analysis phases of software development. The framework introduced in this paper, SD2PN, provides a seamless transition from Sequence Diagrams to Petri Nets. This allows for models to be conveniently designed in UML while taking advantage of the rigorous mathematical analysis afforded by Petri Nets; it also supports the integration the different toolsets across incompatible domains. Petri Nets are well suited for structural and behavioural analysis of a model thanks to their expressive power and flexibility. Moreover, the addition of time properties as a significant feature of the model transformation allows performance analysis to be conducted on real-time and time-sensitive models. The proposed approach has been evaluated with a model of the behaviour of a Personal Area Network. The model was also used as a vehicle for illustrating the difference between the structural and behavioural analysis of conventional Petri Nets and the performance analysis of Timed Petri Nets.

References

[1] D. Sannella, A Survey of formal software development methods, in: Tech. Rept. ECS-LFCS-88-56, Edinburgh University, 1988.
[2] A.P. Sheth, Changing focus on interoperability in information systems: from system, syntax, structure to semantics, in: Interoperating Geographic Information Systems, Kluwer Academic Publishers, 1999.
[3] N. Medvidovic, R.F. Gamble, D.S. Rosenblum, Towards software multioperability: bridging heterogeneous software interoperability platforms, in: Fourth International Software Architecture Workshop (ISAW-4), Limerick, Ireland, 2000.
[4] OMG, OMG Unified Modelling Language (UML) Superstructure 2.1, available at www.omg.org, 2007.
[5] ArgoUML, ArgoUML web site, sourceforge.net/projects/argouml, 2005.
[6] Poseidon, Poseidon for UML, from Gentleware, www.gentleware.com/, 2006.
[7] J. Whittle, Formal approaches to systems analysis using UML: an overview, J. Database Management 11 (4) (2000).
[8] T. Murata, Petri Nets: properties, analysis and applications, Proc. IEEE 77 (4) (1989) 541–580.
[9] P. Bonet, et al., PIPE v2.5: a Petri Net tool for performance modeling, in: XXXIII Conferencia Latinoaméricana de Informática, 2007.
[10] CPNTools, Computer Tool for Coloured Petri Nets, http://wiki.daimi.au.dk/cpntools/.
[11] H. Kühn, M. Murzek, F. Bayer, Horizontal business process model interoperability using model transformation, in: INTEREST'2004 Workshop at ECOOP 2004, Oslo, Norway, 2004.
[12] D. Bertolini, et al., A tropos model-driven development environment, in: 18th Conference on Advanced Information Systems Engineering (CAiSE-06), Springer-Verlag, Luxembourg, 2006.
[13] S. Mauw, M. Reniers, An algebraic semantics of basic message sequence charts, Comput. J. 37 (1994) 269–277.
[14] J. Campos, 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, Springer, Turku, Finland, 2006.
[15] T. Stahl, M. Volter, Model Driven Software Development; Technology Engineering Management, Wiley, 2006.
[16] MDA, Model Driven Architecture, Object Management Group, www.omg.org/mda/, 2005.
[17] MOF, Meta Object Facility (MOF) 2.0 Core Specification, Object Management Group, available at http://www.omg.org, 2004.
[18] ATLAS, ATLAS, Université de Nantes, http://www.sciences.univ-nantes.fr/lina/atl/, 2005.
[19] Kermeta, Triskell Metamodelling Kernel, www.kermeta.org, 2005.
[20] D.H. Akehurst, et al., SiTra: Simple Transformations in Java, in: ACM/IEEE 9TH International Conference on Model Driven Engineering Languages and Systems, 2006.
[21] J.H. Schiller, Mobile Communications, Pearson Education, 2003.
[22] M.A. Ameedeen, 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), München, Germany, 2008, pp. 213–221.
[23] M.A. Ameedeen, B. Bordbar, R. Anane, A model driven approach to analysis of timeliness properties, in: Fifth European Conference on Model Driven Architecture Foundations and Applications (ECMDA 2009), in: Lecture Notes in Comput. Sci., vol. 5562, 2009, pp. 221–236.

[24] B.P. Douglass, Doing Hard Time: Developing Real-time Systems with UML, Objects, Frameworks and Patterns, Object Technol. Ser., Addison–Wesley, 1999.
[25] J. Wang, Timed Petri Nets: Theory and Application, Springer, 1998.
[26] K. Jensen, L.M. Kristensen, L. Wells, Coloured Petri Nets and CPN tools for modelling and validation of concurrent systems, Internat. J. Software Tools Technology Transfer (STTT) (2007).
[27] A.F. Evans, Robert France, Emanuel Grant, Towards formal reasoning with UML models, in: Proc. OOPSLA'99 Workshop on Behavioral Semantics, 1999.
[28] D. Kim, et al., A UML-based metamodeling language to specify design patterns, available from: http://www.cs.colostate.edu/~georg/aspectsPub/WISME03-dkk.pdf, 2003.
[29] S.-K. Kim, A Metamodel-Based Approach to Integrate Object-Oriented Graphical and Formal Specification Techniques, University of Queensland, Brisbane, Australia, 2002.
[30] C. Snook, M. Butler, UML-B: Formal modelling and design aided by UML, in: ACM Transactions on Software Engineering and Methodology, 2006.
[31] R. Marcano, N. Lévy, Transformation rules of OCL constraints into B formal expressions, in: 5th International Conference on the Unified Modeling Language, Dresden, Germany, 2002.
[32] J. Küster-Filipe, Modelling concurrent interactions, Theoret. Comput. Sci. 351 (2) (2006) 203–220.
[33] K. Anastasakis, et al., UML2Alloy: a challenging model transformation, in: ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems, 2007.
[34] AlloyAnalyzer, Alloy Analyzer Website, http://alloy.mit.edu/beta/, 2005.
[35] B. Bordbar, K. Anastasakis, UML2Alloy: A tool for lightweight modelling of Discrete Event Systems, in: IADIS International Conference in Applied Computing 2005, Algarve, Portugal, 2005, pp. 209–216.
[36] OMG, UML 2.0 OCL Specification, Document Id: ptc/03-10-14 2003, OMG Final Adopted Specification, available from: http://www.omg.org/docs/ptc/05-06-06.pdf.
[37] W.M.P. van der Aalst, The application of Petri Nets for workflow management, J. Circuits Systems Computers 8 (1) (1998) 21–66.
[38] J. Vanhatalo, H. Volzer, F. Leymann, Faster and more focussed control-flow analysis for business process models through SESE decomposition, in: Fifth International Conference on Service Oriented Computing, Springer, Vienna, Austria, 2007, pp. 43–55.
[39] J. Desel, J. Esparza, Free Choice Petri Nets, Cambridge University Press, 1995.
[40] F. Baccelli, S. Foss, B. Gaujal, Free choice Petri Net: an algebraic approach, IEEE Trans. Automat. Control (1996).