Transforming UML Sequence Diagrams into Petri Nets

Tony Spiteri Staines
Department of Computer Information Systems, Faculty of ICT, University of Malta Msida MSD 208, Malta

Received: May 2, 2012 / Accepted: June 2, 2012 / Published: January 31, 2013.

Abstract: Sequence diagrams are an abstraction of communication modeling taking place between different entities, objects or classes. Their usefulness and expressivity lies in their ability to describe an execution trace of a particular system at a particular point in time. From their initial use as traditional message sequence charts to the current versions of sequence diagrams, like those found in the UML, these notations have undergone various changes and improvements. Their major use still relies in their ability to express the inter messaging between different entities. Sequence diagrams are not proper formal notations and these lack particular executable semantics precisely because of their usefulness to different scenarios and levels within the process of systems engineering. Various proposals have been given to formalize and improve these notations. Petri nets are graphical formalisms that support rigorous verification. This paper defines and explains the relationship between sequence diagrams and normal Petri nets. This approach is used to transform sequence diagrams into Petri nets. A case study of a banking application is used to illustrate the concept. Findings are discussed. The concepts can be extended to other variants of sequence diagrams.

Key words: Message sequence charts, Petri nets, sequence diagrams, system modeling, UML.

1. Introduction

A sequence diagram represents a set of collaboration between related entities. The messages exchanged between the entities represent actual interactions. Interactions represent dynamic communication and collaboration between objects or entities to achieve a particular result. Unlike communication diagrams, sequence diagrams are more expressive and less compact. A typical use case is traditionally the starting point for a sequence diagram. A use case can be refined into one or more sequence diagrams.

If the concept of quality assurance for information systems is properly understood, this implies that system analysis requires robust and checkable specifications that are properly described. Verification, validation and testing are essential for quality.

Various extensive attempts to improve and formalize sequence diagrams can be found in literature. It makes sense to transform sequence diagrams for various reasons into Petri nets for formal verification. This could be used to measure the reliability and effectiveness of the sequence diagram.

The importance of this work is the natural correspondence between sequence diagrams and normal Petri nets. The conversion or transformation is a simple solution that is applicable to different sequence diagrams. The concept is practical and repeatable and does not have to involve complex non visual formalisms. This means that it can be communicated to different stakeholders. The transformation is useful for any type of system behavior.

The paper is organized as follows: Section 2 explains sequence diagrams; Section 3 discusses the motivation for this work; Section 4 contains related works; Section 5 contains the proposed Petri net solution; Section 6 contains the transformation approach method; Section 7 presents a typical case study scenario illustrating the transformation; Section 8 gives some results and discusses the findings;
2. Sequence Diagrams

Sequence diagrams are graphical notations that depict the sequential ordering of message passing, along with other processes amongst a set of collaborating objects or classes. They include the time dimension. Activity sequence diagrams depict the message event sequence as a relationship to time. This is one feature that distinguishes them from communication or collaboration diagrams. The sequence diagram has two axes: a vertical one and a horizontal one. The horizontal level shows mainly how the communication between objects/entities is carried out. The vertical axis mainly represents the internal operations of the object/entity in relation to the passage of time. Delayed messages have representation in the vertical axis. As their name implies sequence diagram, messages show sequential ordering. If sequential ordering is considered from a temporal viewpoint, the sequential order must have a start point and an endpoint, or multiple starts and ends. Message sequence charts are an expressive visual/graphical notation useful for describing scenarios, as is typical of interworking processes or objects. Normally these are useful in the preliminary development stages as they tend to abstract the proper working of the system. Sequence diagrams are also known as MSC (message sequence charts). MSC are well researched and have formal representation based on a specification language. Literature exists for giving MSC formal rigorous semantic and system representation. It can be argued that MSC are expressively weak, as diagrammatic notations, because they are based on a modest partial event ordering concept.

MSC, sequence diagram and other similar notations, like object interaction overview diagrams, collaboration and communication diagrams, message sequence graphs etc. are not proper formalisms and require extensive extensions or additional support for proper use.

The UML sequence diagram is based on other similar diagrams and represents interactions, temporal event ordering, and relationships as regards specific processes.

In object oriented development, usually after the class definition, is complete, there is the use case design. Sequence diagrams come at a later stage. Sequence diagrams or communication diagrams are used for supporting a particular use case. Finally the model is used for creating the physical system requirements.

3. Motivation

The initial use of sequence diagrams was for object interaction in relation to software design. It must be kept in mind that several other uses exist: business process modeling, communication between systems and computer network communication modeling. Another traditional application of sequence diagrams is in the transformation of requirements expression as use case scenarios. As the requirements phase gains momentum, the sequence diagram takes a more formal role. When this happens, the sequence diagram requires more support and checking or verification. Different models should be considered until a final optimized version is produced. Then the best sequence diagram that explains the major processes is selected. The verification should be done using an executable model that is simple to construct and verify. The models have to be technically and formally correct.

In an ideal situation, the sequence diagram should
capture the requirements in the forms of good scenarios. Bad scenarios need to be detected automatically. Users tend to over complicate the sequence diagrams. Unfortunately, there is little or no indication of what constitutes a good sequence diagram. Ideally, it should be possible to lift off an executable specification for checking the validity of the diagram.

Different work focuses on the formalization of MSC and their verification using more expressive methods. However, most of these notations are not really visual.

The usefulness of using ordinary Petri nets to support sequence diagrams lies in the fact that sequence diagrams represent a form of execution trace. Message passing is a special type of event that is happening in system execution. Usually, no proper state of the system is kept. This implies that the sequence diagram does not actually represent an ongoing system process but rather a snapshot or fragment of a particular temporal scenario or process. This signifies that the sequence diagram on its own forms an incomplete or inconsistent representation of a complete activity.

If model consistency is to be kept, the actual Petri net should also represent or capture this incompleteness. Ordinary Petri nets are restrictive. The constructed Petri net can be considered to be a better refined model.

The Petri net can be considered to be a mechanism of precision because all stages or points in time that are required to complete a message or a process have to be precisely defined. The exact amount of transitions would have to be known. Consequently, any ambiguity or lack of specificity can not exist in the Petri net as opposed to the sequence diagram.

4. Related Work

Many studies have been conducted as regards message sequencing and their representation. The works considered here are just some evidence of important developments for modeling with message communication between different entities. The adaptation of sequence diagrams for different uses is obvious. It can be pointed out that sequence diagrams in their bare format lack precise semantics and can give rise to discrepancies even when modeling the same scenario.

Apart from this, the sequence diagrams used in the UML have an important role in modeling communication between objects or classes. There is a vast amount of literature on UML sequence diagrams and also their formalization. These types of sequence diagrams find their importance for the analysis of many types of information systems, business workflows, distributed processing, network communication, etc..

MSC offer intuitive and visual ways for describing requirements at an early stage in the design or analysis [1]. Here the authors propose and investigate various ways how MSC can be verified based on synchronous or asynchronous interpretation of connecting two MSC. Algorithms are proposed for solving the model checking problem within given bounded parameters. From this work, it is evident that MSC require strict model checking techniques, following proper design.

The more complex the MSC, the more strict the checks that are required. Sometimes model checking techniques are not easily understood, interpreted and are difficult to follow intuitively.

In Ref. [2] MSC are classified into textual forms HMSC (hierarchical message sequence charts). For ordering of MSC different forms are given like visual event ordering, causal ordering and races. Different forms of model checking and their properties are considered. Some solutions and partial solutions are discussed. New problems and issues are identified. Some issues point towards the undecidedness of the problems. The MSC definitely have advantage over other non visual notations or textual representation methods. Other issues for future consideration are race conditions and description.
In Ref. [3] it is clearly indicated that MSC are useful for industry to document processes and objects. The author rightly points out that MSC are expressively weak because of their modest semantic notation for partial event ordering. It is indicated that rigorous or robust MSC language is imperative for supporting use cases and scenarios. An extension to MSC is proposed to specify liveness, i.e. things that have to occur. Certain scenarios need prevention from occurring. The idea of breathing life into MSC is very important.

In Ref. [4] there is the evidence that message sequence charts can not be supported using HA (hybrid automata). This is because HA are based on special bounded checking techniques that can not deal with the “unfeasibility” of the MSC. Triggered MSC are proposed as a feasible solution for scenario based systems requirements model checking from a formal point of view. The formal semantics used here are quite elaborate.

In Ref. [5] TMSC (triggered message sequence charts) are presented, as a well founded framework, for requirements elicitation of distributed systems. The trigger concept ensures that certain requirements be met for communicating with another entity or object. The main operations in TMSC are (1) sequential composition, (2) recursion, (3) parallel composition, (4) delayed choice, (5) internal choice and (6) conjunction. The TMSC are semantically well explained.

In Ref. [6] the visual formalisms of MSC and their original uses are defined. Examples for telecommunication systems and software are given. UML sequence diagrams are a particular variant of MSC adapted for general use. LSC live sequence charts are particular variants of MSC also adapted for general use. LSC like sequence charts can be used for an executable specification. When describing the control flow between different LSGs an execution model is presented. This is called a CTP (communicating transaction process). A Petri net notation is used to represent the control flow of the CTP.

In Ref. [7] layered queuing networks are used to solve multiple resource possession in distributed computing. For modeling the service requests, typical well formed UML sequence diagrams are used. The sequence diagrams are supported using other notations and constructs. Queuing servers are explained. This work clearly shows the importance of combining sequence diagrams and other notations for proper explanation.

5. Proposed Petri Net Solution

The solution that is being proposed is to transform sequence diagrams into a Petri net notation that is closely associated with the sequence diagram’s visual appearance. Both the sequence diagram and the Petri net are based on graphs. Normal Petri nets or place transition nets are used for simplicity and bi-directional transformation.

It is evident that a sequence diagram has one or more starts but normally no termination. This is reflected in the resultant Petri net which can have multiple terminations.

Using the UML 2 superstructure specification [8], sequence diagram semantics and notations and their corresponding Petri net counterparts can be found. The given tables summarize their correspondence. The transformation approach is exhaustive because all the possible constructs have been considered.

Normally two main patterns in a sequence diagram are identifiable: (1) executable specification pattern and (2) message sending pattern. The transformation is intuitive. In the sequence diagram, behavior is depicted along both the vertical and horizontal axis. The vertical axis represents the passage of time and can have executable specification patterns.

6. Transformation Approach

Figs. 1and 2 summarize the rules for transforming a sequence diagram into a Petri net. A compact
representation shows activation in Fig. 1. This can be done because the activation is just a sequential pattern of places and transitions. Normally there are active and passive objects. An active object has a thread of control and must have activation. The active object should have a starting place for its activation, unless this is coming from another object through creation message passing.

Each individual active object or actor that has a continuous activation implies that this requires a sequential Petri net subnet. The number of transitions in the subnet can be considered to have as fixed time period $t$. The longer the activation, therefore the more transitions that would have to be included to represent this.

Conditions for transformation:
1. Sequence diagrams must be well formed;
2. Message returns or return messages must be shown or included;
3. Activation bar must be included;
4. At least one object or class should act as main object or initiator.
7. Case Study

A UML sequence diagram for a typical bank opening a new account is used to illustrate the transformation. Fig. 3 has got three active components or entities which are: bank employee, open new account UI and open new account. The whole sequence of messaging and events are initiated by the open new account. It sends a create message initiating the open new account UI. Once the interface is created, a reply is sent to the open new account. Then the main actor who is the bank employee sends message enter details (details) to open new account UI. The open new account UI forwards the message to open new account which forwards a message check (details) to existing account list. These can be considered to be nested messages, but there is no need to show this specifically in the sequence diagram. If the account is valid then a reply is sent from existing account list to open new account. Next a create new (account) message is used to create an object account. The account object sends a message added to list (account) to existing account list that adds the object to the list. If the account is invalid then an error log is generated. The temporary objects are destroyed. Finally return messages are sent for all the main nested messages. All this is illustrated in Fig. 3.

The transformation rules in Section 6 are used to obtain the Petri net model in Fig. 4.

8. Results and Discussion

It has to be pointed out that in the resultant Petri net shown in Fig. 4, a compact notation is used to show the activation sequence. Only the transitions are shown, the places between the transitions have been omitted. This is a repeatable pattern. The concept is shown in Fig. 1 and Fig. 5. This is used in the fundamental modeling concept Petri net diagram [9]. Concise modeling notations are becoming increasingly important for explaining modern systems.
Using Petri net analysis, it is possible to verify and validate Fig. 4.

If the sequence diagram is not well formed then this will be reflected in the Petri net that is constructed. For sequence diagrams that have many construction errors, it might be impossible to generate a Petri net. If return messages are omitted, it is not possible to achieve a certain amount of precision to determine when the reply would be returned. This would show a certain level of ambiguity in the sequence diagram. If the activations are completely omitted as is done in traditional MSC,
the diagram can be corrected to include activation on all the objects shown.

Some differences between the sequence diagram and the Petri net are listed in Table 1. Some of the major properties or results for the Petri net in Fig. 4 are shown in Table 2. These are important because they can be used for model checking. If a different sequence diagram is transformed then these properties might change. To obtain and verify these properties, the formal and mathematical aspect of the Petri net should be considered. These are not included in this paper.
9. Conclusions

This paper has presented some common aspects of sequence diagrams. A simple and effective method for transforming sequence diagrams into Petri nets has been considered.

It is evident that the conversion process is not always a simple one because, unfortunately, there are no proper rules for constructing sequence diagrams. Many variants of sequence diagrams exist making it difficult to apply a single solution. The transformation method presented has ignored loops, conditional statements, etc. The problem with these is that in sequence diagrams, these constructs are not properly represented at a graphical level. If simple mapping between the sequence diagram and the Petri net are to be maintained, these have to be properly represented using proper message paths. Conditions could be placed on the input arcs of the relevant transitions. Here the focus has been mainly on transforming the message paths to their equivalent notation. The operators and frames placed, just having a label describing a message path as part of a condition, do not provide sufficient detail. Some of these constructs are optional and their use is not always recommended.

The ambiguity and expressive weakness of sequence diagrams is confirmed in Refs. [3, 4] and other related works.

Transformation rules, lookup tables, triple graph grammar mapping, OMG QVT, semantic functions, etc. can be used for creating model to model mapping as required.

This work can be greatly developed by extending the transformation to other classes or notations of Petri nets such as TPN (time Petri nets), CPN (colored Petri nets) and proper fundamental modeling concept Petri net diagrams. Other developments to the transformation method and the Petri net could enhance the result. The Petri net can be transformed into a digraph for other forms of analysis.

References

Transforming UML Sequence Diagrams into Petri Nets


