A Triple Graph Grammar Mapping of UML 2 Activities into Petri Nets

A. Spiteri Staines

***Abstract*—** Model-to-Model mapping has several advantages over relational mapping. In model-to-model mapping an active correspondence is kept between two pairs of models. This is facilitated if graphical models are used. UML 2 activities are based on Petri net like semantics and substantial literature exists explaining their conversion into Petri nets. This paper explains how UML 2 activities can be formally mapped into Petri nets or Petri net semantics from a theoretical, practical and operational point of view adding on previous work of Triple Graph Grammars (TGGs). UML activity constructs are classified and identified. This is useful for creating a basic set of TGG rules. Generic TGG rules are identified and created. The rules are mainly intended for forward transformation. An example is given illustrating the conversion process. The concepts presented can be elaborated further and even extended to other visual models or notations.

***Keywords*—**Activity Diagrams, Petri Nets, Triple Graph Grammars, Unified Modeling Language

I. Introduction

UML 2 activity diagrams are important structured visual modeling notations useful for describing different types of behavior found in computer and information systems [2], [17]. UML 2 activities can be used formally or informally. Practical uses of activity models are for i) web processing, ii) web service composition, iii) business process modeling [21], iv) workflow modeling, v) systems integration, vi) task management and vii) low level tasks like software operations. UML 2 activities are suitable for modeling the diverse requirements of many traditional scenarios. Activities provide for visual modeling that can be easily understood. UML 2 activities evolved from UML 1.x diagrams based on state machines. UML 2 activities do not only specify system behavior but they can also be used for code generation and can be combined with high level languages like BPEL. Activities place particular emphasis on control flows, event sequencing, conditions and coordination. Activities have gained widespread acceptance for modeling different scenarios. Activities can be derived from use cases or constructed directly. The UML 2 superstructure specifies basic rules for node execution based on tokens. UML 2 activities introduce new concepts like collections, streams, exception handling, etc. Activities have gained widespread acceptance for modeling different scenarios. Activities can be derived from use cases or constructed directly. The UML 2 superstructure specifies basic rules for node execution based on tokens. UML 2 activities even introduce more new concepts like collections, streams, exception handling, etc.

Activity processes are suitable for abstraction into activity models. High level models are suitable for transformation into executable processes and different languages. Activity models are not a proper formalism and need proper verification and validation. Transforming activities into Petri nets or Petri net classes seems to be the best solution. This is evidenced from previous work [7]-[9],[18]. Petri nets seem to have a dual identity. They have a graphical representation and a textual or language description.

Until now, most methods of translation of UML activities into Petri nets are based on specific approaches or models. UML activities are translated into Petri nets, colored Petri nets and other formalisms. Translation into colored Petri nets and formalisms normally require more work. Several motivating factors exist for transforming UML 2 activities into Petri nets. Activities can be supported using formalisms like CCS, logics, formal specification languages, etc. However many of these are non visual. It has been explained in [18],[7]-[9] that higher order nets and colored Petri nets seem to be the best choice. However in this work the focus is on presenting a general solution and 'simplifying' the mapping process. For this reason more weight is given on using ordinary place transition Petri nets to explain the idea.

II. Related Works

Different research exists evidencing the need to support UML notations using Petri net models. Some examples are found in [1],[3]-[9],[13]-[16],[23]. One method of transforming UML use case constructs to colored Petri nets (CPN) is based on multi layers [1]. Use cases are the starting point for activity modelling, where no proper formalisms have been used. In [2] a UML 2 activity model for an online multi role playing game is transformed into a special type of Petri net (PEPA net) and analyzed. The transformation process is again informal.

A well structured, semi formal method is presented to translate activities into LGSPNs (labeled generalized stochastic Petri nets) in [3]. These are very useful for

Manuscript sent Nov 25, 2009: revised Feb 2009

A. Spiteri Staines is with the Department of Computer Information Systems, Faculty of ICT, University of Malta, Msida, MSD 2080, Malta, Europe. phone: 00356-21373402; fax: 21312110; (e-mail: toni_staines@yahoo.com, tony.spiteri-staines@um.edu.mt)

performance analysis. Unfortunately the approach seems to be a relational one. Case tools found in the LaQuSo project [5] are used to transform basic activity diagrams into simple Petri nets. HLTPNs (higher level timed Petri nets) have also been indicated for supporting and formalizing the UML.

In [7]-[9] it is explained how activity semantics and constructs are classified and translated into Petri net semantics. The preferred Petri nets class indicated are colored Petri nets or higher order nets.

The UML can be formalized using Petri net like semantics. A CPN based formalization of the UML is presented in [6]. Transforming UML 2 activities into a Petri net semantics has been formalized in [5] and [8]. In [8] a semantic function is described and given. This converts an activity diagram <activity node, activity edge> into a CPN. Practically all these approaches are more about relational issues rather than operational. Petri nets are also found in Fundamental modeling concepts and apply to activity modeling in this context. According to [11] TGGs are suitable for expressing UML activity workflow patterns because of graph-to-graph mapping. TGG can maintain a transformational correspondence between two different models [12]. This correspondence is operational. In the Fundamental modeling concepts method or approach presented in [10], Petri nets are used for modeling behavior in a context similar to activity modeling. In [20] the strengths of UML 2 activities, in conjunction with Petri nets, are used for structural and performance evaluation. This confirms the importance of using Petri nets and UML activities for workflow analysis. A mapping scheme for transforming activities into Petri nets is recommended. In [19] incremental model synchronization is explained. Incremental model synchronization makes use of TGGs. Model synchronization refers to incremental changes that are carried out. Using the TGG concept, incremental changes in a model should correspond to changes in a mapped model.

According to the UML 2 superstructure specification activity diagrams are structured into different classes that have different levels of behavior ranging from simple to more detailed.

III. Motivation

The motivation for transforming activities into Petri nets is that UML 2 activities are based on Petri Net like semantics according to the UML 2 superstructure specification. Activities have a higher level of abstraction. Activities share common properties with Petri nets. This is not properly explained in the UML specification. Both Petri net and activity diagrams are can be classified as types of directed graphs Different approaches to transforming activities into Petri net classes have been suggested. Previous work evidently shows the importance of this. Some are informal, others are semi formal or completely formal. Some of these approaches are quite complex. E.g. a transformation function can be used. These approaches still do not explain the actual transformation process and normally the transformation is too cumbersome to use. Most transformational approaches explain or formalize the actual correspondence. They do not actually explain how to carry out this transformation from a practical perspective. A solution is therefore necessary.

Possible solutions are using QVT, ATL or TGGs. These solutions suggest Model to model mapping. This would definitely be an operational solution.

Model transformation is important in the field of automated software engineering [22]. For this work, model to model mapping using a Triple Graph Grammar approach is being proposed. Model transformation and visual model mapping have become increasingly popular over the years. This is evident from the OMG approach where QVT (query, view, transform) is used to support UML and also with work related to ATL (Atlas transformation language).

Model-to-model mapping offers several advantages over other approaches. Transformation with TGGs is not only relational but also operational as indicated in [12]. The proposed rules best explain one way of transformation which is forward transform. The reverse transform requires the creation of new rules and more information in the Petri net. The Petri net does not capture all the diverse detail in the UML activity unless other information is added.

IV. A Triple Graph Grammar Solution

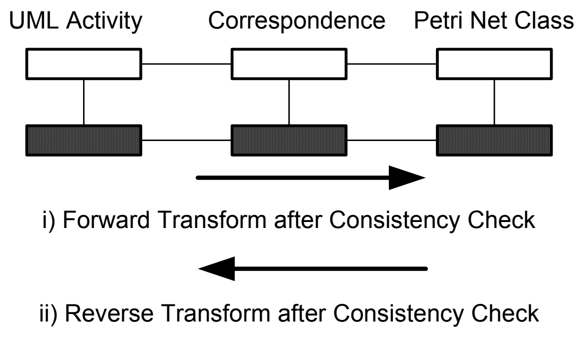
UML Activity
Correspondence
Petri Net Class
i) Forward Transform after Consistency Check
ii) Reverse Transform after Consistency Check

Fig. 1 UML Activity to Petri net TGG Mapping

A. Triple Graph Grammars

Triple graph grammars (TGGs) have been around for several years. They are useful techniques for mapping two different types of graphical models sharing some similar properties. With TGGs it is possible to i) define and ii) declare bi-directional transformations. Relationships between the different models need to be established. A model can be transformed into another model and correspondence is computed incrementally. All changes are recorded and changes can be synchronized. These approaches use similar concepts to those found in TGGs. TGGs have been well researched and documented. There are many different examples of TGGs uses in literature. TGGs describe the dynamic evolution of different models.

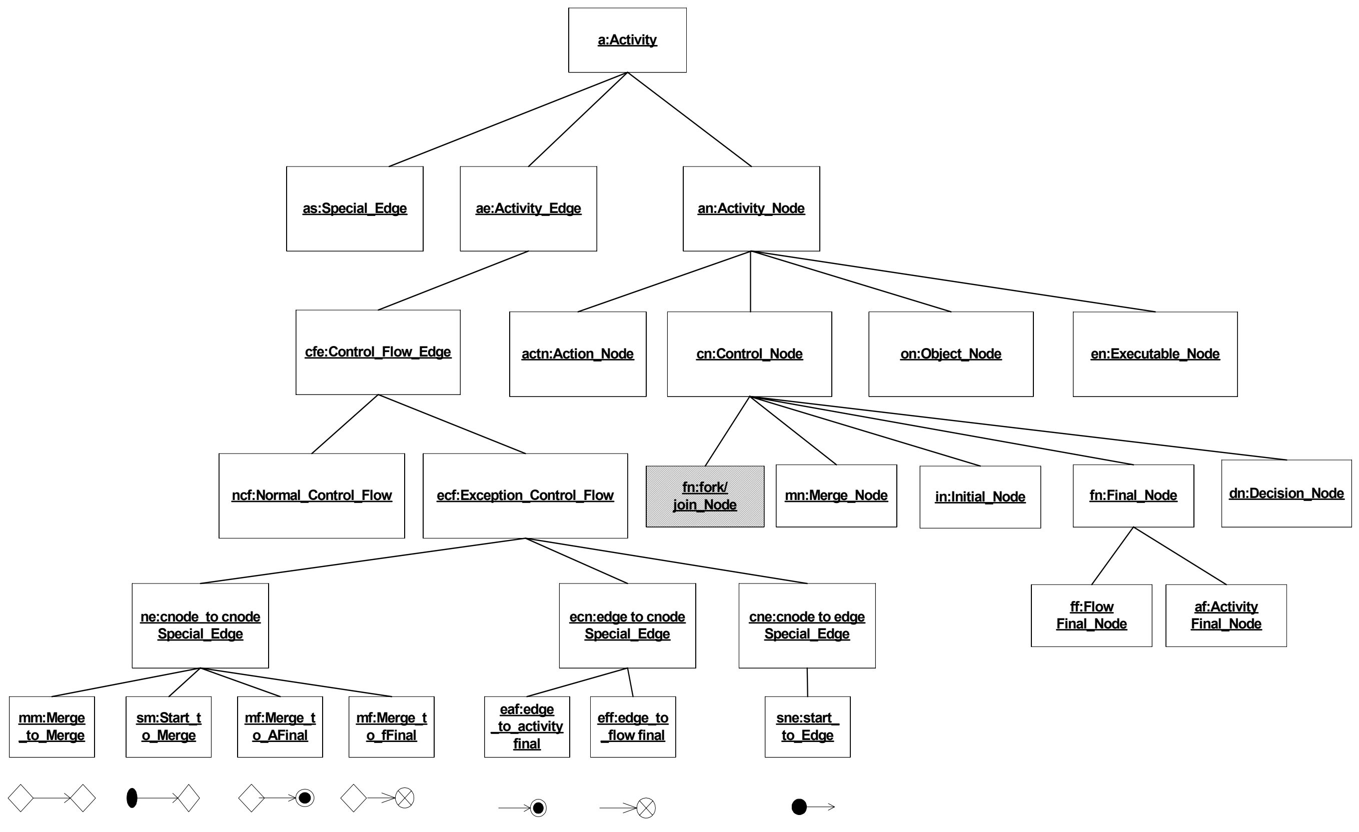

Fig. 2 UML 2 Activity node and edge classification for translation into a Petri net

TGGs are based on graph grammars. In simple terms, graph grammars are visual rules explaining how a graph or part of it, is to be modified according to certain conditions. E.g. nodes and edges are added or removed accordingly. Normally a graph grammar rule has a right hand side and a left hand side that describes the rule.

TGG rules are similar to graph grammar rules with the exception that in TGGs there are three lanes or domains represented [11],[12]. The left side represents the model, the middle represents the correspondence and the right lane represents the transformed model or result. Ideally Correspondency mapping is should link both models from different domains. This is shown in fig. 1. If one model changes, the corresponding model is updated via the application of rules. Rules can be applied: i) never, ii) once or iiii) several times.

B. Understanding UML 2 Activities

Comprehending UML 2 activities is the starting point for this work. UML 2 activities are specified in the OMG UML super structure specification [17]. To create TGG rules the understanding and comprehension of the underlying model relationships is important. The notations can be abstracted into TGG rules for particular relationships. The conversion rules are defined in terms of TGGs. The rules cover all the basic constructs of UML 2 activities. UML 2 activities are classified into seven main types: i) fundamental, ii) basic, iii) intermediate, iv) complete, v) structured, vi) complete structured and vii) extra structured. UML 2 activity nodes have flow of control constructs that can be used for i) synchronization, ii) decision, iii) concurrency, iv) sequence and v) iteration. Each type of activity sub-class addresses the problems and issues within a particular area. E.g. structured activities are suited for traditional programming problems. Structured activities focus on traditional programming, whilst fundamental and basic activities have a level of abstraction making them ideal for high level business process modeling. Fundamental and basic activities are ideal for high level modeling as is the case when describing business processes or workflow. Classes suited to Petri net conversion are i) fundamental, ii) basic and iii) intermediate activities. But this does not preclude other classes. The most important constructs belonging to all the activity sub classes are considered here. It is not specified to which class they belong. From experience and real scenarios these constructs are important regardless from where they are taken. A practical solution is being explained. All subclasses need to be considered in a general manner.

Given several classes of activities according to the UML superstructure specification it is possible to identify different constructs being used, but the most important constructs are used repeatedly in the different classes. The constructs considered are mainly derived from intermediate activities. Intermediate activities inherit from basic activities. Normally intermediate activities can be used for describing different information system scenarios.

C. Classifying UML 2 Activity Notations

Activity nodes normally classify into control node and executable nodes, however action nodes and object nodes are considered. Normally control nodes, object nodes and executable nodes have several subclasses. The subclasses help comprehend the standard behavior specified in the OMG UML superstructure specification.

Technically speaking activities are composed of i) nodes

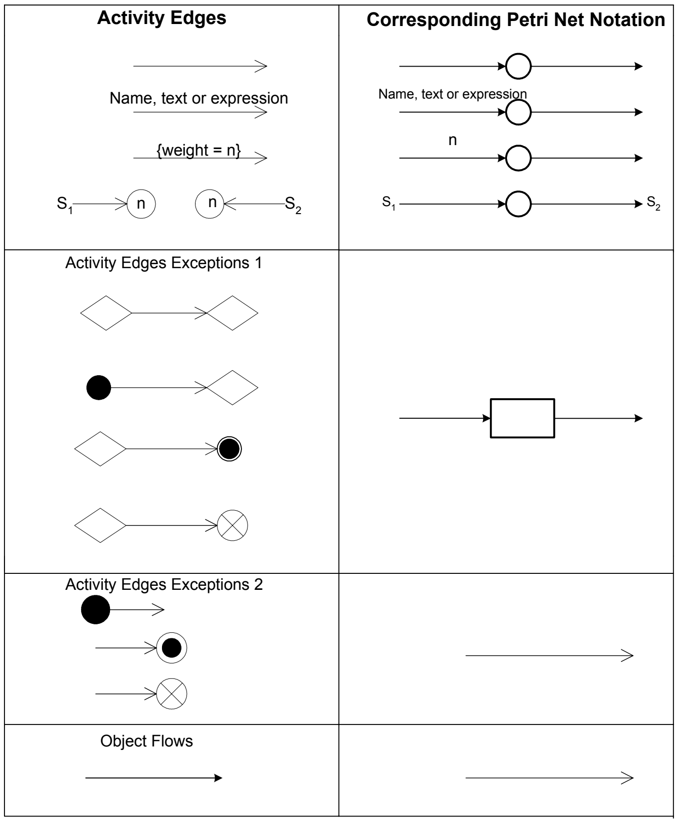

Fig. 3 Activity Edges and corresponding Petri net constructs

and ii) edges. A special edge is used for connecting parts of object nodes. These are ignored here. There are other constructs like signal nodes and edges which can be similarly treated to object nodes.

Fig. 2 explains a generalized the classification of UML 2 activities into nodes and edges as required for transforming them into Petri nets. This is based on previous work presented in [7]-[9], [18]. The actual corresponding Petri net constructs are shown in fig. 3 and 4.

Activity edges can be decomposed into control flow edges. Control flow edges are composed of i) normal control flows and ii) exception control flows. For exception control flows or edges there are i) control node to control node special edges, ii) edge to control node and iii) control node to edge special edges. Control node-to-control node special edges are composed of: i) merge-to-merge nodes, ii) start-to-merge nodes, iii) merge-to-activity final nodes and iv) merge-to-flow final nodes. Edge to control node special edges are composed of: i) edge-to-activity final, ii) edge-to-flow final and control node to edge special edges have start-to-edge. These exceptions require special treatment unlike normal edges. A normal activity edge or normal control flow maps into a Petri

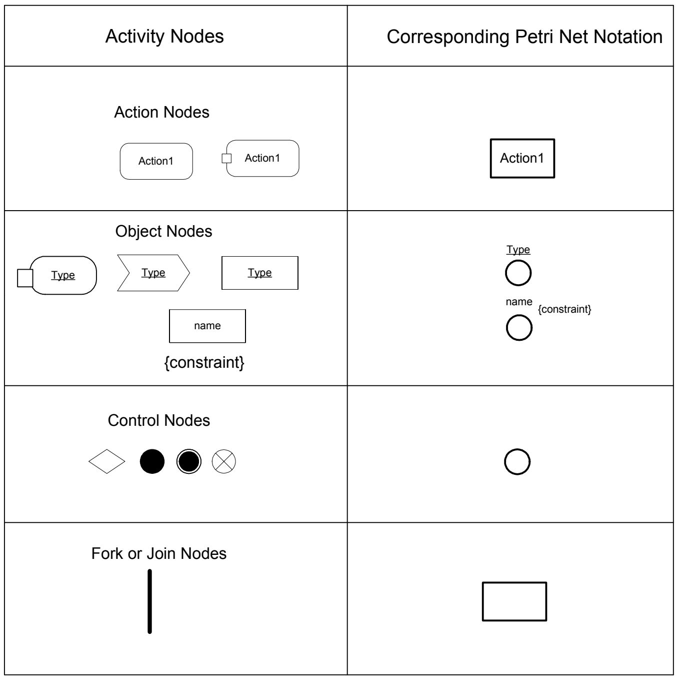

Fig. 4 Activity nodes and corresponding Petri net constructs

net i) input arc, ii) connected to a place and a iii) output arc from the place. A normal action node or executable node translates or maps into a transition. Exception activity edges are explained in fig. 3.

Activity nodes are classified into i) action nodes, ii) control nodes, iii) object nodes and iv) executable nodes. Action nodes and executable nodes convert to Petri net transition types. Control nodes and object nodes convert to Petri net place types. Control nodes can be sub classified into i) fork nodes, ii) merge node, iii) initial node, iv) final node and v) decision node. Final node can be of two types i) activity final or ii) flow final.

Control nodes can be treated similarly and do not constitute an exception. Fork and join nodes are an exception to this. A fork or join node is treated as an executable node and converts into a Petri net transition. Activity nodes are explained in fig. 4.

For conversion the adopted procedure is to start translating the activity model from the starting node visiting every edge and node in the activity diagram applying each TGG rule in sequence. All nodes and edges have to be covered. TGG rules need to be constructed for normal and exception behavior that has been defined above.

D. Triple Graph Grammar Mapping Rules

A basic set of TGG rules is proposed for the forward transformation mapping process.

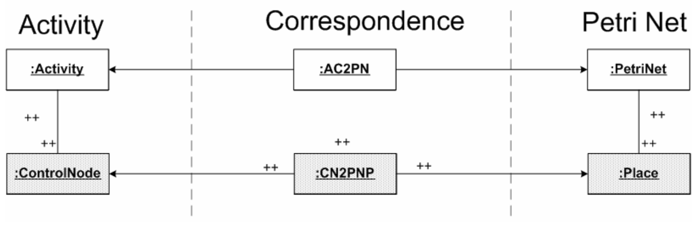

Fig. 5 TGG Rule 1: Add a New Control Node excluding fork or join nodes

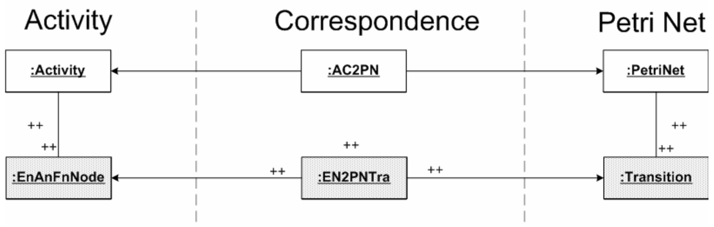

Fig. 6 TGG Rule 2: Add a New Executable Node Action Node or Fork/Join Nodes

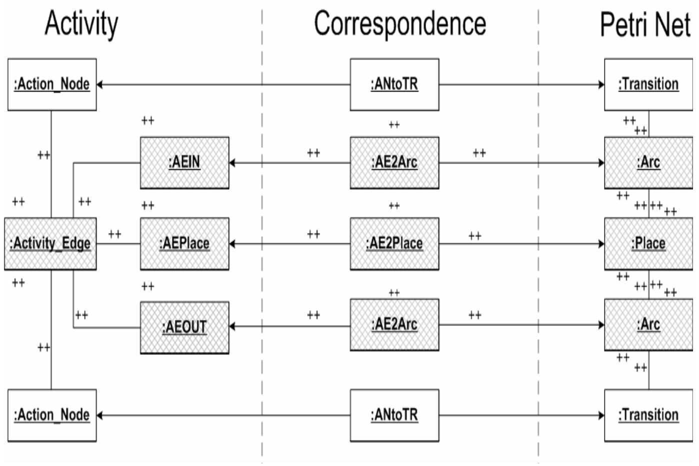

Fig. 7 TGG Rule 3: Insert a Normal Activity Edge

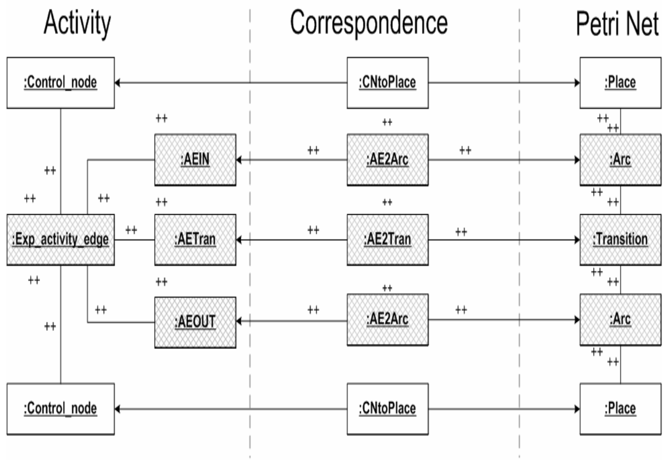

Fig. 8 TGG Rule 4: Insert an Exception Activity Edge between Two Control Nodes

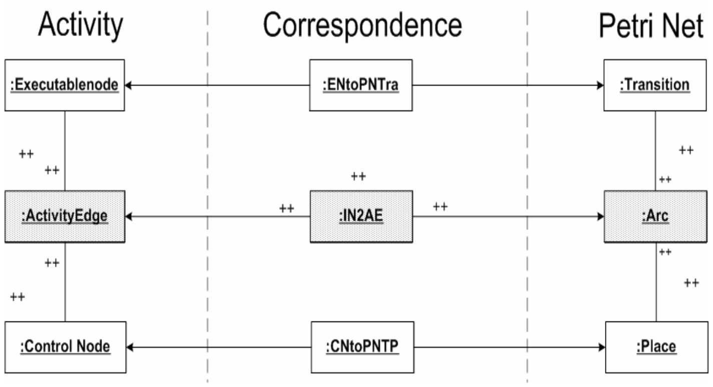

Fig. 9 TGG Rule 5: Insert an Exception Activity Edge for Executable to Control Node

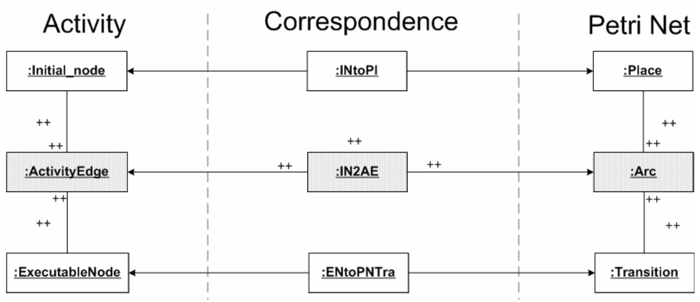

Fig. 10 TGG Rule 6: Insert an Exception Activity Edge for Control Node to Executable Node

shown by ++ are used to illustrate the mapping concept [12]. There are three domains shown in each rule.

This is the i) activity model, ii) correspondence and iii) Petri net. To simplify the diagram rules, the Petri net domain node is not always shown. A total of six TGG rules are created from the classification of the UML activity notations.

The rules created are: Rule 1 Add a New Control Node. This rule excludes fork or join nodes. Rule 2 Add a New Executable Node, Action Node or Fork/Join Node. Rule 3 Insert a Normal Activity Edge between existing Action Nodes. Rule 4 Insert an Exception Activity Edge between Two Control Nodes. Rule 5 Insert an Exception Activity Edge for Executable to Control Node. Rule 6 Insert an Exception Activity Edge for Control Node to Executable Node. These are shown in fig. 5-10.

These rules capture all the main types of activity behavior and exceptions. The rules can be used as the foundation for mapping activities into Petri nets and creating additional rules for the reverse mapping. The rules are explained in detail below.

Rule 1 and Rule 2 are generic rules for i) control node insertion and ii) executable node insertion. At this point, the implication is that these rules just add the counterpart of a control or executable node to the Petri net. Control nodes map into a Petri net place and executable nodes map into a transition. Action, fork and join nodes which are special cases

TGG RULE	ACTION
RULE 1	**ADD A NEW CONTROL NODE** (EXCLUDES FORK/JOIN)
RULE 2	**ADD NEW EXECUTABLE ACTION or FORK/JOIN NODES**
RULE 3	**INSERT NORMAL ACTIVITY EDGE** (between EXECUTABLE,ACTION, FORK/JOIN NODES)
RULE 4	**INSERT EXCEPTION ACTIVITY EDGE** (between TWO CONTROL NODES)
RULE 5	**INSERT EXCEPTION ACTIVITY EDGE** (EXECUTALBE TO CONTROL NODE)
RULE 6	**INSERT EXCEPTION ACTIVITY EDGE** (CONTROL TO EXECUTABLE NODE)

Table 1 Main TGG Rules for Activity to Petri net Forward Transformation

are treated as executable nodes and map into Petri net transitions.

Rule 3: Insert a normal activity edge. Fig. 7 describes the generalized process of inserting a normal activity edge between action nodes and executable nodes. A fork or join construct in the activity model is treated as one of these. This rule has to be applied in the case of fork and join nodes. These correspond to Petri net transitions. Rule 3 is more complex than the other rules. It involves more insertions on the Petri net side. A normal activity edge is inserted between two action nodes or activity nodes. It means that on the Petri net side we have two transitions obtained previously from rule 2. The two transitions are connected by placing an output arc from the first transition, the output arc connects to a place which is connected using an arc to the last transition. This is shown in the activity edge and corresponding Petri net notation in fig. 3.

Rule 4 explains how to insert an exception activity edge between two control nodes. The two control nodes could be any type e.g. merge, start, end, flow final, activity final. Flow final and activity final are treated similarly. These are shown in fig. 2 at the bottom left hand side. Inserting the edge between these nodes corresponds to an output Petri net arc from the first control place. This connects to a transition which has an outgoing arc which inputs to last control place. The existing places are the result of rule1 initially.

Rule 5: Inserts an exception activity edge from an executable node to a control node. This is relatively simple and an arc is inserted between an existing transition and a place.

Rule 6: inserts an exception activity edge between a control node and an executable node. This is similar to Rule 5. Normally the initial node is a control node. But there can be other control nodes like merge, initial etc. Fork and join are excluded because they are treated like executable nodes. This has already been explained above. On the Petri net side an arc is used to connect a place to a transition. The place and transition would have been created using previous rules.

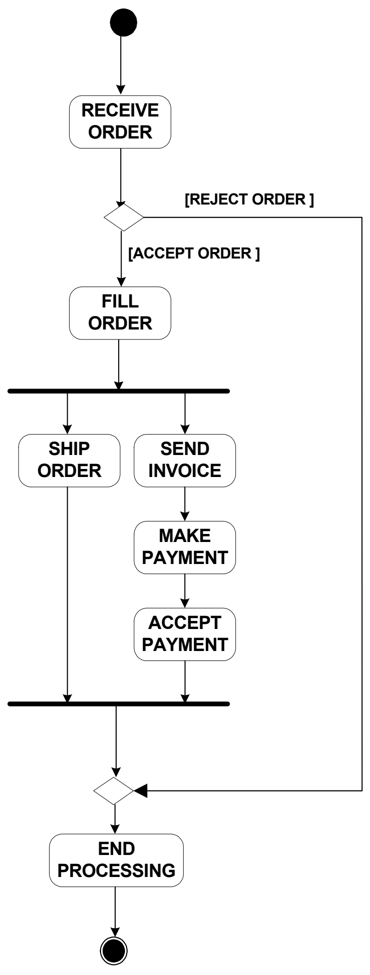

Fig. 11 UML activity for order processing

Table 1 summarizes the six rules for the forward transformation process that have been described.

V. Case Study

An example similar to that found in the UML 2 superstructure specification [17] is presented to illustrate the mapping process. Fig. 11 shows an activity diagram for processing a customer order. Basically a customer order is received. The order is accepted or rejected. If it is accepted it is filled out and then there are some parallel activities like ship order, send invoice, make payment and accept payment. If the order is rejected these steps are skipped and processing goes directly to terminate the order. If the order is accepted then all the steps need to be concluded before termination.

The activity diagram requires the use of all the six TGG rules presented for conversion into a Petri net model. The resultant Petri net shown in fig. 12 depicts the rules that have been applied. They are shown using the value e.g. R1 refers to the application of rule 1. Some rules are applied a number of times whilst other rules like rule 4 is applied only once. There is only one merge-to-merge activity edge. The application if

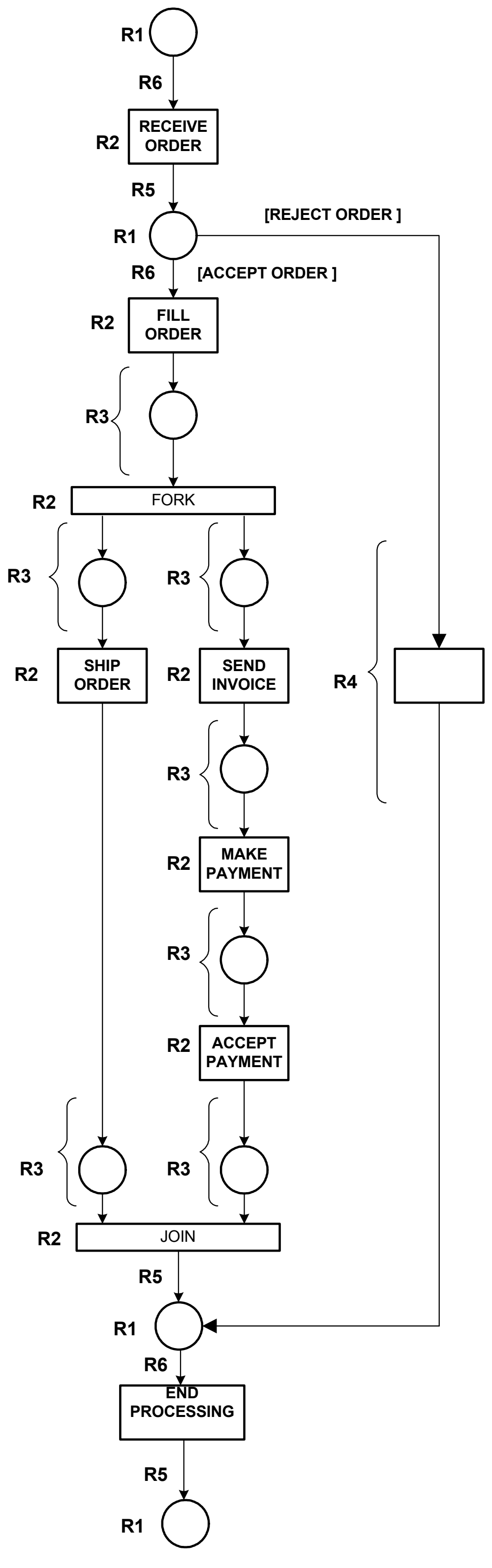

Fig. 12 Corresponding Petri net with rules

Rule 4 is shown in the resultant Petri net as R4.

The TGG rules can be applied to obtain the Petri net shown in fig. 12.

VI. Analysis

The approach used is this work is forward transformation only. The sequence of applying each rule is not shown here. Model correspondence uses graphical syntax.

The left hand side model is called the source and the right hand side model is called the target. The transformation rules presented are mainly intended for forward transform. The basic algorithm to apply these rules is to visit all the activity diagram (graph) nodes and edges from start to finish and sequentially apply each rule. Once all the activity nodes and edges have been matched the Petri net can be generated. Ideally the transformation rules should work the other way round. E.g. from the Petri net we should generate the activity model. There are problems with the reverse transformation. The Petri net needs to match the proper activity notations. Result could be a simplified activity diagram.

There are various possible solutions. E.g. the creation of two matched models initially and adding or deleting nodes on particular sides as needed. This is explained as model integration in [12].

E.g. initial, activity final, flow final, merge, join, fork, decision etc. all require their own rules. The same goes for exception control flows e.g. merge to merge, start to merge, merge to activity final, merge to flow final, edge to activity final.

Rules 1-3 are the most applied rules. Obviously their application frequency depends on the activity diagram.

VII. Discussion

The example presented is quite simple. The formal transformation explained here should work practically for most activity diagrams. This means that this approach is valid when common notations are used. However these rules are manually designed rules. TGG Deletion rules have not been considered. The transformation approach explained is forward transformation. The six rules given cover properly forward transformation. I.e. this implies transforming the activity which is the source into the Petri net which is the result. To get full benefit from TGGs there should also be the possibility of reverse transform. This means that we would have bi-directional transformation.

Reverse transformation implies that from a Petri net model it should be possible to derive the corresponding activity diagram. Although this can be done using the rules given, this is not sufficient. To convert a Petri net backwards into an activity model implies that the Petri net has sufficient detail. E.g. places and transitions in the Petri net must be given specific identities. E.g. a transition has to be labeled as a fork or join transition. Places and arcs require special labeling. This means that from the TGG rules explained more specific or explicit rules have to be created for each different type of activity notation complicating things. However the six TGG rules already defined will be the main starting point for creating the new rules.

Another issue is that an ordinary place transition net does not capture enough detail to support all the different activity notations. A simple solution to this is the labeling of Petri net places, transitions and arcs. A more extensive solution is to use higher order nets and colored Petri nets as described in [18]. However in the latter case the transformation mapping might be more complex.

Simple Petri nets are used because the main idea of this work serves to introduce the TGG approach without complicating things. More work is required to create something fully functional and useable.

VIII. Conclusion and Future Work

The work presented here has dealt with the formal mapping of basic UML 2 activity diagrams into Petri nets. It confirms the usefulness and importance of TGGs. Activity diagrams were selected because they are important visual notations and they are based on Petri net semantics. It has explained how the mapping can be successfully achieved using TGGs. The TGG rules just specify the correspondence mapping of activities and Petri nets graphically using an abstract notation. These rules can be made operational in different ways and applications.

TGGs are useful for formal mapping of two similar visual notations. Many steps and rules are required and have to be repeated several times for transformation. Complex models require the application of several rules a greater number of times. I.e. just adding a start node, requires two rules. Buffer nodes, queues and other complex constructs etc. require special attention that has not been considered as part of this work.

The TGG rules presented are a simplification and generalization of what needs to be done. Only addition rules denoted by ++ have been used. These rules are useful for intuitional transformation. They could be used in other formal transformational approaches.

Actually deletion rules that have not been presented might need consideration. For mapping the control nodes edges and edge insertion see fig. 5-7 these are generalized or generic rules for different activity constructs. From these rules it is possible to create specific rules for every different control node type. E.g. for just adding merge node, final node, initial node, three separate rules need to be created. This means that more than ten rules will be required. The complete conversion will become complex to manage. The use of a CASE tool is recommended.

The TGG rule drawings presented in this paper just give a brief outline of what needs to be done. They definitely can be improved to introduce more detail. They have been constructed indicating the most salient points. The rules can be appropriately colored or shaded to better indicate what is being inserted.

Other transformational concepts like model transformation model integration, model synchronization, node reusability, constraints need to be considered. These are discussed in [12]. Attributes and other advanced concepts require more work.

The work presented identifies the possibility for more research. Other diagrams like state machines and state transition diagrams can be mapped into Petri nets using a similar approach which might be simpler. This necessitates further investigation.

The approach given can be used to understand the complexity of activity models. It is possible to use other classes of Petri nets like higher order nets or colored Petri nets for the transformation process. This would obviously introduce much more complexity in the mapping process.

References

[1] M.E. Shin, A.H. Levis, L.W. Wangenhals, “Transformation of UML-Based System Model to Design/CPN model for Validating System Behavior” , *Proc. of the 6th Int. Conf. on the UML/Workshop on Compositional Verification of the UML Models*, San Francisco CA., Oct 2003.

[2] C. Canevet, S. Gilmore, J. Hilliston, L. Kloul, P. Stevens, “Analysing UML 2.0 Activity Diagrams in the Software Engineering Performance Process” , *WOSP’04*, ACM, Redwood CA., pp. 77-78, Jan 2004.

[3] J.P. Lopez-Grao, J. Campos, “From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering”, *WOSP’04*, Redwood CA., pp. 25-26, Jan 2004.

[4] J. Merseguer, J. Camposm E. Mena, “On the Integration of UML and Petri Nets in Software Development”, *ICATPN’06*, Turku Finland, LNCS 4024, pp.19-36, Jun 2006.

[5] LaQuSo (2007). LaQuSo Work Group / Project, LaQuSo Repository, Eindhoven, www.Laquso.com

[6] J. L. Garrido, M. Gea, “A Colored Petri Net Formalization for a UML-based Notation Applied to Cooperative System Modeling, Interactive Systems: Design, Specification and Verification”, *LNCS 2545*, Springer, pp.16-28, 2002.

[7] H. Störrle, “Structured Nodes in UML 2.0 Activities”, *Nordic Journal of Computing*, Vol. 11, No. 3, pp. 279-302, Sep 2004.

[8] H. Störrle, “Semantics of Control Flow in UML 2.0 Activities*”, Proc. of 2004 IEEE Symposium on Visual Languages and Human Centric Computing*, USA, pp. 235-242, 2004.

[9] H. Störrle, J.H. Hausmann, “Reasoning about UML Activity Diagrams”, *Publ. Assoc. Nordic Journal of Computing*, Vol. 14 No. 1, pp.43-64, 2005.

[10] A. Knöpfel, B. Gröne, P. Tabeling, *Fundamental Modeling Concepts*, Wiley, West Sussex UK, 2005.

[11] C. Lohmann, J. Greenyer, J. Jiang and T. Systä, “Applying Triple Graph Grammars For Pattern-Based Workflow Model Transformations”, *Journal of Object Technology*, Special Issue: Tools Europe 2007, pp. 253-273, Oct 2007. http://www.jot.fm/issues/issue_2007_10/paper13/

[12] E. Kindler, R. Wagner, “Triple Graph Grammers: Concepts, Extensions, Implementations and Application Scenarios”, *Technical Report Tr-ri-284*, University of Paderborn, Paderborn, Germany, 2007.

[13] L. Baresi, M. Pezze, “Improving UML with Petri Nets”, *Electronic notes in Theoretical Computer Science*, Elsevier, Vol 44., No. 2, pp. 107-119,Jul 2007.

[14] E. Borger, A. Cavara, E. Riccobene, “An ASM Semantics for UML Activity Diagrams”, *Proc. of 8th International Conference on Algebraic Methodology and Software Technology*, Iowa City, pp. 293 – 308, May 2000.

[15] Z. Hu, S.M. Shatz, “Mapping UML Diagrams into a Petri Net Notation for System Simulation”, *Proceedings of the 16th International Conference on Software Engineering & Knowledge Engineering (SEKE'2004)*, pp. 213-219, Jun 2004.

[16] P. King, R. Pooley, “Derivation of Petri Net Models from UML Specifications of Communication Software*”, Proc. of 11th Int. Conf. On Tools and Techniques for Computer Performance Eval.*, pp. 262-276, Mar 2002.

[17] OMG UML 2 Superstructure Specification. V2.2, OMG,http://www.omg.org/technology/documents/formal/uml.htm

[18] T. Spiteri Staines, "Intuitive Mapping of UML 2 Activity Diagrams into Fundamental Modeling Concept Petri Net Diagrams and Colored Petri Nets", *Proc. of the 15th ECBS conference*, pp. 191-200, Apr 2008.

[19] I. Madari, L. Angyal, L. Lengyel, "Incremental model synchronization based on a trace model", *Proceedings of the 9th WSEAS international conference on Simulation, modelling and optimization*, Budapest, Hungary, pp. 470-475, 2009.

[20] K. Hee Han, S. Kyu Yoo, B. Kim , "Qualitative and quantitative analysis of workflows based on the UML activity diagram and Petri net" , *WSEAS Transactions on Information Science and Applications*, vol. 6 , no. 7, pp. 1249-1258, Jul 2009.

[21] W. Rungworawut, T. Senivongse, "A Guideline to Mapping Business Processes to UML Class Diagrams" , *WSEAS Trans. on Computers*, vol. 4, no. 11,pp. 1526–1533, 2005.

[22] T. Levendovszky, L. Lengyel, H. Charaf, "Extending the DPO approach for topological validation of metamodel-level graph rewriting rules", *WSEAS Transactions on Information Science and Applications*, Issue 2, Vol. 2, pp. 226- 231, Feb 2005.

[23] A. Spiteri Staines, "Modeling UML Software Design Patterns Using Fundamental Modeling Concepts (FMC)", *Proceedings of the 2nd WSEAS European Computing Conference*, Malta, pp. 192-197, Sep 2008.