Rule Based Bi-Directional Transformation of UML2 Activities into Petri Nets

A. Spiteri Staines

***Abstract*—** Many modern software models and notations are graph based. UML 2 activities are important notations for modeling different types of behavior and system properties. In the UML 2 specification it is suggested that some forms of activity types are based on Petri net formalisms. Ideally the mapping of UML activities into Petri nets should be bi-directional. The bi-directional mapping needs to be simplified and operational. Model-to-Model mapping in theory offers the advantage of fully operational bi-directional mapping between different models or formalisms that share some common properties. However in reality this is not easily achievable because not all the transformations are similar. Previous work was presented where it was shown how Triple Graph Grammars are useful to achieve this mapping. UML 2 activities have some common properties with Petri nets. There are exceptions which require some special attention. In this paper a simple condensed rule based solution for complete bi-directional mapping or transforming UML 2 activities into Petri nets is presented. The solution should be operational, and can be represented using different notations. A practical example is used to illustrate the bi-directional transformation possibility and conclusions are explained.

***Keywords*—** UML 2, Activity diagrams, Petri nets, Bi-directional transformation, Triple graph grammars

I. Introduction

UML 2 activities are fundamentally important visual notations that express the diverse behavior of computer systems and information systems [6]. Their use has been extended to different types of systems and areas not necessarily related directly to computing. Some uses of activities vary from web composition to business workflows [16]-[17] and at lower levels there is modeling of software artifacts like operating systems, file handling, programming etc. UML 2 activities appear to be similar to flowcharts; but the semantics behind them are entirely different. Again UML 2 activities have some similarities to UML state machines or state machine diagrams (SMDs) but again they are not similar. UML state machines are related to modeling system states and have been derived from state transition diagrams (STDs). On the other hand, these activities closely represent a wide spectrum of properties and are targeted towards different levels of stakeholders needs. Activities are useful even on their own, without other UML diagrams to express explicit system behavior for different purposes, and at all levels, hence they need to be expressive and detailed. UML activity diagrams have gained widespread use for different scenarios. They can be used as the initial diagram to study new or existing systems, the latter using reverse engineering concepts. Activities can be modeled directly off use cases.

The UML superstructure specification [6] is the document that explains in detail the different categories and classification of activity types which are i) basic, ii) structured, iii) intermediate, iv) extra structured, v) complete and vi) complete structured. Activity diagrams can be formally verified and supported. The graphical representation of Petri nets can be supported using formal languages, grammars or even textual notations.

UML 2 activities introduce many advanced constructs for error handling, streams, collections, etc. The superstructure document explains the different types of activities introducing rules for node execution based on something similar to token flows. The specification also presents us with activities having a Petri net like semantics.

There are numerous advantages of representing activities as Petri nets. It is possible to formally check the main properties of the Petri net models in detail. This is not always possible with activities. This work focuses on achieving a fully functional bi-directional solution.

II. Related Works

Supporting evidence exists presenting the advantages of mapping activities into Petri nets. From a certain point of view this seems to be the best choice [1]-[3], [7]-[14], [19]. As explained elsewhere, activities are non formal notations or models that require verification and testing. Transforming activities into Petri nets seems a natural choice and higher order nets are more expressible than activities. The possible mapping process has been explained and different approaches have been tried and used. In this work we try to find an optimal simplified solution for bi-directional transformation. This should work from either side. A simplified rule set is given. This should work under any condition. The rules presented are explained and defined in terms of Triple Graph Grammars (TGGs) [4],[5]. However their implementation does not necessarily have to be restricted to TGGs. They could be implemented in a data repository. It has already been examined how UML activities and Petri nets can be combined using TGGs [8]. TGGs are suitable for mapping two graphical models with the main properties being common to both. TGGs allow for the declaration of bi directional transformation relationships. Activities share some common properties with Petri nets. The latter have over three decades of coverage.

Manuscript received Feb 23, 2011. Anthony (Tony) Spiteri Staines, is with the Department of Information Systems, Faculty of ICT, University of Malta, (corresponding phone: 00356-21373402,e-mail: toni_staines@yahoo.com)

Both activity diagrams and Petri nets are based on directed graphs. Different solutions ranging from structured to semi-structured have been previously presented, explaining how activities can be transformed into Petri nets [1]-[3]. However a practical solution has not been outlined. Some of the given solutions present transformation in one direction mainly from the activity diagram to the Petri net and not bi-directionally. Another issue is that some solutions can become quite complex and are useful for very specific one off problems. For many given solutions the transformation process is abstracted. In this work the focus is on a generic solution that should work for most cases. It is explained how the bi-directional transformation can be realized and implemented as required.

There are already possible solutions using (query, view, transform) QVT, (atlas transformation language) ATL or TGGs. These are model to model mapping or transformation based approaches.

In previous work [7],[8] the forward transformation rules were given, explaining how this would work. It was also stated that many new rules might be needed for reverse transformation. There were problems with reverse transform which could require many new rules or the use of higher order nets and colored Petri nets (CPNs) [7],[8]. We shall attempt to present a solution to this, whilst retaining the use of normal place transition nets. Petri nets are well suited for reduction and simplification. Both Petri nets and activity diagrams can be treated as di-graphs. They are well formed and express some strikingly similar properties.

III. Problem Formulation

Model transformation [15], [17],[18] is a key area for MDE (Model Driven Engineering) and MDA addressed by the OMG and UML. The concepts behind QVT, QVTrelations and QVTcore are based on relational mapping between models or notations. The ideas behind QVT show the importance of models and their transformations.

Consider the simple case of transforming an activity model into a Petri net and vice-versa. There are different ways how the rules can be described.

To get full benefit of mapping, reverse transformation needs to be achieved. This means that true bi-directionality between two different models is kept continuously. We shall attempt to give a solution to this, whilst retaining the use of normal place transition Petri nets. For forward transformation six rules were sufficient, however for reverse transformation more new rules must be created. A simplified solution is to include information labeling on both models. This information would serve to identify the appropriate rule both for forward and reverse transform.

E.g. In the activity diagram there are several types of control nodes. E.g. initial node, final node, decision node, etc. A separate rule is required for each of these types. The representation could definitely be included in some grammar or language or even text format.

Here we explain the details how such a solution can possibly work. A good solution needs to be something that is simple, operational and can be applied repetitively to obtain result, i.e. they solution has to be operational and fully functional.

IV. Problem Solution

The proposed solution is to create transformation or mapping rules, and later decompose them or fine tune them to the required level. This will have to include specific information labeling on the elements of both models. This would indicate which rule is needed. Alternatively each model could have its elements labeled as to which rule to use. The information will serve to identify the appropriate rule for the reverse or forward transform. In this manner the number of rules required for the transformation is kept concise!

The sub rules will replace the generic rules if required. The idea can be formulated into a given algorithm.

The rules created for this work can be represented using Triple graph grammars (TGGs). TGG rules can check the activity diagram to the Petri net for a valid correspondence. Normally the source model for starting the transformation would be the activity diagram. The rules are used where the context nodes of the rule can be matched to the existing context source domain side elements which are used to create elements on the target side via a correspondence check. Bi-directional transformation is achievable using only insertion

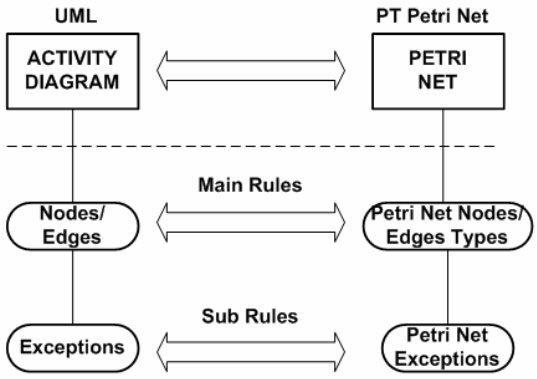

Fig. 1 high level mapping concept of UML activities into Petri nets

or creation rules specified in TGGs using "++" sign. If the activity diagram or the Petri net is generated from scratch, whenever there are changes in the corresponding model TGG creation or insertion rules should suffice. On the other hand if the models are not generated from scratch then other rules might be required.

Using TGGs gives different possibilities: i) two models can be given. I.e. the activity diagram and a Petri net. The correspondence of both models can be checked for validity. ii) a single model. This can be used to generate a new model from scratch. If two graph grammars are mapped structurally, the mapping of the two distinct graph types can be explicitly defined. If TGGs work for a solution, then even multi graph grammars and other forms of grammars can be considered.

A. A Generalized Solution

The actual solution is described below. In essence the activity diagram is composed of nodes and edges. There are several exception types for both nodes and edges. E.g. if control nodes are taken an activity diagram can have several types of control nodes like i) initial node, ii) fork/join node, iii) final node, iv) merge and v) decision. Two solutions are possible either i) create a separate rule for each type of node or ii) use a generic rule. For the former solution information

RULE	GENERIC ACTION	DETAILED ACTION	
RULE 1	Add/Insert a new control node (exclude fork/join)	1.1 Add initial node	
		1.2 Add final node	1.2.1 Add flow final node 1.2.2 Add Activity final node
		1.3 Add merge node	
		1.4 Add decision node	
RULE 2	Add executable action or fork/join nodes	2.1 Add executable node	
		2.2 Add action node	
		2.3 Add fork/join node	

Table. 1 rules 1,2 and sub rules for bi-directional transformation

RULE	GENERIC ACTION
RULE 3	Add normal activity edge (between executable action fork or join nodes)
RULE 4	Insert exception activity edge (between two control nodes)
RULE 5	Insert exception activity edge (executable to control node action)
RULE 6	Insert exception activity edge (control to executable node, action or fork/join)

Table. 2 rules 3-6 for bi-directional transformation

can be added to the models using some grammar or language text for identification.

B. A Solution using Triple Graph Grammars

From the point of TGGs the solution is to add sub rules to the six main rules. This pattern can be repeated in the future even if part of the UML super specification document changes. The sub rules are used to keep track of the A/D node or edge type that is converted. E.g. R 2.2 Add fork/join node => Petri net fork/join place, R1.3 Add merge node => Petri net merge place, R1.1 Add initial node => Petri net initial place.

The solution idea is to decompose the main rules into sub rules as required. The idea can be extended to use any class of Petri nets. The rules are independent of the actual solution and which classes of Petri nets are implemented.

The bi-directional mapping rules, in summarized form, are

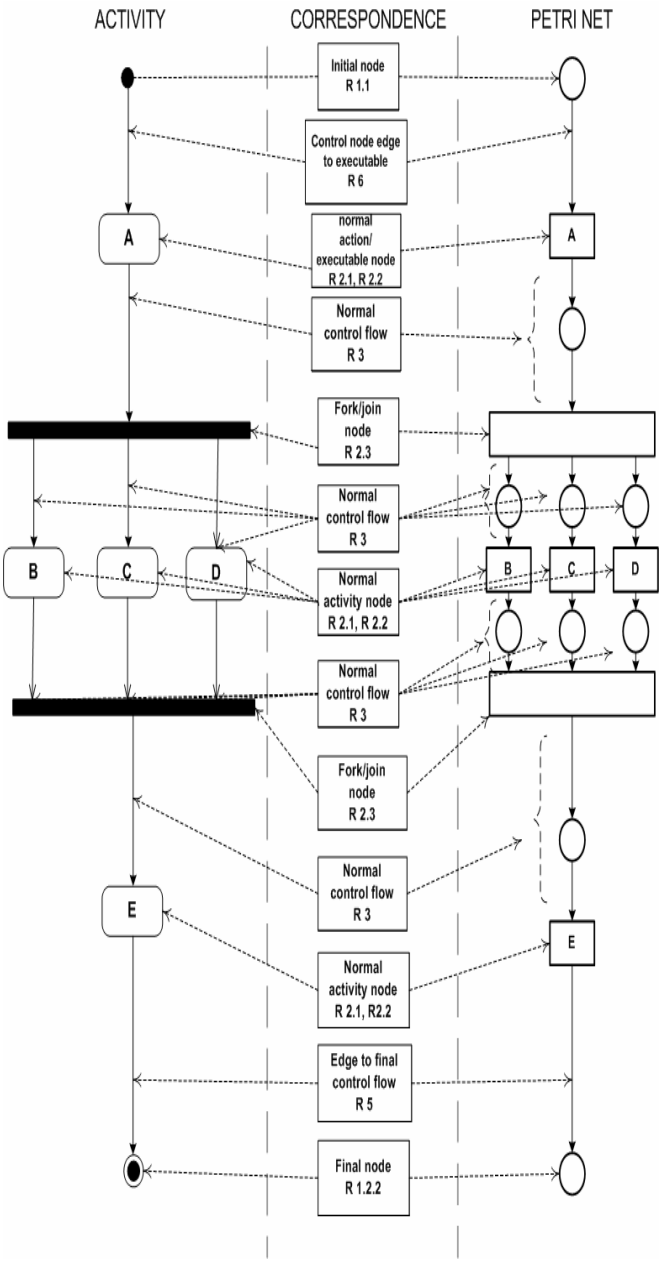

Fig. 2 high level correspondence between activities and Petri nets

shown in tables 1 and 2 respectively. These rules can be implemented using TGGs as has been done in previous work. The classification of UML 2 activity types and their corresponding Petri net counterpart is also explained in detail in [7]. It is enough to state that activity edges and nodes have their Petri net counterparts, but there are several exceptions.

The actual transformation of the actual Petri net element type has not been shown ,as this is described elsewhere [7]. To explain briefly Rule 1 maps the activity control node into a Petri net place. Rule 2 maps the action/executable fork or join into a transition. Rule 3 maps a normal activity edge into a

Petri net input arc connected to a place followed by an output arc. Rule 4 insert exception activity edge maps into a Petri net input arc to a transition followed by an output arc from the transition. Rule 5 connects an executable to a control node, this maps to a Petri net arc between a transition and a place. Similarly Rule 6 maps to a Petri net arc between a place and a transition.

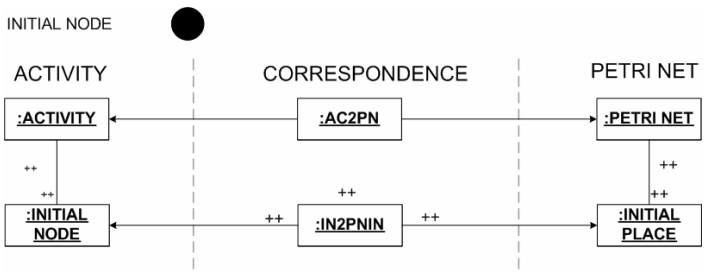

Fig. 3 TGG rule 1.1 for inserting an Initial Node

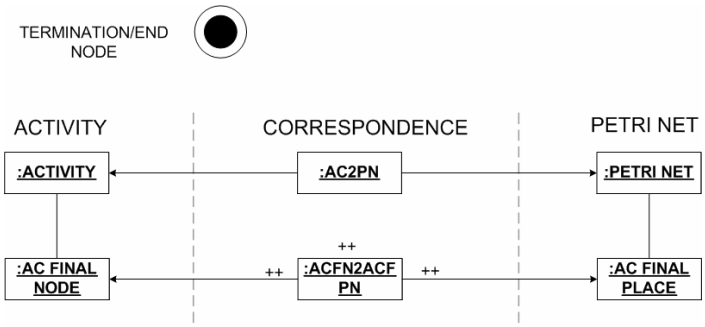

Fig. 4 TGG rule 1.2.2 for inserting an Activity Final Node

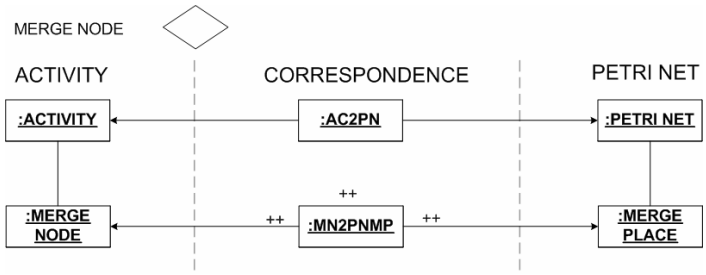

Fig. 5 TGG rule 1.3 for inserting a Merge Node

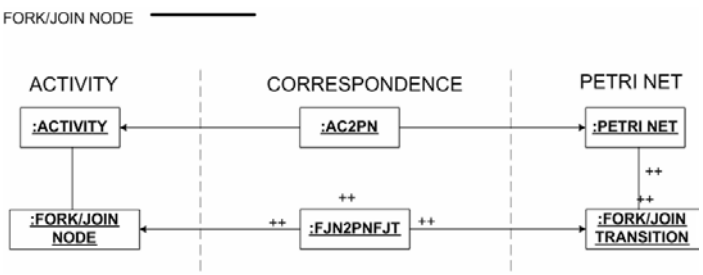

Fig. 6 TGG rule 1.3 for inserting a Fork/Join Node

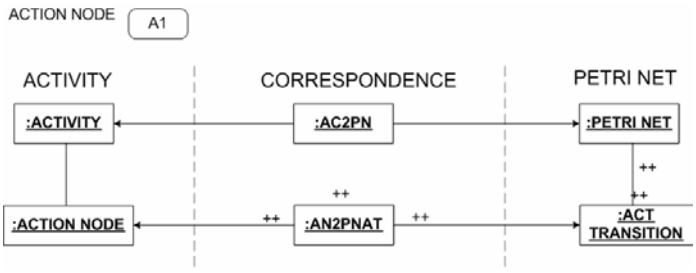

Fig. 7 TGG rule 2.2 for inserting an Action Node

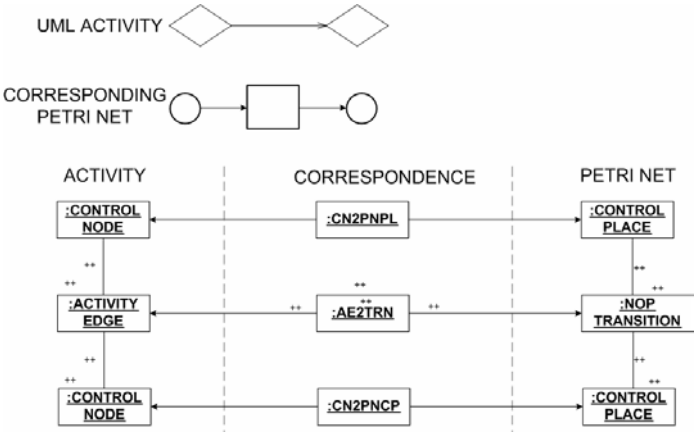

Fig. 8 TGG rule 4 for inserting an exception activity edge between two control nodes

The diagram in fig. 1 illustrates the rule based high level correspondence between activities and Petri nets. Note that not all the rules described have been used for this diagram. This diagram shows how it is possible to achieve bi-directional transformation using the appropriate rule.

From the TGG rules given, it is evident that part of the solution lies in properly labeling the Petri net and the activity. E.g. refer to fig. 3 If an initial node should be labeled as an initial place in the Petri net, this can be used for reverse transformation. The initial place will transform back into an activity diagram initial node. Figures 3-8 show how some of the rules in tables 1 and 2 can be generally applied using TGGs. Fig. 8 shows how an exception activity edge can be inserted between two control nodes. On the Petri net side this translates into an input arc that connects to a non operational transition (NOP) and the output arc from this transition.

V. Case Study

A case study of an online ordering system is used as an example to illustrate the idea presented. There are several steps involved in this order processing system. These are illustrated in fig. 9. All the main different activity node and edge types are used in the diagram. This is done to illustrate the use of the bi-directional transformation.

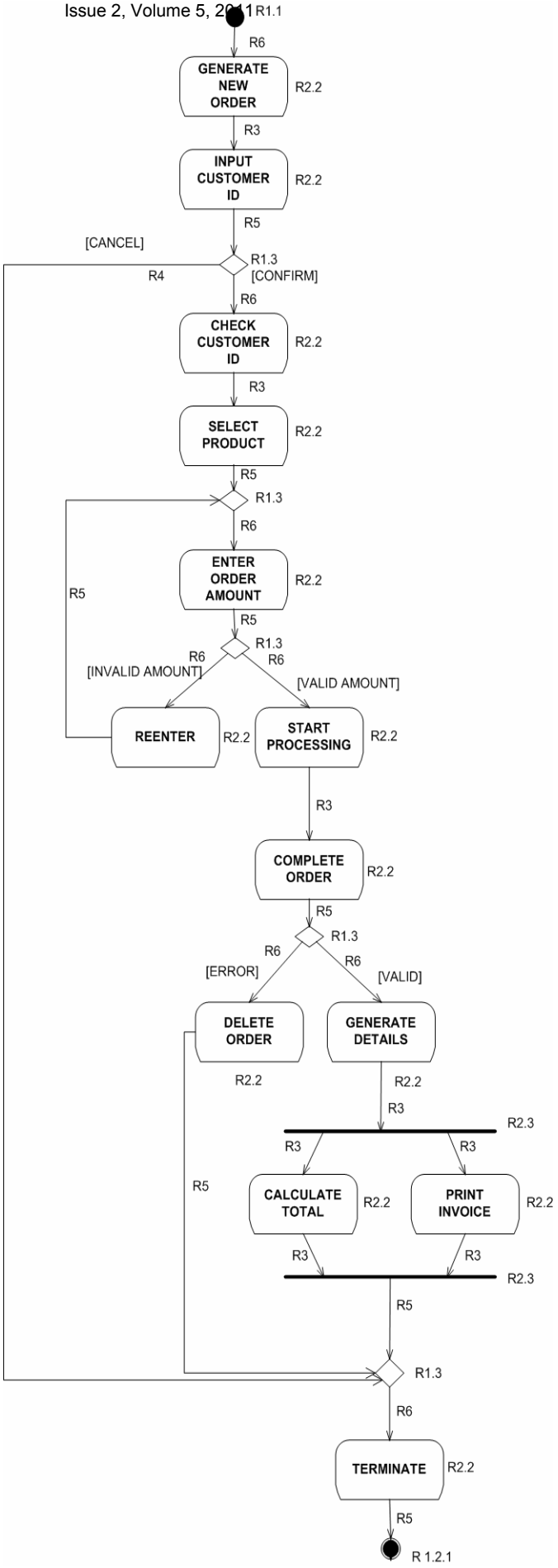

Fig. 9 online system activity diagram

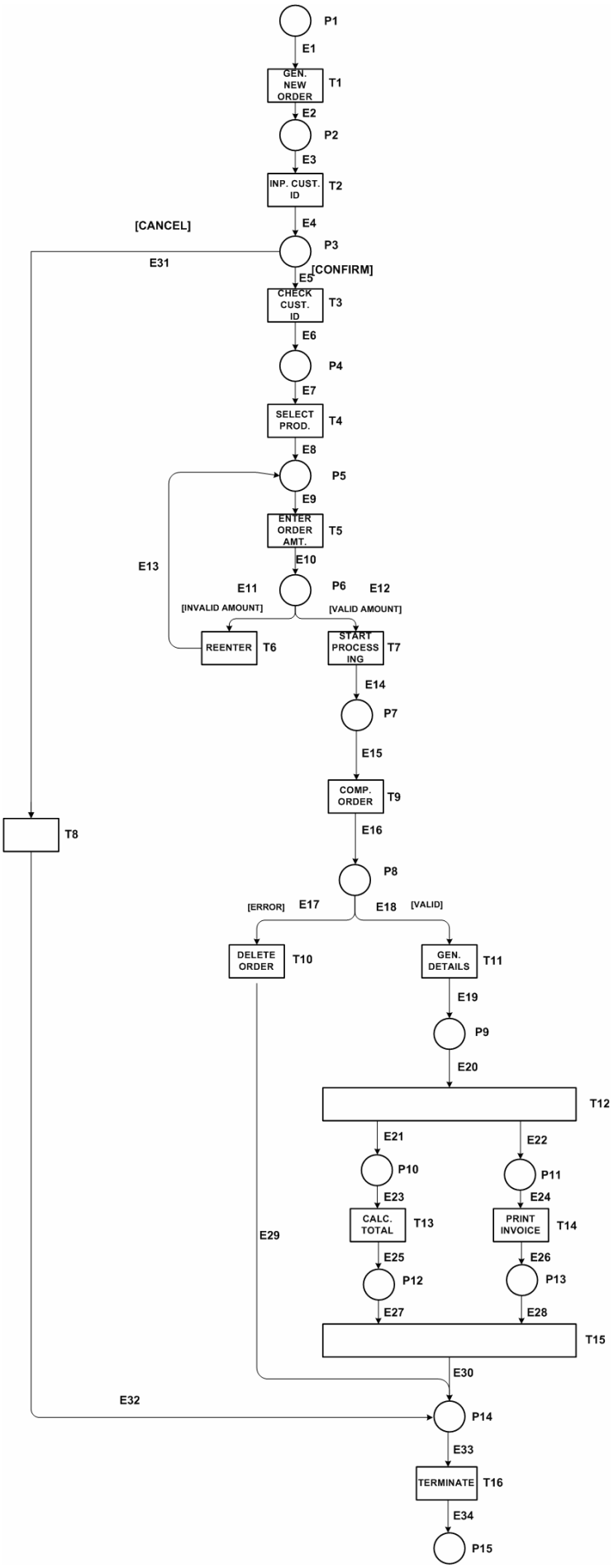

Fig. 10 online system Petri net

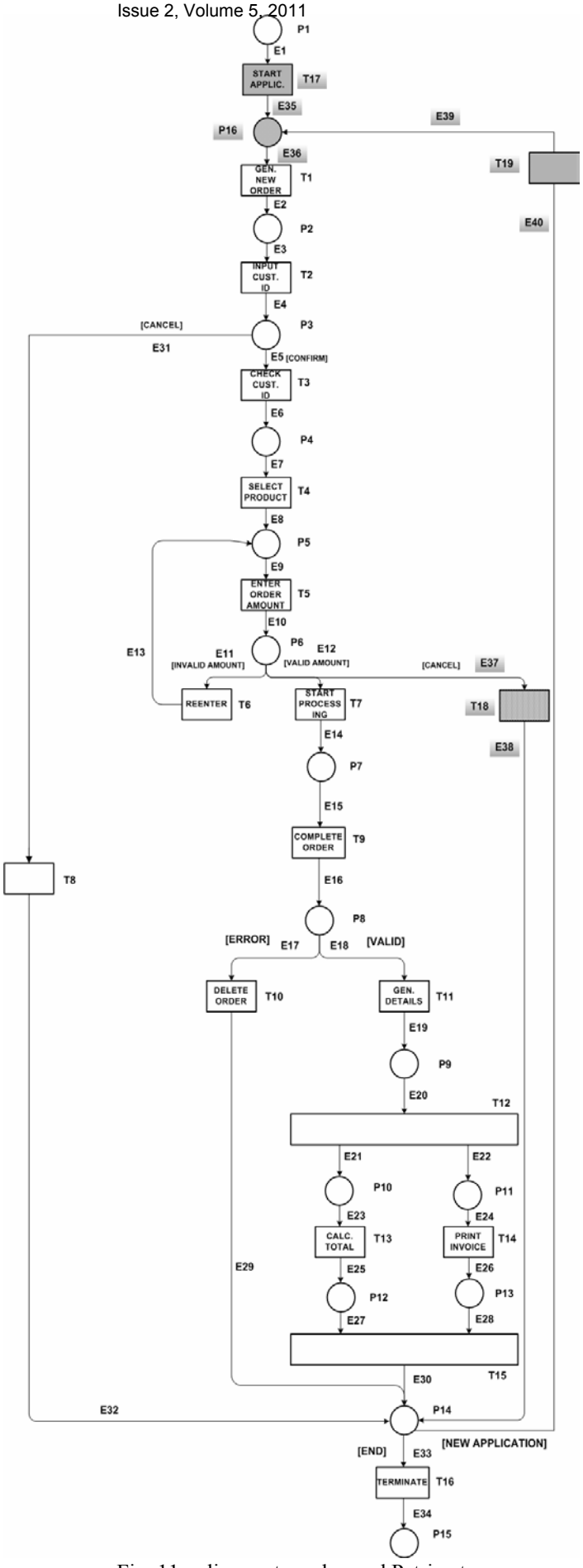

Fig. 11 online system changed Petri net

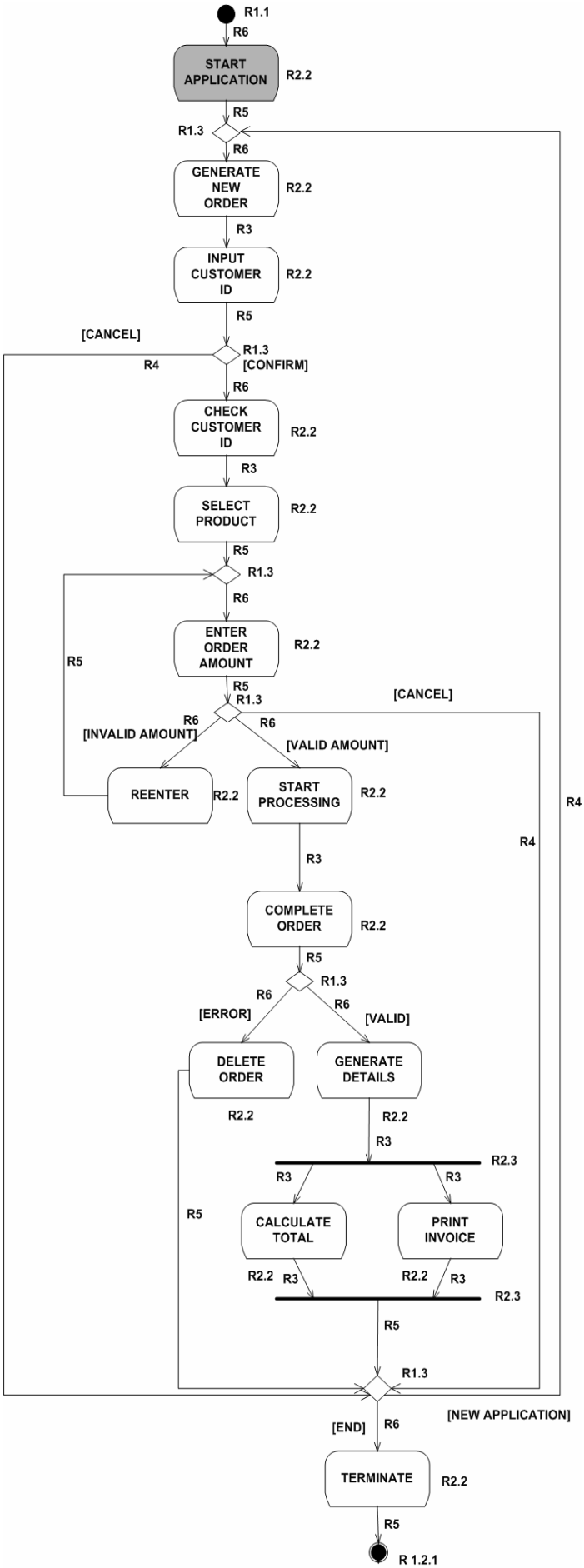

Fig. 12 online system changed activity diagram

The main steps in the activity model start off from generating an order, inputting customer details, selecting a product, entering order amount, start processing, complete order, delete order, generate details, calculate total, generate invoice, terminate etc. Some of these activities are carried out in parallel. E.g. calculate total is carried out in parallel with print invoice. Decision nodes have been used to provide for choosing to go ahead or to delete the order. To illustrate the use of all six transformation rules described, the activity diagram includes many different types of nodes and edges.

The activity diagram in figure 9 shows the case study described above. After applying the rules described in tables 1 and 2 the result is the Petri net shown in figure 10. To clarify, the applied rules are shown on the activity in figure 9 using a short notation. E.g. R2.2 stands for Rule 2.2 which is Add action node. R 3 stands for Rule 3 and stands for add normal activity edge between action executable fork or join nodes.

In applying all the rules in tables 1 and 2 the activity diagram is converted into a Petri net. The Petri net is labeled using Pn for place numbers, Tn for transition numbers and En for edge labeling.

The Petri net labels are used to identify what type of place, transition or edge is inserted. Alternatively a full description can be kept. This is indicated in table 3 which is a fragment of the place description. The description or details would be lined with the Petri net construct used.

PETRI NET	ACTIVITY DIAGRAM CORRESPONDENCY	RULE
P1	INITIAL NODE	R1.1
T1	ACTION NODE	R2.2
E1	CONTROL TO ACTION/EXECUTABLE NODE	R6
E2	NORMAL ACTIVITY EDGE	R3
…	….	….

Table 3 Petri net to activity diagram rule correspondence

To illustrate the bi-directional mapping, if changes are done to the Petri these need to update the activity diagram. Figure 11 illustrates such changes. In this example the possibility of launching a new application or ending are added. There is also a new action called start application and another cancel option.

These require adding T17 E35 E36 P16 E39 E37 E 38, etc. etc. The changes are highlighted and shown in figure 11. Using the corresponding rules e.g. for E37, E38, etc. we use Rule 4. The rules are expressed in tables 1 and 2. On the activity side this transforms to an exception activity edge as it connects two places. This can be identified from the Petri net labeling, etc. as has been previously explained.

As a result of this mapping, there is the changed or updated activity diagram shown in fig. 12.

VI. Conclusion

It has been shown how a simple practical solution for bi-directional mapping of UML 2 activities into Petri nets can be achieved using simple rules that are decomposed to the required level.

The rules can be represented using different notations such as graph grammars, triple graph grammars, multi graph grammars or even in some other relational form.

This idea will work for all activity diagrams that use the most common constructs. Even if the activity diagram has many nodes and edges this conversion is still possible. The resulting Petri net can be modified and using the rules it is possible to generate a new activity diagram or modify the existing one to reflect these changes. Petri nets are more expressible than activity models. The Petri net can be analyzed using Petri net analysis methods like reachability, liveness, etc. The Petri net can be easily converted to a time Petri net (TPN). Different Petri net CASE tools can be used to model the TPN. Also the Petri net can be reduced using Petri net reduction methods and converted back to an activity model, so many new options are available.

The ideas presented can be used for restructuring or reorganizing the Petri net and then the activity model. Petri net structures can be reduced using Petri net rules for reduction. If the activity diagram is complex then even the Petri net structure will reflect this. Finding a reduced Petri net can be used to generate a more simple activity model.

This work shows that the activity diagrams are constructed using certain patterns that are always repeated.

Here the general theoretical idea how to achieve the mapping or the conversion has been given. This still requires to be implemented. The implementation is independent of the actual solution. This mapping could be done manually. It could be possible to use this approach to derive code from the actual diagram but obviously modifications are required. This idea could be extended to other notations including other UML diagrams or other formalisms.

References

[1] H. Störrle, “Structured Nodes in UML 2.0 Activities”, *Nordic Journal of Computing*, vol. 11, no. 3, 2004, pp. 279-302.

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

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

[4] 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, 2007, pp. 253-273, Available: http://www.jot.fm/issues/issue_2007_10/paper13/

[5] E. Kindler, R. Wagner, “Triple Graph Grammers: Concepts, Extensions, Implementations and Application Scenarios”, Technical Report TR-RI-284, University of Paderborn, Paderborn, Germany, 2007, Available: http://www2.cs.uni-paderborn.de/cs/ag-schaefer/Veroeffentlichungen/Quellen/Papers/2007/tr-ri-07-284.pdf

[6] OMG UML 2 Superstructure Specification. V2.2,OMG, Available: http://www.omg.org/technology/documents/spec_catalog.htm

[7] 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*, IEEE, 2008, pp. 191-200.

[8] A. Spiteri Staines, "A TGG Approach for Mapping UML 2 Activities into Petri Nets", *Proc. 9th WSEAS, SEPADS*, Univ. of Cambridge, UK, 2010, pp. 90-95.

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

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

[11] N. Yang, H. Yu, H. Sun, Z. Qian, "Mapping UML Activity Diagrams to Analyzable Petri Net Models", *Proc. of the 2010 10th Int. Conf. on Quality Software*, IEEE, Zhangjiajie, 2010, pp. 369-372.

[12] N. Feng, W. Ming-Zhe, Y. Cui-Rong, T. Zhi-Gong, "Executable Architecture Modeling and Validation", *2nd Int. Conf. ICCAE*, IEEE, Singapore, 2010, pp. 10-14.

[13] 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, 2002, pp.16-28.

[14] L. Baresi, M. Pezze, "Improving UML with Petri Nets", *Electronic notes in Theoretical Computer Science*, Elsevier, vol 44., no. 2, 2007, pp. 107-119.

[15] 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.

[16] 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.

[17] 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.

[18] T. Levendovszky, L. Lengyel, H. Charaf, "Extending the DPO approach for topological validation of metamodel-level graph rewriting rules", *Proceedings of the 4th WSEAS International Conference on Software Engineering, Parallel & Distributed Systems*, 2005.

[19] K. H. Han, S. K. 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*, Issue 7, Volume 6, 2009, pp.1249-1258.