*i*ToNuSMV*: A prototype for enabling model checking of i* models

Novarun Deb, Nabendu Chaki
Department of Computer Science and Engineering
University of Calcutta, Kolkata, WB 700098, India
Email: novarun.db@gmail.com, nabendu@ieee.org

Aditya Ghose
Decision Systems Lab
School of Computing and Information Technology
University of Wollongong, NSW 2522 Australia
Email: aditya@uow.edu.au

***Abstract*—Goal model like i* are inherently sequence agnostic whereas standard model checkers like *SPIN* or *NuSMV* accept extended finite state models which represent an ordering of state transitions that can occur within a system. These two notions are mutually conflicting and, thus, we cannot check temporal properties (like compliance rules), described using temporal languages (like Linear Temporal Logic (LTL), or Computational Tree Logic (CTL)), on goal models by feeding them as input to a standard model checker. The *i*ToNuSMV* tool aims to bridge this gap such that the compliance of i* models towards temporal constraints can be verified in the early requirements phase of software development itself.**

***Keywords*-model transformation, i* model checking, i* compliance**

I. Introduction

Unlike dataflow and workflow models, goal models do not capture sequences of activities within the system or enterprise being designed. Although different types of goal model analysis techniques exists, the sequence agnostic nature of goal models makes them prone to errors arising out of non-compliance towards temporal properties and constraints. Since goal models have their own motivation, quite distinct from those of process models or workflow models, researchers have come up with completely different analysis techniques that provide new insights into the system or enterprise being developed.

Horkoff and Yu [3] have documented an exhaustive survey of the existing goal model analysis techniques. According to this survey, model checking techniques have been combined with planning and simulation to check goal models (like i*) [5], [7]. Fuxman [6] uses the NuSMV model verifier to check temporal constraints on goal models where as Giorgini et al.[8] have used the Datalog model checker to enhance the security and trust of i* models. Standard model checkers (like NuSMV) accept extended finite state models as input. Finite state models capture some sort of sequential information that represent the possible state transitions that a system can go through. Since goal models are sequence agnostic they cannot be fed into model checkers directly. We aim to provide a significant contribution in this direction by proposing the *i*ToNuSMV* tool that performs model transformation of goal models to finite state models.

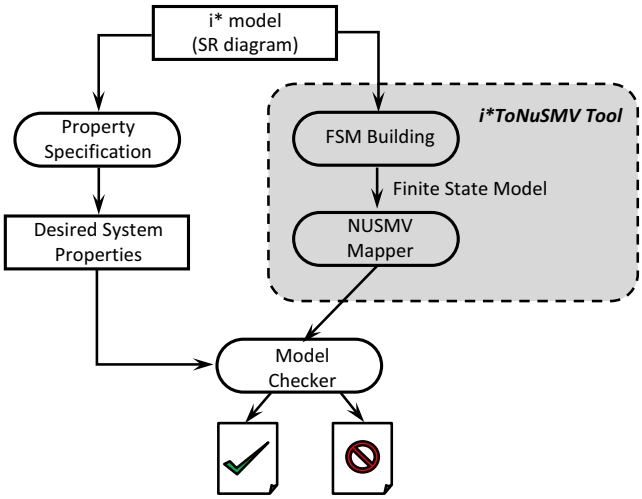

Fig. 1. The *i*ToNuSMV* tool

Figure 1 illustrates the architecture of the proposed solution. The *FSM Building* module generates a finite state model corresponding to the given i* model and the *NUSMV Mapper* module maps the generated finite state model to the NuSMV input format. The output of the *i*ToNuSMV* tool can be fed directly into the NuSMV model verifier and can be checked against temporal properties, behavioral characteristics or compliance rules written using LTL, or CTL.

II. The i*ToNuSMV Tool

A. The input module

The *i*ToNuSMV* prototype does not provide a graphical interface for drawing i* models. Rather, it takes a textual representation of the i*-SR diagram as input. For the i* model shown in Figure 2, the corresponding textual representation is intuitively derived as follows:

- **Actor** *Doctor*
- **Goal** *ProvideHealthcare*
- **MeansEnd** *Symptoms_Treat*() | *Symptoms_EMR_Treat*()
  - **Task** *Symptoms_Treat*
    - **DecomposesTo** *ObtainSymptoms*() ... **and so on**.

We perform a simplified lexical analysis of this textual input by tokenizing the text, identifying keywords, operators

2332-6441/16 $31.00 © 2016 IEEE
DOI 10.1109/RE.2016.62

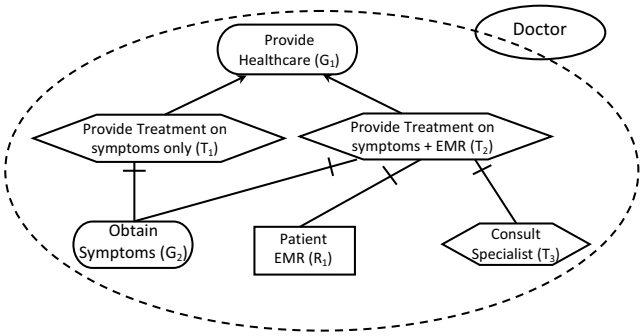

Fig. 2. An illustration for modeling the Healthcare domain

and user-defined model artifacts. We proceed to identify the tree structure of the model artifacts and begin our model transformation procedure from the root of this tree structure. In the example shown in Figure 2, the root node is *Provide Healthcare*.

B. The model transformation module

After obtaining the desired tree structure, we proceed to generate the finite state model corresponding to the given SR-diagram. We use the *Semantic Implosion Algorithm* (SIA) [1] for converting the given i* model to a finite state model. The algorithm, as proposed by the authors, proposes a methodology for exploiting the semantics of SR-diagrams and creating a finite state model with minimum number of state transitions.

SIA uses the notion of each model artefact going through three states - *Not_Created*(NC), *Created_Not_Fulfilled*(CNF), and *Fulfilled*(F) - as proposed by Fuxman in [2]. SIA controls an explosion of the state transition space by mapping *k*-element means-end decompositions to *k*-conditional branch structures and *k*-element task decompositions to *k*-dimensional hypercube lattices. A detailed illustration of the algorithm and the significant improvement achieved w.r.t. the state transitional space complexity, has been documented in the original article [1].

The URL for accessing the original article of the Semantic Implosion Algorithm is http://dx.doi.org/10.1016/j.jss.2016.03.038 .

C. The mapper module

The mapper module takes the extended finite state model produced by the SI-algorithm and maps it to the input language of the NuSMV model verifier. We assign identifiers with all goals, tasks, and resources that appear in the given SR-diagram. Each such identifier can have three possible values - `NC`, `CNF`, `FU` - corresponding to the three states (Not_Created, Created_Not_Fulfilled, and Fulfilled) as mentioned in the previous section. All identifiers are initialized to the `NC` value which marks the initial state of our finite state model. The final state of the state model is denoted by any state where the root node has the value `FU`. The state transitions of the finite state model are captured using `next()` value assignments in the NuSMV input language.

D. The platforms used

The front end of the tool has been developed in the Microsoft Visual Basic environment. The 64-bit binaries have been generated using the Pelles C platform.

III. Conclusion

The *i*ToNuSMV* tool is a research prototype that takes a user-defined textual representation of an SR-model as input. This can prove to be cumbersome for users as they have to derive these textual descriptions manually. We are currently working on integrating our tool with the tGRL language and jUCMNav framework.

A major limitation of the prototype is that it does not support inter-actor dependencies. This is a major drawback as strategic dependencies are an integral part of the i* framework. The Semantic Implosion Algorithm addresses dependencies but we have not yet implemented this in our prototype.

There is no particular reason for the authors to choose the i* goal modelling framework. The tool can be extended to any goal modelling framework that supports AND-decompositions and OR-decompositions of goals. This flexibility arises from the generic nature of the Semantic Implosion Algorithm.

Acknowledgement

We acknowledge the contribution of TCS Innovation Labs and the Technical Education Quality Improvement Programme (TEQIP), University of Calcutta, in funding this research. We also acknowledge the contribution of Nitesh Shukla and Irshad Hussain, in the coding process, as part of their masters project.

URL of the i*ToNuSMV prototype

The *i*ToNuSMV* tool can be freely downloaded from the following URL - *http://cucse.org/faculty/tools/*.

References

[1] Novarun Deb, Nabendu Chaki, Aditya Ghose, "Extracting finite state models from i* models", Journal of Systems and Software, Elsevier (2016), doi:10.1016/j.jss.2016.03.038.

[2] Ariel D. Fuxman, "Formal Analysis of Early Requirements Specifications", MS thesis (2001), Department of Computer Science, University of Toronto, Canada.

[3] Jennifer Horkoff, Eric Yu, "Analyzing Goal Models – Different Approaches and How to Choose Among Them", Proceedings of the 2011 ACM Symposium on Applied Computing (SAC '11), pages 675-682.

[4] V. Abdelzad, D. Amyot, S.A. Alwidian, and T.C. Lethbridge, "A Textual Syntax with Tool Support for the Goal-oriented Requirement Language", 8th International i* Workshop (iStar@RE 2015), Ottawa, Canada. CEUR-WS 1402, pages 61-66.

[5] V. Bryl, P. Giorgini, and J. Mylopoulos, "Supporting Requirements Analysis in Tropos: a Planning-Based Approach", Proceedings of the Pacific Rim International Workshop on Multi-Agents (PRIMA'07), Springer, 2007, Vol. 5044, pages 243-254.

[6] A. Fuxman, L. Liu, M. Pistore, M. Roveri, and J. Mylopoulos, "Specifying and analyzing early requirements: some experimental results", Proceedings International. Requirements Engineering Conference (RE'03), IEEE Press, 2003, pages 105- 114.

[7] D. Gans, D. Schmitz, T. Arzdorf, M. Jarke, and G. Lakemeyer, "SNet Reloaded: Roles, Monitoring and Agent Evolution", Proceedings International Workshop on Agent-Oriented Information Systems (AOIS'04), Springer, 2004, LNCS 3508, pages 68-84.

[8] P. Giorgini, F. Massaci, J. Mylopoulos, and N. Zannone, "Requirements Engineering meets Trust Management: Model, Methodology, and Reasoning", Proceedings of the International Itrust Conference, Springer, 2004, LNCS 2995, pages 176-190.