Annals of DAAAM for 2011 & Proceedings of the 22nd International DAAAM Symposium, Volume 22, No. 1, ISSN 1726-9679
ISBN 978-3-901509-83-4, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria, EU, 2011
Make Harmony between Technology and Nature, and Your Mind will Fly Free as a Bird
Annals & Proceedings of DAAAM International 2011

SCENARIOS AND MODEL BASED DESIGN

BLASKOVIC, B[runo]; RANDIC, M[irko] & DEMBITZ, S[andor]

Abstract: *This paper presents model based design of scenarios. A model for the scenario is declarative structure consisting of four parts, declarative, initial, transition and final part, respectively. First, the set of algorithms for model transformations are introduced. After that, simple example is given. Proof of concept and rapid tool prototype development are main concept for generating scenario program skeleton code and scenario model checking. At the end, results are discussed together with future research directions.*
Key words: *model transformation, model checking, spin*

1. INTRODUCTION

In this paper model based scenario design is described. We focus on algorithms for verified scenario definition and scenario skeleton code generation. For that purpose we first introduce data structure ScD for scenario declaration (ScD – **Sc**enario Declaration). In ideal situation scenario is derived from formal description of UML (Omg-Uml, 2009) sequence or Statechart diagrams to Spin model checker. Properties are expressed with temporal formula and analyzed. Program code is derived from UML specification.

In real situation scenario is prose text with figures, tables and diagrams. Designer can define code from scenario with tools like Stmc (State Machine Compiler) for (***, 2009) code generation and Spin (Holzmann, 2003) for model checking. We have three related models: the semi-formal prose model of a scenario, the model for code generation and the model for verification (model checking). Stmc is implemented in Java language with Stmc input language, Spin is open source C application with Promela as input language. In order to perform model based design, tools should be connected together. One approach stimulates designers to code directly in target programming language. The second approach for scenario implementation consists of direct coding in Stmc and Promela from scenario prose description. In this paper we will use the model based design approach. First, declarative scenario has been defined. Data structure describing scenario model contains enough information for scenario definition and transformation. After that, Stmc and Promela models are obtained through model translations.

2. SCOPE AND MOTIVATIONS

Scenario is partial description of the system. During system requirement and specification phase scenarios can describe various viewpoints of the system behavior (Uchitel et al., 2009). The set of all scenarios describe total system behavior.

There are other possibilities to describe the system architecture and behavior. Nowadays, in wide usage is UML (Omg-Uml, 2009) *de facto* standard between software system design community. In typical case scenario is specified with UML sequence diagram or UML Statechart. Of course, there is other specification formalism like SDL, MSC, Z…

In this place, we have to mention the long-term goal for research direction: "How to derive the application directly from high level description?" For that goal, we will use the formal method as high level specification and implementation language. But we only have partial answer to this question, suitable only for narrow problem domain and limited practical usage. We apply the formal methods to generate skeletons of program modules and test cases. For that purpose, UML sequence diagram is used to describe scenarios on the highest level. Starting from scenario description, we should perform the following steps:

- from scenario define scenario declaration ScD or UML sequence diagram
- from scenario declaration ScD define Stmc and Promela models

Several algorithms can be detected in this phase:

- scenario prose description to scenario declaration (ScD) translation. This is designer task realized without tool support.
- scenario description ScD is translated to code skeleton
- scenario description ScD is translated to model checker
- algorithms for model checking and state machine programming.

In our case, model based design has been realized with tool integration and model transformations. For each tool, input data follows ScD translation into the tool input language. That enables rapid tool prototype development. In this way we get the platform for our research goals: tools interoperability, formal methods interworking, design for validation, declarative and automatic programming.

Consider for example, the testing of property "program implementing scenario must always reach end state with desired variable values". During software development designer manually code the programs, yielding thousands lines of code. After that testing (Amman et al., 2008) is performed using chosen coverage strategies. As it is well known, ideal number of tests is always far beyond available time and human resources.

In our case, ScD specification, Promela models and Stmc program have common representation which correspondents to finite state machine and control flow graphs, respectively. Previous experience shows that when we "test" scenario in earlier development phase, concise description in ScD form yields Promela and Stmc models that are possible to define from ScD. After that, Stmc generates program skeleton and Spin checks desired property for scenario declaration. Besides that, relevant test cases can be derived from Promela models for testing purposes.

3. THE MODEL AND ALGORITHMS

ScD is the primary model of declarative scenario specification. ScD must contain enough data to define Promela or Stmc models. We use Petri net like structure (Grahlmann, 1998) for ScD description with algorithm complexity within linear limits: $O(n) \sim k*n$, $k<3$.

ScD is quadruple *(decl, init, tran, fine)* where:

- *decl* describes declaration part of scenario: global, local variables, signals and components
- *init* define preconditions. That includes initial state and initial values for all variables
- *tran* enumerates transitions: transition is in the form (*pre*, *post*, *label*) that means previous, post state and label as event/action, respectively
- *fine* define final conditions: end state(s), and values for variables

We define the following set of algorithms:

- scd2prml translates ScD or scenario description to Promela model.
- scd2stmc translates ScD to state machine input. After that Stmc tool generate program code. This algorithm translates ScD data structure to data structure suitable for Stmc input.
- StMC2tgc generates code. This algorithm is built into the Stmc tool
- prml2tc generates test cases. Test cases are defined as Spin counterexamples. That means that designer have to add constraints to Promela model. That includes the addition of "assert" commands. Note that algorithm for prml2tc is built in Spin. If multiple scenarios exist, they must have unique global variables.

4. EXAMPLE

We illustrate previous concepts with simple example from sensor networks (Sekovanic, 2011), with four states, three events and single action, without variables. Note that declaration never comes to the end state:

"If signal ev2() is detected activate service A"

```
DECLARE
  State det;
  State cen;
  State rig; State end_fin;
INIT
  State = det
TRANS
  det > cen ev1()
  cen > rig ev2()/act3(A)
  rig > det ev3()
FINAL
  STATE == end_fin AND ev3()== ();
```

Promela model has no assertions or temporal logic LTL formula. The following output illustrates counterexample and sequence of events to the error:

```
pan:1: invalid end state (at depth 3)
pan: wrote adi.prml.trail

#processes: 2 trail ends after 4 steps
4:proc1 (adi) adi.prml:26 (state 19) ev1()
4:proc0 (adi) adi.prml:55 (state 3) ev2()
4:proc1 (adi) adi.prml:26 (state 19) act3(A
4:proc0 (adi) adi.prml:55 (state 3) ev3()
```

5. RELATED WORK

In (Grahlmann, 1998) the pep structure for Petri Net definition has been described. The most important fact that has influence on this paper is the simplicity of parsing (in our case achieved with Perl script), and no need for XML libraries. This is acceptable for rapid prototype and for "proof of concept". We use modified pep-like structure (ScD) to define data types in tool chain.

Spin (Holzmann, 2003) is industrial strength model checker designed for the verification of concurrent reactive software. Spin has Promela as input language. Promela language model of concurrent system consists of the set of processes that represent dynamic behavior of the system. All Promela instructions are guards similar to Dijkstra guarded commands. Spin calculate traces from Promela model and check the properties with LTL temporal logic formula.

In (Uchitel et al., 2009) scenarios are translated to labeled transition system. Stmc (***, 2009) translates the finite state machine to code. Stmc supports many languages like C, C++, Perl , Java … However Stmc has no support for concurrency or threads. Software model checking with Clips interpreted Petri nets is presented in (Blaskovic & Randic, 2010).

6. CONCLUSION

This approach helps user to detect design errors in early development stage without deep understanding of model checking. Besides that, program code skeletons and the basic set of test cases are prepared.

Our contribution is declaration definition with ScD structure. After that, "proof of concept" tool chain for studding rapid prototyping of scenarios is developed. We find the following (+) pro- and (-) contra- arguments in our approach:

(+-) "as good as model we define". Design errors in ScD can have catastrophic impacts on design.

(-) user must have deep knowledge of temporal logic and model checking mechanism for industrial relevant examples

(+) scenario is part of the system, too big or too small scenarios are not feasible. If model is possible to verify due to space time constraint, we have optimum scenario size.

(+-) testing is still "must do" for designer.

(-) declaration has similarity with business modeling (BPEL) or with some UML and other design tool. Strict UML semantic in relevant tool will decline our approach.

Further research plans will implement scenario descriptions as high level language for telecommunication software development. We are planning the set of utilities for simultaneous skeleton code generation and software model checking.

7. REFERENCES

Ammann, P., Offutt, J. (2008). Introduction to software testing. Cambridge University Press

Blaskovic, B., Randic, M. (2010). Interpreting Petri nets with Clips for software model checking. In: Katalinic, B., editor. Proceedings of the 21st International DAAAM Symposium; Vienna, Austria, pp. 0111-0113

Grahlmann, B. (1998). Combining Finite Automata, Parallel Programs and SDL Using Petri Nets. In: Steffen, B., editor. Proceedings of TACAS'98 (Tools and Algorithms for the Construction and Analysis of Systems); vol. 1384 of Lecture Notes in Computer Science. Springer-Verlag, pp.102-117

Holzmann, G. (2003). Spin Model Checker: the Primer and Reference Manual. Addison-Wesley Professional; first ed.

OMG Unified Modeling Language (2009). OMG Document Number: formal/2009-02-04, Infrastructure Version 2.2, Available on: http://www.omg.org/spec/UML/2.2/Infrastructure

Sekovanic, I. (2011). Specification and testing of e-service scenario using model transformation. Master's thesis, Faculty of Electrical Engineering and Computing, ZOEEM, CRS laboratory (in Croatian language)

Uchitel, S., Brunet, G., Chechik, M. (2009). Synthesis of partial behavior models from properties and scenarios. IEEE Transactions on Software Engineering, 35(3):384-406. DOI http:// doi.ieeecomputersociety.org/ 10.1109/ TSE.2008.107

*** http://smc.sourceforge.net; 2009. *Accessed on: June 2011.*