POLITECNICO DI TORINO
Repository ISTITUZIONALE

An approach to refinement checking of SysML requirements

Original
An approach to refinement checking of SysML requirements / Makartetskiy, Denis; Sisto, Riccardo. - STAMPA. - (2011), pp. 1-4. (Intervento presentato al convegno IEEE 16th Conference on Emerging Technologies & Factory Automation (ETFA) tenutosi a Toulouse (France) nel 5-9 Sept. 2011) [10.1109/ETFA.2011.6059147].

Availability:
This version is available at: 11583/2460420 since: 2023-09-11T13:19:32Z

Publisher:
IEEE

Published
DOI:10.1109/ETFA.2011.6059147

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

25 April 2024

An approach to refinement checking of SysML requirements

Denis Makartetskiy and Riccardo Sisto
Politecnico di Torino
C.so Duca degli Abruzzi 24
10129 Torino, Italy
{denis.makartetskiy, riccardo.sisto}@polito.it

Abstract

During last years, the importance of safety aspects in industry has significantly increased. System engineering modeling language SysML is widely used in order to manage increasing complexity of embedded systems. Being just a modeling language, SysML does not provide integrated means of verification and validation for its models. Therefore, additional efforts are needed for checking consistency of models.

This work shows efforts towards integrating embedded systems modeling with verification measures, namely, with refinement checking (checking whether a system description is really an implementation of another, more abstract, system description) applied to statemachines linked to SysML requirements. We show how such verification can be done automatically with the help of externally implemented tools.

1 Introduction

The popularity of modeling languages has been growing significantly over the last two decades. There is a steady trend in factory automation and other safety critical industries, such as aerospace, automotive and railway to a model-driven development. Since model-driven development is based on models, the language which system is being modeled with becomes a subject for standardization activities. The latter demand for Validation and Verification (V&V) activities to be performed on these models thus demonstrating their internal consistency. In SysML requirements have been introduced as modeling elements. This approach enables strong connection between system architecture and behaviour on different abstraction levels with requirements imposed on a system/sub-system. Since in many cases requirements have complex hierarchical structure, i.e. some requirements are decomposed into other low-level requirements, it is important to keep track of adhering to some rules. For example, each low-level requirement has to realize a high-level requirement.

Industry has been recently experiencing largely growing demand in techniques and tools facilitating V&V activities for complex embedded systems. This is the case especially for safety critical systems, for which it is not only a natural demand but also a requirement of development standards.

With SysML it is handy to define requirements and relationships between them. Various model checking tools are good at checking properties expressed in formal languages. But there is still a technological gap between requirements and skill-oriented verification activities on reactive systems resulting from these requirements. The main reason for this gap is that requirements as a rule are expressed in natural language while formal analysis tools can normally check properties formulated in temporal logic. The problem is that only specialists can write correct temporal properties. The possibility to base formal analysis on properties described with a simple language could give large positive impact on workflows applied in industries where complex systems are produced.

This paper proposes a way for lightweight introduction of formal methods and automated verification in SysML models of safety-critical embedded systems. The key remark is that system engineers using SysML are likely to be skilled in writing statemachines while they are normally not skilled in writing temporal logic properties. For this reason, earlier attempts to introducing automated formal verification in SysML, which were based on model checking properties expressed in temporal logics, are probably not the best way. If SysML requirements are formalized by statemachines that describe the desired system behaviour, refinement relationships between requirements with attached statemachines can be checked automatically without requiring skills in temporal logics. This simple consideration is developed in this paper. The aim is to show how this can be put into practice, by exploiting state of the art formal verification tools based on Communicating Sequential Processes (CSP).

2 Related work

Verification of UML statemachines has been a subject for research during more than the last 10 years. UML statecharts have a complex underlying semantics. There were numerous implementations constructing a bridge between semi-formal and formal representations. [4] gives a general overview of existing approaches.

Taking some of the results coming from this line of research, the distinguishing characteristic of our work is to apply formal verification of UML statemachines in the context of requirements verification with the SysML modelling language.

A significant amount of research on formal analysis of requirements has been done. Just to mention one among the most advanced initiatives, the work in [3] resulted in the RATSY software. Our approach differs from most works in this line of research because our final aim is to integrate already available formal semantics for UML state machines and state-of-the-art refinement checking tools into an integrated environment for internal consistency checking of requirements in SysML models.

Two papers are most closely related to our work. In [2] refinement checking is done on requirements partially expressed with Controlled Natural Language and partially derived from Motorola mobile phone interfaces, by exploiting the FDR model checker. In contrast, our approach is not based on home-grown modeling techniques that suit only specific environments (Motorola phones in particular) but is smoothly integrated with a generic modeling environment.

Another related work is [7] where the SysML requirement diagram is augmented with property stereotypes linked to requirement elements. The authors claim to use model-checking for safety properties described by a UML profile, though no verification results have been reported in the paper. Also, the authors address safety properties while our focus is refinement checking.

Summing up, there were some works on adjacent areas, but there were no proposed methods for fully automatic verification of requirements modelled with widely used modeling languages. The natural idea of linking verification activities to requirements to the best of our knowledge has not been elaborated up to techniques ready to be widely applied during development processes. The current paper represents a first step for filling this gap.

3 Integrated Refinement Checking

3.1 Communicating Sequential Processes

CSP [6] is a mathematical modelling language oriented basically on the description of concurrent systems and interaction between them. CSP is based on events and processes. Processes are described by events and other operators such as parallel execution operators, choice operators, etc. Usually, CSP is applied for the description of safety-critical systems.

CSP can be used to describe the semantics of UML Statemachines so that the latter can be translated into corresponding CSP processes. Since CSP models are event-driven, state-based UML Statemachines become event-driven models after translation. Events in CSP models correspond to transitions in UML State charts, and processes correspond to states.

3.2 Refinement Checking

Refinement checking is a formal verification technique that compares two models written in the same concurrent specification language (in contrast to model checking where a property written in temporal logic is verified on a model written in another formal language). The output is a statement of whether or not one model refines another model. There are several definitions of refinement relation. In this work *trace refinement* is considered.

A trace of a model P refers to one of its possible executions. A trace is an ordered list of labels representing the time-ordered events and/or states occurring in that execution of P. Formally, given models P and Q one says that Q trace refines P if all the traces of Q correspond to possible traces of P.

The traces of Q can contain new trace elements (state transitions in terms of UML statemachines, events in terms of CSP) interleaved with trace elements of P. For more details on trace refinement see [1]. In the sequel of this paper, the term refinement denotes trace refinement.

The output of refinement checking represents a *YES/NO* answer and a counterexample (if the answer is NO). The notion of counterexample in refinement checking defines a trace which is present in one description, but absent in another one while it should be present in both.

Usually, refined models contain more transitions (events) with respect to abstract ones. Events introduced during refinement are normally disregarded when checking refinement, i.e. only the projection of the refined trace on the alphabet of the abstract trace is considered.

PAT [8] is a recently developed framework that implements various model checking techniques and refinement checking too on CSP models. The translation from UML Statecharts to CSP# (CSP + syntactic sugar) is included in PAT, which makes this tool very interesting for our purposes.

3.3 General Workflow

The aim of our work is to integrate refinement checking into the modeling process. When dealing with requirements on different levels it is important to have evidence that more detailed requirements do not contradict the less detailed requirements they are derived from. Obviously, such a checking cannot be performed automatically for plain text descriptions while it is possible when some degree of formalization is introduced in requirements. Though many formal descriptions are available

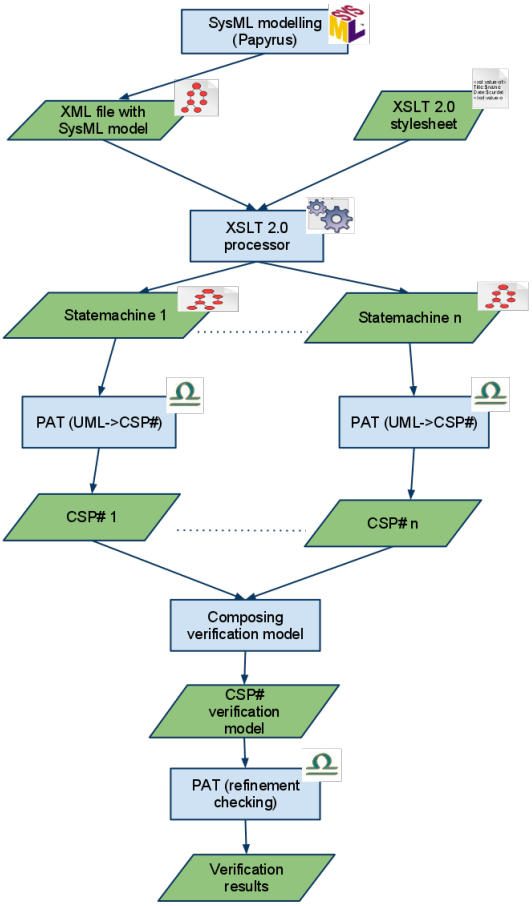


Figure 1. Dataflow for refinement checking of SysML Requirements diagram

for use, this paper focusses on reactive systems describable by UML statemachines due to reasons mentioned in Chapter 1. Pure UML, though including FSMs, is too general for system modelling and does not include specific support for requirements. This is why this paper focuses on SysML which incorporates specific constructs for representing requirements and their relationships. In SysML, state machines can be associated with requirements, thus providing the basis for assigning formal semantics to requirements. When this is done, if two requirements are bound by a refinement relationship, this relationship can be verified to hold on the state machines that formalize the two requirements.

The overall workflow for refinement checking of SysML requirements is depicted in Fig.1.

The requirement diagram is usually serialized into XML. We assume that state-machines are already assigned to requirements. In our first experiments we used MDT Papyrus [5] for SysML modelling, but any other modelling environment that is either open-source or provides powerful API or OLE Automation (or similar) facilities is suitable for our approach. Statemachines must be extracted from the serialized format. This can be done using a proper XSLT transformation, as we demonstrated by implementing the transformation stylesheet. Note that only XSLT 2.0 supports output to more than one XML file. After having obtained a single file for each statemachine associated with a requirement, each file must be translated into a language suitable for refinement checking. Fortunately, the PAT tool includes a function that derives a CSP description out of the XML representation of a UML statemachine. On the next step, automatic composing of the refinement verification model is required. Manual composing is easy, and usually it does not require much time, but many cases have to be considered in order to make it automatic. For the time being, we experimented with manual composition, leaving automation as future work. During composition, newly introduced events must be "hidden", which can be achieved by using the CSP hiding operator, and repeated names for processes have to be disambiguated. Finally, the refinement verification model consists of two corrected CSP# models corresponding to the original FSMs and an assertion of type "modelA refines model B in trace semantics". This can be analyzed by a tool like PAT.

In order to give feedback about refinement checking, results must be reflected in the modeling environment. There are two principal pieces of information about refinement checking:

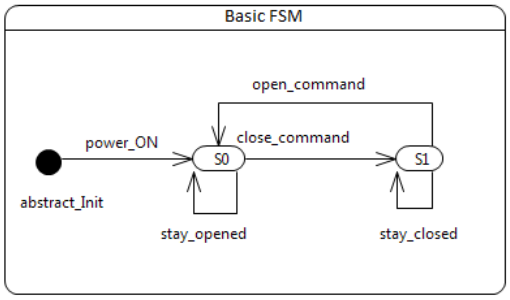
1. whether refinement is valid or not;
2. counterexample (path existing in implementation and inexistent in specification) in case refinement is not valid.

Both messages can be easily embedded in a suitable box inside the modeling environment as textual data, but highlighting the counterexample on the model is preferable as it significantly improves user experience. Another possibility for counterexample representation could be generating a UML sequence diagram with a counterexample when refinement is proved to be wrong. At the moment these features are left as future work.

4 Case Study

This section partially demonstrates the developed concept applied to a quite simple example. Let us assume that requirements have been collected and formalized within some enterprise that produces machinery for factory automation. Let us assume requirements we want to perform refinement checking on are:

1. **Basic requirement:** Mechanical press has to be able to move continuously between two basic states - "opened" and "closed".
2. **Detailed requirement:** Mechanical press should be able to move continuously between two basic states - "opened" and "closed" but it should support an alarm system that would prevent detail under pressure and



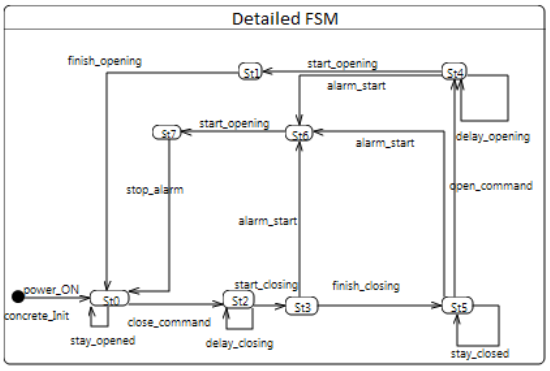


Figure 2. Statemachines corresponding to requirements

press itself from possible damage. The press must open whenever alarm is detected.

The two requirements could be represented by the state machines in Fig. 2.

Since the detailed statemachine contains transitions/events that are inexistent in the basic one we must communicate to refinement checker not to take into account these transitions/events using the CSP hiding operator.

The final input to PAT for refinement checking is a single CSP model with corrected descriptions of both statemachines and an assertion saying that the detailed statemachine refines the concrete one.

4.1 Verification results

After having composed the verification model and launched it in PAT we get the results in table 1.

Method used	on-the-fly trace refinement checking using depth-first search
Assertion (Implement() refines Spec())	VALID
Visited States	9
Total Transitions	16
Time Used	0.0059882s
Estimated Memory Used	8634.68KB

Table 1. Verification Results

5. Conclusions and Future Work

In this paper, a concept of linking high level models with refinement checking has been introduced, explained and demonstrated on a simple case study. This work is still ongoing since this approach has not yet been smoothly integrated into the modeling environment, though its core logic has already been implemented. For example, automatic composing of CSP verification models is still in progress.

The presented technique adequately suits all modeling languages which use the same paradigm for requirements modeling used by SysML. In particular EAST-ADL for modeling automotive embedded systems.

Incorporating Requirements Analysis Tool with Synthesis [3] into requirements modelling can be done in order to enable advanced requirements analysis techniques such as consistency checking and realizability of requirements (checking if there exists such a system that realizes a given requirement), but this kind of integration has been left as future work.

Also, it is important to develop case studies of realistic complexity, in order to check the real applicability of the proposed technique to industrial projects.

References

[1] R. J. R. Back and J. von Wright. Trace refinement of action systems. In *Structured Programming*, pages 367–384. Springer-Verlag, 1994.

[2] C. Bertolini and A. Mota. Using refinement checking as system checking. In *Iberoamerican Workshop on Requirements Engineering and Software Environments*, pages 17–30, 2008.

[3] R. Bloem, A. Cimatti, K. Greimel, G. Hofferek, R. Knighofer, M. Roveri, V. Schuppan, and R. Seeber. Ratsy a new requirements analysis tool with synthesis. In *Computer Aided Verification*, Lecture Notes in Computer Science, pages 425–429. Springer Berlin / Heidelberg, 2010.

[4] D. Drusinsky. *Modeling and Verification Using UML Statecharts: A Working Guide to Reactive System Design, Runtime Monitoring and Execution-based Model Checking*. Newnes, 2006.

[5] The Eclipse Foundation. *Papyrus Modelling Software*, 2011. URL: `http://wiki.eclipse.org/MDT/Papyrus`.

[6] C. A. R. Hoare. *Communicating sequential processes*. Prentice-Hall, Inc., 1985.

[7] J.-F. Petin, D. Evror, G. Morel, and P. Lamy. Combining sysml and formal models for safety requirements verification. In *22nd International Conference on Software and Systems Engineering and their Applications*, 2010.

[8] J. Sun, Y. Liu, and J. S. Dong. Model checking csp revisited: Introducing a process analysis toolkit. In *Proceedings of the Third International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2008)*, Communications in Computer and Information Science, pages 307–322. Springer, 2008.