Three-Phase Approach for Formalization and Validation of High-level requirements for Safety Critical Applications in Mobile Ad-hoc Sensor Networks

Machha.Narender
Research Scholar, Sri Sai University of Technology & Medical Sciences

Abstract - Mobile ad hoc and sensor networks (MAHSNs) are expected to become the fabric of modern societies. Despite considerable advancements, these networks are yet unable to surmount many operational challenges especially in safety-critical large-scale applications. The validation of requirements is a fundamental step in the development process of safety-critical systems. In safety critical applications such as aerospace, avionics, security, economy and quality of life, the use of formal methods is of paramount importance both for requirements and for design validation. Nevertheless, while for the verification of the design, many formal techniques have been conceived and applied, the research on formal methods for requirements validation is not yet mature. The main obstacles are that, on the one hand, the correctness of requirements is not formally defined; on the other hand that the formalization and the validation of the requirements usually demands a strong involvement of domain experts. I report on a methodology and a series of techniques that I developed for the formalization and validation of high-level requirements for safety-critical applications. The main ingredients are a very expressive formal language and automatic satisfiability procedures. The language combines first-order, temporal, and hybrid logic; automated techniques are used for language processing for the beacon messages such as automated tag extraction and discourse representation theory.

Keywords: MAHSNS, First-order, Temporal, Hybrid

I. INTRODUCTION

Formal methods are widely used in the development process of safety-critical systems. The application of formal verification techniques relies on the formalization of the system's design into a mathematical language. Several formal languages are available according to the different aspects that are relevant to the verification, and many design tools can automatically formalize the design into one of these languages. The verification techniques typically trade-off the automation of the analysis with the expressiveness of the specification language. State-of-the-art approaches mix *model checking* and *theorem proving* in order to tackle the verification of infinite-state systems with a sufficient level of automation.

In the context of *safety-critical applications*, the choice of the language used to formalize the requirements is still an open issue, requiring a delicate balance between expressiveness, decidability, and complexity of inference. The difficulty in finding a suitable trade-off lies in the fact that the requirements for many real-world applications involve several dimensions. On the one side, the objects having an active role in the target application may have complex structure and mutual relationships, whose modeling may require the use of rich data types. On the other side, static constraints over their attributes must be complemented with constraints on their temporal evolution. Constraints are put on the data where required to support validation of the formal algorithm. Pre and post conditions are defined for correct operation of communication in terms of messages

In this paper, I report on a methodology and a series of techniques that we developed for the formalization and validation of high-level requirements for safety-critical applications. The methodology is based on a three-phase approach that goes from the informal analysis of the requirements, to their formalization and validation [CRST08a]. The methodology relies on two main ingredients: a very expressive formal language and automatic satisfiability procedures. The language combines first-order, temporal, and hybrid logic [CRST08b, CRST09, CRT09]. The satisfiability procedures are based on model checking and satisfiability modulo theory. The tool [CCM+09] integrates, within a commercial environment, techniques for requirements management and model-based design, and advanced techniques for formal validation with the model checker NuSMV [CCGR00].

II. A METHODOLOGY FOR THE FORMALIZATION AND VALIDATION OF REQUIREMENTS

My methodology has been presented in [CRST08a]. It consists of three main steps:

• *Informal analysis.* The first activity in the methodology is the informal analysis of the set of requirements. In this phase, first the requirement fragments are identified and categorized on the basis of their characteristics. Then, they are structured according to their dependencies.

• *Formalization.* The second phase consists of the formalization of each categorized requirement fragment identified in the informal analysis by specifying the corresponding formal counterpart.The link between informal and formal is used for requirements traceability of the formalization against the informal textual requirements, and to select directly from the textual requirements document a categorized requirement fragment to validate.

• *Formal validation.* The third phase aims at improving the quality of the requirements and increasing the confidence that the categorized requirement fragment and its corresponding formalized counterpart meet the design intent. It consists of the definition of a series of validation problems 70 Formalization and Validation of S.-C. Requirements and the analysis of the results given by an automatic validation check. The problems include three main types of checks; namely, checking logical consistency, scenario compatibility, and property entailment:

– *Logical consistency* to formally verify the absence of logical contradictions in the considered formalized requirement fragments. It is indeed possible that two formalized requirement fragments mandate mutually incompatible behaviors. Note that this check does not require any domain knowledge.

– *Scenario compatibility* to verify whether a scenario is admitted given the constraints imposed by the considered formalized requirement fragments. Intuitively, the check for scenario compatibility can be seen as a form of simulation guided by a set of constraints. The check for scenario compatibility can be reduced to the problem of checking the consistency of the set of considered formalized requirement fragments with the constraint describing the scenario.

– *Property entailment* to verify whether an expected property is implied by the considered formalized requirement fragments. This check is similar in spirit to model checking, where a property is checked against a model. Here the considered set of formalized requirement fragment plays the role of the model against which the property must be verified. Property checking can be reduced to the problem of checking the consistency of the considered formalized requirement fragments with the negation of the property.

If one of the checks reveals a problem, two causes are possible: the first one is that the formalization is not correct due to an improper use of the formal language or to an ambiguity of the informal specification; the second possibility is that there is a flaw in the informal specification that needs to be corrected. An inspection of the diagnostic information can be carried out in order to discriminate among the two possibilities in order to take the most appropriate corrective action. In fact, the above checks not only produce a yes/no answer, but they can also provide the domain expert with diagnostic information, mainly in the form of:

– *Traces*. When consistency and scenario checking succeeds, it is possible to produce a trace witnessing the consistency, i.e. satisfying all the constraints in the considered formalized requirement fragments. Similarly, when a property check fails the tool provides a trace witnessing the violation of the property by the formalized requirement fragments.

– *Unsatisfiable core*. If the specification is inconsistent or the scenario is incompatible, no behavior can be associated to the considered formalized requirement fragments; in these cases, the tool can also generate diagnostic information in the form of a minimal inconsistent subset. This information can be given to the domain expert, to support the identification and the fix of the flaw.

2.1 A property specification language for safety-critical applications

The success of the methodology relies on the availability of a specification language which is enough expressive to represent the requirements of safety-critical applications, and enough simple to be used by domain experts and analyzed with automatic techniques.

In order to specify requirements in the context of safety-critical applications we adopt a fragment of *first-order temporal logic*. The first-order component allows to specify constraints on objects, their relationships, and their attributes, which typically have rich data types. The temporal component allows specifying constraints on the temporal evolution of the possible configurations. I have enriched the logic with constructs able to specify hybrid

aspects of the objects' attributes such as derivatives of the continuous variables and instantaneous changes of the discrete variables. The logical formulas are consequently interpreted over hybrid traces where continuous evolutions alternate with discrete changes. Finally, the logic has been designed in order to be suitable for an automatic analysis with model checking techniques.

As described in [CRST09], we use a class diagram to define the classes of objects specified by the requirements, their relationships and their attributes. The class diagram basically defines the signature of the first-order temporal logic. The functional symbols that represent the attributes and the relationships of the objects are flexible in the sense that their interpretation change at different time points. Quantifiers are allowed to range over the objects of a class, and can be intermixed with the temporal operators.

The basic atoms of the logic are arithmetic predicates of the attributes and relationships of objects.As described in [CRT09], the "next" operator can be used to refer to the value of a variable after a discrete change, while the "der" operator can be used to refer to the first derivative of continuous variables during a continuous evolution.

The temporal structure of the logic encompasses the classical linear-time temporal operators combined with regular expressions. This combination is well established in the context of digital circuits and forms the core of standard languages such as the Property Specification Language (PSL) [EF06].

On the lines of PSL, I also provide a number of syntactic sugar which increases the usability of the language by the domain experts. This includes natural language expressions that substitute the temporal operators, the quantifiers, and most of the mathematical symbols.

2.2 Model checking techniques for requirements validation

The validation process of the proposed methodology relies on a series of satisfiability checks: consistency checking is performed by solving the satisfiability problem of the conjunction of the formalized requirements; the check that the requirements are not too strict is performed by checking whether the conjunction of the requirements and the scenario's formulas is satisfiable; finally, the check that the requirements are not too weak is performed by checking whether the conjunction of the requirements and the negation of the property is unsatisfiable.

Unfortunately, the satisfiability problem of the chosen language is undecidable. The undecidability comes independently from the combination of temporal and first-order logics, from the combination of the uninterpreted functions and quantifiers, and from the hybrid component of the logic. Nevertheless, I want to keep such expressiveness in order to faithfully represent the informal requirements in the formal language. Thus, I rely on automatic albeit incomplete satisfiability procedures.

First, I fix a number of objects per class so that it is possible to reduce the formula to equi-satisfiable one free of quantifiers and functional symbols [CRST09]. As described in [CRST08b], I can automatically find a bound on the number of objects for classes under certain restrictions.

Second, I translate the resulting quantifier-free hybrid formula into an equi-satisfiable formula in the classical temporal logic over discrete traces. In this case, I exploit the linearity of the constraints over the derivatives to guarantee the existence of a piecewise-linear solution and to encode the continuity of the continuous variables into quantifier-free constraints.

Third, I compile the resulting formula into a Fair Transition System (FTS) [MP92], whose accepted language is not empty if the formula is satisfiable. For the compilation we rely on the works described in [CRT08, CRST08b]. I apply infinite-state model checking techniques to verify the language emptiness of the resulting fair transition system. In particular, I used Bounded Model Checking (BMC) [BCCZ99], particularly effective in solving the satisfiable cases and producing short models, 72 Formalization and Validation of S.-C. Requirements and Counterexample-Guided Abstraction Refinement (CEGAR) [CGJ+00], more oriented to prove the unsatisfiability cases.

The language non-emptiness check for the FTS is performed by looking for a lasso-shape trace of length up to a given bound. I encode this trace into an SMT formula using a standard BMC encoding and I submit it to a suitable SMT solver. This procedure is incomplete from two point of views: first, I am performing BMC limiting the number of different transitions in the trace; second, unlike the Boolean case, I cannot guarantee that if there is no lasso-shape trace, there does not exist an infinite trace satisfying the model (since a real variable may be forced to increase forever). Nevertheless, I find the procedure extremely efficient in the framework of requirements validation.

In order to prove the emptiness of the FTS, I use predicate abstraction. I adopt a CEGAR loop,where the abstraction generation and refinement are completely automated. The loop consists of four phases: 1) *abstraction*, where the abstract system is built according to a given set of predicates; the abstract state space is computing by

passing to the SMT solver an ALLSAT problem; 2) *verification*, where the non-emptiness of the language of the abstract system is checked; if the language is empty, it can be concluded that also the concrete system has an empty language; otherwise, an infinite trace is produced; the abstract system is finite so that I can used classical model checking techniques; 3) *simulation*: if the verification produces a trace, the simulation checks whether it is realistic by simulating it on the concrete system; if the trace can be simulated in the concrete system, it is reported as a real witness of the satisfiability of the formula; the trace is simulated by checking the satisfiability of the SMT problem; 4) *refinement*: if the simulation cannot find a concrete trace corresponding to the abstract one, the refinement discovers new predicates that, once added to the abstraction, are sufficient to rule out the unrealistic path; also this step is solved with an SMT solver.

III. RELATED WORK

Several works faced with the problem of the formal specification and validation of requirements. Some of them focused on the problem of formalizing natural language specifications, other focused on the formal specification languages to be used in such a task, other proposed a methodological approach to the requirements representation and validation.

On the first side, works such as [FGR+94] and [AG06] aim at extracting automatically from a natural language description a formal model to be analyzed. However, their target formal languages cannot express temporal constraints over object models. Moreover, they miss a methodology for an adequate formal analysis of the requirements. Other works such as [GMM90, BDZ97] provided expressive formal languages to represent the requirements. Although, the proposed languages have some similarities with ours such as the adoption of first-order temporal logic, they do not allow specification of hybrid aspects which are necessary for safety-critical applications. Also these works miss a methodology for the analysis of the formal requirements and the verification algorithms are perform either with interactive theorem proving or with model checking restricted to propositional sub-cases.

Several formal specification languages such as Z [Spi92], B [Abr96], and OCL [OMG06] have been proposed for formal model-based specification. They are very expressive but require a deep background 74 Formalization and Validation of S.-C. Requirements in order to write a correct formalization. Alloy [Jac02] is a formal language for describing structural properties of a system relying on the subset of Z [Spi92] that allows for object modeling. An Alloy specification consists of basic structures representing classes together with constraints and operations describing how the structures change dynamically. Alloy only allows to specify attributes belonging to finite domains (no Reals or Integers). Thus, it would have been impossible to model the Train position as requested by the ETCS specifications. Although Alloy supports the “next” operator (“prime” operator) to specify the temporal evolution of a given object, it does not allow to express properties using LTL and regular expressions.

Among the methodological approaches, in [HJL96], a framework is proposed for the automated checking of requirement specifications expressed in Software Cost Reduction tabular notation, which aims at detecting specification problems such as type errors, missing cases, circular definitions and nondeterminism. Although this work has many related points to our approach, the proposed language is not adapt to formalize requirements that contain functional descriptions of the system at high level of abstraction with temporal assumptions on the environment. Formal Tropos (FT) [SPGM05, FLM+04] and KAOS [DDMvL97, vL09] are goal-oriented software development methodologies that provide a visual modeling language that can be used to define an informal specification, allowing to model intentional and social concepts, such as those of actor, goal, and social relationships between actors, and annotate the diagrams with temporal constraints to characterize the valid behaviors of the model. Both FT and KAOS are limited to propositional LTL temporal constraints, and thus not suitable for formalizing safety-critical requirements.

IV. CONCLUSION

In this paper I described a recent research line that we are pursuing in the context of requirement validation for safety-critical applications. I developed an end-to-end methodology for the analysis of requirements, which combines informal and formal techniques. The property-based approach guarantees traceability, by allowing for a direct correspondence between the components of the informal specification and their formalized counterparts. The formal specification language mixes linear-temporal logic with first-order and hybrid components. Automatic albeit incomplete techniques based on model checking are used to check consistency, entailment of required properties, and possibility of desirable scenarios.

In the future, I shall pursue the following lines of activity. First, we will investigate the application of automated techniques for Natural Language Processing (e.g. automated tag extraction, discourse representation theory), in order to increase the automation of the first phase of the methodology. Second, I shall explore extensions to the expressiveness of the formalism, the relative scalability issues of the verification tools.

REFERENCES

[1] [Abr96] J.-R. Abrial. *The B-book: assigning programs to meanings*. Cambridge University Press, 1996.A. Cimatti, M. Roveri, A. Susi, S. Tonetta 75

[2] [AG06] V. Ambriola and V. Gervasi. On the Systematic Analysis of Natural Language Requirements with CIRCE. *Autom. Softw. Eng.*, 13(1):107–167, 2006.

[3] [BCCZ99] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *TACAS*, pages 193–207, 1999.

[4] [BDZ97] Philippe Du Bois, Eric Dubois, and Jean-Marc Zeippen. On the Use of a Formal R. E. Language -The Generalized Railroad Crossing Problem. In *RE*, pages 128–, 1997.

[5] [CCGR00] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A new symbolic model checker.*STTT*, 2(4):410–425, 2000.

[6] [CCM+09] Roberto Cavada, Alessandro Cimatti, AlessandroMariotti, CristianMattarei, AndreaMicheli, Sergio Mover, Marco Pensallorto, Marco Roveri, Angelo Susi, and Stefano Tonetta. Eurailcheck: Tool support for requirements validation. In *Proceedings of the 24th IEEE/ACMInternational ConferenceAutomated Software Engineering (ASE 2009)*, 2009. to appear.

[7] [CGJ+00] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement. In *CAV*, pages 154–169, 2000.

[8] [CRST08a] A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. From Informal Requirements to Property-Driven Formal Validation. In *FMICS*, LNCS, L'Aquila, Italy, sep 2008. Springer.

[9] [CRST08b] A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. Object models with temporal constraints. In *SEFM*, pages 249–258. IEEE Computer Society, 2008.

[10] [CRST09] A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. Formalizing requirements with object models and temporal constraints. *Journal of Software and Systems Modeling (SoSyM)*, 2009. DOI 10.1007/s10270-009-0130-7.

[11] [CRT08] A. Cimatti, M. Roveri, and S. Tonetta. PSL Symbolic Compilation. *IEEE Trans. on CAD of IntegratedCircuits and Systems*, 27(10):1737–1750, 2008.

[12] [CRT09] A. Cimatti, M. Roveri, and S. Tonetta. Requirements Validation for Hybrid Systems. In *CAV 2009*,LNCS, pages 188–203. Springer, 2009.

[13] [DDMvL97] R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde. GRAIL/KAOS: an environment for goal-driven requirements engineering. In *ICSE'97*, pages 612–613. ACM, 1997.

[14] [EF06] C. Eisner and D. Fisman. *A Practical Introduction to PSL*. Springer-Verlag, 2006.

[15] [FGR+94] A. Fantechi, S. Gnesi, G. Ristori, M. Carenini, M. Vanocchi, and P. Moreschini. Assisting Requirement Formalization by Means of Natural Language Translation. *Formal Methods in System Design*,4(3):243–263, 1994.

[16] [FLM+04] A. Fuxman, L. Liu, J. Mylopoulos, M. Roveri, and P. Traverso. Specifying and analyzing early requirements in Tropos. *Requirements Engineering*, 9(2):132–150, 2004.

[17] [GMM90] Carlo Ghezzi, Dino Mandrioli, and Angelo Morzenti. Trio: A logic language for executable specifications of real-time systems. *Journal of Systems and Software*, 12(2):107–123, 1990.

[18] [HJL96] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. Automated consistency checking of requirements specifications. *Trans. Softw. Eng. Methodol.*, 5(3):231–261, 1996.

[19] [Jac02] D. Jackson. Alloy: a lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, 2002.

[20] [MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems, Specification*.Springer, 1992.

[21] [OMG06] OMG. *Object Constraint Language: OMG available specification Version 2.0*, 2006.

[22] [SPGM05] A. Susi, A. Perini, P. Giorgini, and J. Mylopoulos. The TroposMetamodel and its Use. *Informatica*,29(4):401–408, 2005.

[23] [Spi92] J. M. Spivey. *The Z Notation: a reference manual, 2nd edition*. Prentice Hall, 1992.76 Formalization and Validation of S.-C. Requirements

[24] [vL09] Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to SoftwareSpecifications*. Wiley, 2009.