VARED: Verification and Analysis of Requirements and Early Designs

Julia Badger David Throop Charles Claunch

I. Motivation

Requirements are a part of every project life cycle; everything going forward in a project depends on them. Good requirements are hard to write, there are few useful tools to test, verify, or check them, and it is difficult to properly marry them to the subsequent design, especially if the requirements are written in natural language. In fact, the inconsistencies and errors in the requirements along with the difficulty in finding these errors contribute greatly to the cost of the testing and verification stage of flight software projects [1].

Large projects tend to have several thousand requirements written at various levels by different groups of people. The design process is distributed and a lack of widely accepted standards for requirements often results in a product that varies widely in style and quality. A simple way to improve this would be to standardize the design process using a set of tools and widely accepted requirements design constraints. The difficulty with this approach is finding the appropriate constraints and tools. Common complaints against the tools available include ease of use, functionality, and available features. Also, although preferable, it is rare that these tools are capable of testing the quality of the requirements.

II. Tool Description

The VARED tool chain (Figure 1) aims to provide an integrated environment to analyze and verify the requirements and early design of a system. The input to the tool are natural language requirements that have been written to some common standards. The requirements are then automatically parsed into a formalized requirements language and checked for quality using Natural Language Processing (NLP) techniques. The Requirements Conversion Engine (RCE) then automatically converts the formalized requirements into Linear Temporal Logic (LTL) statements. LTL is an appropriate form for using formal methods both on and with the requirements. The Logical Consistency Checker (LCC) will formally analyze the LTL version of the requirements for satisfiability and vacuity with respect to an appropriate state model, seamlessly incorporating automatic testing and verification of the requirements statements into the design process.

Two existing tools have been adapted to design and verify early system designs based on environment models and the formal requirements statements. The SBT Checker and the state model will help designers create controllers and system models with the appropriate structure needed for the InVeriant symbolic model checker. InVeriant will formally verify the controllers or system model against the LTL version of the requirements statements, which both incorporates automatic testing and verification for the early design stage of software development, and also formally ties the requirements and early design together.

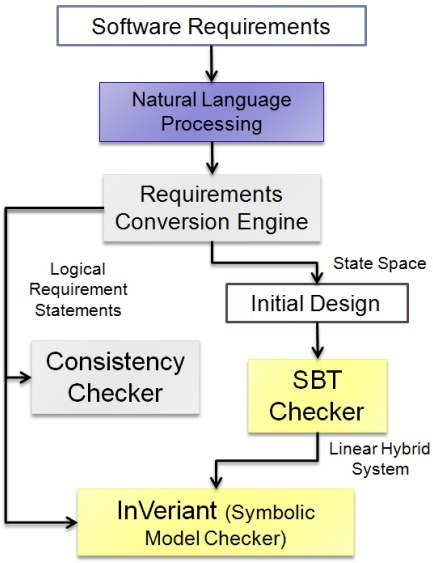

Fig. 1. VARED- integrated requirements and early design tool chain.

A novel feature of the VARED toolchain is the use of a state model of the system and its environment. This state model is developed from the scope and the high-level requirements of the system under design. The model is created using State Analysis-derived methods [2] and is used to aid nearly all pieces of the VARED tool chain, including the natural language processing, the LCC, SBT Checker, and InVeriant. Brief descriptions of the tool's components are as follows.

A. STAT/Edith

The natural language processing tool for VARED is called Edith and was built on more than a decade of work on the underlying STAT parsing tool. STAT (Semantic Text Analysis Tool) [3] is a NLP tool that was originally built to parse through space shuttle problem reports. The Edith extension allows STAT to be used to parse requirement statements into a structure that can be parsed by the Requirements Conversion Engine, if possible. Edith parses the sentences structure, which is naturally normalized by the fairly structured English language found in requirements statements, and replaces common words and phrases with general words associated with LTL semantics. It also queries the state model to find the appropriate

form that the subject and other noun phrases should take, so that the formal requirement statements can be used with the state model downstream in the tool chain. If Edith encounters a sentence structure or noun phrase that it does not recognize, the requirement statement is returned to the user with an error message; otherwise, a structured English statement conforming with the state model terminology is sent to the RCE.

B. Requirements Conversion Engine

The RCE consists of two parts. The first part is a third-party open source package called SALT that translates structured English to LTL. The second part takes the LTL output and converts it to a form that can be used by the downstream tools. SALT (Structured Assertion Language for Temporal Logic) [4] is a temporal specification language designed to create concise statements that are used in model checking and runtime verification. In particular, SALT is configured to output LTL as understood by SMV (Symbolic Model Verifier) [5]. The RCE returns requirements statements written in LTL in PANDA syntax [6], which is used by the Logical Consistency Checker and InVeriant.

C. Logical Consistency Checker

The LCC provides two functionalities for checking the quality of the formal requirement statements against the state model. First, it uses PANDA [6] to format the LTL formulas in conjunction with formalized state model properties to create an SMV input file. This input file checks the given formula against a universal model to check the satisfiability of the statement with respect to the state model. Second, the LCC will take the combined requirements and state model LTL statements and check them for vacuity [7]. Informally, an LTL formula is vacuous in an expression if that expression does not affect the value of the formula. The LCC provides the user feedback on requirements statements that are vacuous or unsatisfiable so that the user can correct the initial requirement or the state model of the system as necessary.

D. SBT Checker and InVeriant

InVeriant is a symbolic model checker that is capable of doing safety checking on a class of linear hybrid automata (LHA) that have a special property called state-based transitions [8]. This property ensures that the system covers the state model- that is, each state represented in a model of the system and its environment has a corresponding location in the model of the controller. Using the SBT Checker, this property can be checked for modularly in smaller automata that combine to create an automaton that models the entire system under control. The InVeriant model checker then combines the individual LHA into the model for the system under control using both the controller LHA inputs and the state model of the system. It then checks the overall system design against the formal requirements statements to ensure that they hold in all states of the system design. Both tools provide the user feedback if the models do not pass the respective checks, allowing the user to redesign the system model as appropriate to meet the state-based transitions property or to satisfy the requirement statement.

III. Contributions and Future Work

Though natural language processing of requirements is broadly studied (i.e., [9], [10]) and the formal methods algorithms utilized in the VARED tool are on par with current technology, the major contribution in this work is combining the two while taking a system-level approach. The design method that this tool uses requires the requirements engineer to create a model of the system and the environment from the requirements and supporting documentation. Then, using that model, the VARED tool helps designers to both analyze the quality of requirements as well as provides traceability of the requirements through the early design stage, where the controller is designed and verified against the requirements. This state model based approach is unique to both natural language processing and symbolic model checking tools available today.

The VARED tool has been developed to support systems whose controllers can be described as linear hybrid automata with certain properties. Because the properties and controller design depend heavily on the fidelity of the state model constructed, this tool is appropriate for analyzing and developing systems in the early stage of design. The choice of LTL as the specification language introduces limitations on the set of requirements that can be analyzed; however, the tool is expandable to other types of logics as needed. Several other future directions for the model checkers could be implemented to expand the usefulness of the VARED tool, such as introducing timing, liveness properties, and branching.

References

[1] D. Peercy, *Software Quality Engineering Course Guide*. SEMATECH, 1995.

[2] D. Dvorak, M. Indictor, M. Ingham, R. Rasmussen, and M. Stringfellow, "A unifying framework for systems modeling, control systems design, and system operation," *IEEE Conference on Systems, Man, and Cybernetics*, October 2005.

[3] J. T. Malin, C. Millward, F. Gomez, and D. R. Throop, "Semantic annotation of aerospace problem reports to support text mining," *IEEE Intelligent Systems*, vol. 25, pp. 20–26, 2010.

[4] A. Bauer, M. Leucker, and J. Streit, *SALT: Structured Assertion Language for Temporal Logic*. Springer Berlin / Heidelberg, 2006, vol. LNCS 4260, pp. 757–775.

[5] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang, "Symbolic model checking: 10^{20} states and beyond," in *Proc. of the Fifth Annual IEEE Symposium on Logic in Computer Science*, 1990, pp. 428–439.

[6] K. Y. Rozier and M. Y. Vardi, "A multi-encoding approach for LTL symbolic satisfiability checking," in *FM 2011: Formal Methods*. Springer, 2011, pp. 417–431.

[7] A. Gurfinkel and M. Chechik, "Robust vacuity for branching temporal logic," *ACM Transactions on Computational Logic (TOCL)*, vol. 13, no. 1, p. 1, 2012.

[8] J. M. B. Braman, "Safety verification and failure analysis of goal-based hybrid control systems," Ph.D. dissertation, California Institute of Technology, 2009.

[9] L. Mich and R. Garigliano, "NL-OOPS: A requirements analysis tool based on natural language processing," in *Proceedings of Third International Conference on Data Mining Methods and Databases for Engineering, Bologna, Italy*, 2002.

[10] F. Iwama, T. Nakamura, and H. Takeuchi, "Constructing parser for industrial software specifications containing formal and natural language description," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12, 2012, pp. 1012–1021.