Symbolic Verification of Requirements in VRS System

Oleksandr Letychevskyi
Glushkov Institute of Cybernetics
Kyiv, Ukraine
lit@iss.org.ua

Thomas Weigert
Uniquesoft LLC
Palatine, USA
thomas.weigert@uniquesoft.com

***Abstract*—VRS (Verification Requirements Specifications) system is a tool for processing formal requirements during the initial stage of software, hardware, or system development. Symbolic modeling and deductive methods are used for detection of issues such as safety violations, deadlocks, nondeterminism, or livelocks. The formal representation of requirements also supports the generation of test suites as well as the synthesis of a design model.**

***Index Terms*—symbolic modeling, verification of requirements, Use Case Maps, basic protocols**

I. Introduction

Usage of formal methods to verify system requirements has become relevant due to the growth in complexity of the designs common in hardware and software industries. Usually verification is by a formal proof of properties of an abstract mathematical model of the system captured by the requirements. Examples of such mathematical models are finite state machines, labeled transition systems, Petri nets, or process algebras. There are two main approaches to establish properties of such models: Model checking [1] is an exhaustive exploration of the states and transitions of the mathematical model. It subsumes different techniques such as abstract interpretation [2], symbolic simulation [3], abstraction refinement [4], and others. Model checking is supported by a number of tools, such as SPIN or BLAST. The other approach is deductive verification where the requirements are presented as a set of assertions and the properties are established by theorem provers such as HOL, Z3, or Isabelle.

For the implementation of verification procedures the system requirements must be expressed formally. Some formal requirements languages are close to the programming languages (VDM, CASL). UML, SysML, and special graphical notations such as URML are also leveraged for the formal representation of requirements.

VRS [5] relies on both above approaches. Symbolic simulation in VRS allows detecting reachability of the violation of correctness properties by considering symbolic states of a system and also generates a set of tests. A static requirements checking is based on formal proving of statements about system properties and involves deductive tools.

The use of symbolic and deductive methods is difficult for engineers due to the complexity and unfamiliarity of formalization and verification methods. VRS and its associated methods aim at making symbolic techniques more amenable to industrial usage by front-ending mathematical techniques with familiar or easy to learn notations, including expressive graphical presentations.

VRS is the result of over 10 years of experience in the industrial application of formal methods to the requirement gathering stage. It was deployed in a large number of projects at Motorola and Uniquesoft LLC in various subject domains, from telecommunications to networking, microprocessor programming, and automotive systems.

II. Input Language

The input language of VRS is UCM (Use Case Maps), standardized as part of URN (User Requirements Notation) in ITU-T recommendation (Z.151) which provides a scenario-based approach to requirements specification. Similar graphical approach is implemented in a tool AUTO/MAUTO (produced by Inria) [6] for specification and verification of requirements for concurrent system presented by process algebra. It can deal with model checking systems. UCM allows an easy and natural expression of sequences of events with synchronization and structure. VRS extends UCM by the language of basic protocols as developed at the Glushkov Institute of Cybernetics [7] which represents behavioral requirements as reactions of a system to externally triggered events. Such local behaviors are modeled by basic protocols which consist of three components:

- The precondition defines the state of the environment of the system at the point when the basic protocol is applicable.
- Process actions are presented as MSC (Message Sequence Chart) diagrams that show input and output signals and local actions.
- The postcondition defines the change of the environment in response to the application of the basic protocol.

Pre- and postcondition are represented as formulae of the basic logic language. It supports attributes of numeric and symbolic types, arrays, lists, and functional data types. The following example of a basic protocol (Fig.1) is taken from the specification of a well-known telecommunication protocol.

The order and synchronization of events and basic protocols is defined by means of UCM diagrams in a graphical notation. UCM diagrams are oriented graphs with initial and

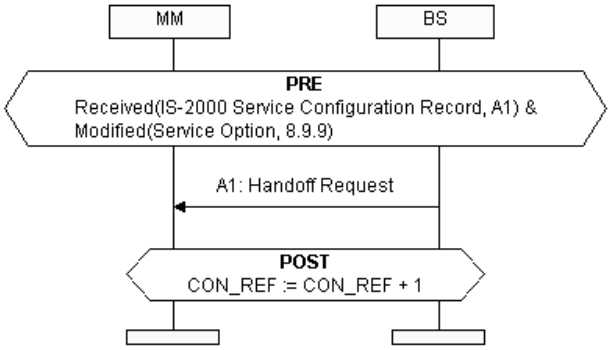

Figure 1. Basic protocol example of the requirement for a telecommunication protocol: "If an IS-2000 Service Configuration Record was received in A1: Handoff Request message and the service option was modified via section 8.9.9, the MM shall modify the attribute CON_REF by adding 1 to CON_REF"

final states. Nodes of the graph represent events in a system. The basic protocols notation captures the atomic actions (responsibilities) of a UCM map.

The UCM notation is a convenient tool for the description of parallel processes and their synchronization. An example of a UCM diagram is given below (Fig 2.).

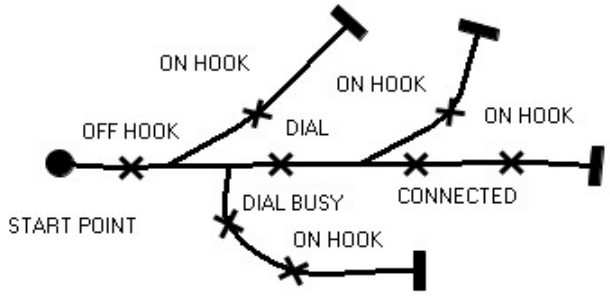

Figure 2. Example of UCM notation for requirements of phone actions in a telephony protocol

It is possible to synthesize high-level design models in terms of UML state models from these scenario models.

III. Symbolic Verification

The VRS system supports the following techniques of verification of a specification.

Checking the consistency of requirements means detection of non-determinism and ambiguities in behavioral requirements. Often such issues are deeply hidden in specifications and could entail subsequent errors and misunderstanding by developers.

Incompleteness detection helps finding of deadlock situations in formal requirements. The system establishes a trace that leads to a deadlock situation and identifies its causes. A trace is graphically presented as a MSC diagram.

Safety violations are detected by means of symbolic modeling or static proving. Liveness of a system is checked by finding the reachability of the necessary property. Livelock detection identifies situations where a system may or otherwise be nonresponsive.

Different from model checking, the VRS system is based on symbolic methods that involve deductive systems such as provers or solvers. Formal requirements present the system at a high level of abstraction where deductive techniques are most suitable. Deductive systems provide proofs of assertions in a first-order theory, resultant from the integration of theories of integer and real linear inequalities, enumerated data types, uninterpreted function symbols, and queues. This technique allows the verification of the requirements for systems with a large or infinite number of states.

Symbolic techniques also support incremental verification that is important for the development of large systems. Different parts of a formal requirements specification can be verified separately and encapsulated into enlarged entities to avoid repeating the verification of such components. For a system with high degree of feature interaction each feature can be verified separately first and their interaction can be verified without examining the individual behaviors repeatedly.

IV. Test Suite Generation

The formal presentation of requirements in UCM notation together with basic protocols gives the possibility to generate a test suite at the given level of abstraction. A set of traces can be generated from formal requirements. Traces will be obtained in MSC notation and can be converted into standard test formats. Traces contain input and output signals and local actions of the system together with the set of states of the environment that contains possible values for the system attributes. These states are symbolic and cover potentially large sets of attribute values.

The generation of traces is implemented based on different coverage criteria such as requirements statement coverage or UCM branch coverage.

References

[1] Doron Peled, Patrizio Pellicone, Paola Spoletini, "Model Checking", Wiley Encyclopedia of Computer Science and Engineering, 2009.

[2] Patrick Cousot, "Formal Verification by Abstract Interpretation", Lecture Notes in Computer Science, 2012, vol. 7211, pp. 3—7, Springer.

[3] Robert B. Jones, "Symbolic Simulation Methods for Industrial Formal Verification", 2002, Springer.

[4] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, Helmut Veith, "Counterexample-Guided Abstraction Refinement", Lecture Notes in Computer science Volume, 1855, 2000, pp. 154-169.

[5] A. Letichevsky, A. Godlevsky, O. Letychevskyi, S. Potienko, V. Peschanenko, "The properties of predicate transformer of the VRS system," Cybernetics and System Analyses, №4, 2010, pp. 3-16.

[6] A. Bouali, S. Gnesi, S. Larosa, "The Integration Project for the JACK Environment", Report, Centrum voor Wiskunde en Informacia, CS-R9443, 1994

[7] A. Letichevsky, O. Letychevskyi, T. Weigert, J. Kapitonova, V. Volkov, S. Baranov, V. Kotlyarov, "Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specications," Computer Networks, vol. 47, 2005, pp. 662-675.