Translation of Safety-Critical Software Requirements Specification to Lustre

Dongchul Park
6-240 EE/CS Building
University of Minnesota
Minneapolis, MN 55455
park@cs.umn.edu

Abstract - **SpecTRM-RL (Specification Tools and Requirements Methodology-Requirements Language) is a modeling language for describing safety-critical software requirements. However, SpecTRM-RL does not support formal verification, which plays a very important role in developing safety-critical systems and software. Lustre is a dataflow synchronous language designed for programming reactive systems. Lustre supports the analysis and formal verification as well as code generation. Therefore, by translating SpecTRM-RL into Lustre, it not only will endow verification function to SpecTRM-RL, but also will make it possible that SpecTRM-RL supports various analysis approaches of codes by using previously developed translator which converts Lustre into NuSMV, PVS, and SAL.**

In this paper, I present the rules to translate SpecTRM-RL to the Lustre language, and also present an empirical study in which we practically translate a SpecTRM-RL requirements document into Lustre using the rules proposed. This study shows that SpecTRM-RL can be effectively converted into Lustre so that it can support formal verification.

I. INTRODUCTION

Compared to a decade ago, software is widely used in safety-critical fields such as aircraft, cars, and medical instruments. So both finding errors early in the software development cycle with specification languages such as RSML^{-e} and SpecTRM-RL and software verification with analysis tools such as PVS (Prototype Verification System) and NuSMV (a new Symbolic Model Verifier) play a pivotal position in the software development [1].

SpecTRM (Specification Tools and Requirements Methodology) is a toolset to assist the development of software-intensive safety-critical systems and software [2]. Thus, SpecTRM emphasizes system requirements and specification because most decisions that affect safety are made early in the product life cycle [3]. SpecTRM-RL (SpecTRM-Requirements Language) is a modeling language for describing such requirements. However, SpecTRM and SpecTRM-RL do not support formal verification which plays a very important role in developing safety-critical systems. This is their most critical weak point of them with respect to safety-critical system development.

Lustre is a dataflow synchronous language designed for programming reactive systems such as automatic control and monitoring systems [4][5]. Lustre supports the analysis and formal verification; so if we translate SpecTRM-RL models into Lustre codes, we can give SpecTRM-RL the features of verification and analysis Lustre originally has. Lustre is also widely used as an intermediate language in translations because we can easily and efficiently parse and manipulate the Lustre codes [6]. In particular, since the translator from Lustre to NuSMV, PVS, and SAL (Symbolic Analysis Laboratory) codes has been already developed by the Crisys group at the University of Minnesota [7], only if we translate SpecTRM-RL into Lustre, can the translated codes be automatically converted into other codes for various analysis tools such as NuSMV, PVS, and SAL [8] [9]. This not only endows a formal verification function to the SpecTRM-RL model but also makes it possible for the SpecTRM-RL model to support various analyses by using those analysis tools.

In this paper, I present the rules to translate SpecTRM-RL into Lustre, and also present an empirical study in which I translated two specific SpecTRM-RL models (Altitude Switch and Cruise Control) into Lustre with the rules I proposed. I also performed black box testing to verify both results. The studies show that SpecTRM-RL can be effectively converted into Lustre.

The main contribution of this paper is the development of a set of rules to translate a safety-critical software requirements specification (SpecTRM-RL) model to Lustre. As a result, this translation enables the SpecTRM-RL model to have formal verification as well as various kinds of analytical approaches by using the previously developed translator, which, ultimately, makes SpecTRM a more powerful toolset for safety-critical systems and software development.

This paper is organized as follows. In section 2, related studies are briefly discussed. Section 3 gives the translation rules I proposed and several examples with BNF grammar, and section 4 describes translation issues and approaches to address those issues. Section 5 explains the experimental setup and results, and finally, section 6 gives the conclusion on the problem.

II. RELATED WORK

A. SpecTRM-RL

As mentioned in the introduction, SpecTRM-RL is a modeling language used in SpecTRM to describe the behavior of system components. Although this language was initially developed to describe reactive embedded control system

T. Sobh (ed.), Innovations and Advanced Techniques in Computer and Information Sciences and Engineering, 157–162.

software, it has been successfully applied to other kinds of components including hardware [10]. SpecTRM-RL models are very readable and reviewable because the syntax of the language is regular and simple; so various kinds of stakeholders including domain experts, engineers, and managers can review this model with no specialized training.

A SpecTRM-RL model is a collection of model elements that describe the behavior of a component [11]. Even though model elements consist of these several elements, all of the model elements have some common traits; first of all, all begin with a declaration in a box at the top of the elements telling what kind of model element definition follows. Second, the next line shows the name of the model element. Third, the next part of the model element is a set of attribute-value pairs. Lastly, below the attribute-value pairs every model element has some kind of definition (See Fig.1).

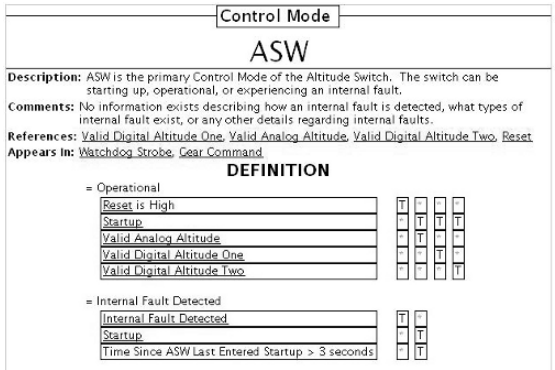

Fig. 1: A sample part of SpecTRM-RL model (mode element)

Model elements

A SpecTRM-RL model consists of 6 model elements that describe the behavior of a component: outputs, modes, states, macros, functions, and inputs.

Each output model element describes the output behaviors of the system. So each output describes the value of the output as well as the conditions with which the output is triggered. In particular, output model elements begin with triggering conditions as an output definition which is an AND/OR table; so when the triggering conditions holds, the output is sent, otherwise, it is not sent.

All components, typically, will have major operating modes that strongly affect the way it responds to inputs. SpecTRM provides mode model elements to depict such groupings of behaviors.

SpecTRM-RL includes a model element for describing the inferred system state used by the controller to trigger outputs. The definitions of states, much like modes above, consist of several transitions. Every state, especially, must have a transition to 'unknown'. Lots of accidents tend to occur because the automated controllers do not have any requirements to handle unknown cases.

In order to abstract common logic and to increase readability and understandability, macros are generally used in the SpecTRM-RL model. Macros take a piece of an AND/OR table from another part of the model and endows it with a name; so the definition of macros consists of a single AND/OR table evaluating them as true or false.

If there are complex calculations that cannot be represented within an AND/OR table, it is hard to handle such calculations. To solve this problem, SpecTRM-RL provides a function definition language. The body of the function is composed of a series of statements that perform the calculation. Each function must have certain attributes which detail when it will be evaluated, what type of value it returns and what arguments it takes as well as what their types are.

In most systems, inputs come from sensors which measure and evaluate values from the controlled process or from the operator controls. Inputs have a type of integer, real, duration, or enumerated sets. Inputs always have to transition to one of the following: new value, current value, or obsolete. In particular, the 'obsolete' transition is required of every input because many of the accidents thus far have resulted from software acted on data which is too old to be reliable.

B. Lustre

Lustre is a dataflow synchronous language designed for programming reactive systems. This supports their analysis and formal verification. In addition, Lustre is also widely used as an intermediate language in translations because it is parsed and manipulated easily and efficiently[4][12].

Streams and nodes

Lustre operates on streams constituted by a finite or infinite sequence of values and an associated clock defining the logical instants when the stream is active [13]. So the Lustre program can be seen as a set of equations operating on streams; any variable and expression denote flow; in other words, a pair which is composed of a possibly infinite sequence of values and a clock representing a sequence of times. A flow takes the n-th value of its sequence of values at the n-th time of its clock.

Lustre programs are composed of nodes which are self-contained modules with inputs, outputs, and internally declared variables. Lustre nodes are similar to functions or methods in other languages such as C and Java; however, they do not interact with global variables and values. Instead, they can retain their own state information because Lustre has a 'pre( )' expression which describes their previous value or state information. Thus, in Lustre, it is not necessarily guaranteed that the same inputs to a node will yield the same outputs. However, it always guarantees that the same input trace will yield the same output trace.

Time and Clocks

In particular, the concept of 'clock' is of consequence in Lustre. The Lustre language is basically based on the concept of logical time which is an infinite, well ordered sequence of logical instants [13]. So it should be noted that the concept of