SpecTRM: A CAD SYSTEM FOR DIGITAL AUTOMATION

Nancy G. Leveson, Massachusetts Institute of Technology, Cambridge, MA
Jon Damon Reese, Safeware Engineering Corporation, Seattle, WA
Mats P.E. Heimdahl, University of Minnesota, Minneapolis, MN

Introduction

In the system engineering of complex systems that include digital automation, the most vexing and potentially costly problems arise in the early stages of development. Few adequate tools exist to assist in developing system requirements and architectures and translating the system requirements to software requirements. Serious unsolved problems also exist at the other end of the lifecycle in changing or upgrading automated control tasks without introducing errors. In addition, these two system development phases present the most serious and unsolved problems in certification and hazard analysis.

SpecTRM-RL (Specification Tools and Requirements Methodology) is a CAD system for digital automation. It is not intended to replace engineers, but instead to use the latest in research ideas to assist engineers in managing the requirements, design, and evolution process. SpecTRM emphasizes:

- Finding errors early in development so they can be fixed with the lowest cost and impact on the system design.
- Tracing not only requirements but design rationale (including safety constraints) throughout the system design and documentation.
- Building required system properties into the design from the beginning rather than emphasizing assessment at the end of the development process when effective response is limited and costly.
- Supporting the construction of families of related systems and the reuse of the early parts of the system development process.

Complex systems cannot be built successfully without the interaction of multiple disciplines—the most challenging systems include electromechanical components, computers, and humans. While we train engineers to be experts in individual fields, these complex heterogeneous systems require knowledge across engineering disciplines. The introduction of integrated product teams or other approaches to organizing projects so specialists work together have helped to some degree, but the problem goes deeper than simply management structures. SpecTRM attempts to provide bridges between diverse groups of system designers and builders. It does this by providing modeling, specification, and analysis tools (1) to ease communication and coordinated design of components and interfaces and (2) to provide seamless transitions and mappings between the various development and maintenance stages.

SpecTRM was designed to act as a workbench for teams of engineers to enhance communication by using common models and analysis tools that execute on the models (see Figure 1). At the heart of the system development process using SpecTRM is an executable model of the system components. Our modeling language (SpecTRM-RL) emphasizes readability and requires little training. At the same time, it has a formal foundation, which allows formal analysis on the models built. In addition, visualization tools are used to assist in understanding the models and to provide the most appropriate views of the system for the task being performed. Visualization tools are also included to provide animations of the output of the model as it executes.

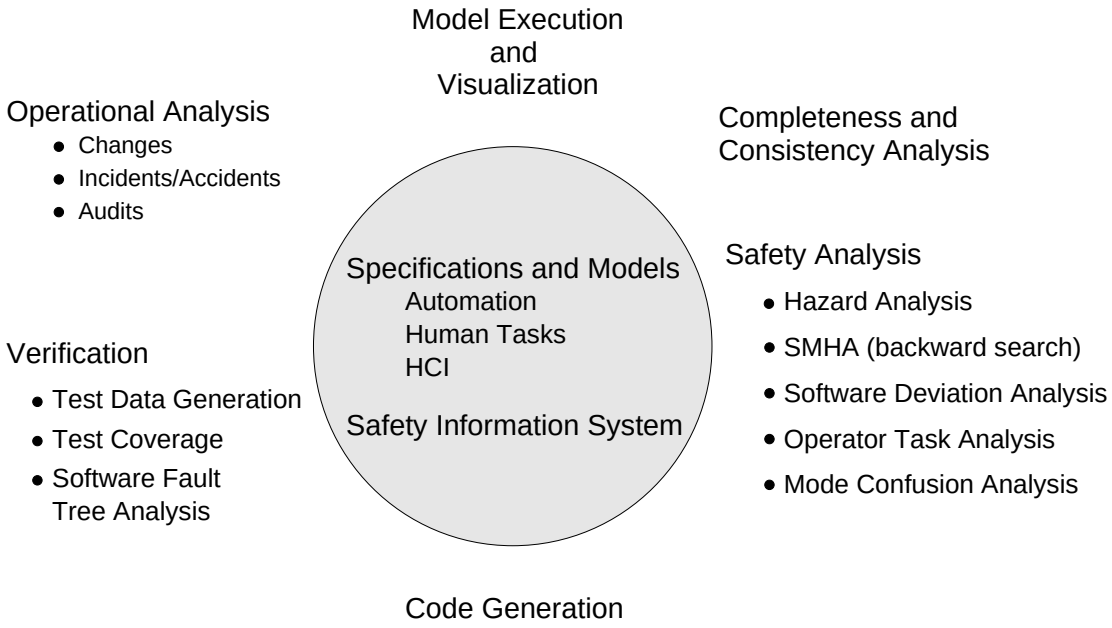

Figure 1. The SpecTRM Systems Engineering Workbench for Digital Systems.

Tools are also included to generate or assist in generating various types of analyses such as fault trees, hazard analysis, human error analyses, robustness analysis, and consistency and completeness analysis. We plan to add tools for timing analysis, reliability analysis, test data generation, blackbox test coverage analysis, and code generation. A safety information system is included to assist in maintaining hazard logs and audit trails. But instead of being separate from the development process, hazard and safety information is tightly integrated into the environment in which safety-related decisions are made.

SpecTRM is written entirely in Java and is supported on most Windows, Macintosh, and UNIX platforms. The architecture is based on a client-server paradigm where the client side can be launched from Java-enabled Web browsers or as a stand-alone, thus supporting both individual and shared use of SpecTRM design and development artifacts (such as specifications, models, safety information system contents, and modeling results). SpecTRM supports revision control, encryption, and access control.

SpecTRM is an example of technology transfer from the university to industry. Safeware Engineering Corporation is building commercial quality tools to support SpecTRM and will also act as a conduit from university research to continually upgrade the SpecTRM CAD system with additional tools and analysis capabilities as they are invented. Parts of SpecTRM have been used experimentally on real systems including a flight management system and TCAS II. NASA has contracted to use this methodology in the safety assessment of the upgrades they are building for the U.S. Air Traffic Control System.

The rest of this paper describes some of the unique features of SpecTRM that differentiate it from other system engineering tools and environments.

Intent Specifications

Many of the goals stated above are achieved through our unique specification methodology. Intent specifications are a new way of structuring system specifications that is based on research in the fundamental principles

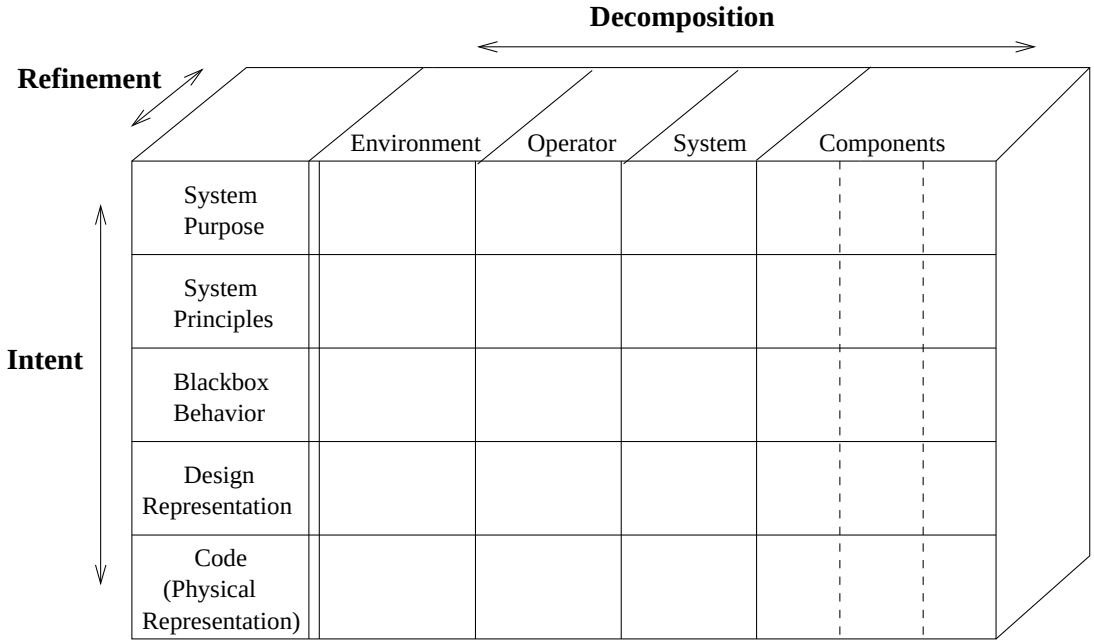

Figure 2. The form of a SpecTRM-RL specification.

of problem-solving and abstraction that humans use to make complex tasks intellectually manageable. The problems in performing system engineering and software engineering activities are rooted in complexity and intellectual manageability. Psychologists have found that complexity itself is not a problem if humans are presented with meaningful information in a coherent, structured context. "People don't mind dealing with complexity if they have some way of controlling or handling it … if a person is allowed to structure a complex situation according to his perceptual and conceptual needs" [3].

One approach found to be effective in dealing with complexity is the use of hierarchical abstraction—that is, structuring the situation such that the problem solver can transfer the problem to a different level of abstraction. Most specifications use decomposition and refinement to provide hierarchical abstraction, where each level of the usual specification can be thought of as providing "what" information while the next lower level describes "how." Intent specifications add a type of hierarchical abstraction based on goals or purpose and thus organize the levels of the specification not only in terms of what and how but also in terms of "why." By organizing the specification in this way, higher-level purpose or intent for design decisions can be immediately determined.

Systems and software are continually changing and evolving; they must be designed to be changeable and the specifications must support evolution without compromising the confidence in the properties that were initially verified. One requirement for correct and safe system evolution is knowing the rationale behind a design. Unfortunately, the reasons why something was done a certain way are often not recorded. Maintainers of a system may not know why a particular design feature was included and inadvertently remove it while making a change. Recording all the design rationale is difficult and may not be effective if the necessary information is difficult to find when a change is made. Intent specifications record and link the rationale behind design decisions ("why" information) directly in the

system specification so that it is easy to find and use.

System and software specifications in SpecTRM are organized along three abstraction dimensions: intent, refinement, and decomposition (see Figure 2). The vertical dimension specifies the level of intent at which the problem is being considered, i.e., the language or model that is currently being used. The decomposition and refinement dimensions allow users to change their focus of attention to more or less detailed views within each level or model. The information at each level is fully linked to related information at the levels above and below it.

The intent dimension has five levels of abstraction. The highest level of the specification contains the overall goals and safety constraints. Some of the information here is generated through the preliminary hazard analysis process.

The next lower level contains the underlying scientific principles upon which the design at the lower levels is based and through which the goals and constraints at the highest level are satisfied. Models and specifications at this level may be subjected to operations research and other types of system analyses to evaluate alternative designs (such as various aircraft spacing and routing alternatives for an ATC system) with respect to the higher-level goals and constraints. The third level contains a blackbox functional behavior model of the system components. These models are executable and formally analyzable. The fourth and fifth levels contain design and implementation information.

Because each level is mapped to the appropriate parts of the intent levels above and below it, traceability of not only requirements but also design rationale and design decisions is provided from high-level system goals and constraints down to code (or physical form if the function is implemented in hardware) and vice versa.

Each level of an intent specification supports a different type of reasoning about the system and represents a different model of the same system. The model at each level is described in terms of a different set of attributes or language. Level 1 (System Purpose) assists system engineers in their reasoning about system-level properties such as goals, constraints, hazards, priorities, and tradeoffs. The second level allows engineers to reason about the system in terms of the physical principles and laws upon which the design is based. The third level enhances reasoning about the logical design of the system as a whole and the interactions between components as well as the functional state without being distracted by implementation issues. The lowest two levels provide the information necessary to reason about individual component design and implementation issues. The mappings between levels provide the relational information that allows reasoning across the hierarchical levels and tracing from high-level requirements down to implementation and vice versa.

The intent information represents the design rationale upon which the specification is based and thus design rationale is integrated directly into the specification. Each level also contains information about underlying *assumptions* upon which the design and validation is based. Assumptions are especially important during analysis of operations, such as safety audits. When conditions change such that the assumptions are no longer true, then a new safety analysis should be triggered.

Because the separation of human factors and the design of the human-computer interface from the main system and component design can lead to serious deficiencies in each, we have attempted to integrate both types of specifications at each level and across levels. Interface specifications and specifications of important aspects of the system environment are also integrated into the intent specification. Finally, each level of the intent specification includes a specification of the requirements for

and results of the verification and validation activities at each level.

Safety Information System

Setting up a comprehensive and usable safety information system can be time consuming and costly, but such a system is crucial to the success of safety efforts and resources invested in it are well spent. Studies of organizations and accidents have shown that an effective safety information system ranks second only to top management concern about safety in discriminating between safe and unsafe companies matched on other variables [1, 2].

Control of any activity requires information—a crucial aspect of any management system is the feedback of information on which to base decisions. Information can be used to describe, to diagnose, to compare, to evaluate, and to improve. For example, a safety information system can provide information necessary (1) to identify and control hazards and to improve designs and standards, (2) to evaluate the effectiveness of proposed or implemented safety controls, (3) to evaluate the implications of suggested changes for their safety impact, (4) to compare models and risk assessments with actual behavior, and (5) to detect trends and deviations that presage an accident.

Documenting and tracking hazards and their resolution are basic requirements for any effective safety program. A complete hazard log and audit trail will show what was done and how and why safety decisions were made during system development, operational use, maintenance and evolution. The safety information system should also contain the most recent update of the System Safety Program Plan and the status of all the activities included in the plan, results of hazard analyses, incident and accident information including corrective action, trend analysis data, etc. Interfaces with various project databases, such as system and software configuration and control, should be well defined.

One of the most difficult aspects of maintaining a safety information system is dissemination of information in a useful form. If information is not presented to decision makers in a meaningful way, use of and learning from the data is inhibited. The information needs to be presented in a form that people can learn from, apply to their daily jobs, and use throughout the life cycle of projects. Also, the method of presenting the data should be adaptable to the cognitive styles and models of the users. SpecTRM tightly integrates the safety information system into the process and engineering environment in which safety-related decisions are made. The support provided for complete traceability from hazard analysis to design and implementation is valuable in both designing safety into the system and ensuring that safety is not compromised during operational use and system evolution.

Modeling Language

Building models is not easy and will (and should) only be done if there is enough payoff from the process. We believe that formal modeling can provide the most assistance at the system design phase where functionality is allocated to individual components. It is at this phase where tools are most needed to assist in design validation and tradeoff decisions. Models of the allocated component behavior can be executed and both formally and informally analyzed to ensure that required system functionality has been incorporated, the system design and architecture exhibits various desirable properties such as fault tolerance and safety, human—machine interactions have been appropriately designed, and tradeoffs between conflicting goals are resolved adequately. The formal modeling language in SpecTRM, called SpecTRM-RL (SpecTRM Requirements Language) is specifically designed to assist at this phase of system development. It is used in Level-3 of an Intent Specification.

SpecTRM-RL reflects lessons we learned from our use of an earlier language,

called RSML, that we designed for the FAA to specify the requirements for TCAS II, an aircraft collision avoidance system[1]. Like many of the currently popular specification languages, SpecTRM-RL uses a state machine as its underlying formal model. And like those languages, SpecTRM-RL is executable and has an associated suite of visualization and analysis tools. However, SpecTRM-RL is unique in several ways.

The first is the way that SpecTRM-RL deals with complexity. The complexity of our new systems is starting to overwhelm our current tools and cognitive capabilities. Engineers are attempting to build systems where the interactions between components cannot be thoroughly planned, understood, anticipated, and guarded against. Digital automation is exacerbating the problems by allowing greater levels of coupling with more integrated, multi-loop control and large numbers of dynamically interacting components. Besides allowing and sometimes encouraging new levels of complexity, computers introduce new types of failures modes not handled well by traditional approaches to designing for reliability and safety. In addition, computers introduce new types of problems in the interactions between components, including the interaction between humans and automation. Increased complexity and coupling is making it difficult for the designers to consider all the potential system states or for operators to handle all normal and abnormal situations and disturbances safely and effectively.

SpecTRM-RL has been designed to assist with intellectual manageability, both in terms of the difficulty inherent in building models of complex systems and in the design and engineering of complex systems themselves. Models are completely blackbox, that is, they describe component behavior only in terms of outputs and the inputs that stimulate or trigger those outputs: The model does not include any information about the internal design of the components themselves, only their externally visible behavior. The overall system behavior is described by the combined behavior of the components, and the system design is modeled in terms of these component behavior models and the interactions and interfaces between the components.

Blackbox modeling allows separation of concerns so that the review and analysis of the specified blackbox functional behavior of a system component can be separated from the review and analysis of the internal design and implementation of the component. We believe that black box functional specifications are closer to the mental model of the engineer evaluating the system architecture than specifications that are complicated by the addition of internal component design decisions. Readability and reviewability will be enhanced by using languages that allow building models that are semantically close to the reviewer's mental model of the system. That is, the *semantic distance* between the model in the expert's mind and the specification model should be minimized. Using blackbox models reduces this distance compared to models that also include component design information.

SpecTRM-RL has been tailored to support the specification of requirements for reactive and, in particular, control systems. For example, the language supports specifying complex systems in terms of operational modes. Modes are abstractions that allow categorizing classes of system behaviors and therefore enhance the understanding of complex system behavior. However, the mode structure itself can become complex and some mode-related behavior of automated systems can lead to mode confusion on the part of users. By including the specification of component behavior in terms of modes, SpecTRM-RL encourages the design of simplified mode structures and also allows mode-related analyses, such as mode confusion analysis (see below).

[1] Another paper at this conference describes those lessons learned. See "Experiences From Specifying the TCAS II Requirements Using RSML".

In our experience using a state-machine modeling language (such as RSML) for modeling the TCAS II system requirements, we found that some features of these languages are error-prone. For example, the semantics of internally broadcast events were difficult for reviewers to understand and led to different interpretations of specified behavior by different readers of the models. SpecTRM-RL substitutes less ambiguous and complex modeling facilities. The semantically complex features of general-purpose modeling languages are not needed when building blackbox models only.

In specifying the TCAS II system requirements, we also learned a lot about readability and understandability of formal models—out specification was reviewed by a large number of people having varied backgrounds and knowledge, such as computer scientists, aeronautical engineers, and pilots. Ideally, specification languages should be both formally analyzable and readable without graduate-level mathematical training.

Readability is critical. While automated tools are helpful and may even be necessary to analyze some aspects of large and complex models, our experience in using these models for industrial projects is that the most important errors will be found by expert (human) review. The analysis tools we build and the mathematical theories upon which they are based cannot possibly incorporate all the domain-specific knowledge (e.g., for TCAS this knowledge includes FAA rules and procedures, basic aeronautical engineering, human factors and cognitive psychology) required to find subtle safety-critical flaws in a complex system design.

On the other hand, the complexity of these systems leads to the need to use formal models and automated assistance to support human navigation and understanding of the models and system specifications and to perform automated analysis where possible. Note, however, that any potential design flaws found by automated tools will need to be evaluated by humans. Thus readability and understandability of the models is a requirement for human processing of the mathematical analysis results.

We solve this dilemma in SpecTRM-RL by providing both a readable specification language we believe is semantically close to a designer's mental model of the system as well as a formal modeling language that underlies the more readable specification language. The automated analysis tools are based on the underlying formal model. In SpecTRM-RL, this formal model is called the Requirements State Machine (RSM), which is a form of Mealy automaton. Only the most intuitive aspects of this formal model need be understood by readers or even builders of SpecTRM-RL specifications, and these features can be described in a few minutes.

Flawed understanding of requirements is a major cause of software-related accidents. Incomplete specifications play a significant role in these misunderstandings. Previously, we defined a set of criteria describing what is needed for a specification of a process-control system to be sufficiently complete with respect to safety. Some of these criteria are derived from mathematical aspects of the formal model underlying the specification language. Others are related to lessons learned from accidents and incidents. Many are related to human-computer interaction. A few of the criteria, namely those related to mathematical completeness, can be checked by our automated tools (and such checking for mathematical completeness is provided for most commercial and research modeling languages). Our other criteria, many of which are derived from basic control engineering concepts or from experience, have been used primarily in checklist form in industry. In designing SpecTRM-RL, we included many of the required features in the language syntax so they can either be checked by a parser or can easily be checked manually.

Analysis Tools

SpecTRM includes a suite of tools (Figure 1) that (1) assist in building, using, and maintaining the specifications and models, (2) perform various types of automated analysis on SpecTRM-RL models, and (3) perform or assist in performing various system development activities.

The intent specification editor is used to create, change, and use intent specifications. Tools also assist in building SpecTRM-RL models and visualizations. We are planning to include facilities to provide the analyst with various views of the models, both during model development (design) and evaluation (execution).

The consistency and completeness tool can be used to identify some types of incompleteness (specifically, mathematical completeness) and nondeterminism in SpecTRM-RL models. The tool detects all conditions for which no behavior has been specified and identifies parts of the model with inconsistent and nondeterministic behavior (different behaviors under the same conditions). The specification can be analyzed in small pieces and need not be finished, which means the requirements model can be analyzed incrementally as it is being constructed.

A set of safety analysis tools is also provided for requirements and hazard analysis. These tools include support for backward analysis (starting from a hazardous state and identifying predecessor states to determine how and if the hazard can be reached), forward robustness analysis (to evaluate the operation of the software in an imperfect environment), and some forms of human task and mode confusion analysis (to identify features of the required software behavior that may lead to critical human errors). Additional analysis tools are planned but will not be part of the initial release of the tool set.

Finally, SpecTRM will include tools to assist in later phases of the system and software engineering process including (1) verification (test data generation from the SpecTRM-RL models, blackbox test coverage analysis, and software fault tree analysis), (2) code generation, and (3) operational analysis (e.g., tracing the effect of proposed changes on safety).

Summary

SpecTRM is a system engineering environment to support modeling and analysis during requirements generation, functional decomposition and tradeoff analysis, subsystem specification, implementation, verification, and system maintenance and evolution. A general goal is to build bridges among disciplines by providing integrated specifications and modeling tools that can be used by system engineers, software engineers, hardware engineers, and human factors experts. We also hope to provide seamless transitions and mappings between the various system development and maintenance stages.

Because many automated real-time systems have safety-critical aspects, SpecTRM provides support for hazard analysis and building safety into the design. The safety information and activities on a project are integrated into the development and decision making environment.

Beta testing of the basic components of SpecTRM will begin in Fall 1998. The first full release of the tool set is planned for June 1999.

References

[1] U. Kjellan. Deviations and Feedback Control of Accidents. in J. Rasmussen, K. Duncan, and J. Leplat (eds.), New Technology and Human Error, pages 143-156, John Wiley & Sonds, New York, 1987.

[2] N.G. Leveson. Safeware: System Safety and Computers. Addison-Wesley Publishing Company, 1995.

[3] J.R. Newman. Extensions of Human Capability Through Information Processing and Display Systems. Technical Report SP-2560, System Development Corporation, 1966.