Draft Intent Specifications (including SpecTRM-RL) User Manual

Nancy G. Leveson

MIT
Safeware Engineering Corporation

August, 1999

1 Preface

This manual provides instructions for writing Intent Specifications, including an informal description of the SpecTRM-RL language and hints on creating SpecTRM-RL models. Several examples of intent specifications exist and can be found on either the Safeware Engineering website (www.safeware-eng.com) or on sunnyday.mit.edu/SpecTRM. The most current small example is of an altitude switch. We also have most of an intent specification for TCAS-II, but note that the design of SpecTRM-RL has changed in minor ways since the TCAS example was created. Example specifications of air traffic control software, the mobility and positioning software for a robot, an unmanned autonomous helicopter, and some spacecraft control software are in various stages of preparation and will appear on the websites when completed.

Intent specifications are based on fundamental ideas in system theory and cognitive engineering[1]. The goal is not simply to record information but to provide specifications that support human problem-solving and the tasks that humans must perform in software development and evolution. A paper describing intent specifications and the rationale behind their design can also be found on the web sites cited above (it is in the queue to be published in IEEE Trans. on Software Engineering).

2 Overview

An intent specification differs from a standard specification primarily in its structure: Hierarchical abstraction is based on *intent* ("why") rather than simply the more usual *what* and *how*. Because each level is mapped to the appropriate parts of the intent levels above and below it, traceability of design rationale and design decisions is provided from high-level system requirements and constraints down to code (or physical form if the function is implemented in hardware) and vice versa. One of the goals of intent specifications is to include safety information within the system specification or at least links to the safety database. To be used in decision making, safety analysis information must be available to the system and specialist engineers when they are making design decisions and in a form that they can use.

There are five levels in an intent specification, each level supporting a different type of reasoning about the system. The information at each level includes emergent information about the level below and represents a different model of the same system. Each level is not simply a refinement (which is done within each level) but describes the system in terms of a different set of attributes or language. Figure 1 shows the overall structure.

[1] *Cognitive engineering* is a term that has come to denote the combination of ideas from systems engineering, cognitive psychology, and human factors to cope with the challenges of building high-tech systems composed of humans and machines. These challenges have necessitated augmenting traditional human factors approaches to consider the cognitive capabilities and limitations of the human element in complex systems.

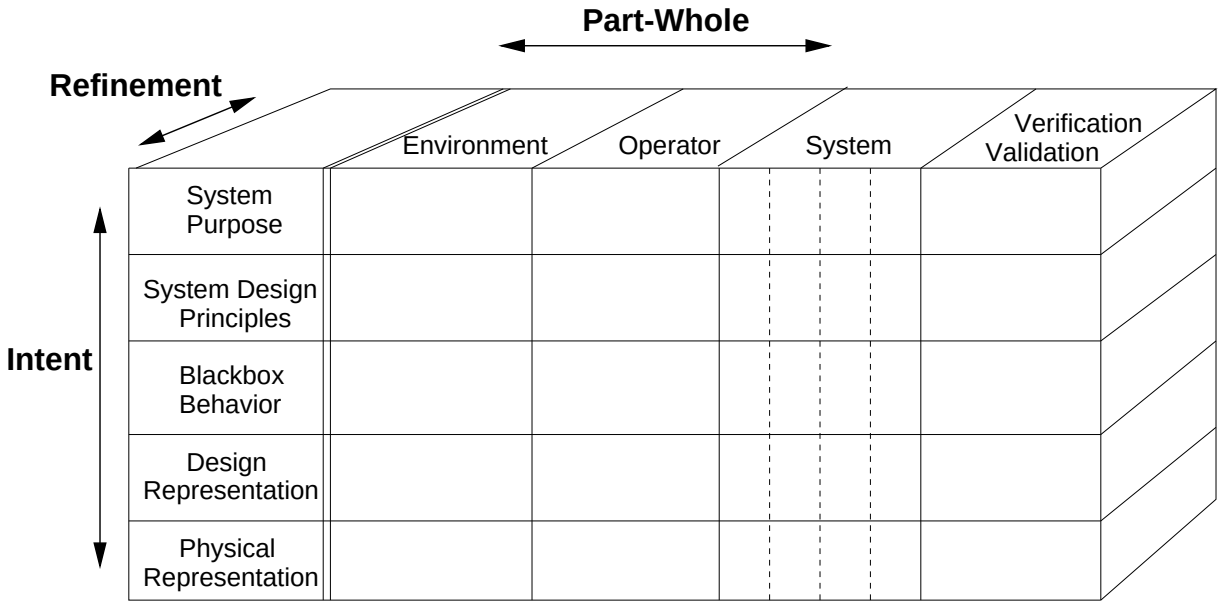


Figure 1: The structure of an intent specification.

The highest level of an intent specification assists system engineers in their reasoning about system-level properties such as goals, constraints, hazards, priorities, and tradeoffs among them. The second level, System Design Principles, allows engineers to reason about the system in terms of the physical principles and laws upon which the design is based. The third or Blackbox Behavior level, enhances reasoning about the logical design of the system as a whole and the interactions between the 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 used to explain a decision or to record fundamental information on which the design is based.

Assumptions are especially important in safety analyses; operational safety depends on the accuracy of the assumptions and models underlying the design and hazard analysis processes. The operational system should be monitored to ensure (1) that it is constructed, operated, and maintained in the manner assumed by the designers, (2) that the models and assumptions used during initial decision making and design were correct, and (3) that the models and assumptions are not violated by changes to the system, including workarounds and unauthorized changes in proce-

dures, or by changes in the environment. Operational feedback on trends, incidents, and accidents should trigger reanalysis when appropriate. Linking requirements, design features, and assumptions underlying them throughout the document with the hazard analyses will assist in performing maintenance and evolution activities.

Assumptions underlying safety and other analyses may be (and usually are) included in a safety analysis document (or at least should be), but are not usually traced to the parts of the implementation they affect. Thus the system safety engineer may know that a safety analysis assumption has changed (e.g., the pacemakers are now being used on children rather than the adults for which the device was originally designed and validated), but it is a very difficult and resource-intensive process to figure out what parts of the design used that assumption unless traceability is maintained to and within the system specification.

Each of the five intent levels is also organized in terms of the more common part-whole abstractions, i.e., parallel decomposition and refinement. Each level contains information about the environment, the operator and operational procedures, the human-machine interface, and the system (and its components). Each level of an intent specification also includes a specification of the requirements and results of verification and validation activities of the information at That specification level.

The specification as a whole allows a seamless transition from system to component (including software) specifications and the integration of formal and informal aspects of system and software development. Because the structuring is based on what is known about human problem solving, we believe that this type of specification will enhance human processing and use of specifications and will also enhance our ability to engineer for quality and to build evolvable and changeable systems without degrading quality. The structure is designed to facilitate the tracing of system level requirements and constraints into the design and the assurance of various system properties (such as safety) in the initial design and implementation as well as reduce the costs of implementing changes and reanalysis when the system is changed, as it inevitably will be.

Although various types of tools might be used to create intent specifications, Safeware Engineering Corp. has a set of tools that assist in creating intent specifications as well as executing the executable parts of the specification and performing various types of analysis. This tool set is still in early development but an alpha test version including some basic functionality is currently in "use" testing. Additional tools and facilities will be added as they become available.

In the SpecTRM tool set, the first number or letter in a link tells you where it is located (and what type of information is in it):

Number 1-5: Requirement on Levels 1 to 5, respectively
G: Goal (Level 1)
EA: Environmental Assumption (Level 1)
EC: Environment Constraint (Level 1)
OP: Operator behavioral requirement, assumption, or constraint (Level 1)

L: System Limitation (Level 1)
C: Non-safety-related design constraint (Level 1)
SC: Safety-related design constraint (Level 1)
FTA-x: Line x of the Fault Tree Analysis

The rest of this manual describes the information to be found at each of the five levels of an intent specification and the language at each level that is supported by the SpecTRM tools. Note that although the levels of the intent specification roughly correspond to the basic phases of the system engineering process, a pure "waterfall" model is not being implied. Every real process contains iteration and skipping around among phases. But the final documentation should be organized around and reflect, as Parnas has suggested, a "rational" design process. Such an organization will make the documentation easier to understand. Most of the information included in an intent specification is usually documented somewhere for any complex system; intent specifications simply prescribe a structure that will make it easier to find and use the information when needed for system engineering and software engineering tasks.

3 Level 1: System Purpose, Constraints, and Limitations

The highest level of an intent specification contains the (1) system goals and requirements, (2) system-level design constraints, (3) assumptions and limitations of the system, (4) assumptions about the environment in which the component will operate, and (5) results of analyses for system-level qualities such as preliminary and system hazard analyses. Most of the information at this level is generated during the conceptual design phase, but information will probably be added or changed during later design phases. This level will be the starting place for those unfamiliar with the system when trying to learn about it and to understand why it was designed the way it was.

We recommend putting the following types of information on this level, although not every category of information may be appropriate for every system and additional categories of information may also be useful for some systems. Examples of most of these sections can be found in our system specification for TCAS-II.

Introduction: A brief overview of the system for those who have no information about it.

Historical Perspective: Relevant historical information about the development of the system or of related systems (perhaps previous versions and why they are being superceded).

Environment Description: The "givens" or environment in which the system being specified will function including the components that already exist or at least are not being designed as part of this system development. In TCAS, this section includes descriptions of relevant properties of the Mode-S transponder and its control panel; altitude reporting data that will be sent to the TCAS unit (high-level type information only, detailed physical descriptions are included in lower levels of the specification); antennas; aircraft discretes read by TCAS; aircraft identification, attitude, and heading information available to TCAS; the relation of TCAS warnings to other aircraft alerts and warnings (e.g., the system priority for windshear, Ground Proximity Warning System, and TCAS alerts, and the behavior expected of TCAS when TCAS is inhibited by one of these alerts).

Environment Assumptions: The assumptions about how the environment in which the system being designed will operate. The correct operation of the system as well as the hazard analyses will be based on these assumptions. Examples for TCAS: The other aircraft have operating transponders; all aircraft have legal identification numbers; altitude information is available from intruding targets with a minimum precision of 100 feet; threat aircraft will not make an abrupt maneuver that thwarts the TCAS escape maneuver.

Environment Constraints: Constraints on the design of the environment to assure correct behavior of the system being designed. For TCAS, for example, the behavior or interaction of non-TCAS equipment with TCAS must not degrade the performance of the TCAS equipment. Constraints on the environment are similar to assumptions about the environment (and could be combined into one section), but constraints reflect how the new system can be integrated into and operate within a larger system while assumptions specify what assumptions the system being designed is making about the environments within which it will operate. Any environment assumptions or constraints linked to the hazard analysis may lead to restrictions on the use of the system or to the need for system safety analyses to determine that the requirements hold for any larger system in which the system being specified is used.

System Functional Goals: High-level goals for the system. Usually, in the early stages of a project, goals are stated in very general terms. One of the first steps in defining system requirements is to refine the goals into testable and achievable high-level requirements (the assumption is made here that requirements must be measurable and testable to be called a requirement). For TCAS, a high-level goal is to [G1] *Provide affordable and compatible collision avoidance system options for a broad spectrum of National Airspace System users.*

High-Level Requirements: Testable and achievable refinements of the system functional goals. For G1 above, this might include a requirement to

1.1: *Provide collision avoidance protection for any two aircraft closing horizontally at any rate up to 1200 knots and vertically up to 10,000 feet per minute.*

> ASSUMPTION: *Commercial aircraft can operate up to 600 knots and 5000 fpm during vertical climb or controlled descent (and therefore two planes can close horizontally up to 1200 knots and vertically up to 10,000 fpm).*

As another example:

1.2: *TCAS shall operate in enroute and terminal areas with traffic densities up to 0.3 aircraft per square nautical miles (i.e., 24 aircraft within 5 nmi).*

> ASSUMPTION: *Traffic density may increase to this level by 1990, and this will be the maximum density over the next 20 years.*

Design and Safety Constraints: Restrictions on how the system can achieve its goals. For example, TCAS is not allowed to interfere with the ground-level air traffic control system while it is trying to maintain adequate separation between aircraft. Avoiding interference is not a goal or purpose of TCAS—the best way to achieve it is not to build the system at all. It is instead a constraint on how the system can achieve its goals, i.e., a constraint on the potential system designs. Because of the need to evaluate and clarify tradeoffs among alternative designs, separating these two types of intent information (goals, requirements, and design constraints) is important.

For safety-critical systems, constraints should be further separated into normal and safety-related. Examples of *non-safety constraints* for TCAS-II are:

C1: *The system must use the transponders routinely carried by aircraft for ground ATC purposes.*

C2: *No deviations from current FAA policies and philosophies must be required.*

Safety-related constraints should have two-way links to the system hazard log and perhaps links to any analysis results that led to that constraint being identified. Hazard analyses specified on this level are linked to Level 1 requirements and constraints on this level, to design features on Level 2, and to system limitations (or accepted risks). Links are created and maintained automatically by the Safeware tools.

Example TCAS safety constraints are:

SC1: *The system must generate advisories that require as little deviation as possible from ATC clearances.*

...

SC5: *The system must not disrupt the pilot and ATC operations during critical phases of flight nor disrupt aircraft operation.*

SC5.1: *The pilot of a TCAS-equipped aircraft must have the option to switch to the Traffic-Advisory-Only mode where TAs are displayed but display of resolution advisories is inhibited.*

ASSUMPTION: *This feature will be used during final approach to parallel runways, when two aircraft are projected to come close to each other and TCAS would call for an evasive maneuver.*

Note in SC5.1 how *refinement* occurs at the same level of the intent specification. As stated earlier, the other intent specification levels are not refinements but rather specify emergent (intent) properties about the level below and use different models and languages (attributes).

System Limitations: Limitations may be related to the basic functional requirements, e.g.,

L1: *TCAS does not currently indicate horizontal escape maneuvers and therefore does not (and is not intended to) increase horizontal separation.*

or to environment assumptions, e.g.,

L2: *TCAS provides no protection against aircraft with nonoperational transponders.*

L3: *Aircraft performance limitations constrain the magnitude of the escape maneuver that the flight crew can safely execute in response to a resolution advisory. It is possible for these limitations to preclude a successful resolution of the conflict.*

L4: *TCAS is dependent on the accuracy of the threat aircraft's reported altitude. Separation assurance may be degraded by errors in intruder pressure altitude as reported by the transponder of the intruder aircraft.*

ASSUMPTION: *This limitation holds for existing airspace, where many aircraft use pressure altimeters rather than GPS. As more aircraft install GPS systems with greater accuracy than current pressure altimeters, this limitation will be reduced or eliminated.*

Limitations often involved accepted risks, i.e., hazards that could not be completely eliminated, mitigated, reduced to an acceptable level, or in some other way resolved satisfactorily, e.g.,

L5: *TCAS will not issue an advisory if it is turned on or enabled to issue resolution advisories in the middle of a conflict (→FTA-405)*[2].

[2]The pointer to FTA-405 denotes the box labelled 405 in the Level-1 fault tree

L6: *If only one of two aircraft is TCAS equipped while the other has only ATCRBS altitude-reporting capability, the assurance of safe separation may be reduced (→FTA-290).*

Finally, limitations may be related to problems encountered or tradeoffs made during the system design process (recorded on lower levels of the intent specification). For example, TCAS has a Level 1 performance monitoring requirement that led to the inclusion of a self-test function in the system design to determine whether TCAS is operating correctly. The following system limitation relates to this self-test facility:

L7: *Use by the pilot of the self-test function in flight will inhibit TCAS operation for up to 20 seconds depending upon the number of targets being tracked. The ATC transponder will not function during some portion of the self-test sequence.*

Limitations will commonly have links or pointers to operational procedures and entries in user manuals. For example, L7 might be linked to a corresponding entry in the Aircraft Flight Manual for TCAS-II.

Operator Requirements: Operator behavior assumptions, requirements, and constraints. This information is used in the design of the human-computer interface, the system logic, operator tasks and procedures, operator documentation (e.g., aircraft flight manuals), and training plans and programs. Example TCAS II operator requirements are:

O1: *After the threat is resolved, the pilot shall return promptly and smoothly to his/her previously assigned flight path.*

O2: *The pilot must not maneuver on the basis of a Traffic Advisory only.*

Explanations would normally be provided for the rationale underlying each of these requirements as well as links to other parts of the system intent specification related to these operator requirements.

Human-Interface Requirements: Requirements and constraints on the human-computer interface (controls, displays, aural alerts).

Hazard and Other System Analyses: Analysis results for system-level (emergent) properties such as safety or security. For the TCAS specification, a preliminary hazard analysis (including fault-tree analysis) is included and linked to other sections of this Level 1 specification such as the safety-critical design constraints, system limitations, related system requirements, etc. and also to design decisions based on the hazard analysis that appear in lower levels of the specification. Whenever changes are made to the system (during development or during maintenance and evolution),

the safety of any proposed change needs to be evaluated. This process can be difficult and expensive. By providing links throughout the levels of the intent specification, it should be possible to assess whether a particular design decision or piece of code was based on the hazard analysis or a safety-related design constraint before it is changed and to determine whether the change will affect safety.

Level 1 Verification and Validation: Each level contains information about required V&V procedures for the level as well as a record of the results. At this highest conceptual level, the V&V activity might involve reviews by various groups.

4 Level 2: System Design Principles

The second level of the specification contains the basic system design and scientific and engineering principles needed to achieve the behavior specified in the top level. It answers the question "why" for the design decisions in the level below and describes any basic principles or assumptions upon which the system design depends. It describes how the requirements above will be achieved and how the constraints will be enforced. The horizontal dimension again allows abstraction and refinement of the basic system design principles.

Information at this level may be specified using English or other types of engineering and mathematical notations such as differential equations. The information on this level should be specified using notations familiar to and commonly used by engineers. Assumptions and the rationale upon which the design features are predicated is again included throughout the Level 2 specification.

For TCAS, this level includes such general principles as the basic *tau* concept, which is related to all the high-level alerting goals and constraints:

2.1: *Each TCAS-equipped aircraft is surrounded by a protected volume of airspace. The boundaries of this volume are shaped by the tau and DMOD criteria.*

2.1.1: *TAU: In collision avoidance, time-to-go to the closest point of approach (CPA) is more important than distance-to-go to the CPA. Tau is an approximation of the time in seconds to CPA. Tau equals 3600 times the slant range in nmi, divided by the closing speed in knots.*

2.1.2: *DMOD: If the rate of closure is very low, a target could slip in very close without crossing the tau boundaries and triggering an advisory. In order to provide added protection against a possible maneuver or speed change by either aircraft, the tau boundaries are modified (called DMOD). DMOD varies depending on own aircraft's altitude regime. See Table 2.*

The principles are linked to the related higher level requirements, constraints, assumptions, limitations, and hazard analysis as well as linked to lower-level system

design and documentation. As stated, assumptions used in the formulation of the design principles should also be specified at this level. For example, the TCAS design has a built-in bias against generating advisories that would result in the aircraft crossing paths (called *altitude crossing advisories*).

2.36 *A bias against altitude crossing RAs is also used in situations involving intruder level-offs at least 600 feet above or below the TCAS aircraft. In such a situation, an altitude-crossing advisory is deferred if an intruder aircraft that is projected to cross own aircraft's altitude is more than 600 feet away vertically* ($\downarrow Alt\_Separation\_Test_{m-351}$).

> ASSUMPTION: *In most cases, the intruder will begin a level-off maneuver when it is more than 600 feet away and so should have a greatly reduced vertical rate by the time it is within 200 feet of its altitude clearance (thereby either not requiring an RA if it levels off more than* ZTHR[3] *feet away or requiring a non-crossing advisory for level-offs begun after* ZTHR *is crossed but before the 600 foot threshold is reached).*

The example above would include a pointer down to the part of the black box behavior specification on Level 3 ($Alt\_Separation\_Test$) that embodies the design principle.

As another example of the type of links that may be found between Level 2 and the levels above and below it, consider the following. TCAS II advisories may need to be inhibited because of an inadequate climb performance for the particular aircraft on which TCAS II is installed. The collision avoidance maneuvers posted as advisories (called RAs or Resolution Advisories) by TCAS II assume an aircraft's ability to safely achieve them. If it is likely they are beyond the capability of the aircraft, then TCAS II must know beforehand so it can change its strategy and issue an alternative advisory. The performance characteristics are provided to TCAS II through the aircraft interface. An example design principle (related to this problem) found on Level 2 of the intent specification is:

2.39: *Because of the limited number of inputs to TCAS for aircraft performance inhibits, in some instances where inhibiting RAs would be appropriate it is not possible to do so (↑L3). In these cases, TCAS may command maneuvers that may significantly reduce stall margins or result in stall warning (↑SC9.1). Conditions where this may occur include .... The aircraft flight manual or flight manual supplement should provide information concerning this aspect of TCAS so that flight crews may take appropriate action (↓ [Pilot procedures on Level 3 and Aircraft Flight Manual on Level 4).*

[3]The vertical dimension, called ZTHR, used to determine whether advisories should be issued varies from 750 to 950 feet, depending on the TCAS aircraft's altitude.

Finally, principles may reflect tradeoffs between higher-level goals and constraints. As examples:

2.2: *Tradeoffs must be made between necessary protection (G1) and unnecessary advisories (SC5). This is accomplished by controlling the sensitivity level, which controls the tau, and therefore the dimensions of the protected airspace around each TCAS-equipped aircraft. The greater the sensitivity level, the more protection is provided but the higher is the incidence of unnecessary alerts. Sensitivity level is determined by ...*

2.38: *The need to inhibit* CLIMB *RAs because of inadequate aircraft climb performance will increase the likelihood of TCAS II (a) issuing crossing maneuvers, which in turn increases the possibility that an RA may be thwarted by the intruder maneuvering (↑SC7.1, FTA-1150), (b) causing an increase in* DESCEND *RAs at low altitude (↑SC8.1), and (c) providing no RAs if below the descend inhibit level (1200 feet above ground level on takeoff and 1000 feet above ground level on approach).*

This level of the TCAS specification includes system design principles for the general TCAS-II system components, the surveillance and collision avoidance logic, performance monitoring, pilot tasks and procedures, and the pilot-TCAS interface.

Level 2 Verification and Validation: V&V of the Level 2 system design principles may involve simulations, experiments, engineering analyses, and other validation procedures.

5 Level 3: Blackbox Behavior

This level describes the blackbox behavior of the system components, including humans, the logical aspects of the interfaces between the components (detailed physical design is found at Level 4), and any assumed relevant environment behavior (e.g., performance requirements on equipment interacting with the system being specified). The description includes no internal component design information and behavior is described only in terms of externally visible variables, objects, and mathematical functions.

The environment description on Level 3 includes the assumed behavior of the external components (such as altimeters and transponders for TCAS), including perhaps failure behavior, upon which the correctness of the system design is predicated. It also contains a description of the interfaces and communication between the system and its environment.

Differing formats are appropriate for the various types of information contained at this level, but Safeware Tools include support for a specification language called

SpecTRM-RL used for modeling blackbox software requirements (blackbox software behavior). SpecTRM-RL models are executable and, because they have an underlying formal model, they are formally analyzable while still being reviewable and readable with minimal training. We are in the process of developing an operator procedure modeling language with the same underlying formal model although most likely a different representation or interface to users. By basing the specification on the same underlying model, operator procedures can potentially be executed and analyzed along with the other models of the system components. Various types of analysis tools for SpecTRM-RL specifications are under development or planned for the future.

This level does not contain information about physical design or implementation. Any of the components theoretically could be implemented using analog or digital technology although practical considerations will normally limit the implementation medium. The levels above (Level 1 and Level 2) will answer questions about the intent or purpose of the information in this level. Again, links are maintained to provide relational information.

The V&V section on this level will normally provide testing requirements for the logical blackbox behavior and other types of analysis (such as completeness analyses) and results. Because SpecTRM-RL models are executable, they can be subjected to dynamic analysis (testing), static analysis, as well as used in hardware-in-the-loop simulation. Thus the requirements can be tested and validated before the code is written.

Safeware Engineering Corporation and their university partners (MIT and the University of Minnesota) are developing techniques and tools for defining test coverage of blackbox requirements specifications and for generating test data from the Level 3 SpecTRM-RL models. In addition, if this level contains information about operator behavioral requirements (such as operational procedures and task analyses), then this section will also contain information about the validation of these procedures (e.g., use testing and simulation).

5.1 SpecTRM-RL Description

"Designing Specification Languages for Process Control Systems: Lessons Learned and Steps to the Future" (by Leveson, Heimdahl, and Reese) describes the rationale behind the design of SpecTRM-RL. Here we include only an informal description for users of the language. The formal syntax and semantic description will be provided in a separate document.

The design of SpecTRM-RL is greatly influenced by our desire to provide a combined specification and modeling language. Most formal modeling languages are too difficult to read to be used as the system specification language (which needs to be reviewed and used by people with a large variety of backgrounds and expertise). However, the practicalities of system development are such that rarely will there be resources to provide a separate modeling effort for the specification, and the con-

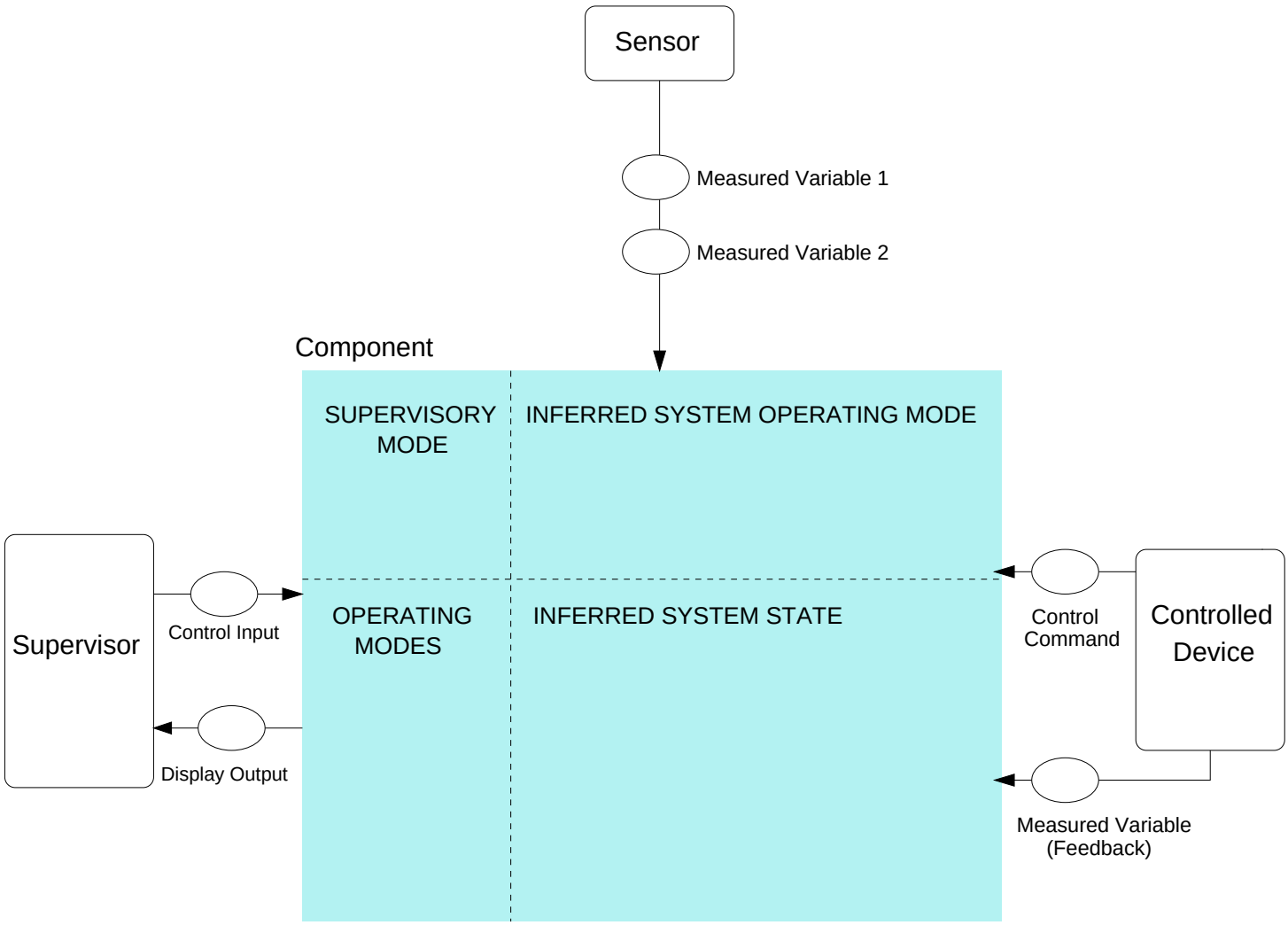


Figure 2: The Parts of a SpecTRM-RL Specification.

tinual changes common to most software development projects will require frequent updates to ensure that the formal model is consistent with the current requirements and system design. SpecTRM-RL was designed to satisfy both objectives: to be easily readable enough to serve as the official system specification of the behavioral requirements and to have an underlying formal model that can be executed and subjected to mathematical analysis.

A SpecTRM-RL specification is composed of four main parts (Figure 2): (1) a specification of the supervisory modes of the controller being modeled, (2) a specification of its operating modes (3) a model of the controlled process (or *plant* in control theory terminology) that includes the inferred operating modes and system state (these are inferred from the measured inputs), and (4) a specification of the inputs and outputs to the controller.

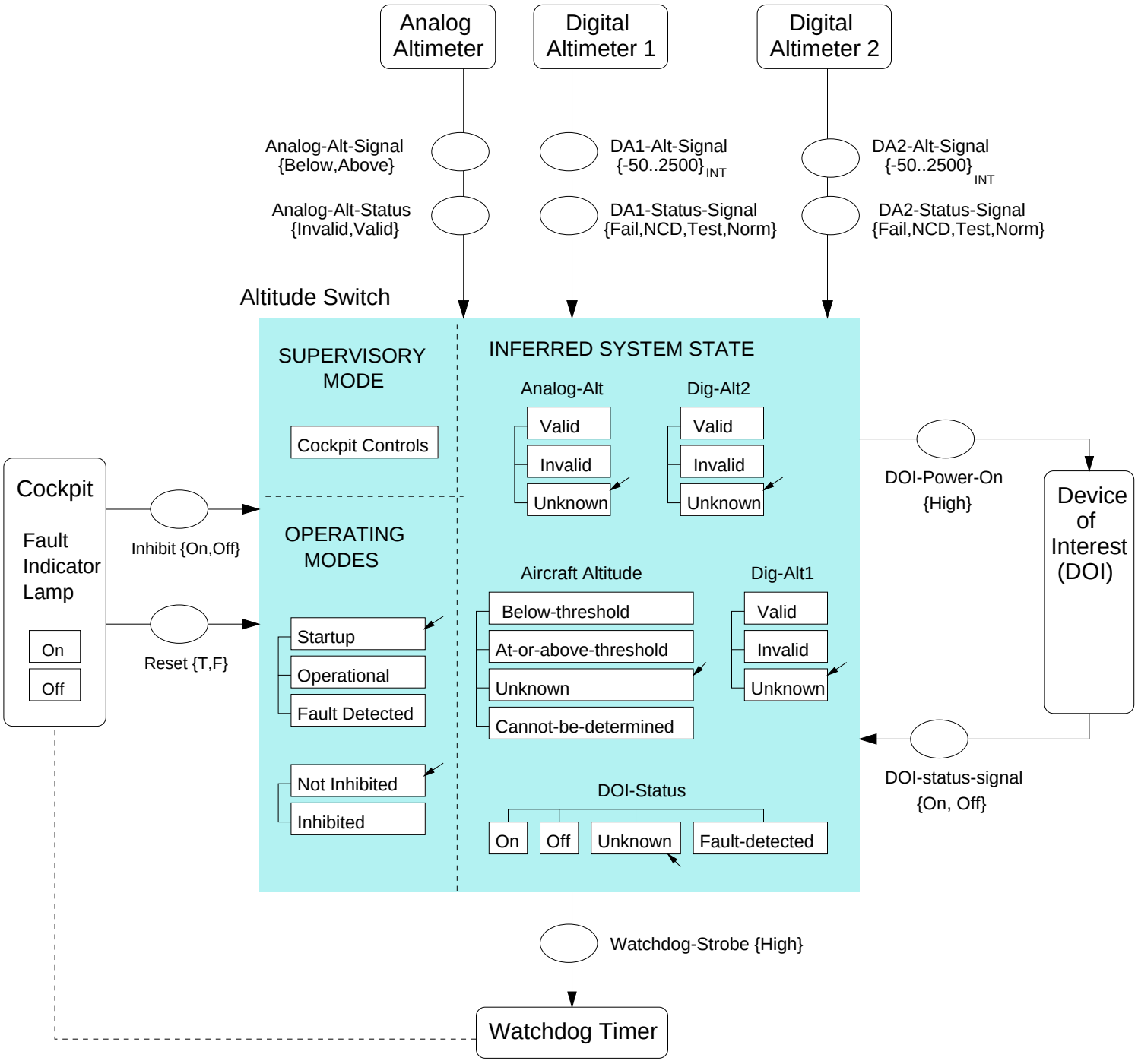


Figure 3: Example of the Graphical Model

Figure 3 shows the graphical part of the specification for a simple altitude switch. The graphical notation mimics the typical engineering drawing of a control loop. The supervisory interface is shown to the left of the main controller model. Every automated controller has at least two interfaces: one with the supervisor(s) that issues instructions to the automated controller (the supervisory interface) and one with each controlled system component (controlled system interface).

The supervisory interface consists of a model of the operator controls and a model of the displays or other means of communication by which the component relays information to the supervisor. Note that the interface models are simply the logical view that the controller has of the interfaces—the real state of the interface may be inconsistent with the assumed state due to various types of design flaws or failures. By separating the assumed interface from the real interface, we are able to model and analyze the effects of various types of errors and failures (e.g., communication errors or display hardware failures). In addition, separating the physical design of the interface from the logical design (required content) will facilitate changes and allow parallel development of the software and the interface design. During development, mockups of the physical screen or interface design can be generated and tested using the SpecTRM-RL simulator to generate outputs.

In the altitude switch example, there are two control inputs: a reset signal that has the value *true* or *false* and an inhibit button that inhibits operation of the altitude switch. The inhibit button can either be in the *on* or *off* position. The only display related to the altitude switch is a fault indicator lamp that can also either be *on* or *off*. There is only one supervisory mode—cockpit controlled—which is shown in the upper left quadrant of the component model.

The bottom left quadrant of Figure 3 provides information about the operating modes for the controller itself. These are not internal states of the altitude switch (which are not included in our specifications) but simply represent externally visible behavior about the controller's modes of operation. In the example, there are three modes: *startup*, *operational*, and internal-fault-detected (which will result in the fault indicator light being lit in the cockpit and cessation of activity until the reset button is pushed) as well as *inhibited* and *not inhibited*.

The right half of the controller model represents inferred information about the operating modes and states of the controlled system (the *plant* in control theory terminology). A simple plant model may include only a few relevant state variables. If the controlled process or component is complex, the model of the controlled process may itself be represented in terms of its operational modes and the states of its subcomponents. In a hierarchical control system, the controlled process may itself be a controller of another process. For example, the flight management system may be controlled by a pilot and may itself issue commands to a flight control computer, which issues commands to an engine controller. If, during the design process, components that already exist are used, then those plug-in component models could be inserted into the SpecTRM-RL process model. Parts of a SpecTRM-RL model are easily

reused or changed to represent different members of a product family.

In the altitude switch example, defining the control algorithm requires using information about the altitude level with respect to a given threshold, the inferred status of the DOI, and the validity of the altimeter information being provided.

Inputs representing the state of the plant (monitored or measured variables) are shown with arrows pointing to the controller. For the altitude switch, these variables provide the current status (on or off) of the device of interest (DOI) that the altitude switch turns on and inputs about the status and value of three altimeters on the aircraft (one analog and two digital) that provide information to the altitude switch about the current measured altitude of the aircraft as well as the status of that information (i.e., normal operation, test data, no computed data provided, or failed),

The output commands are denoted by outward pointing arrows. In the example, they include a signal to *power-on* the device (DOI) and a strobe to a watchdog timer so that proper action can be taken (by another system component) if the altitude switch fails. The outputs in this example are simple "high" signals on a wire or line to the device.

In the altitude switch example, defining the control algorithm requires using information about the altitude level with respect to a given threshold, the inferred status of the DOI, and the validity of the altimeter information being provided.

It is important to note that the internal design of the altitude switch is not included in the model. The altitude switch operating modes are externally visible (and must be known for the pilot to understand its operation) and the aircraft model is used to describe the externally visible behavior of the altitude switch in terms of the process being controlled (and not in terms of its own internal data structures and algorithms).

Because of the simplicity of the altitude switch example, there are a few general aspects of a SpecTRM-RL model that are not included, all involving the ability to specify modes. Modes are abstractions on states and are not necessary for defining blackbox behavior. They are useful, however, in understanding or explaining the behavior of complex systems. SpecTRM-RL allows specifying several additional types of modes: supervisory modes, control modes, and controlled-system operating modes, and display modes.

Supervisory modes are useful when a component may have multiple supervisors at any time. For example, a flight control computer in an aircraft may get inputs from the flight management computer or directly from the pilot. Required behavior may differ depending on what supervisory mode is currently in effect. Mode-awareness errors related to confusion in coordination between multiple supervisors can be defined (and the potential for such errors theoretically identified from the models) in terms of these supervisory modes.

Control Modes control the behavior of the controller itself. They may be used to control the interpretation of the component's interfaces or to describe the component's required process-control behavior.

A third type of mode, *controlled-system* or plant operating modes, can be used

to specify sets of related behaviors of the controlled system (plant) model. They are used to indicate its operational status. For example, it may be helpful to define the operational state of an aircraft in terms of it being in takeoff, climb, cruise, descent, or landing mode.

In systems with complex displays (such as Air Traffic Control systems), it may also be useful to define various *display modes*.

The SpecTRM-RL language contains a graphical notation, output message specifications, mode and state variable definitions, operator interface control inputs and display outputs, input variable definitions, output variable definitions, macros, and functions. Each of these features is described, in turn, below.

5.1.1 Graphical Notation

The SpecTRM-RL notation is driven by the intended use of the language to define a blackbox function from outputs to inputs. SpecTRM-RL has a greatly simplified graphical representation (compared to RSML or Statecharts), which is made possible because we eliminated the types of state machine complexity necessary for specifying component design but not necessary to specify the input/output function computed in a pure blackbox requirements specification. Some features are also the result of wanting to provide a useful interface for the designer when the model is being executed.

Input and output variables are denoted by ovals. We use this notation because the SpecTRM-RL simulator allows the designer to step through executions of the models. Currently the active state and mode values light up, and we hope eventually to have the current value of each input and output variable displayed in the oval (the tools currently show these in a different window). Next to the oval is the name of the input or output variable and a description of its possible contents.

State values in square boxes represent inferred values. Such variables are necessarily discrete in value[4], and thus can be represented as a state variable with a finite number of possible values. In practice, such state variables almost always have only a few relevant values (e.g., altitude below a threshold, altitude at or above a threshold, cannot-be-determined, and unknown). Values for state variables in the plant model are required in SpecTRM-RL to include an "unknown" value. The meaning and purpose of the unknown state are described below.

The possible values for a state variable are shown with a line connecting the boxes. The line simply denotes that the values are disjoint, that is, the variable may assume only one value at a time. A small arrow pointing at a box denotes the default (startup) value for the state variable or mode.

[4]If they are not discrete, then they are not used in the control of the function computation but in the computation itself and can simply be represented in the specification by arithmetic expressions involving input variables.

5.1.2 Output Message Specification

Everything starts from outputs in SpecTRM-RL. By starting from the output specification, the specification reader can determine what inputs trigger that output and the relationship between the inputs and outputs. Other state-machine specification languages, such as RSML or Statecharts, do not explicitly show this relationship (although it can be determined, with some effort, by examining the specification).

The general format of an output specification is shown in the appendix and an example specification for the altitude switch example in Figure 4. Some of the notation (particularly that at the top of the specification) is included in order to assist with building complete specifications and stem from our description of criteria for complete process-control specifications (see Chapter 15, "Software Hazard and Requirements Analysis" in *Safeware* by Leveson).

For output variables, the following information can and should be included: destination of the output, acceptable values, any initiation delay or completion deadline along with any required exception-handling behavior if the deadlines cannot be met, feedback information about how the controller will determine that the output command has been successfully implemented (see "Output to Trigger Event Relationships" in Chapter 15 of *Safeware*), and any other output commands that reverse this output. In many systems, it is important to indicate a maximum time in which the output command remains effective before it is executed. After this time, the output essentially "times out" and should not be executed. This information can either be provided in the output message (if the timeout is implemented by the output command actuator) or included in the reversal information if the controller must issue a reversal to undo the command's effect. Reversal information is also useful in identifying accidentally omitted behavior from the specification, i.e., most output actions to provide a change in the controlled plant or supervisory interface have complementary actions to undo that change. References are pointers to other levels of the intent specification related to this part of the specification.

The conditions under which an output is triggered (sent) is simply a predicate logic statement over the various states, variables, and modes in the specification. From our experience in specifying complex systems, we have found that the triggering conditions required to accurately capture the requirements are often extremely complex. The propositional logic notation traditionally used in computer science notations to define these conditions did not scale well to complex expressions and quickly became unreadable (and error prone). To overcome this problem, we use a tabular representation of disjunctive normal form (DNF) that we call AND/OR tables[5].

[5]For those familiar with other state-machine specification languages that use tables, such as SCR, it is useful to note that tables are used very differently here. In such languages, the actual transitions between states are described in a table. We do not do this in SpecTRM-RL; instead, the tables are used simply to represent one predicate logic statement about the conditions on one transition arrow between states. Therefore our tables are of much more limited size and their use scales up to very large and complex system specifications while still remaining relatively small.

The far-left column of the AND/OR table lists the logical phrases of the predicate. Each of the other columns is a conjunction of those phrases and contains the logical values of the expressions. If one of the columns evalutes to *true*, then the entire table evaluates to *true*. A column evaluates to *true* if all of its elements match the truth values of the associated predicates. A dot or empty square denotes "don't care."

The tables are divided into two parts: an upper part denoting the relevant operating modes of the controller for that column and a lower part describing any additional conditions for triggering the output. We have found that this separation assists in completeness checking, particularly when humans are writing and reviewing specifications.

For the altitude switch power-on output in the example shown in Figure 4, the output is triggered (sent) when all the following conditions are true: the altitude is below the threshold, the DOI is off, the altitude switch is operational, and the previous altitude was at or above the threshold (the requirements for the altitude switch say that if the switch is turned off while the aircraft is below the threshold altitude, the DOI is not powered on again until the aircraft is above the threshold altitude and again passes down through it). The *Prev* built-in function allows referring to previous values of state variables and modes, inputs, or outputs.

Subscripts will be used in future versions of SpecTRM-RL to denote whether a variable in the table is an input variable (i), a state variable (s), a macro (m) or a function (f). A number will be attached to the subscript to indicate the page on which the input variable, state variable, macro, or function is defined. The current tool does not yet support this feature, although this information can be easily determined by a reader of the specification.

Every output command column must include a reference to the operating mode(s) under which the command is sent. It is assumed that if not specified, then the output cannot occur in that mode.

5.1.3 Constants

The specification of constant values represents a tradeoff design decision in most specification languages. There are two choices: (1) use the constant value everywhere it is referenced and when changes are made all these locations must be identified and updated or (2) assign an identifier name to the constant, and use that identifier (instead of the constant) throughout the specification and have the reader look up the current value in a table somewhere.

If the constant values, such as the altitude threshold, are included directly in the specification (rather than included in a table somewhere other than where it is used), the specification will be easier to read and review because the reviewer will not need to continually flip to another page where the constant is defined when reading the specification. On the other hand, changing the value of the constant when the requirements change can then be tricky and error prone. Using a table simplifies the

Output Command

DOI-Power-On

Destination: DOI

Acceptable Values: {high}

Initiation Delay: 0 milliseconds

Completion Deadline: 50 milliseconds

Exception-Handling:

Feedback Information:

Variables: DOI-status-signal

Values: high (on)

Relationship: Should be on if ASW sent signal to turn on

Min. time (latency): 2 seconds

Max. time: 4 seconds

Exception Handling: DOI-Status changed to Fault-Detected

Reversed By: Turned off by some other component or components. Do not know which ones.

Comments:

References: ↑ 2.4.5 ↓ 4.7

CONTENTS

= discrete signal on line PWR set to high

TRIGGERING CONDITION

Operating Mode	Operational	T
	Not Inhibited	T
State Values	DOI-Status = On	F
	Altitude = Below-threshhold	T
	Prev(Altitude) = At-or-above-threshold	T

Figure 4: Example of an Output Specification

updating problem because the change need only be made in one place.

We provide a compromise solution to this specification language design problem that satisfies both the readability and changability requirements. The constant itself is used throughout the specification with a subscript that attaches an identifier to it. The constant identifiers are maintained in a table in the specification, but the reviewer need not refer to this table (it is only included for convenience). When a constant is changed, for example, an altitude threshold, the tools can automatically search for instances in the specification and update them all.

5.1.4 Input Variable Definition

Our desire to enforce completeness in the language itself leads to language features that allow the inclusion of information (if relevant) about arrival rates, exceptional condition handling, data age requirements, etc. No input data is good forever; after some point in time it becomes obsolete and should not be used. We provide a special value, *obsolete*, that any input variable can assume a specified time after it is received.

The specification of control and display inputs and outputs are similar to inputs and outputs from the controlled process. The general forms are shown in the appendix and an example in Figure 5. In the example, the definition essentially says that the value comes from the *altitude* field in the DA1-message and is assigned when a message arrives. If no message has arrived in the past 2 seconds, the previous value is used. If the last message arrived more than 2 seconds before, the data is considered obsolete. The input variable also starts with the obsolete (undefined) value. Because of the similarity of the form of most input definitions, we may simplify the notation in the future.

Supervisory modes may need to be specified for input values to determine which inputs to use at any time.

5.1.5 State Variable Definition

State variable values are inferred from the values of input variables or from other state variable values. Figure 6 shows a partial example of a state variable description for the altitude switch.

SpecTRM-RL requires all state variables that describe the process state to include an *unknown* value. This value is the default value upon startup or upon specific mode transitions (for example, after temporary shutdown of the computer). This feature is used to ensure consistency between the computer model of the process state and the real process state upon startup or after leaving control modes where processing of inputs has been interrupted. By making Unknown the default state value and by changing to the unknown value upon changes to control modes where inputs are not being process, the use of an unknown state value forces resynchronization of the model with the outside world after an interruption in processing inputs. Many accidents have been caused by the assumption that the process state does not change

Input Value

DA1-Alt-Signal

Source: Digital Altimeter 1

Type: integer

Possible Values (Expected Range): -20..2500

Exception-Handling: Values below -20 are treated as -20 and values above 2500 as 2500

Units: feet AGL

Granularity: 1 foot

Arrival Rate (Load): one per second average

Min-Time-Between-Inputs: 100 milliseconds
Max-Time-Between-Inputs: none

Obsolescence: 2 seconds

Exception-Handling:

Description:

Comments:

References: ↑2.11 ↓4.2

Appears in: Altitude

DEFINITION

= FIELD (Altitude in DA1-Message)

Receive DA1-Message FROM Digital-altimeter-1	T

= PREV (DA1-Alt-Signal)

Receive DA1-Message FROM Digital-altimeter-1	F
Time > Time (DA1-Message arrived) + 2 seconds	F

= Obsolete

Receive DA1-Message FROM Digital-Altimeter-1	F	
Time > Time (DA1-Message arrived) + 2 seconds	T	
Startup		T

Figure 5: Example of an Input Specification

State Value

Altitude

Obsolescence: 2 seconds

Exception-Handling: Because the altitude-status-signals change to obsolete after 2 seconds, altitude will change to Unknown if all input signals are lost for 2 seconds.

Description: When at least one altimeter reports an altitude below the threshold, then the aircraft is assumed to be below the threshold. ↑2.12.1

Comments:

References: ↑2.12 ↓4.10

Appears in: DOI-Power-On

DEFINITION

= Unknown

Startup	T		
Controls.Reset = T		T	
Analog-ALT = Unknown			T
Dig-Alt1 = Unknown			T
Dig-Alt2 = Unknown			T

The altitude is assumed to be unknown at startup, when the pilot issues a reset command, and when no recent input has come from any altimeter.

= Below-threshold

Analog-Valid-and-Below$_m$	T		
Dig1-Valid-and-Below$_m$		T	
Dig2-Valid-and-Below$_m$			T

At least one altimeter reports a valid altitude below the threshold..

= At-or-above-threshold

Analog-Valid-and-Above$_m$	T	T	T	F	T	F	F
Dig1-Valid-and-Above$_m$	T	T	F	T	F	T	F
Dig2-Valid-and-Above$_m$	T	F	T	T	F	F	T

At least one altimeter reports a valid altitude above the threshold and none below.

= Cannot-be-determined

Analog-Alt = Invalid	T
Dig-Alt1 = Invalid	T
Dig-Alt2 = Invalid	T

No valid data is received from any altimeter (all report test or failed status).

Figure 6: Example of an Inferred State Variable Specification

Macro

Dig1-Valid-and-Below

Description:

Comments:

References: ↑ 2.5.1 ↓ 4.3, 4.8.2

Appears in: Altitude

DEFINITION

Dig1-alt = Valid	T
DA1-Alt-Signal < 2000_{THRES}	T

Figure 7: Example of a Macro Specification

while the computer is idle or by incorrect assumptions about the initial value of state variables on startup or restart.

If a model of a supervisory display is included in the specification, the unknown is used for state variables in the supervisory display model only if the state of the display can change independently of the software. Otherwise, such variables must specify an initial value (e.g., blank, zero, etc.) that should be sent when the computer is restarted.

5.1.6 Macros and Functions

Macros are simply named pieces of AND/OR tables that can be referenced from within another table. For example, the macro in Figure 7 is used in the definition of the state variable *altitude* in the altitude switch example. Its use is not necessary, but simplifies the specification of altitude and thus makes it easier to understand while also allowing easy changes or reuse. Macros, for the most part, correspond to typical abstractions used by application experts in describing the requirements and therefore add to the understandability of the specification. In addition, we found this a convenient feature for expressing hierarchical abstraction and enhancing hierarchical review and understanding of the specification. For very complex models (e.g., a flight management system), we have found that macros are almost required for humans to be able to handle the complexity involved in constructing the specification.

Rather than including complex mathematical functions directly in the transition

tables, functions may be specified separately and referenced in the tables.

The macros and functions, as well as other features of SpecTRM-RL not only help structure a model for readability; they also help organize models to enable specification reuse. Conditions commonly used in the application domain can be captured in macros and common functions can be captured in reusable functions. Naturally, to accomplish reuse, care has to be taken when creating the original model to determine what parts are likely to change and to modularize these parts so that substitutions can be easily made.

6 Hints on Developing SpecTRM-RL Models

We have found that users of requirements modeling languages often use these languages to describe internal component design, which makes the specifications very hard for experts to review. SpecTRM-RL was designed specifically to avoid that problem and to provide assistance in writing blackbox requirements specifications, but the process can seem overwhelming when the specifier is faced with a complex system and a blank sheet of paper. This section includes techniques that I find useful in building SpecTRM-RL models. There is no "correct" way to solve a problem, of course, and others may find other procedures easier for them. I provide these guidelines simply for beginners who need help getting started or for those having trouble devising their own heuristics.

Any software requirements and blackbox behavioral modeling should start from requirements. I find that the latest design techniques that start with defining objects and only later consider functional requirements result in designs that are unnecessarily complicated, inefficient, and difficult to review and validate (with respect to system requirements). I suspect that they actually exacerbate the difficult problems associated with changing requirements (which they are supposed to assist with), but I have no evidence of that. While such object-oriented approaches may well be the best for general information and business systems, I believe that starting from required functions rather than objects is the most appropriate approach for specifying and designing embedded control systems.

1. Start with a blank graphical model. Fill in obvious components but don't worry if there is not much there at first. It will be filled in as the specification process proceeds.

2. Next start with the outputs. As Parnas has said, if inputs are not used to determine outputs, nobody cares whether you read them or not. The required outputs can be determined from the functional requirements. If, for example, you are designing a mobility and positioning system for a robot, the functional requirements will lead you to quickly determine that the software must provide movement and manuevering commands.

3. Determine the triggering conditions for each output, i.e., those conditions under which a particular output must be produced. You will probably find that you miss some conditions at first and will need to return to the output specifications to add conditions that were omitted in the early attempts to build the model. Consider each of your identified operating modes and whether the value of the output will depend on the mode.

4. Specifying the triggering conditions will lead to recognizing inferred system states and input values that are needed, and these should be added to the graphical state model as they are identified.

5. Working backward to the inferred states and thence to inputs will identify additional inputs and inferred states necessary to describe the required behavior.

6. Take the completeness criteria (the tools will in the future assist with this process) and make sure each criterion is satisfied in the specification. A checklist and how to use it with SpecTRM-RL specifications will be provided soon and tools to assist with this process are also planned.

7. The final step is to get the specification reviewed by others. Visualization tools are planned to assist with understanding and reviewing the specification.

7 Level 4: Physical Design Representations

This level of an intent specification contains the normal physical design representations with links to the level above. The contents will depend on whether the particular function is being implemented using analog or digital devices and what it is. The design intent information may not all be completely linked and traceable upward to the levels above—for example, design decisions not based on higher level requirements or constraints, such as the use of a particular graphics package because the programmers are familiar with it or it is easy to learn. Knowing that these decisions are *not* linked to higher level purpose is important during software maintenance and evolution activities.

We currently have no specific suggestions for formats. Our sample TCAS-II specification contains the Human–Computer Interface Specifications, the Aircraft Flight Manual, Software Design Specifications, Hardware Design Specifications, and Verification (Testing) Requirements. It is likely that the information will actually be kept in separate documents and the specification simply contain appropriate pointers. The Safeware tools allow on-line specifications to contain hyperlinks so that accessing information from different documents is simplified.

8 Level 5: Physical Implementation

This level of the specification contains the actual software, hardware assembly and installation instructions, training requirements or training plan, etc. Again, the information will most likely be kept in several different physical documents.

9 Appendices

Various types of information may be included in appendices. For TCAS, we included reference algorithms, physical measurement conventions, a glossary, and an index.

10 Appendix: Format of SpecTRM-RL Components

Output Command

Name

Destination:

Acceptable Values:

Units:

Granularity:

Exception Handling:

Hazardous Values:

Timing Behavior:

Initiation Delay:

Completion Deadline:

Output Capacity Assumptions: (rate at which actuators can accept and react to data produced by component)

Load: (number of outputs N within an interval of time D)

Min time between outputs:

Max time between outputs:

Hazardous timing behavior:

Exception-Handling:

Feedback Information:

Variables:

Values:

Relationship:

Min. time (latency):

Max. time:

Exception Handling:

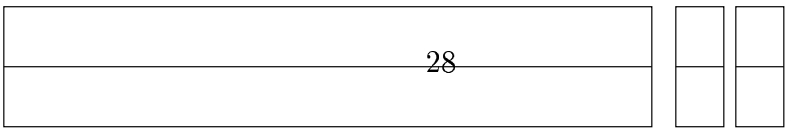
Reversed By:

Comments:

References:

DEFINITION

= ...



=

Operating Mode

Name

Description:

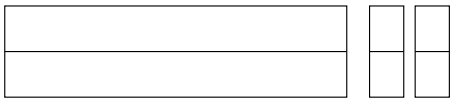
Comments:

References:

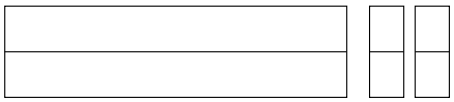
Appears in:

DEFINITION

= ...



= ...



.

Figure 9: General Form of a Operating Modes Specification

State Value

Name

Obsolescence:

Exception-Handling:

Related inputs:

Description:

Comments:

References:

Appears in:

Related Inputs: (so can check that behavior specified for all values of related inputs, assumes stays in same state for other inputs)

DEFINITION

= Unknown

Startup	T	
...		

This table must include all conditions under which the value changes to unknown, such as startup, timeouts, transitions to particular modes or states such as maintenance or off-line mode, ...

= ...

Figure 10: General Form of an Inferred State Variable Specification

Input Value

Name

Source:

Type:

Possible Values (Expected Range):

Units: (the applicability of these will depend on the value type)
Granularity:
Exception-Handling:

Timing Behavior: (continuous, periodic, random, s-r)

Load: (number of inputs N within an interval of time D)
Min-Time-Between-Inputs:
Max-Time-Between-Inputs:
Max-Time-Before-First-Input:
Related Outputs: (if s-r)
Latency : Min-time-after-output
Min-time-after-output
Exception-Handling:

Obsolescence:
Exception-Handling:

Description:

Comments:

References: ↑ ↓

Appears in:

DEFINITION

= FIELD (...)

Receive xx-Message from ...	T	Event that triggers change

= PREV (..)

Receive xx-message from ...	F	Conditions under which previous value used.
(timing statement)		

= Obsolete

Startup	T	
Receive xx-message from ...		F
(timing statement)		T

Figure 11: General Form of an Input Specification

Control Input

Name

I'M NOT SURE ABOUT THE FORM OF THIS YET

Source:

Type:

Possible Values (Expected Range):

Units:

Granularity:

Exception-Handling:

Timing Behavior:

Load: (number of inputs N within an interval of time D)

Min-Time-Between-Inputs:

Max-Time-Between-Inputs:

Exception-Handling:

Obsolescence:

Exception-Handling:

Description:

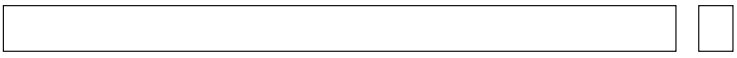
Comments:

References: ↑ ↓

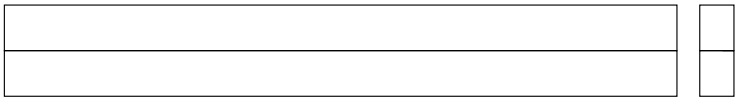
Appears in:

DEFINITION

= FIELD (...)



= PREV (..)



= Obsolete

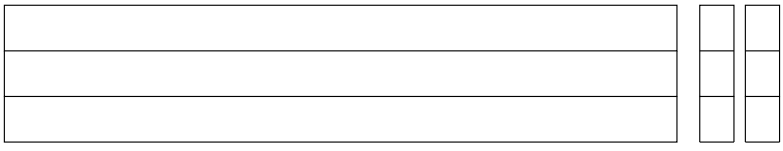


Figure 12: General Form of a Control Input Specification

Display Output

Name

I'M NOT SURE ABOUT THE FORM OF THIS YET

Destination:

Type: (discrete values, queue)

Acceptable Values:

Units:
Granularity:
Hazardous Values:

Update Requirements:

Update Delay:
Update Completion Deadline:
Output Capacity Assumptions: (rate at which humans can accept and react to data produced by component)

Update Load: (number of outputs N within an interval of time D)
Min update rate:
Max update rate:

Deletion Requirements (including data age):
Hazardous timing behavior:
Exception-Handling:

Failure Indication: (usually blank or message)

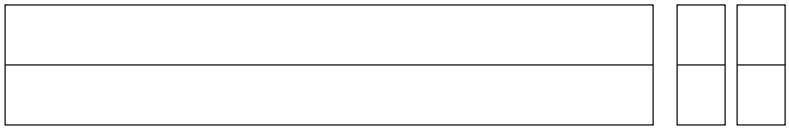
Reversed By:

Comments:

References:

DEFINITION

= ...



= ...

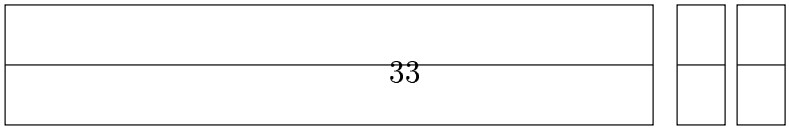


Figure 13: General Form of a Display Output Specification

Macro

Name

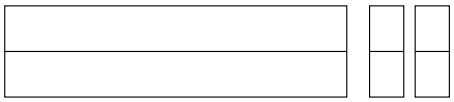
Description:

Comments:

References:

Appears in:

DEFINITION



.

Figure 14: General Form of a Macro Specification