An Approach to Designing Safe Embedded Software*

Nancy G. Leveson

Massachusetts Institute of Technology, Room 33-313, Cambridge MA 02139, USA
leveson@mit.edu,
http://sunnyday.mit.edu

Abstract. The complexity of most embedded software limits our ability to assure safety after the fact, e.g., by testing or formal verification of code. Instead, to achieve high confidence in safety requires considering it from the start of system development and designing the software to reduce the potential for hazardous behavior. An approach to building safety into embedded software will be described that integrates system hazard analysis, user task analysis, traceability, and informal specifications combined with executable and analyzable models. The approach has been shown to be feasible and practical by applying it to complex systems experimentally and by its use on real projects.

1 Introduction

Embedded systems often involve control over processes that are potentially dangerous or could involve large losses (including loss of the system itself). Safety and human factors are often considered at too late a stage in embedded system development to have adequate impact on system design: It has been estimated that 70-90% of the decisions relevant to safety are made in the early conceptual design stages of a project [Joh80]. Relying on after-the-fact safety assessment emphasizes creating an assessment model that proves the completed design is safe rather than constructing a design that eliminates or mitigates hazards. Too often, after-the-fact safety assessment leads to adjusting the model until it provides the desired answer rather than to improving the design.

In the same way, when the human role in the system is considered after the basic automation is designed, the choices to ensure usability and safety in human–computer interaction are limited to interface design, training, and human adaptation to the newly constructed automation. This approach has been labeled *technology-centered design* and has been accused of leading to "clumsy" automation [WCK91] and to new types of accidents in high-tech systems, such as new fly-by-wire aircraft [SW95]. Most of these accidents have been blamed on pilot error but more accurately can be described as flaws in the overall system and software design. Automation has the potential to overcome human perceptual and cognitive limits and to reduce or eliminate specific common human

* This work was partially supported by NSF ITR Grant xxx

errors. At the same time, it also has the potential for leading to accidents if not designed correctly.

This paper describes an integrated safety and human-centered approach to developing software-intensive systems along with some specification, modeling and analysis tools for implementing it. SpecTRM (Specification Tools and Requirements Methodology) is both a methodology and supporting toolset for building embedded, software-intensive, safety-critical systems that focuses on the system engineering aspects of software and the development of safe and correct requirements. Most of the focus in computer science research has been on the implementation of requirements with much less work on the system-level aspects of embedded software development, i.e., how the software will interact with the other components in the system. Perhaps this lack of emphasis in research and tools is why so many problems seem to arise at this interface. To reduce communication problems, industry has developed Integrated Product Teams, where the software developers work closely with the system engineers. Simply putting people together on a team, however, is not adequate to solve the problem—common models and ways to communicate about system design issues are necessary for effective communication to take place.

SpecTRM is based on the principle that critical properties must be designed into a system from the start. As a result, it integrates safety analysis, functional decomposition and allocation, and human factors from the beginning of the system development process. Because neither formal or informal specifications alone are adequate to develop embedded software, SpecTRM uses both to accumulate the information needed to make tradeoff and design decisions and to ensure that desired system qualities are satisfied early in the design process when changes are easier and less costly. Because almost all accidents related to software have involved requirements errors and not coding or implementation errors, requirements specification and validation is emphasized.

The methodology is supported by a new specification structuring approach, called Intent Specifications, that supports traceability and documentation of design rationale as the development process proceeds. While most of the information specified is not written in a formal language (and does not need to be), some parts of the development process can benefit greatly from having formal, analyzable models. At the same time, most errors in requirements specifications will be found by application experts who understand the engineering and other requirements on the system. The formal modeling language, SpecTRM-RL (SpecTRM Requirements Language), was designed with readability as a primary criterion and therefore we believe the models that result can be used as an unambiguous communication medium among the system developers and software implementers.

The next section outlines the overall methodology and describes the general goals behind intent specifications. Then the types of information specified at each level of an intent specification is described along with associated analyses.

2 The SpecTRM Approach to Designing Embedded Systems

The steps of the basic system engineering approach underlying SpecTRM are shown in Figure 1. The middle column represents the general engineering tasks. The right column shows special safety engineering activities and those in the left column represent human factors engineering. The separation is only shown to emphasize the safety and human factors activities; in any real project, they should be tightly integrated with the general engineering tasks but they are often separated—with unfortunate results. The process also involves more iteration and feedback than shown. While it may look like the process implies a pure waterfall model, the steps could be embedded in other lifecycle processes. Performing the later steps before the earlier ones, however, may result in a lot of backtracking and wasted effort or result in unsafe and difficult to use systems. The life cycle appropriate for embedded systems is very different than one appropriate for developing office software.

The steps of the process are supported by a new specification approach called Intent Specifications [Lev00a] and automated tools to assist with model construction, recording of design rationale, and traceability. The models are important in evaluating designs while design rationale and traceability are critical for certifiability and maintainability.

Embedded software will evolve and change continually throughout its life. Maintaining safety in such a changing environment requires high-quality specifications that include detailed descriptions of the externally visible behavior of the existing components as well as the rationale for the system design choices. The integration of new components must be based on the design and constraints of existing components and the surrounding environment, and any changes to the current system must be analyzed for their effect on system requirements, operator tasks and responsibilities, safety constraints, and human factors.

Determining whether a requirements, design, or implementation change has a potential effect on system safety also requires a level of traceability not normally found in specifications. Although such traceability implies more planning and specification effort at the beginning of a project, the effort will allow changes to be made much more quickly and easily. It could be prohibitively expensive, for example, to generate a new hazard and safety assessment for each system change that is proposed. Being able to trace a particular design feature or implementation item to the original hazard analysis will allows decisions to be made about whether and how that feature or code can be changed. The same is true for changes that affect operator activities and basic task allocation and usability principles. In some regulated industries, traceability is required by the certification authorities.

Intent Specifications organize system specifications not only in terms of the usual *what* and *how* (using refinement and part-whole abstractions), but also in terms of *why* (using intent abstraction) and integrate traceability and design rationale into the basic specification structure. They include both natural language and formal executable models. The design of Intent Specifications is

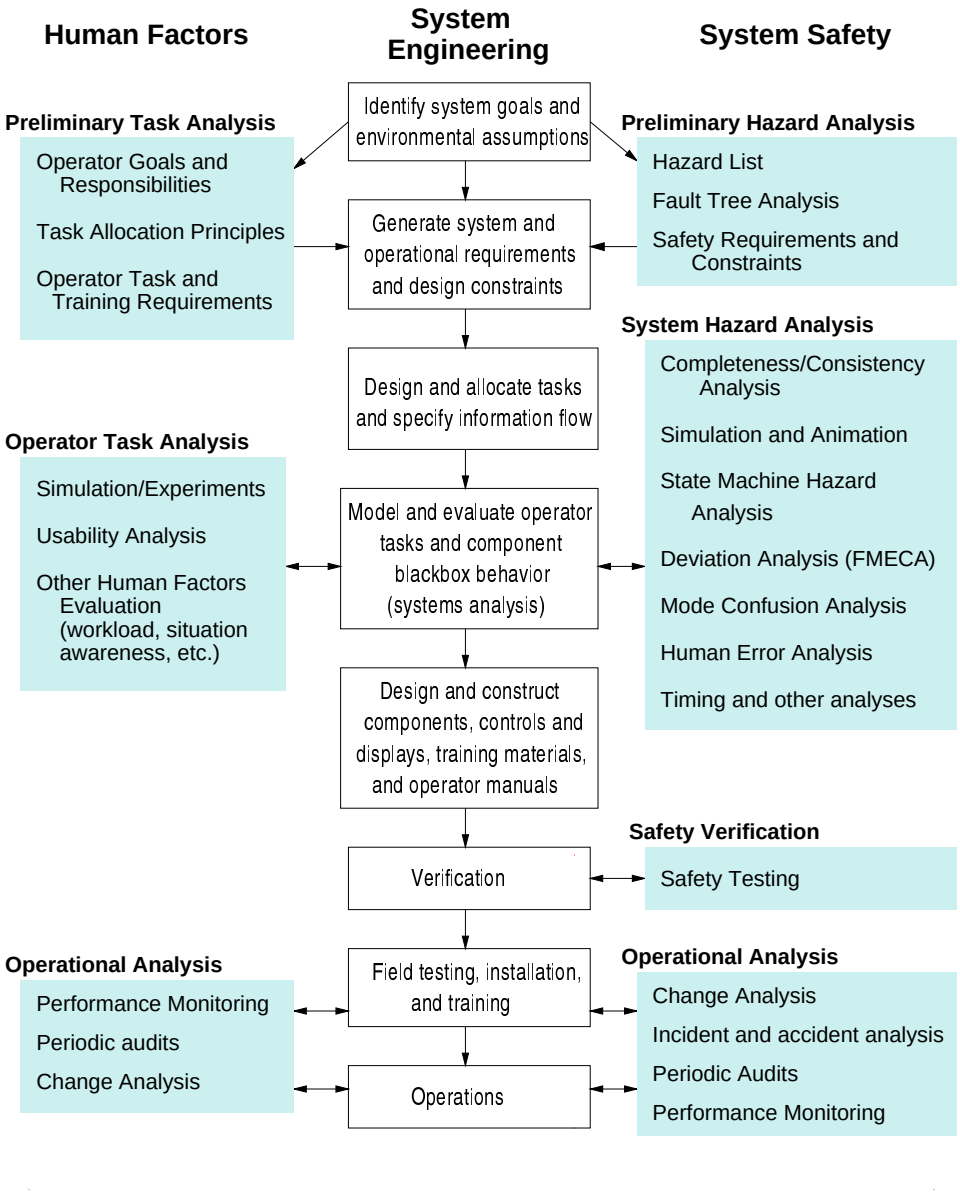


Fig. 1. A Safety and Human-Centered Approach to Building Embedded Systems

based on fundamental research on human problem-solving, systems theory, and basic system engineering. For a description of the systems theory and cognitive psychology research underlying Intent Specifications, see [Lev00a].

There are seven levels in an Intent Specification, each supporting a different type of reasoning about the system. A level does not represent refinement of the information at the level above, but instead contains a different view of the system. At the same time, the levels do not contain redundant information, but (as in basic system theory) instead higher levels represent constraints on the levels below. Each level also includes information about the verification and validation of the system model at that level. By organizing the specification in this way and linking the information at each level to the relevant information at the next higher and lower level, higher-level purpose or intent, i.e., the rationale for design decisoins, can be determined. In addition, by integrating and linking the system, software, human task, and interface design and development into one specification framework, intent specifications support an integrated rather than stovepiped approach to system design.

Curtis *et.al.* [CKI88] did a field study of the software requirements and design process for 17 large systems. One of the characteristics they found that appeared to set exceptional designers apart from their colleagues was the ability to map between the behavior required of the application system and the computational structures that implement this behavior. The most successful designers understood the application domain and were adept at identifying unstated requirements, constraints, or exception conditions and mapping between these and the computational structures. A goal of Intent Specifications is to support and foster this understanding.

The seven levels of an intent specifiation are shown in Figure 2. The top level represents the management view of the project and the second the customer view. The third level is the system engineering view while the fourth level represents the interaction between system and software engineers. The fifth and sixth levels contain the usual information developed and used during software development. The lowest level supports an operational view of the system. Note that the ordering of the levels does not imply an ordering of the activities. Most projects involve both top-down and bottom-up development, and the pieces of the Intent Specifications will be filled in as the related activities are performed. The only requirement is that at the end of the development phase, the intent specification is complete enough to support system maintenance and evolution.

In the following description of the information at each level, parts of an example specification we produced for an Air Traffic Control Conflict Detection Tool (called Mid Term Conflict Detection or MTCD) are used. MTCD is currently being evaluated by Eurocontrol for possible use in the European airspace.

Level 0: Management Perspective

The top level of an intent specification contains management plans, system safety plans, and other planning documents. Links to lower levels allow management to maintain a view of the status and results of the process.

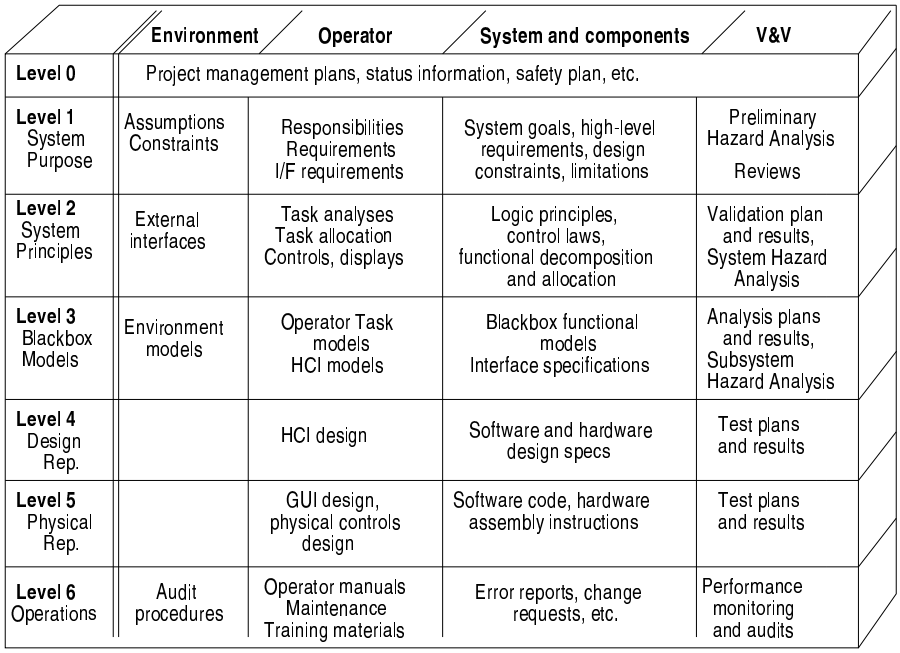


Fig. 2. The Structure of an Intent Specification

Level 1: Conceptual Design

One of the first steps in embedded software development is to identify high-level functional goals and the assumptions and constraints on the software design arising from the environment in which it will be used. For example, two high-level goals for MTCD are:

G1: To provide a conflict detection capability to air traffic controllers for all flights in the area of operation.
G2: To help keep the workload of the controllers within acceptable and safe limits despite an expected increase in traffic.

The success of any embedded software will rest on how well it fits within the larger system. For example, the ATC system within which MTCD fits consists of the conflict detection function itself, planning and tactical air traffic controllers for the sector, and the human–machine interface. MTCD interacts directly with the real-time flight data processing system, the environment data processing system, and a recording function, and indirectly with various automated decision aids, such as an arrival sequencing manager and monitoring aids. Two example assumptions about the interaction of MTCD with the real-time flight data processing system (FDPS) are:

Env-As-FDPS-01: FDPS will provide MTCD with system trajectories for all eligible flights.

Env-AS-FCPS-03: FDPS will inform MTCD when a flight leaves the area of operation.

Because we believe system design must consider human factors and system safety from the very beginning in order to achieve high levels of usability and safety, the first steps in the methodology involve a preliminary hazard analysis (PHA) and a preliminary controller task analysis (PTA). The PHA starts from agreed upon system hazards, such as violation of minimum separation between aircraft or entry of an aircraft into a restricted area, and identifies system behavior that could lead to those hazards.

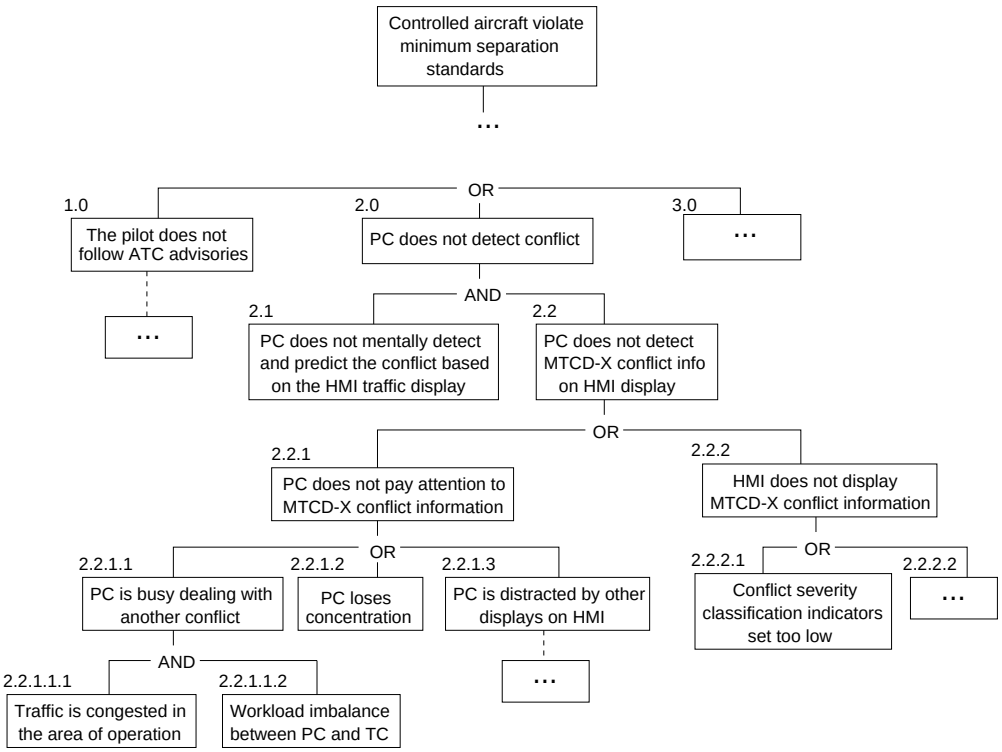


Fig. 3. A Piece of a Fault Tree for Violating Minimum Separation

Figure 3 shows a piece of the fault tree for the violation of minimum separation between controlled aircraft. The fault tree is used to derive requirements and design constraints related to safety. Each leaf node in the fault tree must either be traced to an operational or training requirement for the controller tasks or to an MTCD requirement or design constraint (and thence to the design feature used to eliminate or mitigate it). If a leaf node cannot be eliminated or mitigated, it must be accepted (and documented) as a necessary limitation of the

system. Such limitations may in turn require changes in the operation or design of the overall air traffic management system. Information derived from the fault tree may also be used in the Preliminary Task Analysis (and vice versa).

The hazard analysis and system engineering processes are iterative and mutually reinforcing. In the beginning, when few system design decisions have been made, the hazard analysis may be very general. As the system design emerges, the hazard analysis will become more detailed and will impact additional design decisions. For example, the need for conflict severity categorization for MTCD was identified in the PHA and leads to a requirement:

MTCD-08: *MTCD shall support conflict severity categorization.*

The box labelled 2.2.2.1 (*Conflict severity classification indicators are set too low*) leads to requirements and design constraints related to conflict severity classification indicators, how they are set and how they can be changed, the conditions under which conflicts are displayed, and the need for feedback to the controller about the current value of the conflict severity categorization indicators. For example, the conflict detection function might include a requirement to allow the operators to change the conflict severity thresholds. At the same time, there may be a constraint on the controller tasks and interface design that requires permission before the conflict categorization indicators can be changed by the controller.

A Preliminary Task Analysis (PTA) is also performed at this early concept development stage and interacts closely with the concurrent PHA process. The PTA consists of cognitive engineers, human factors experts, and operators together specifying the goals and responsibilities of the users of a new tool or technology, the task allocation principles to be used, and operator task and training requirements.

For MTCD, we started by specifying all of the tactical controller (TC) and planning controller (PC) responsibilities, not just those directly affected by MTCD. All responsibilities were included because any safety or usability analysis will require showing that MTCD-X does not negatively impact any of the controller activities. For instance, the PC (Planning Controller) is responsible for detecting sector entry or exit conflicts and formulating resolution plans with the PC of the adjacent sector and the TC (Tactical Controller) of the current sector. The TC, on the other hand, is responsible for in-sector tactical conflict detection and for implementing the plans formulated by the PC for entry or exit conflicts.

The second step in the PTA is to define the task allocation principles to be used in allocating tasks between the human controllers and the automation. This process uses the results of the Preliminary Hazard Analysis, previous accidents and incidents, human factors considerations, controller preferences and inputs, etc. For example, some task allocation principles for conflict resolution might be:

> *The human controller will have final authority as far as the use of the prediction tool, the need for intervention, and the criticality of the sit-*

uation. The controller will be responsible for devising solutions to the conflict.

These principles, together with the PHA and high-level system goals, are used to write requirements for the controller tasks, the automated function, and the human-machine interface. For MTCD, the PHA, the controller responsibilities, and the task allocation principles may lead to the following operator requirements (among others):

MIT-OP-R02: The PC shall plan traffic using MTCD output, and where a problem persists shall notify its existence and nature to the TC.
MIT-OP-R03: If incorrect or inconvenient behavior (e.g. high rate of false alarms) is observed, the controller shall not use the MTCD function.
MIT-OP-R07: The controller shall address conflicts detected by MTCD in a criticality-based rather than time-based order.

The final step of the PTA is the generation of operator task and training requirements and constraints.

The system goals and environmental assumptions and constraints along with the results from the PHA and PTA are then used to generate a complete set of system requirements (including functionality, maintenance, management, and interface requirements), operational requirements, and design constraints.

Level 2: System Design

Using the system requirements and design constraints as well as the other information that has been generated to this point, a system design (or alternative system designs) is created and tasks are allocated to the system components (including the operators) to satisfy the requirements, task allocation principles, and operational goals. The specification at this level contains *system* design, not *software* design. Note that this process will involve much iteration as the results of analysis, experimentation, review, etc. become available. We have found that natural language is the most appropriate specification language at this level combined with standard engineering and mathematical notations, for example, differential equations or control block diagrams for documenting control laws.

Level 3: Allocating and Validating Requirements

The next step involves validating the system design and requirements and performing any tradeoff and evaluation studies that may be required to select from among a set of design alternatives. Various types of operator task analyses and system hazard analyses play a part in this process.

The methodology includes using formal models in SpecRLM-RL to assist in this evaluation and validation process. Designers construct formal, blackbox models of the required component behavior and operator tasks. The SpecTRM-RL modeling language was designed with readability and reviewability of the

models by various domain experts as a major goal. The models act as a communication medium among everyone involved and therefore must be easily understandable and unambiguous.

An earlier version of the current modeling language (called RSML) was used to specify the official requirements for TCAS II [LHH94]. One requirement of that project was to provide a specification language that could be read and reviewed by any interested parties with minimal training (less than an hour). Our latest version of the formal modeling/specification language attempts to enhance readability and reviewability by reducing even further the semantic distance between the reviewer's mental model and the specification. We also elminated the features we found to be very error-prone, such as internal events. Our current research is focused on visualization techniques for complex, formal specifications and the use of domain-specific notations.

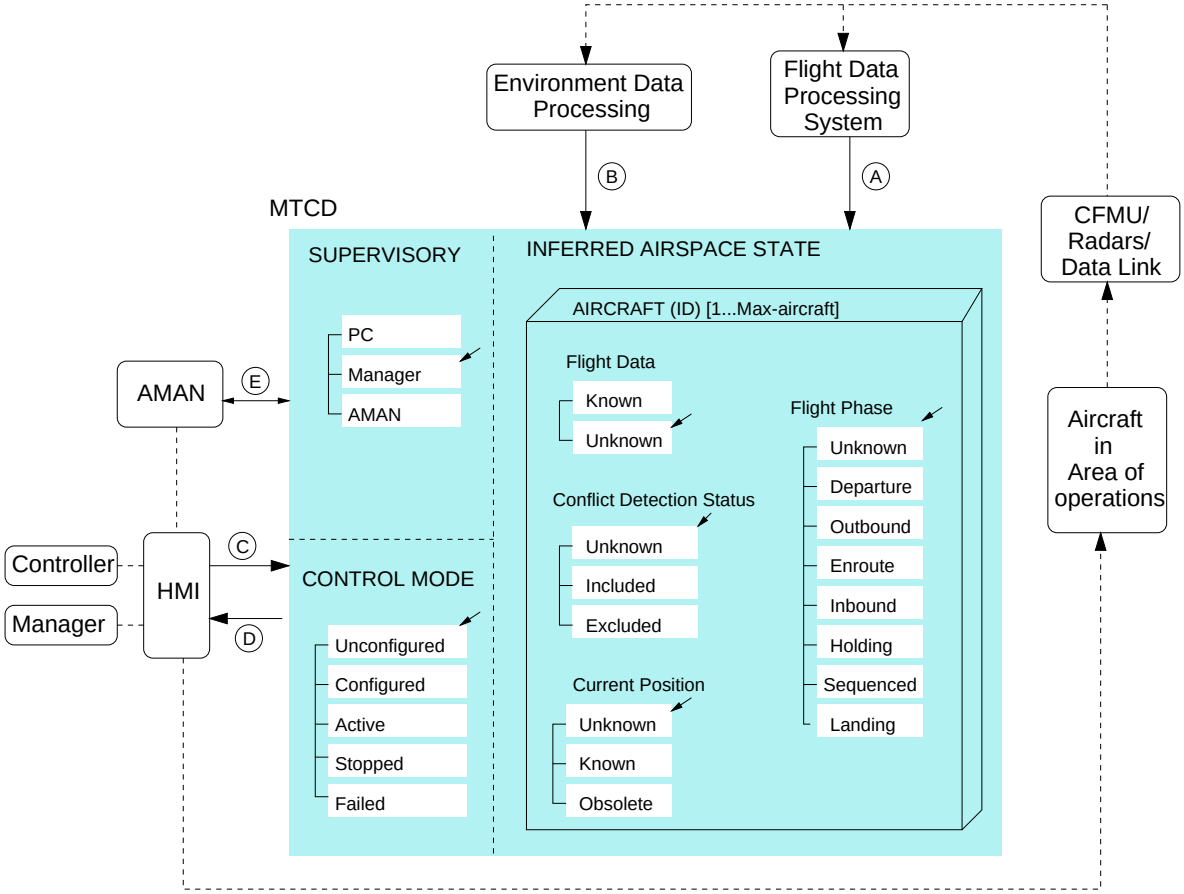
The blackbox component behavior models are built on an underlying state machine model [JLHM91]. SpecRLM-RL blackbox models combine a graphical notation with tabular descriptions of the legal state changes. Figures 4 and 5 show pieces of our SpecTRM-RL model for MTCD. The MTCD model could be combined with a model of the airspace and models of the other system components and executed or analyzed together in a system simulation environment.

The graphical part of the model (as shown in Figure 4), is drawn in the form of a control loop showing the direct interactions of MTCD with other system components (the environment data processing system, the flight data processing system, and the controller working position). A future planned interface with a new arrival manager tool (AMAN) is shown.

A SpecTRM-RL model of a component itself (in this case MTCD) usually has four parts:

- **Display modes**: the display mode will affect the information to be provided to the controller. A display mode specification is not needed for MTCD
- **Supervisory modes**: the supervisory mode specifies who is using the component at any time, which affects which operations are legal. In this case the supervisor may be the PC, the operations manager, or AMAN.
- **Component control modes**: the mode the automation is in. In the case of MTCD, these include unconfigured, configured, active, stopped, and failed.
- **Inferred system state**: a model of the inferred state of the controlled system, in this case, the airspace in the area of operation.

The controlled system (airspace) state at any time is inferred from the inputs received and may be incorrect if those inputs are incorrect or not timely. The airspace model within MTCD consists of a model of the assumed state of each of the aircraft being evaluated for conflicts. The model of MTCD shown has, for each aircraft, state variables representing the status of the flight data from that aircraft, the conflict detection status, the flight phase (needed because separation criteria will vary with flight phase), and the status of the current position information. Note that accidents occur when this inferred airspace state differs from the real state. The validation phase involves assuring that the model is



(A) FDPD → MTCD-X

For each aircraft:

Flight_ID
Aircraft_Type
Nav_Capabilities
Class_of_Flight
Current_Position (X,Y, Level)
Trajectory

(E) AMAN → MTCD-X

MTCD-X → AMAN

Undefined at this time

(B) EDPD → MTCD-X

For each airspace:

Airspace_ID
Upper_Level
Lower_Level
Boundary
Start_Time_of_Restriction
End_Time_of_Restriction
Type_of_Airspace
Separation_Parameters
Uncertainty_Parameters

For each parallel route:

Route_ID
Separation_Parameters

Area _of_Operation

(C) HMI → MTCD-X

Stop_MTCD-X
Start_MTCD-X
Configuration_Params
Include_Aircraft (ID)
Exclude_Aircraft (ID)

(D) MTCD-X → HMI

For each conflict::

Conflict_ID
Conflict_Type
Severity
Conflict_Data

Fig. 4. Part of a SpecTRM-RL Model of MTCD

correct and that the overall system is robust against errors in the information received about the current airspace state.

A complete model also needs to specify the conditions under which each of the MTCD control modes is used, the conditions under which the outputs are generated and their content, and how each of the inferred state variables is assigned a value. Figure 5 shows the logic for selecting the MTCD operating mode. The conditions under which each of the four values for operating mode become enabled are described using AND/OR tables. The operating mode takes a particular value when the *table* associated with that value evaluates to TRUE, which in turn happens when any *column* of the table evaluates to TRUE. A column is TRUE when each row satisfies the truth value shown (with a dot denoting "don't care"). In the example, the MTCD status becomes ACTIVE if either (1) the previous mode was CONFIGURED and an area of operation is received by MTCD *or* (2) the previous mode was STOPPED and a start command is received.

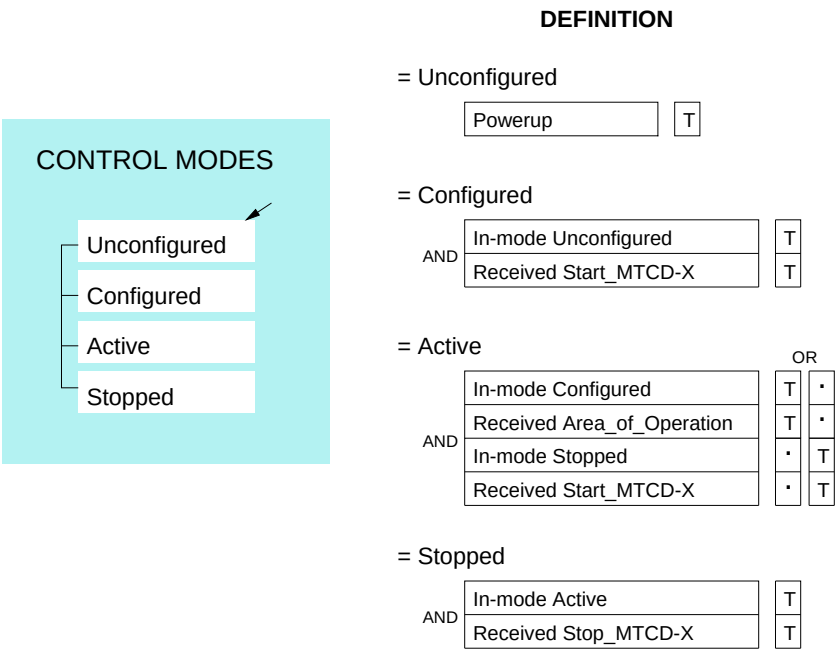


Fig. 5. The Logic for Selecting the Current Operating Mode

An executable human task modeling language has also been defined. In previous experimentation, we found that a different notation was more useful for modeling human tasks than that used for describing the automation behavior. Both generate the same type of underlying formal model, which should allow integrated execution and analysis of the system as a whole, both the automation and the user tasks. An important aspect is the specification of the communica-

tion between the various controllers as well as between the controllers and the automation. We have shown how these task models can be used to find features of the combined automation and task design that can lead to mode confusion and other human errors [RZK00].

In addition to being reviewable by aviation and air traffic management experts, the formal models are executable and can be executed alone or integrated into an ATC simulation environment. Animation and visualization of the executing models assist in understanding and evaluating the proposed design of the automation and controller tasks. The executable specifications can also be used in experiments involving controllers to evaluate human factors. An advantage of executable specifications over prototypes or special simulation languages is that the specification can be changed as the evaluation proceeds. At the end of the evaluation stage, a final specification is ready for implementation without having to reverse engineer a specification from a prototype.

Because the modeling language is based on a formal mathematical model, various types of automated mathematical analysis can also be applied. We have developed techniques for analysis of consistency and completeness, robust operation in an imperfect environment, reachability of hazardous states, and potential mode confusion. A new hybrid (continuous and discrete) version of the basic modeling language (SpecTRM-H) allows safety analysis of the conflict detection algorithms themselves [Neo01].

Requirements errors and incompleteness account for most of the accidents in which digital automation has been involved. It is, therefore, particularly important that the requirements specification distinguish the desired behavior from that of any other, undesired behavior, that is, the specification must be precise (unambiguous), complete, and correct (consistent) with respect to the encompassing system requirements. We have built prototype tools to check our specifications for consistency and some aspects of mathematical completeness [HL96]. Other important completeness aspects are enforced by the design of SpecTRM-RL itself [Lev00b] or can be checked using inspection or simple tools.

Robustness can be evaluated using an automated technique called Software Deviation Analysis [RL87]. SDA determines how the software will operate in an imperfect environment by examining the effects of deviations of system parameters (inputs). The input to the SDA tool is an input deviation, for example, "*the altitude reported by the radar data processing function is lower than the actual altitude.*" The output is a list of scenarios, where a scenario is defined as a set of deviations in the software inputs plus constraints on the software execution states that are sufficient to lead to a deviation in an identified safety-critical output. The deviation analysis procedure can optionally add further deviations as it constrains the software state, allowing for the analysis of the effects of multiple, independent failures.

Other tools can be used to assist in system and subsystem hazard analysis [LS87]. Information from these analyses is useful in eliminating hazards from the design or in designing controls and hazard mitigation. For example, one tool assists the designer in tracing back through the model from hazardous states to

determine if and how they are reachable. Backward search can also reveal how the system can end up in a hazardous state if a failure occurs. Our backward reachability analysis on discrete state models has recently been augmented to include continuous states (a hybrid model) [Neo01], using some basic techniques from control theory. Neogi has experimentally applied this approach to evaluate the safety of the conflict detection algorithm used in MTCD.

Finally, the specification/model of the blackbox automation behavior can be evaluated for its potential to lead to mode confusion [LRK97]. We have identified six automation design categories have been identified as leading to mode confusion, based on accidents and simulator studies: (1) ambiguous interface modes, (2) inconsistent automation behavior, (3) indirect mode changes, (4) operator authority limits, (5) unintended side effects, and (6) lack of appropriate feedback. Analysis procedures are being developed to detect these features in SpecTRM-RL models. We have experimentally tested these ideas on real helicopter and aircraft flight management system software.

Once the engineers are happy with the operator task and logical system design, detailed design and construction of the system components, controls and displays, training materials, and operator manuals can begin.

Level 4 and 5: Component Design and Implementation

For some applications, the code can be automatically generated from the SpecTRM-RL models. For others, the real-time requirements require hand crafting of the code. In general, many of the software specification and design techniques currently popular do not afford the level of traceability and safety assurance necessary in safety-critical systems. We assume that reuse will start from the SpecTRM-RL specification because analysis at the system level will be necessary. Therefore, some of the design methodologies focused on reusing code components will be less important in this type of system. We are currently developing SpecTRM-RL macro components for spacecraft design, as well as a spacecraft-specific modeling language built on top of SpecTRM-RL, to evaluate the practicality and feasibility of this approach to reuse.

This level will contain most of the information about system and middleware design and hardware decisions. Many or most of the proposals for specifying and designing such computer system components will fit into the SpecTRM approach.

Level 6: Operations

Ensuring safety does not stop with development. Operations need to be monitored and periodically audited to ensure that the assumptions underlying the original hazard analysis and the safety-related design features hold in the current system. Operators change their behavior and the environment is very likely to change. The traceability and documentation of design rationale included in

Intent Specifications should be useful in deriving auditing and performance monitoring procedures and metrics and in making any necessary changes to the software.

3 Status and Future Extensions

A methodology for building safety into embedded software has been described. The approach integrates system hazard analysis, operator task analysis, traceability, and documentation of design rationale as well as executable and analyzable models into the development process. A commercial toolset to support SpecTRM is in development and is currently being used to support industrial projects.

References

[CKI88] B. Curtis, H. Krasner and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(2): 1268–1287, 1988.

[JLHM91] M.S. Jaffe, N.G. Leveson, M.P.E. Heimdahl, and B.Melhart. Software requirements analysis for real-time process-control systems . *IEEE Trans. on Soft. Eng.*, SE-17(3), Mar 1991.

[HL96] Heimdahl, M.P.E. and Leveson, N.G. Completeness and Consistency in Hierarchical State-Based Requirements. *IEEE Trans. on Soft. Eng.*, SE-22, No. 6, June 1996.

[Joh80] Johnson, W.G. *MORT Safety Assurance Systems*, Marcel Dekker, Inc., 1980.

[Lev00a] Leveson, N.G. Intent Specifications. *IEEE Trans. on Soft. Eng.*, Jan. 2000.

[Lev00b] Leveson, N.G. Completeness in Formal Specification Language Design for Process-Control Systems. *ACM Formal Methods in Software Practice*, Aug 2000

[LHH94] Leveson, N.G., Heimdahl, M.P.E., Hildreth, H., and Reese, J.D. Requirements Specification for Process-Control Systems. *IEEE Trans. on Soft. Eng.*, SE-20, No. 9, Sept. 1994.

[LRK97] Leveson, N.G., Reese, J.D., Koga, S., Pinnel, L.D., and Sandys, S.D. Analyzing Requirements Specifications for Mode Confusion Errors. *Int. Workshop on Human Error, Safety, and System Development*, Glasgow, March 1997.

[LS87] Leveson, N.G. and Stolzy, J.L. Safety Analysis Using Petri Nets. *IEEE Trans. on Soft. Eng.*, Vol. SE-13, No. 3, March 1987, pp. 386-397.

[Neo01] Neogi, N. Hazard Elimination Using Backward Reachability and Hybrid Modeling Techniques. Ph.D. Dissertation, Aeronautics and Astronautics, MIT, May 2002.

[RL87] Reese, J.D. and Leveson, N.G. Software Deviation Analysis. *International Conference on Software Engineering*, Boston, May 1997.

[RZK00] Rodriguez, M., Zimmerman, M., Katahira, M., de Villepin, M., Ingram, B., and Leveson, N.G. Identifying Mode Confusion Potential in Software Design. *Digital Aviation Systems Conference*, Philadelphia, October 2000 .

[SW95] Sarter, N.D. and Woods, D. "How in the World did I Ever Get into That Mode?" *Human Factors 37*, 5–19.

[WCK91] Wiener, E.L., Chidester, T.R., Kanki, B.G., Palmer E.A., Curry, R.E., and Gregorich, S.E. The Impact of Cockpit Automation on Crew Coordination and Communications. NASA Ames Research Center, 1991.

This article was processed using the LaTeX macro package with LLNCS style