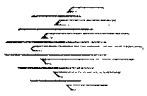


Carnegie Mellon
Software Engineering Institute

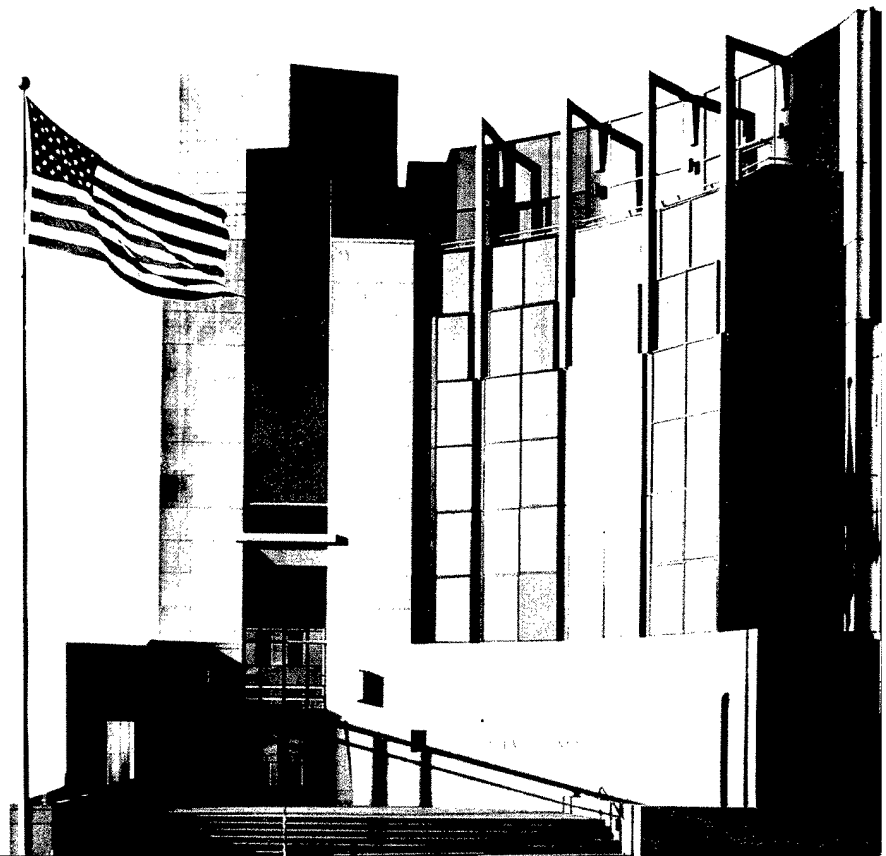
Model-Based Verification: Abstraction Guidelines

John Hudak
Santiago Comella-Dorda
David P. Gluch
Grace Lewis
Chuck Weinstock

October 2002

Performance-Critical Systems

Technical Note
CMU/SEI-2002-TN-011



Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admission, employment, or administration of its programs or activities on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state, or local laws or executive orders.

In addition, Carnegie Mellon University does not discriminate in admission, employment or administration of its programs on the basis of religion, creed, ancestry, belief, age, veteran status, sexual orientation or in violation of federal, state, or local laws or executive orders. However, in the judgment of the Carnegie Mellon Human Relations Commission, the Department of Defense policy of "Don't ask, don't tell, don't pursue" excludes openly gay, lesbian and bisexual students from receiving ROTC scholarships or serving in the military. Nevertheless, all ROTC classes at Carnegie Mellon University are available to all students.

Inquiries concerning application of these statements should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-2056.

Obtain general information about Carnegie Mellon University by calling (412) 268-2000.

Model-Based Verification: Abstraction Guidelines

John Hudak
Santiago Comella-Dorda
David P. Gluch
Grace Lewis
Chuck Weinstock

October 2002

Performance-Critical Systems

Technical Note
CMU/SEI-2002-TN-011

Contents

List of Figures

Abstract

Model-Based Verification (MBV) is a systematic approach to finding defects (errors) in software requirements, designs, or code. The approach judiciously incorporates mathematical formalism, in the form of models, to provide a disciplined and logical analysis practice, rather than a "proof of correctness" strategy.

This technical note presents a number of abstraction techniques that can be used to build essential models of system behavior in the context of MBV and details a methodology for creating state machine models using those techniques. In building essential models, abstraction is used to hide details and expose the entities, variables, states, and transitions needed to construct a state machine model. Through illustrative examples, this technical note identifies the types of simplifications that are useful and effective, and highlights the importance of the perspective in determining what important elements to include in an abstracted model.

1 Introduction

Model-Based Verification (MBV) is a systematic approach to finding defects (errors) in software requirements, designs, or code [Gluch 98]. The approach judiciously incorporates mathematical formalism, in the form of models, to provide a disciplined and logical analysis practice, rather than a "proof" of correctness strategy. MBV involves creating essential models of system behavior and analyzing these models against formal representations of expected properties.

The artifacts and the key processes used in Model-Based Verification are shown in Figure 1. Model building and analysis are the core parts of Model-Based Verification practices. These two activities are performed using an iterative and incremental approach, where a small amount of modeling is followed by a small amount of analysis. A parallel compile activity gathers detailed information on errors and potential corrective actions.

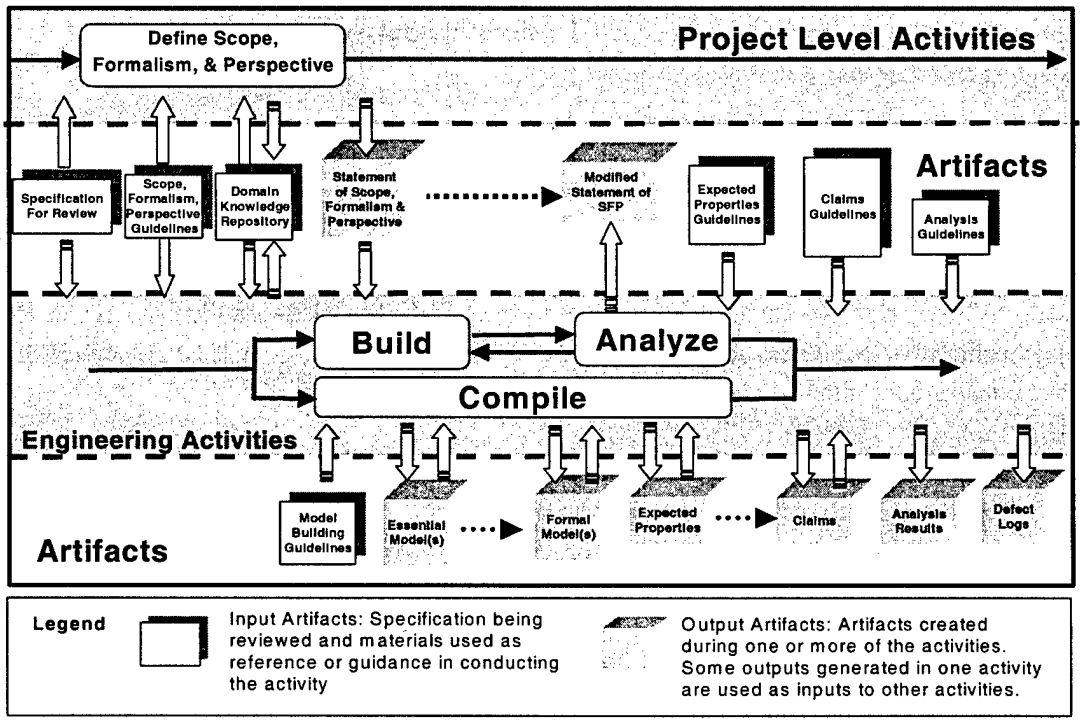


Figure 1: Model-Based Verification Process and Artifacts

Essential models are simplified formal representations that capture the essence of a system, rather than provide an exhaustive, detailed description of it. Through the selection of only critical (important or risky) parts of the system and appropriately abstracted perspectives, a reviewer using model-based techniques can focus the analysis on the critical and technically

difficult aspects of the system. The discipline and rigor required to create a formal model in and of itself uncovers errors even before the model is analyzed.

Once the formal model is built, it is analyzed. Within this analysis, potential defects are identified both while formulating claims about the system's expected behavior and while formally analyzing the model using model-checking tools. Model checking has been shown to uncover especially difficult to identify errors: the kind of errors that result due to complexity associated with multiple interacting and inter-dependent components [Clarke 95, 96]. These include embedded as well as highly distributed applications.

A variety of different formal modeling and analysis techniques are employed within Model-Based Verification [Gluch 98, Clarke 96]. The choices are based upon the type of system being analyzed and the technological foundation of the critical aspects of that system. These choices involve an engineering tradeoff among the technical perspective, formalism, level of abstraction, and scope of the modeling effort.

The specific techniques and engineering practices of applying Model-Based Verification to software verification have yet to be fully explored and documented. A number of barriers to the adoption of Model-Based Verification have been identified including the lack of good tool support, expertise in organizations, good training materials, and process support for formal modeling and analysis.

In order to address some of these issues, the SEI has created a process framework for Model-Based Verification practice. This process framework identifies a number of key tasks and artifacts. Additionally, the SEI has produced a series of technical notes that can be used by Model-Based Verification practitioners. Each technical note is focused on a particular Model-Based Verification task, providing guidelines and techniques for one aspect of the Model-Based Verification practice. These technical notes address abstraction in building models, generating expected properties, generating formal claims, and interpreting the results of analysis.

This technical note focuses on the abstraction techniques for exposing the elements needed when creating a model. It also describes a systematic approach model building, using abstraction, and how perspective is used to guide the approach. This technical note addresses the specific topic of using abstraction in MBV. The reader is referred to Gluch for information that is used as a precursor to this activity [Gluch 01]. Gluch also provides a general outline for the entire MBV practice and provides a guide to the technical notes that have been developed in support of specific MBV activities [Gluch 02b].

Through this manual, we hope you will

- gain an understanding of the role and techniques one uses in abstraction
- develop a working understanding of several proposed methodologies that aid in the development of essential state-machine models

2 Abstraction Techniques

Modeling and analysis are used to explore the functional behavior of a system or its components. For example, a model of an airfoil can be used to explore its response to control laws under various flight conditions. Models tend to be based on and guided by first principles (e.g., mathematical equations that describe resultant forces based on load dynamics). These principles rely on the scientific foundations of the application (e.g., the physics of flight) to guide the building and analysis of the models. In contrast, modeling as used in the Model-Based Verification of software captures (models) the desired behavior of systems or components based upon the specification of their behavior and potential implementation in software, examining it from various viewpoints (or perspectives). These perspectives include fault-tolerance, safety, consistency, etc.

Abstraction is used to reduce the complexity of a model by including only the parts of the system necessary for the issues being investigated. The goal of abstraction is to prune away unnecessary detail. This enables the modeler to explore, substantiate, or disprove intended behaviors of a system while maintaining the validity of the model [Frantz 95]. Engineers and scientists routinely use abstraction in problem solving. This section of the technical note presents some abstraction techniques and relevant examples. The techniques discussed are:

- variable elimination
- enumeration
- reduction
- non-determinism
- grouping based on commonalities
- decomposition

Various stages of abstraction and modeling of a system are generally referred to as being representative of a 'level of abstraction.' The convention for these levels is that the removal of detail results in a higher level of abstraction whereas adding detail results in a lower level of abstraction. The term 'granularity' is often used to refer to the level of detail. Levels of abstraction and levels of granularity are often used interchangeably.

2.1 Variable Elimination

Variable elimination [Heitmeyer 98, Bharadwaj 99] removes parts of the system that are not relevant to the properties and behavior to be demonstrated or proven. Irrelevant variables can be identified by looking at dependencies and then removed. Again, consider a process control computer system and its applications software. Let's assume we are investigating the

safety aspects of the operation of a gas turbine. The gas turbine receives superheated gas from a pressure vessel that is heated by a furnace. The temperature of the furnace (T) causes the gas to expand within the pressure vessel, elevating the pressure inside of it (p), which then feeds the turbine, causing it to increase its speed (s). Under certain conditions, increasing the temperature, will result in an increase in pressure of the gas, which would result in an increase in speed of the turbine. One approach to constructing a model would be to express the relationship between the temperature and pressure, and the pressure and the speed of the turbine. This would involve the three variables of T, p, and s. If the safety issues being investigated are focused on temperatures only, then one could model the system considering only the temperature (T) and the resultant speed (s). The relationship between the temperature (T) of the furnace and the speed (s) of the turbine would be the property of interest.

Another approach to variable elimination can be considered. Suppose we are interested in variable P1 in a six-variable model consisting of P1, P2, P3, T1, T2, and T3. In the model relationships exist among the variables. Specifically P1, is dependent on P2 and P3. P3 has no other dependencies. P2 however is dependent on T2. The fact that P2 depends upon T2 can be used to eliminate P2.

2.2 Enumeration

Enumeration is a technique that represents the range of the values of a continuous variable as a set of abstracted terms. The general approach is to partition the range of the variable into a set of subparts. Each subpart can then be enumerated by a separate variable. Specifically, consider a process control task that monitors a range of temperatures. The control system must perform a certain task (for example, alarming) during a specific portion of the temperature range. Suppose the total monitored range is 0-100 degrees F. When the temperature is less than 70 degrees F, the control system must assert T1_light = Green. All temperature values in this range are safe. If the temperature is from 70 degrees F to and including 85 degrees F a caution alert, T1_light = Yellow, must be asserted. If the temperature is greater than 85 degrees F a warning alert, T1_light = Red, must be asserted. In this case, the temperature range can be abstracted and represented by an enumerated variable that has three values: {normal, high, and dangerously high}.

2.3 Reduction

Reduction is a technique that decreases the size of individual parts of a system while preserving relevant characteristics needed to verify the behavior of the system. The reduction choices are made based upon what behavior is to be investigated; this is the modeling perspective. For example, consider a network message packet consisting of a destination address, sending address, message identifier, message body, end-of-transmission character, and checksum. Suppose that the fault tolerance capabilities of the routing technique are to be investigated. Reviewing the specification shows that the destination address and the checksum are used to ensure correct network communication. Therefore, when developing a

model a reasonable abstraction is to treat the message packet as if it contained only the destination address and checksum, ignoring the other components of the message in the model. The other components are not used in fault/error detection or response or other functions involved in providing fault tolerance capabilities in the system.

Reduction and variable elimination are similar in that they both reduce the amount of detail in the system. They differ in that reduction simply removes the non-essential information, whereas variable elimination takes advantage of dependencies that exist among some variables, and uses one or a subset of the variables in the model. In practice, this helps to reduce the number of states in the model. If one of the dependent variables is of concern, one can determine its behavior given the behavior of the other variables, and the (usually mathematical) relationship among them.

2.4 Non-Determinism

Performing abstraction by using *non-determinism* involves allowing arbitrary choices at decision or transition points in a model. In this technique, the details in the logic used to make a choice among alternatives are ignored. For example, consider a user interface that has three modes of displaying information, AUTOMATIC, MANUAL, and MONITOR. Each display mode visually changes the aspect ratio of the display, areas for interactive controls, and areas for display—only values. The display communicates to the control system via a serial communication line. Based upon the contents of a DISPLAY_MODE variable in the communication model, the intelligent display would perform the necessary computations to change the display appearance and update its internal data store. If the Statement of Perspective states that the objective in the analysis is to determine that communication was successful and that a display would result, non-determinism could be used to produce a simplified model. The model would allow an arbitrary choice of DISPLAY_MODE, and would ignore computational and logical details associated with redisplaying the information. Thus, the model would focus on communication and successful display, as established in the statement of perspective, rather than on the details of the selection logic.

2.5 Grouping (Commonality)

Grouping is an explicit many-to-one mapping of variables or entities into a single descriptor. The issues the model is being used to explore as described in the Statement of Perspective guides the grouping. The goal of this technique is to group entities into a smaller set or to regroup entities to facilitate modeling and analysis.

As an example, consider the climate control system of a tall building that monitors temperature (6 sensors) and flow of heating water (5 sensors) distributed throughout the building. Depending on the Statement of Perspective, it may be appropriate to group these 11 sensors into 2 operational groups (temperature and pressure) or into 3 zones of 2 input types each: flow and temperature (6 distinct abstracted variables).

2.6 Decomposition

Decomposition is a technique for systematically partitioning a system into structural or functional components. While this approach is not traditionally considered an abstraction technique, it is effective in helping to make decisions about what is needed in a model and what is not and in understanding individual components of a system and their interrelationships. For example, a process control computer may be decomposed into the central processing unit (CPU), the analog-to-digital (A/D) subsystem, the digital-to-analog (D/A) subsystem, a digital input/output (I/O) subsystem, a communication channel to an operator display, and another communication channel to a process control computer at some other level or area. The interrelationships may be that the CPU communicates with the A/D and D/A subsystems via a special I/O bus, with the digital I/O via an RS-232 bus, with the operator display via a universal serial bus, and with other computers via Ethernet.

Suppose each of the I/O subsystems variables is placed into a common memory area and the protocol for inserting the data, maintaining the data, and retrieving the data is of concern. The details of how a data value is actually acquired from the outside world are not of interest and may be neglected. In this example, it is sufficient to assume that the data value is obtained from the A/D channel and is placed in memory location X at some specified rate. The details of the sampling process can be ignored.

If the decomposition is appropriately done, each component can be modeled (at the necessary level of detail), and analyzed separately. Moreover, results from individual models (components) can be recombined at a higher level of abstraction.

3 Model Building and Abstraction

This section presents an approach for building essential models. Section 1, Figure 1 represents a process view of the MBV activities. The top half shows project-level activities with the lower half depicting the engineering activities. Project-level activities involve a variety of personnel (e.g., project managers, lead hardware engineers, lead software engineers, software engineers, and other stakeholders). At the project level, decisions are made on what issues should be explored and verified using MBV. These decisions result in the Statement of Scope, Formalism, and Perspective. (Refer to Gluch for a detailed discussion regarding the purpose and generation of these documents [Gluch 01].) Scope defines what parts of the system will be explored and modeled. Formalism defines the tool(s) to be used, and their associated documentation. Perspective defines the objectives of the modeling effort. The Statement of Scope, Formalism, and Perspective guides subsequent MBV activities, particularly the Build activity that is shown in the Engineering Activities area of Figure 1. The abstraction techniques that were presented in Section 2 are used principally within the Build activity.

An amplification of the Build activity is shown in Figure 2. It involves a two-phased approach: (1) achieving an understanding of the system and formulating intermediate representations of it and (2) composing a state-machine model.

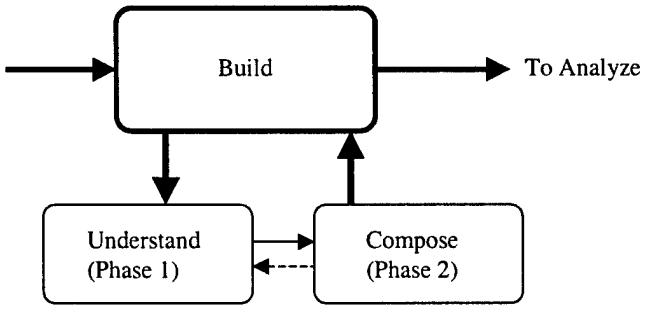


Figure 2: The Two Phases of the Build Process

The goal of Phase 1 (Understand) is to develop a working knowledge of the major components and their structural and operational interrelationships for the system being investigated. The artifacts produced as a result of the investigation, together with other existing project documents (e.g., requirements) are used in the composition of the model in Phase 2 (Compose). The back arrow between Understand and Compose indicates that it is often necessary to revisit artifacts used in the Understand phase to resolve issues in composing the model. The remainder of this section will detail the activities and artifacts within the two phases of the Build process.

3.1 Artifacts Involved in the Build Activity

The Build activity is based on:

- information (primarily documents, but also expert knowledge) that usually exists as part of a general software engineering effort
- documents that are generated as a result of the MBV process

The artifacts involved in the entire Build process are listed below.

- Specification for Review: This is the specification that is being analyzed using MBV techniques. It may include
  - system specifications: the written, functional description of the system to be built.
  - software requirements specifications: the formal description of the software to be built.
  - hardware architecture diagram: a diagram illustrating the major hardware components and their structural and functional relationships. This diagram is often a component of the Specifications for Review. If not available, it may be generated by people involved in the Build activity.
- Statements of Scope, Formalism and Perspective: expresses what area/subsections are to be modeled, what tool(s) and language are to be used to do the modeling, and what properties are to be satisfied by the modeling (e.g., fault tolerance, safety, completeness). This is generated as part of the MBV project activities.
- Acronym list: a list of the acronyms and their meanings. This list is often generated in the Build process. If it already exists, it is usually augmented.
- Definition of terms: the definition of components, subassemblies, and related elements. It is usually part of the Specifications for Review or Software Requirements Specifications. If not, it is generated as part of the Build process. If it already exists, it is usually augmented.
- Assembly diagram: a diagram (usually structural) showing a high-level, static view of the physical architecture of the assembly [Firesmith 93]. Generally this is a hierarchical view of the system that contains the major hardware components, external interfaces, etc. It is usually developed as part of the Build process.
- Context diagram: a diagram that documents the domain of study by showing the set of data flows that cross into and out of the domain. [Jackson 01]. It lays the foundation for structuring and analyzing the problem by showing all the domains and interfaces that must be taken into account. The domain in this case is the problem area (or sub-area) that is being addressed.
- Issues list: a table of items generated within the Build process that are not fully discussed or may be ambiguous in the specifications. The understanding and clarification of these issues is generally important to the Build process.

These artifacts are discussed in the next section within the context of an example problem. Examples of each artifact are included in the Appendix.

3.2 Sample Problem

To aid in the discussion of the build activity, a detailed example of an automobile electronic dashboard is presented. Appendix A illustrates the hardware architecture of the electronic dashboard. There are two microprocessors that communicate over a redundant dual Controlled Area Network (CAN) bus. The engine control computer monitors various engine sensors that provide a combination of both digital and analog inputs. The engine microcomputer also produces an analog output to the throttle. (Note that only signals relevant to the dashboard and cruise control example have been shown.) The dashboard control computer also monitors various digital inputs such as the cruise control On/Off button, Coast button, and so forth. The dashboard control computer also puts out various digital signals to the named digital displays. Some of the signals monitored by the engine control computer must be communicated to the dashboard control computer (e.g., wheel rotation, brake sensor) because it contains the cruise control application software. Communication is supported by the CAN bus—a serial line protocol developed primarily for the auto industry. The protocol contains rules for message formatting, retries, bus failure, and so forth.

The electronic dashboard is composed of a cruise control, speedometer, and gauge cluster subassemblies as shown in Appendix A-2. Beneath each subassembly are the control signals relevant to each function.

The software requirements specification document for the cruise control unit provides the basis of this discussion (Appendix J). It is assumed that a prior MBV activity generated the statements of scope (Appendix K), formalism (Appendix L) and perspective (Appendix M). The statement of Scope defines the cruise control as the subassembly to be investigated via MBV techniques. The statement of Perspective states what properties are to be verified by modeling the cruise control. In this example, the investigation is to determine whether the operational modes are consistent among themselves and whether the unit can always enter the off state from any state (a fail-safe condition). These are not the only properties that could have been investigated. For example, since the cruise control application requires data that is gathered from another computer (the engine control computer), message timing and synchronization could be investigated.

3.3 Phase 1: Understand

Figure 3 shows the activities and artifacts of the first phase within the Build activity of MBV. The basic steps in this phase are to review input artifacts and to generate output artifacts using various abstraction techniques, in order to develop an understanding of the problem domain. Once the output artifacts are generated, the next phase in the process, Compose, is initiated. The inputs to the Understand activity are shown on the left hand side, and the output artifacts are shown below the activity. The specifications, Statements of Scope, Formalism, and Perspective, and the acronym list and definition of terms (if available) exist prior to the initiation of the Build activity; they are disseminated to the engineers who will be

doing the modeling. There may be a variation in the level of system understanding, domain knowledge, and experience possessed by each engineer. This may range from novice to expert. This discussion of the Build activities assumes the engineer involved is new to the project area and relatively inexperienced.

In Phase 1: Understand, an engineer becomes familiar with the system's essential elements. The activities in this phase focus on identifying, documenting, and organizing the content required for building models. The outcomes of these efforts form the basis for the development of a concise set of models that faithfully capture the behavior of the system, as prescribed in the Statements of Scope, Formalism, and Perspective. Critical to this phase is the use of abstraction techniques. All of the abstraction techniques described earlier in this document can be used. But certain techniques are more applicable and hence most often applied in this phase. These include *decomposition, grouping,* and *reduction.* The input documents are read in detail, with the goal of developing a good working knowledge of the problem area.

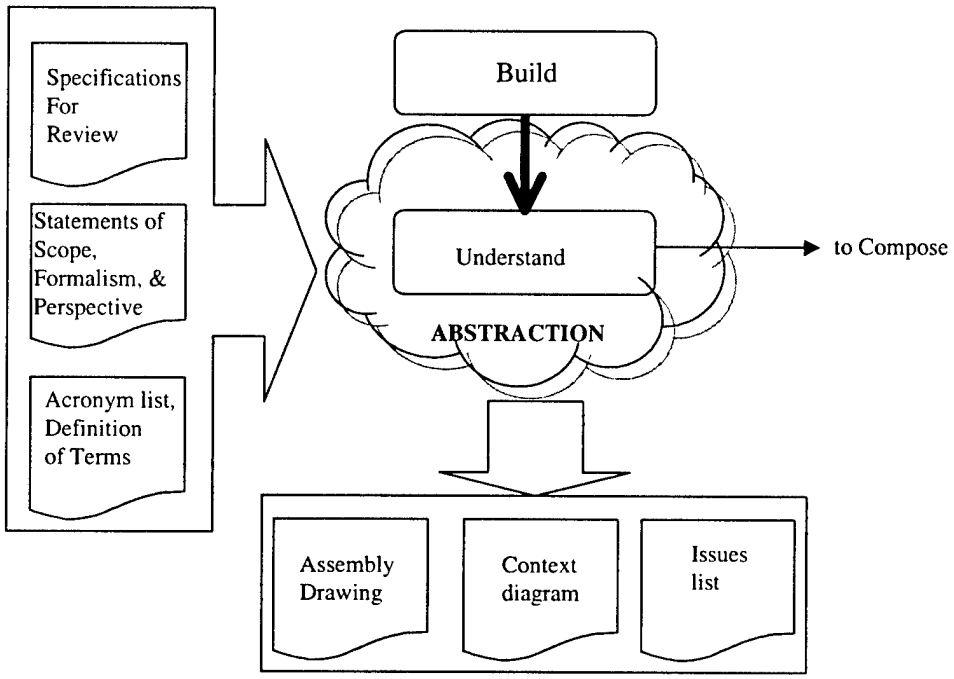


Figure 3: Phase 1 of the Build Activity: Understand

In order to systematically extract the necessary information needed for modeling, three output artifacts can be developed in this phase:

- assembly diagram: The assembly diagram shows the major hardware components of the system and their organizational relationship to the entire system. It provides a comprehensive but abstracted view of the system. This is especially valuable to an

engineer unfamiliar with the system or the application domain. The assembly diagram can be based upon a variety of conventions (e.g., object-oriented or structured design).

- context diagram: The context diagram scopes the problem area that is being investigated and shows the external interfaces. To further augment understanding of the system's operation, the data that is exchanged between the application and the external components is noted on the drawing. As one reads through the requirements, the discovery of information enables the partial construction of a context diagram as well as aiding in the creation of the assembly diagram. For example, the relationships discovered can help to refine the groupings within the assembly diagram. In essence this is an iterative endeavor that involves adding refinements to both artifacts, as the system becomes better understood.
- issues list: The issues list is developed throughout the entire Understand activity. It is a collection of various kinds of questions and statements about the area under investigation.

The three artifacts are not usually developed sequentially (as the listing may imply). Moreover, one or more of them may only be partly developed or not be included in a particular MBV effort or already exist or be replaced by a similar artifact, depending on the specific practices of an organization. Whatever alternative is chosen it is important that the efforts and outcomes of this phase provide a solid foundation, as embodied in the artifacts defined here, for the build phase.

In practice, enough information contained in a top-level specification is sufficient to largely complete the assembly diagram. As one develops the assembly diagram, questions can be added to the issues list. As one develops a deeper understanding of the problem area, sufficient information about the subassemblies and components can be attained to begin development of the context diagram. As the context diagram is developed, additional questions may be added to the issues list. As new relevant information is uncovered by reading through the specification, it is added to the appropriate diagram. The procedure to complete the diagrams is somewhat iterative; the degree of iteration usually depends on how well the input set of documents is organized and the degree of familiarity of the engineer with the problem domain.

The development of each output artifact is detailed below.

3.3.1 Developing the Assembly Diagram

The first artifact generated to help organize this information is the assembly diagram. The purpose of the assembly diagram is to provide a high-level, static view of the physical architecture of the assembly [Firesmith 93]. It generally consists of major entities of the system, a list of the physical components of each major entity, and the relationship among them.

Abstraction is used to help organize the components of the system into a group of logically related subcomponents in a way that supports the intended goal of the analysis. For example, considering the dashboard, grouping the sensors according to the functionality they support

as opposed to their type (analog or digital) makes sense if one is to investigate the operation of the cruise control. The nature of the data that is sensed (e.g., speed) is more relevant to the functionality than the way the data is formatted. In the development of the assembly diagram, the *decomposition* and *grouping* abstraction techniques are often employed. Frequently, the problem is partitioned into separate structural and functional component sets. The partitioning process usually exposes a hierarchical relationship among components. Exploring the hierarchical relationships aids in developing a deeper understanding of the problem area.

The assembly diagram for the electronic dashboard example is given in Appendix A-2. The dashboard has been decomposed into three major subassemblies: cruise control, speedometer, and the gauge cluster. The functionalities of each grouping are

- cruise control: maintain the setpoint speed of the vehicle
- speedometer: compute and display the average and instantaneous speed of the vehicle
- gauge cluster: monitor and display the status of important vehicle operational parameters

Identifying the constituent physical components can further refine each of the subassemblies. In this example, this includes the identification of broad categories—buttons, sensors, and displays. The decomposition into subassemblies is based on their perceived functional relationship to the physical subassembly. The functionality of each component is generally described in the System Requirements Specification (SRS) or similar document. The particular specifications involved and the level of detail will vary from project to project. Arrows show the dependencies of each subassembly. For example, the cruise control will need the information from the wheel rotation sensor in order to compute whether the correct speed is being maintained. The dependency is often data: What data is the component delivering to the subassembly? The assumption here is that the hardware component producing the data is usually contained within the hardware entity whose function we are attempting to understand. This may not always be the case—the data may be used by other hardware components. The assembly drawing is developed showing where each component is physically located (e.g., the circuit board or electronic enclosure). The flow of data will be better addressed in the discussion of the context diagram in the next section.

Keep in mind that this is the modeler's first step in trying to understand what the system is and how it is intended to work, and that the end goal is to construct an essential model. An essential model contains the necessary information to verify the intended behavior of the system. Producing the assembly diagram is the first step in this process.

The next step is to identify subcomponents within the assembly diagram that are part of the scope of the modeling and analysis effort. In addition we want to develop an operational understanding of how the components work together; specifically, what processing occurs and what major pieces of information flow through the system. Developing the context diagram helps to answer these questions.

3.3.2 Developing the Context Diagram

Having identified major hardware components and their structural associations, we now need to understand the intended operation of each component within the components. In addition, we need to begin to understand what variables those components accept, derive and manipulate, and if they must be visible within the model. It's generally useful to view components as having activation events and resultant intended actions. For example, input devices such as buttons are pushed by someone, and have the effect of signaling or causing something to happen. Output devices such as displays are activated by some signal (changing data in a register or refreshing) and result in displaying information. Determining these relationships is very helpful in constructing the context diagram.

The resultant context diagram for the cruise control example is shown in the Appendix D. The creation of this diagram resulted from carefully reading the specifications, determining (confirming) the cruise control entities, and determining their operation. It should be noted that there can be a hierarchical organization to context diagrams. Context diagrams define the focus of the area to be investigated. For the electronic dashboard application, Appendix C contains the top-level context diagram.

Abstraction is also used in developing the context diagram. It usually takes the form of *eliminating* non-essential information in order to succinctly state the component's functions. A technique often employed in developing the context diagram is *reduction*, in which non-essential details are removed, while keeping the information necessary to verify the system's behavior. *Decomposition, grouping*, and *variable elimination* are also techniques often used in this activity.

In the cruise control example, consider the reduction in the sensors in going from the dashboard context diagram to the cruise control context diagram. For the most part, this is a fairly straightforward pruning of unnecessary sensor input. In this example, sensors related to fuel level, oil pressure, engine temperature, and so on, do not have an effect on the cruise control application. The data provided by the wheel rotation sensor is used by both the cruise control and the speedometer/odometer components. It is therefore contained in the cruise control context diagram.

In some cases, key pieces of information, such as variables, are attached to a specific function. For example, in the electronic dashboard application, the function of the wheel rotation sensor is to update the speedometer and odometer. The key information provided by this sensor could be called VEHICLE_SPEED. As indicated in the statement of perspective, it is clear that the speed of the vehicle is a relevant entity. It is needed in the formation of expected properties and in developing claims.

Depending on how the VEHICLE_SPEED variable will be used in exercising the state diagram, more or less detail in its representation may be needed. The key principle is to minimize the complexity of the representation while preserving an appropriate fidelity of the

representation. For example, one could represent the VEHICLE_SPEED as discrete integers in the range from zero to 100 MPH. But a review of the specifications indicates that the allowable operating conditions of the cruise control unit are within 35-85 miles per hour. It would therefore seem reasonable to use the *enumeration* abstraction technique to subdivide the vehicle speed range into three distinct values. These would be

1. under_speed (less than 35 mile per hour)
2. within_speed (within the inclusive range of 35-85 mile per hour)
3. and over_speed (greater than 85 miles per hour)

It is important to realize that performing this enumeration allows the necessary information (the speed variable) to be present in the model by eliminating the detail in the range information. This also allows one to check the operational characteristic stated in the Statement of Perspective (i.e., should not be allowed to operate when the speed is below 35 MPH.) Other abstraction techniques that can be applied in this activity are *decomposition, reduction,* and *grouping.*

3.3.3 Developing the Issues List

Within the Understand activity, we are simultaneously updating the acronym and definition lists as we come across items particular to our area of analysis. We are also generating an issues list. Initially the issues list consists of notes to the engineer regarding a number of things that "don't seem quite right." It may contain all or some of the following:

- information/operations that are missing, incomplete, or conflicting
- questions about supplied information or operations
- insightful statements about design tradeoffs and techniques used
- observed defects in the documentation

Generating the list is cyclical, with questions being answered, revisited, and added. The unresolved items are carried into the next phase and investigated if relevant to the area being verified.

A sample Issues List for the cruise control application is shown in Appendix N. The general columns of the list include

- date: date the issue is discovered
- document title: the title (number and revision if applicable) and the location in the document where the issue is relevant
- description: an accurate but concise description of the issue
- resolution: who the issue was reported to and what the resolution is

This is an example organization of the issues list, columns should be added as needed to satisfy organizational requirements. The primary function of this list is to document ways in

which the specification may be incorrect, vague, or incomplete. Items on this list may eventually turn into "defects" that have been detected while developing a model. This is part of the strength of the MBV approach.

Briefly reviewing the issues presented in the example, we can provide a general categorization of the issues. The first issue addresses the fact that no active braking signal from the cruise control software appears to exist. One approach to regulating the speed of a vehicle is to apply the brakes in a controlled manner if the vehicle exceeds a speed setpoint. Another viewpoint is to let the drag of the transmission and engine slow the car down. Since no mention of either (or any alternative) method is described, it is a good idea to clarify this point. If braking is to be an overt action from the controller, this observation could be termed a defect in the specification. If the vehicle is to be slowed down by the drag of other components, then this issue amounts to nothing more than needing a clarification. Similarly, the third issue falls into the same category. No mention of wheel slip detection is discussed. The second issue is more of an implementation detail but could be important if fault detection or prevention software is to be developed as part of this application. It would not affect the modeling effort in any way.

3.3.4 Summary of Phase One Activities

Phase One activities can be summarized as

- generate an assembly diagram. This will help develop a working understanding of the major hardware components of the system being investigated.
- generate a context diagram. This will set the context for the problem being investigated and help develop a working understanding of the data flow of the subsystem.
- generate an issues list. This will bring to light structural and behavioral questions that need to be clarified.
- augment the acronym and definition list with any additional terms. This will aid one's understanding of the system and its behavior.

The goal of the Understand phase of the MBV process is to develop a working understanding of the hardware components and their physical organization with respect to each other; and to gain insight into the behavior of the application software as it interacts with the various hardware components. This understanding is accomplished by developing the various diagrams and artifacts. All of these artifacts will be used in the Compose phase. The issues list will serve as an aid to guide subsequent model composition activities, primarily because it highlights behavioral details that may be important for the actual model development. The carryover and use of these documents are further discussed in the Compose phase and illustrated in Figure 4.

3.4 Phase Two: Compose

Phase Two activities are aimed at generating the state chart (s) or state diagrams that will be translated into a formal modeling language. A state chart captures the behavioral information

required for verification. Figure 4 illustrates the information and activities associated with this phase.

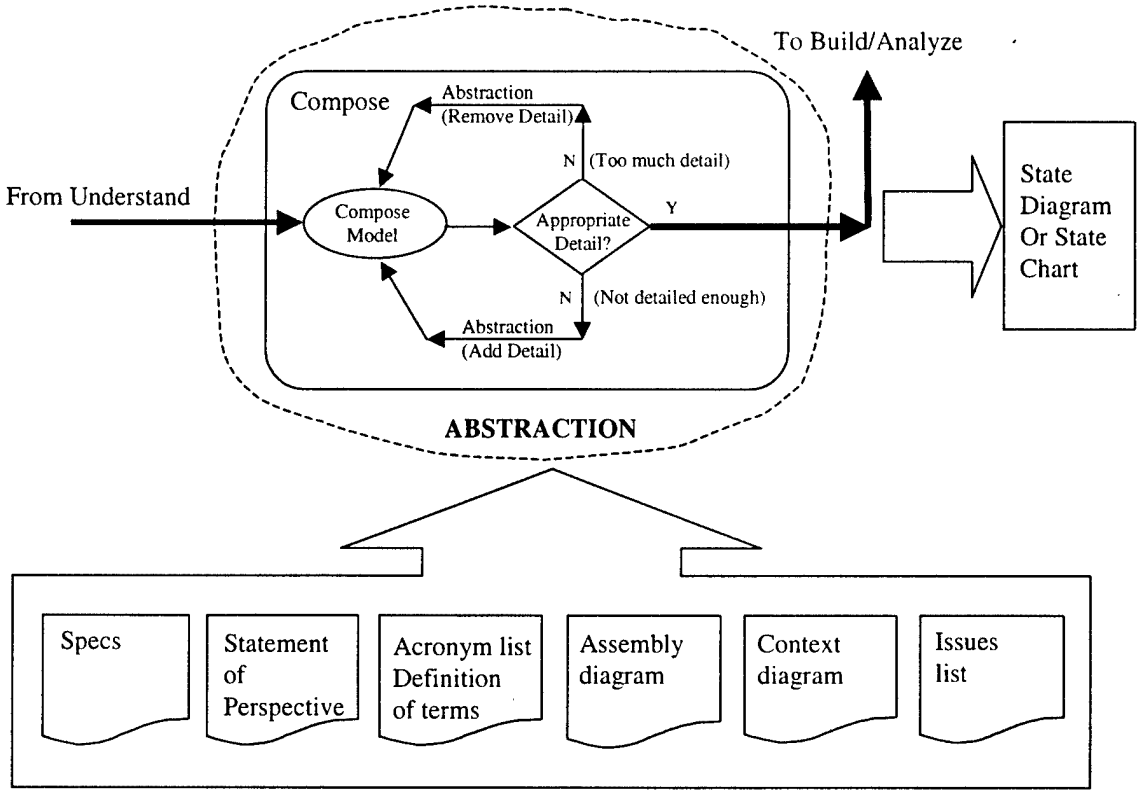


Figure 4: Phase 2 of the Build Activity: Compose

The goal of this phase is to generate an essential model. In this discussion, and for the cruise control example, we consider essential models that are represented as state charts or state diagrams [Booch 98]. As shown in Figure 4, the inputs from the Compose phase are documents that were either generated in the previous phase, or artifacts that resulted from project-level activities (as shown in Figure 1).

One point worth noting, however, is the importance of the Statement of Perspective. It serves as the guideline to the abstraction activity, addressing issues of what behavior needs to be explored, what components are relevant to the process, and what components are not to be considered. The modeler should frequently refer to the perspective statement during the Build activity to help resolve questions regarding components to include/exclude, behavioral aspects (e.g., fault tolerance, completeness), and detail. The more precise the Statement of Perspective, the more efficient the model-building activity can be.

It is sometimes the case that elements of the issues list must be addressed in order to develop a model that is representative of the intended behavior. In addition to the Statement of Perspective, the issues list should often be reviewed as well.

Composing an essential model (e.g., in this discussion developing a state chart or state machine representation) is an incremental and iterative activity as shown in the Compose bubble in Figure 4. It is incremental in the sense that initially a rather coarse-grained, high-level abstract representation of the behavior of the system is developed (assuming a top-down approach to developing the state representation). The work of extracting the additional information along with using the abstraction techniques to represent the entities is essentially what is depicted in the lower feedback loop in Figure 4 (the "Not detailed enough" path).

Sometimes it is the case that one has exposed too much detail and must reorganize it into a larger representative entity. This process is shown in the upper feedback path in the "Compose" block in Figure 4 (the "Too much detail" path).

Another observation worth mentioning is that iteration between Understand and Compose, as shown in Figure 1, is possible and even likely. It may be the case that although one has moved into the phase of composing models, some additional understanding of the problem domain is necessary. One must therefore refer back to the requirements, and perhaps modify the assembly or context diagrams. The amount of iteration is determined by many factors: primarily the familiarity of the engineer with the problem domain, but also the amount of detail contained within the perspective statement.

3.4.1 Abstraction and Model Composition

This section will discuss some guidelines of applying abstraction and how it relates to model composition. Not included in this section is a detailed method for state machine composition.

The Compose phase involves two interrelated considerations that must be simultaneously addressed throughout the process. These are

- determining and applying the appropriate level of abstract representations
- structuring the state chart or state diagram (essential model)

The Compose activity involves an evolution of the essential model that proceeds from an initial representation to a completed abstraction for analysis. The initial representation reflects preliminary judgments on the abstractions that are needed. As the questions identified above are addressed additional insight is gained and decisions are revisited and the models are enhanced. Often the initial model represents only a portion of the scope that is ultimately to be included. Developing a state chart can be a challenging activity because in many cases, determining the relevant states is not always straightforward.

3.4.2 Determining the Appropriate Abstract Representations

Understanding the problem is largely accomplished in Phase One. The input–output relationships are determined, structural (assembly) diagrams are composed, functional relationships are established in the context diagrams, and initial selections of variable names

and content are developed. The guidance for the exposure of these relationships and variables is contained in the statement of perspective.

There is a set of questions that can be used throughout the build processes that can help to ensure a solid understanding of the problem and to mature that understanding. Keeping these questions in mind while reading the specification helps organize the information and places the information in a relative priority. The specific questions for a particular problem are built from a set of basic forms. These are

- What are the conditions necessary for.....?
- What are the operational modes (phases, sequences, etc.) in.....?
- What is the chunk of data that is manipulated/determined/provided in .....?
- What is that chunk of data composed of?

Constantly asking and answering these questions while being guided by the perspective helps to establish the necessary information at the applicable level of granularity.

Once the problem is understood, it must be mapped into an abstract state machine representation. One way to begin is by enumerating the possible unique sequences of inputs or configurations of the system. These will help define the states of the state machine. The approach is to identify the entities and their relationships, the variables and associated events, and the identification of unique states. To keep the state diagram or state chart simple and readable, we include only transitions that explicitly cause a state change.

Optimizing the number of states is effectively creating the essential model. Only the number of states necessary to verify the expected properties should be derived. As the requirements are reviewed, many transition criteria and states may be exposed. Quite often, a group of states can be combined into a single abstracted state. This abstraction technique of mapping many to one is quite useful. This minimization (optimization) of states is seen in the loop decision path in Figure 4. Effectively, we are asking if we have exposed and modeled the behavior to the appropriate level of detail.

Let's demonstrate how applying the guidelines discussed above in the cruise control example can result in generating a state diagram. The first thing to do would be to outline the actions that occur when each input is activated. Suppose the cruise control application is switched from OFF to ON. Although not discussed in detail in the specification, one could envision the following behavior of the cruise control application:

- variables are initialized
- outputs are set to predetermined safe values
- a quiescent (idle) state is achieved
- the system is ready to execute some action

The requirements don't describe exactly what variables are to be initialized since these are internal to the implementation and need not be exposed at this level of abstraction. So at a very high level of abstraction, the cruise control can consist of two states: OFF and ON, where OFF could be viewed as the cruise control application is in a quiescent state but 'off line' and waiting for the action of the OFF-ON switch, and ON would indicate that the cruise control application is now 'on-line' (Appendix E). We now ask if this model at this level of abstraction is sufficient for this investigation. Is this level of detail sufficient given the example Statement of Perspective? The abstracted two-state model is not sufficiently detailed to allow the investigations indicated in the Statement of Perspective. One reason is that the Specific Guidelines section of the Perspective establishes the specific functions to be investigated (Set speed, Coast, Resume, and a quiescent state in which the throttle actuator is disengaged). These modes of operation will cause the vehicle to accelerate, coast or maintain some predefined speed. These are clearly some substate of the ON state.

To expand the model, we need to describe additional substates of the ON state that capture the behavior associated with the functions to be investigated. In the two-state model we have the notion of the cruise control being ON, in some neutral state, but no explicit actions are identified. One way of determining substates within a system is to consider the actions of the controlled devices. As seen in the cruise control context diagram (Appendix D), the throttle actuator is the only controlled device. It can be engaged or disengaged. Perhaps two substates within ON could be DISENGAGED and ENGAGED (Appendix F).

Is this level of detail sufficient given the example Statement of Perspective? To answer this, it is helpful to outline the actions that may occur when invoking the "Set Speed" function. This is fairly well described in the "Set Speed Functionality" section of the sample specification. There is the notion of some speed range for the function to be active. Presuming those predefined speed conditions are met, the set speed function would most likely capture the current speed of the vehicle, which would become the speed setpoint) and monitor the current instantaneous speed on a periodic basis to see if it is within the tolerance specified (+/- 2 MPH) of the speed. If the current instantaneous speed should fall below the threshold, the throttle actuator would need to send an 'engage' message to the throttle actuator controller to accelerate the vehicle. If the speed should go above the threshold, the throttle actuator would send a 'disengage' message. This suggests that the ENGAGED state could be composed of three substates: ACCELERATING, DECELERATING, and MAINTAINING (Appendix G). Note there is no discussion in the specification that describes a function of the cruise control to apply the brake to slow the car or any reference to a controlled deceleration, so it is not really an overt control action. For this particular system, the drag produced by the transmission is sufficient to decelerate the vehicle in a smooth manner. Therefore disengaging the throttle actuator will cause the vehicle to gradually reduce its speed. The behavior of deceleration captured in the DECELERATING state can effectively be encapsulated in the DISENGAGED state. The reduction technique (Section 2.3) has been used to abstract the behavior into three states and eliminate the DECELERATING state (Appendix H).

An acceptability test is again applied to the three states just described. This is performed by determining if the provisions for the behavior described in the statement of perspective are included. We can accommodate the three enumerated values of the speed range (under_speed, within_speed, over_speed) developed in Section 3.3.2— Developing the Context Diagram. This can be checked by noting the behavior that would occur when each of the speed range conditions exist. For the under_speed and over_speed cases, the cruise control would remain in a disengaged state. For the within_speed case, the vehicle could accelerate to a current instantaneous speed within a tolerance band, by entering the ACCELERATING state. Once the current instantaneous speed is within the tolerance band, a MAINTAINING state would be entered. Should the current instantaneous speed fall below the tolerance band, the vehicle would need to accelerate, thus re-entering the ACCELERATING state. Should the current instantaneous speed exceed the tolerance band, the vehicle would need to be slowed, and this would be accomplished by transition to the DISENGAGED state. If the current instantaneous speed would then fall out of the tolerance band, the vehicle would need to accelerate, causing the transition to the ACCELERATING state.

The state machine that satisfies the behavior specified in the Set Speed Functionality section of the Software Requirements-Electronic Dashboard (Appendix J) is depicted in Appendix H. The diagram shows the states, and the transitions needed to transition to the other states. The transition labels (Disengage, Accelerate, Maintain, etc.) describes the events, but not the conditions that cause the event to occur. These conditions, sometimes called 'guards' must be satisfied in order to effect the transition. For example, using the Set Speed function, one can write a guard based on the operational description in the Set Speed Functionality section of the Software Requirements-Electronic Dashboard (Appendix J). The first condition called out in the Set Speed Functionality is

> "If the instantaneous speed of the vehicle is greater than 35 MPH and less than or equal to 85 MPH and the cruise control is disengaged, engaging the SET SPEED button shall result in the cruise control maintaining the current instantaneous speed."

The one of the guards (there could be more due to conditions when in other modes, i.e., Resume, Coast, etc.) on the Accelerate arc from Disengaged to Accelerating that would capture the conditions of the cruise control specification would look like

Accelerate = (mode=set_speed & 35<instantaneous_speed<=85)

In a similar fashion, each condition in the Set Speed function would be reviewed and applied to the appropriate event in the state diagram.

As a side issue, when the actual model is implemented in a modeling language, these guard conditions are expressed within the model. Additional abstractions are made in the expression of the guards. For example, the enumeration technique, described in Section 2.3 can be applied to the "35<instantaneous_speed<=85" portion of the guard condition. The

phrase could be re-cast to a Boolean variable 'cc_operational_range' that would be true if the speed was greater than 35 MPH and less than or equal to 85 MPH. Additional reasons to consider abstraction of model variables is discussed in [Lewis 01].

The state diagram that has been developed thus far is most likely incomplete because the additional functionality expressed in the Accelerate, Coast, Disengage, and Off descriptions has not been fully investigated. From having developed the artifacts in the Understand (Section 3.3) and Compose (Section 3.4) sections, a good comprehension of the behavior of the system has been developed. To review, the assembly diagram of the automobile basically depicts the entities associated with the cruise control system (i.e., buttons, switches, sensors, actuators, and displays). Having composed the assembly diagram allows one to develop the relationships among the entities (i.e., an entity can provide information to something, the information can be represented in a variable). The relationships can also reflect the notion of classes of objects that have some generic properties applicable to any reference to the object (i.e., an on-off switch is a specific instance of two-position, normally open switch). The information contained in the context diagram for the cruise control illustrates how the entities can change the behavior of the cruise control system as characterized by causing interrupts and eventually sending messages to the cruise control application. The controlled entities (e.g., throttle actuator) are also displayed in the context diagram. In general, the changes in variables signify an event, which can sometimes cause a change in state behavior. For example, causing the output to the throttle actuator to increase will cause the vehicle to accelerate. Identification of variables associated with each intended function of the cruise control will help in determining the appropriate states necessary to describe the intended behavior. The state is a set of values for the attributes that define the behavior of the cruise control system. As shown previously, investigating the Set Speed function revealed that changes to the throttle actuator would result in the vehicle accelerating or decelerating. The state diagram developed to this point can be refined by reviewing each of the remaining operational scenarios of the cruise control application (e.g., Accelerate, Coast, Disengage, Resume, and Off) and identifying any additional events that would cause transitions to existing states and/or help to identify new state. Applying the questions outlined in Section 3.4.2 will help to identify the variables necessary for modeling. Identifying those variables and the behavior associated with them may lead to the identification of additional states. Using this general approach results in the state diagram show in Appendix I.

Having arrived at a state diagram that appears to reflect the behavior of the system necessary for analysis, we can review it for completeness with respect to the intended analysis (Figure 4). This section (3.4) has provided guidelines that help in the construction. This information contained in the perspective statement should be reviewed and the state machine behavior evaluated with respect to the behavioral issues contained in the statement of perspective. For example, one of the issues in the Statement of Perspective addresses the completeness of the operational states. This implies that the actions of each input should result in the intended behavior, as outlined in the specifications. Each input can be systematically checked. As another example, certain functions can only happen within a predefined range. Are the

events described by the change in the variables contained within the state diagram? Performing similar reviews of each point in the perspective statement will help to ensure a complete state diagram. There will be cases, however, where during subsequent analysis activities some information will be incomplete, most often the addition of variables necessary for composing claims against the model. This will necessitate augmenting the state model with additional variables (and perhaps additional states). This modification activity is depicted in Figure 1, in the Engineering Activities portion of the chart, specifically the Build, Analyze, and Compile activities.

3.4.3 Obtain an Abstract Representation of the State Machine

With an essential state diagram, one can translate the states into the language of the modeling tool. At this point, claims can be crafted to validate the intended behavior expressed in the expected properties. (See the technical note that describes how to craft and express the claims [Dorda 01].)

It should be noted that producing a large state machine might give rise to a state explosion problem. In a state explosion, the size of the model exceeds the capabilities of a model-checking tool; that is, impractically large execution times or computer memory requirements or both are required for the analysis. Should this occur, it will be necessary to revisit the abstraction process to further reduce the number of states. The state explosion issue and resolution techniques are further discussed in Lewis [Lewis 01].

4 Summary

Abstraction is a fundamental process for reducing the complexity of a representation and aiding in system analysis. It is applied both to grasp and efficiently combine the essential elements of a system. The perspective, which is used in determining what parts of the problem are necessary and what parts can be hidden, drives the abstraction process.

The process for building an essential model is divided into the two phases of Understand and Compose. These provide a framework for developing an essential model. The Understand phase involves the use and development of artifacts that result in intermediate representations of the system and its components. A number of abstraction techniques are employed throughout this phase. The Compose phase uses the results of the Understand phase to develop essential state machine models of the system. Abstraction techniques are used iteratively in this phase as part of a process of assessing how well an abstracted model meets the expectations of the perspective, as defined in of the Statement of Scope, Formalism, and Perspective.

Abstraction processes are not simple sequential algorithmic processes. Consequently, both phases of the Build activity involve multiple iterations. These include iterations among activities within and between the phases. One of the main issues to be addressed throughout the Build activity is the significance of the information presented within the relevant project documentation (e.g., SRS). In particular, this involves determining just how much detail is needed and how far to proceed in exposing, eliminating, and rearranging system components.

Our experiences with pilot projects have shown that domain knowledge and the quality of the perspective that is defined in the Statement of Scope, Formalism, and Perspective are important components in knowing when to stop the modeling process. This information can go a long way to establishing when enough information has been abstracted to expose the elements necessary for verification.

Our experiences gained in pilot projects have also provided insight into a methodology that can aid in efficiently and effectively generating models. Artifacts have been identified that provide useful information to the individual modeler, as well as expose defects, even though a verification effort was not specifically aimed at revealing those types of defects. The methodology described in this technical note may be influenced and subsequently modified as more application systems are analyzed employing MBV.

Appendix A Hardware Architecture: Electronic Dashboard

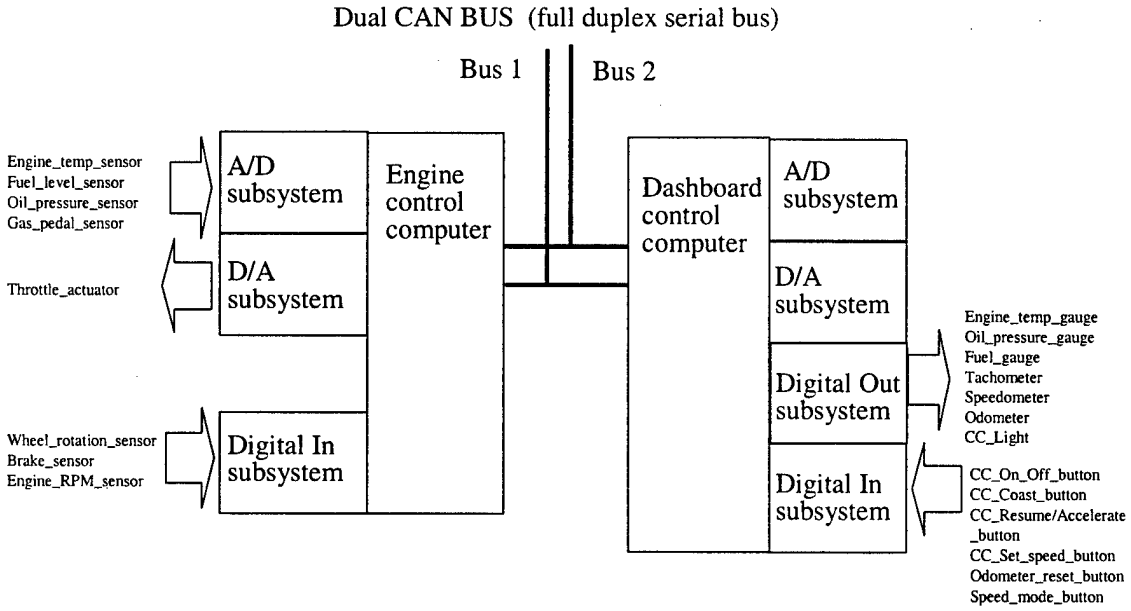


Figure 5: Dashboard/Cruise Control Hardware Architecture

Appendix B Assembly Diagram: Electronic Dashboard

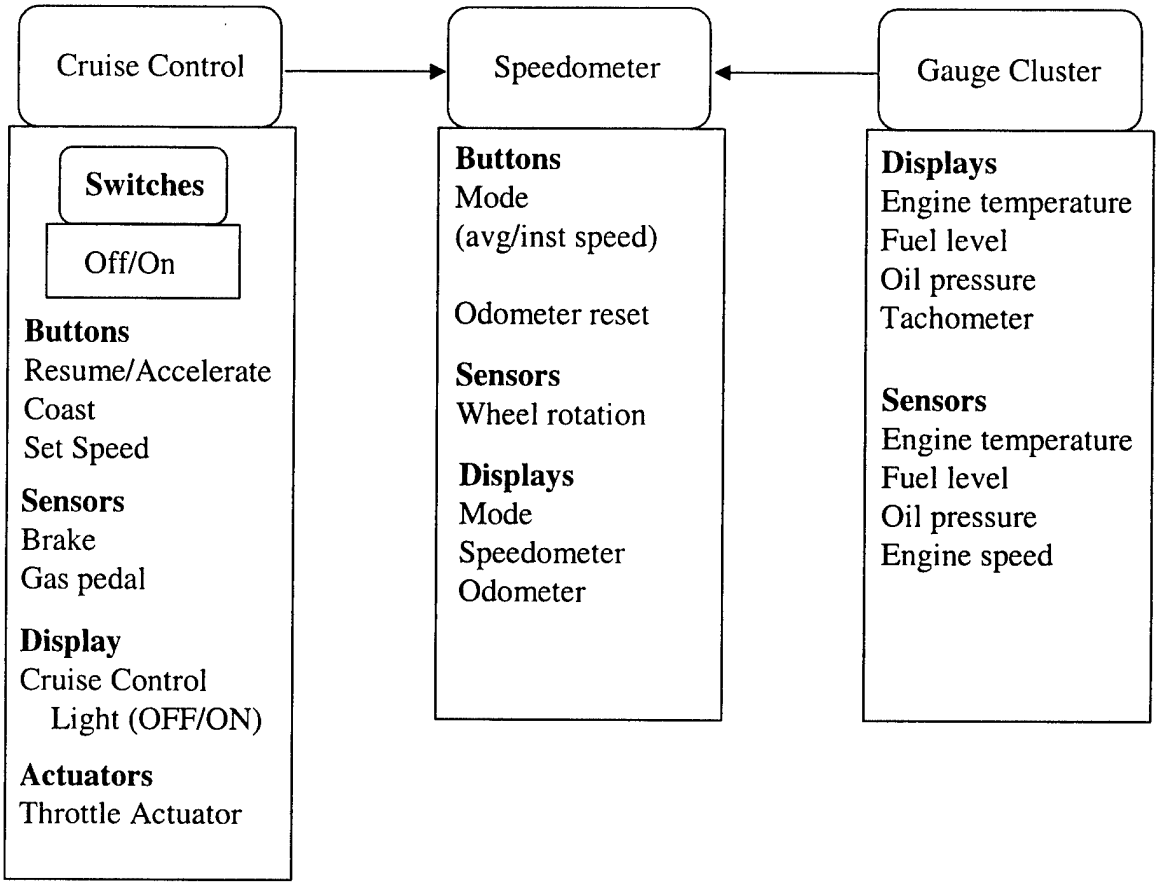


Figure 6: Assembly Drawing—Automobile Dashboard and Cruise Control

Appendix C Context Diagram: Electronic Dashboard

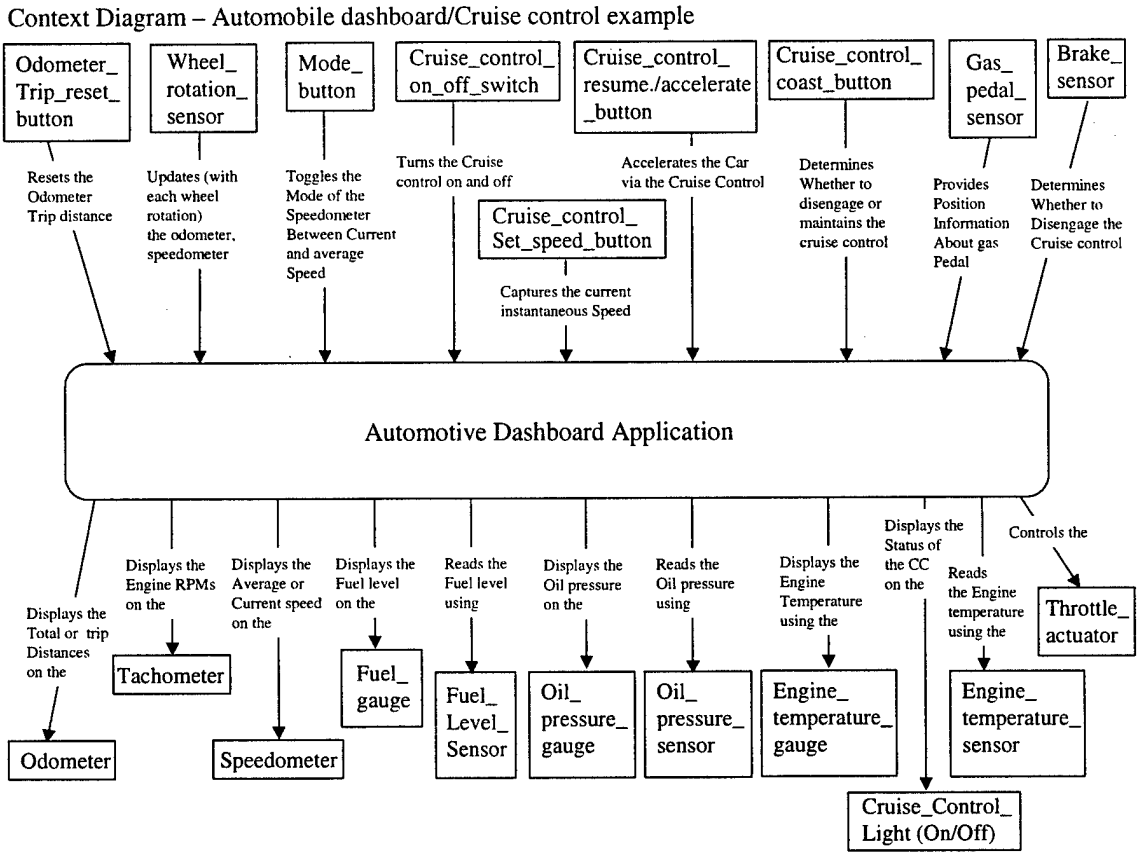


Figure 7: Context Diagram—Automobile Dashboard/Cruise Control Example

Appendix D Context Diagram: Cruise Control

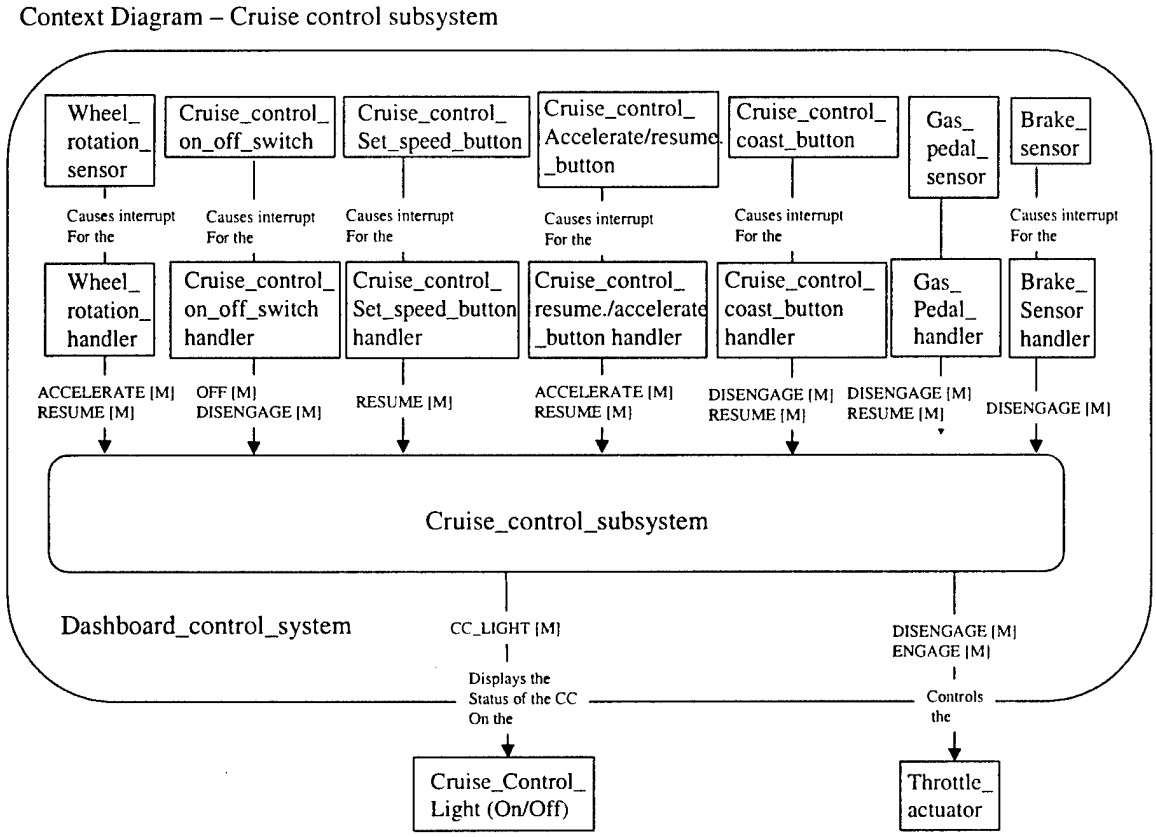


Figure 8: Context Diagram of the Cruise Control

Appendix E Simplified State Diagram: Cruise Control

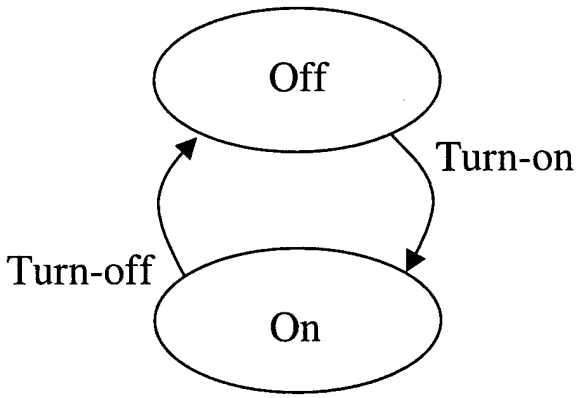


Figure 9: A Simplified State Diagram of the Cruise Control

Appendix F Expanded State Diagram: Cruise Control

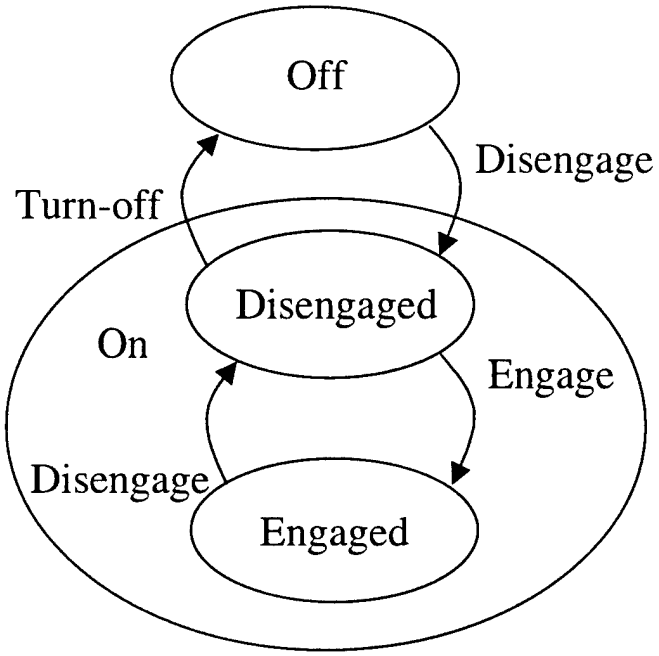


Figure 10: Expanded State Diagram Showing Hidden States

Appendix G State Diagram: Cruise Control

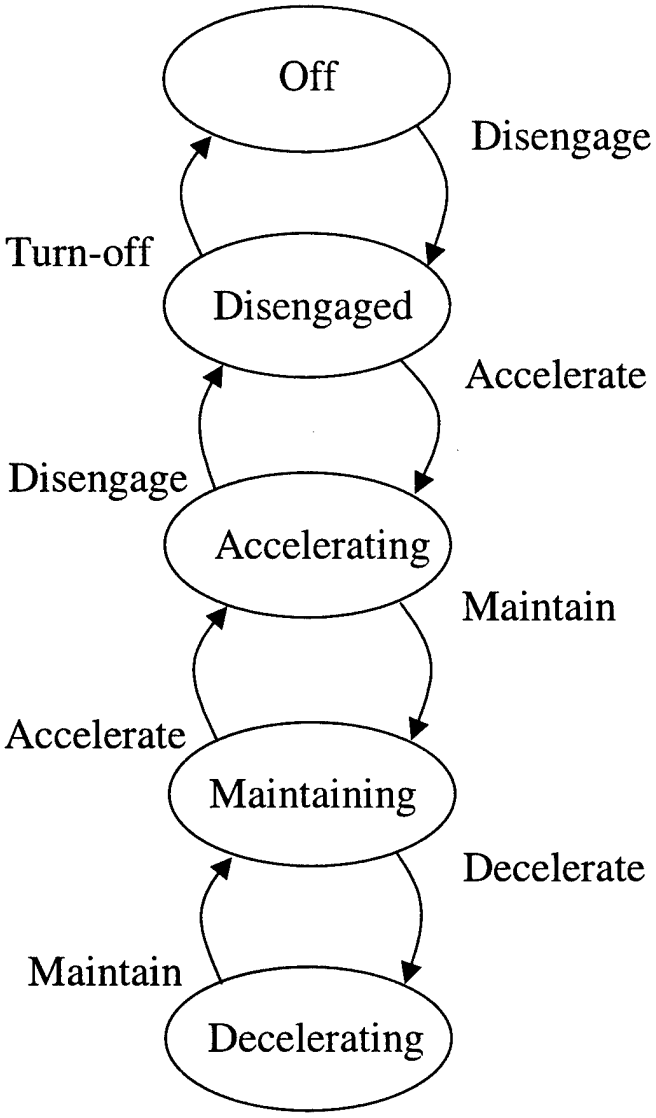


Figure 11: Expanding the Speed Setpoint Behavior

Appendix H Collapsed State Diagram: Cruise Control

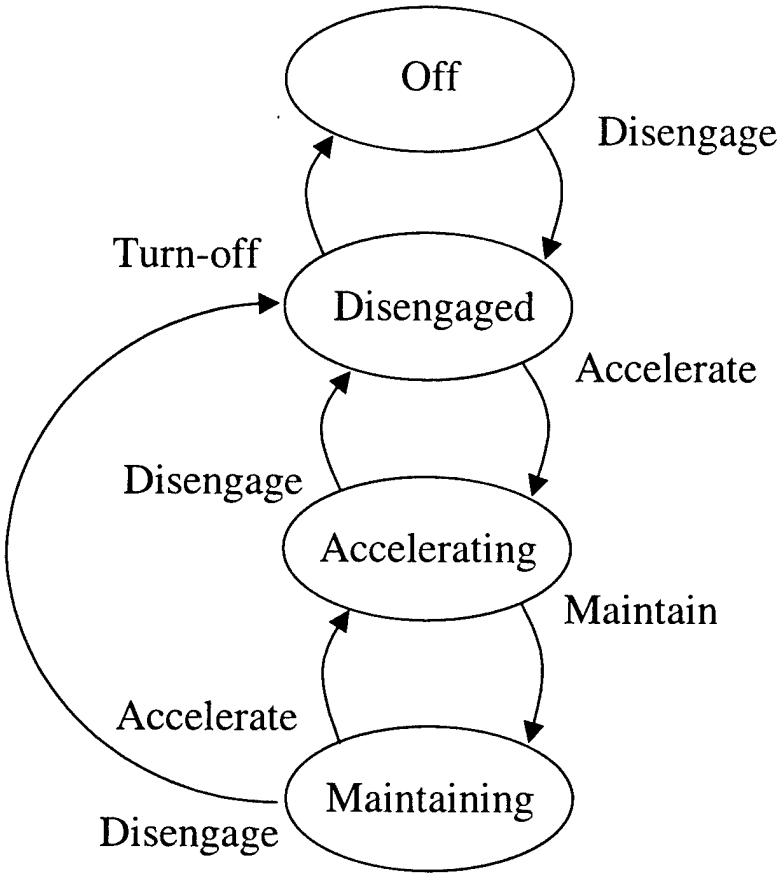


Figure 12: Collapsing the Speed of Setpoint Behavior

Appendix I: Complete State Diagram: Cruise Control

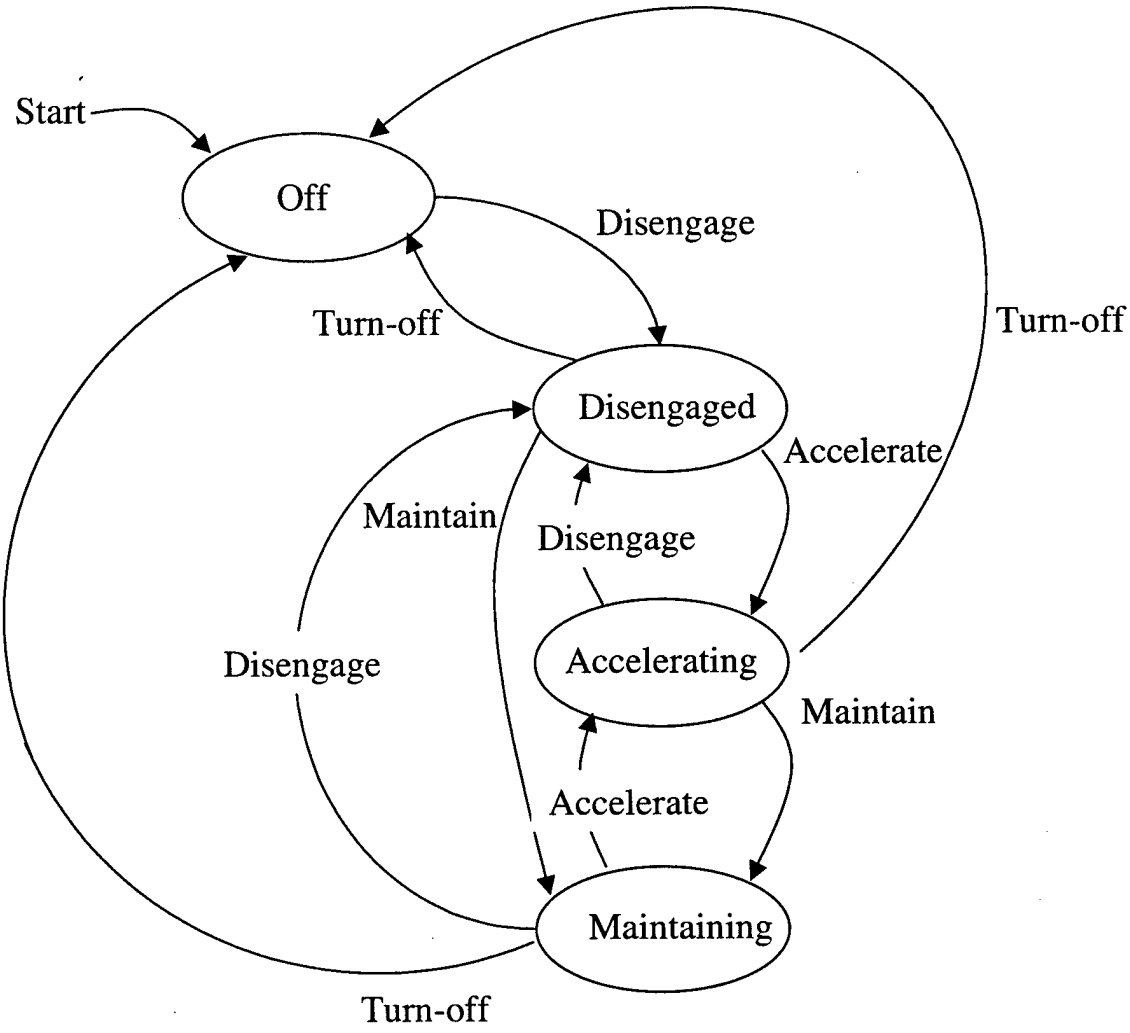


Figure 13: Complete State Diagram of the Cruise Control

Appendix J Software Requirements: Electronic Dashboard

This example software requirements specification describes the overall functionality of an electronic dashboard for an automobile. The main purpose of this example is to describe in detail the functionality of the cruise control system and its operation.

System Description

The purpose of the automobile dashboard application assembly is to

1. provide the user interface functionality (input and output) for the cruise control
2. display the specified gauges
3. update the values of the gauges using data from the appropriate sensors
4. provide the functionality for the specified inputs (buttons, switches, etc.)

The Assembly diagram for the dashboard application can be seen in Appendix C.

The dashboard shall consists of three subassemblies:

- cruise control
- gauges
- speedometer

Cruise Control Subsystem Description

The cruise control subassembly is to implement the cruise control functions by grouping and controlling the visibility of all software components that provide the cruise control capability.

Hardware Interface—Data Input

Data input into the cruise control is via the following hardware interfaces:

- Cruise control acceleration button: a two-position, spring return button that provides a binary indication of the acceleration button being ON (Accelerate) or OFF (resume previous control mode). The 'rest' or 'normal' position of the switch is OFF.

- Cruise control coast button: a two-position, spring return button that provides a binary indication of the coast button being either ON (pushed in) or OFF (out).

- Cruise control on-off switch: a two-position, detent switch that provides a binary signal to cruise control application to be either ON or OFF.

- Cruise control set speed button – a two-position, spring return button (press to 'set speed') that provides a binary indication to signal the cruise control that the current vehicle speed is to be used as the desired speed setpoint for the cruise function.

- Gas pedal sensor: an absolute range of numeric values indicating the position of the gas pedal.

- Brake pedal sensor: a two-position, spring return button that provides a binary indication of the brake pedal being depressed or released.

Hardware Interface - Data Output

Data output to the cruise control is via the following hardware interfaces:

- Throttle actuator: a linear displacement actuator connected to the throttle.

- The cruise control ON/OFF indicator: this digital output shall be asserted when the cruise control is turned ON.

User Interface Control Requirements

The user of the cruise control system interacts with it using the following input devices:

- cruise control ON/OFF switch: enables the cruise control application

- cruise control accelerate/resume button: enables the vehicle to resume to a previously set speed function if the cruise control is in the disengaged mode, or, accelerates the vehicle if in the maintaining speed mode.

- cruise control coast button: enables the coast function if the cruise control is in the maintaining speed or accelerating mode.

- cruise control set speed button: the button is depressed to set the current speed of the vehicle as the desired speed setpoint.

Required Capabilities

This section describes the desired functionality of the cruise control. Power-up and power-down sequences are not discussed in this specification.

Cruise Control ON/Enabled Functionality

Upon enabling of the cruise control application, the states of the input devices should be read by the controller, and the throttle actuator should be disengaged (moved to a neutral position) by sending a disengage message to the throttle actuator controller, and the cruise control ON light shall be illuminated. The cruise control shall not be allowed to operate if any of the following conditions occur:

- The state of the input devices does not match the required state for startup (Accelerate/Resume button is OPEN, and, the coast button is OPEN, and the brake pedal is DEPRESSED, cruise control watchdog timer is OK).

- The instantaneous speed of the vehicle is less than or equal to 35 miles per hour.

Set Speed Functionality

If the instantaneous speed of the vehicle is greater than 35 MPH and less than or equal to 85 MPH and the cruise control is ON, pushing the SET SPEED button shall result in the cruise control recording and maintaining the current instantaneous speed as the desired speed (speed setpoint).

If the current instantaneous speed becomes greater than 2 miles per hour (+ 2 MPH), the cruise control shall decelerate the vehicle by disengaging the throttle actuator until an increase in speed is called for.

If the current instantaneous speed becomes less than 2 miles per hour (- 2 MPH), the cruise control shall accelerate the vehicle by sending an accelerate message to the throttle actuator.

Resume Speed Functionality

If the current state of the cruise control is disengaged and the current instantaneous speed of the vehicle is greater than 35 miles per hours, and the Accelerate/Resume button is engaged, the cruise control application shall send an accelerate message to the vehicle to the previous speed setpoint. The acceleration rate shall be 5 feet/sec**2. The vehicle will continue to accelerate until one of two conditions occur:

- The brake pedal is depressed: at which point the cruise control application will disengage the throttle actuator.

- The previous speed setpoint is reached. At this point the cruise control will maintain the current speed setpoint.

Coast Functionality

If the cruise control is accelerating to a current speed setpoint or maintaining a current speed setpoint and the coast button is pressed (actuated), the cruise control shall immediately disengage the throttle actuator for the duration that the coast button is engaged. Releasing the coast button shall result in the cruise control attempting to maintain the desired speed setpoint.

Disengage Functionality

The cruise control shall enter a disengaged state if any of the conditions occur, no matter what state the cruise control is in:

- the brake pedal is depressed. The current desired speed is saved in memory.

- the watchdog timer associated with the cruise control times out (a fault has occurred). The current desired speed is not saved in memory.

The Off Functionality

The cruise control shall immediately enter the off state when the ON/OFF button is moved to OFF, no matter what the state the cruise control. It will send the disengage message to the throttle actuator, and set the internal desired speed to zero.

The Accelerate for Passing Functionality

To accelerate while the cruise control application is maintaining a desired speed setpoint, depress the accelerator. When the pedal is released, the vehicle shall return to the set desired speed.

Fault Detection and Handling

A fault is detected in the cruise control application by the following:

Watchdog Timer

If the watchdog timer is not reset within the predefined time limits, the cruise control application shall perform the following actions:

1. Immediately disengage the throttle actuator.

2. Blink the cruise control ON indicator at a 1-second duty cycle.

Special Requirements

This section describes special requirements or operational issues of the cruise control application.

Digital Inputs

The digital inputs of the cruise control shall be interrupt driven. This includes the following:

1. Cruise control ON/OFF switch
2. Coast button
3. Accelerate/Resume button
4. Set Speed button
5. Gas pedal sensor
6. Brake sensor

Moving Down Hills

When the vehicle is moving down hills, it is possible for the vehicle to gain speed, even though the cruise control is engaged. Since the brake is not controlled by the cruise control system, it is necessary for the operator to apply the brake to slow the vehicle, thereby disengaging the cruise control (which will disengage the throttle actuator).

Passing a Vehicle

In order to pass a vehicle with the cruise control engaged, the operator can press the gas pedal to accelerate the vehicle. Upon releasing the gas, the vehicle will coast to the currently set speed and the cruise control will operate to maintain the current speed setpoint.

Appendix K Statement of Scope: Electronic Dashboard

The Model-Based Verification activities should focus on the cruise control subsystem of the electronic dashboard. Components of this subassembly include the switches (On/Off), buttons (Resume/Accelerate, Coast, Set Speed), sensors (Brake, Gas pedal), as well as the display (Cruise control on/off) and the throttle actuator. The software that we are investigating is resident on the dashboard control computer, although some variables are needed from the engine control computer. In addition, the throttle actuator contains control laws that respond to messages sent to it. The control laws are not to be investigated.

Rationale: These systems form the basis for the cruise control functionality.

Appendix L Statement of Formalism: Electronic Dashboard

State machine modeling and the SMV model-checking tool will be used. Other approaches and tools may be applied as needed. These will be assessed based upon the results of the analysis effort. Changes will be made as appropriate through the normal project tracking and planning processes.

Appendix M Perspective Statement: Cruise Control

General Description

The object of the verification effort is to check the cruise control for proper operation. To that end, the following characteristics are to be investigated and verified:

1. The investigation is to focus on the consistency of the operational states of the cruise control. Specifically looking at the conditions for entering and exiting various operational states, and also ensuring that no matter what operational state is active, there is always a way to return to the off state.
2. Explicitly check that depressing the brake pedal will disengage the cruise control system, no matter what state the cruise control is in.
3. Explicitly check that between the speed range of 0<=x<=35 miles per hour that the cruise control will not activate.

Specific Guidelines

Consider the following critical aspects and issues:

1. There is no 'memory' of states from the previous on condition when the system has been turned off, then on again. When this occurs, the cruise control application should enter a quiescent state, with the throttle actuator disengaged.
2. outputs and associated attributes (persistent, transient, periodic, etc.)
3. Validate the 'normal' modes of operation: Setting and maintaining a speed setpoint, coasting from a maintained speed setpoint, resuming to a speed setpoint, and disengaging whenever the brake pedal is actuated.
4. Validate the allowed transitions: disengage to setting a setpoint.
5. Any abnormal situation should result in disengagement of the cruise control.

Issues Requiring Attention

An interesting aspect of the design is that all the push button actuations are implemented via interrupts. The effects of the interrupts will be conditioned via the implementation to be persistent for period of time. Are there any combinations of input and persistence that need to be guarded against?

Another aspect is that there is no explicit means to decelerate the vehicle. The throttle controller contains logic to accelerate the vehicle at a certain rate when it receives an engage message. The throttle actuator controller will gradually increase the throttle according to a control law programmed in the throttle actuator controller. The throttle actuator controller responds to a disengage message by disengaging the throttle actuator and allowing the drag of the transmission to slow the vehicle. How is this condition handled and does it compromise the safe operation of the system?

System Attributes Specifically Ignored

Ignore the interaction with the other monitored engine variables, as well as messages from other computers on the bus. Ignore the condition of multiple faults within the system.

Appendix N Issues List

Project Name: Automobile Electronic Dashboard
Engineer: M.V.B.

Date	Document And location	Description	Resolution
2/12/02	Electronic Dashboard SRS Page 4, paragraph 1.2.6 Data Output	There is no mention of any device to perform braking nor description of braking. Is any such device present in this implementation? If the vehicle coasts down hill with the CC on, it could overspeed.	Submitted to Review committee 2/15/02. Pending resolution.
2/22/02	Electronic Dashboard SRS Page 3, paragraph 1.2.5.2 Speed Set Functionality	Are the values of 35 MPH and 85 MPH to be hard coded or will they be modifiable in the field by the appropriate diagnostic/programming device?	Submitted to Review committee 2/15/02. Resolution: values to be hard coded.
3/4/02	Electronic Dashboard SRS Page 3, paragraph 1.2.5.2 Speed Set Functionality	It seems possible that the wheel on which the rotation sensor is mounted could slip on ice, resulting in providing an incorrect indication of the speed of the vehicle. No discussion in spec about this condition. Does this need to be looked at in hardware? Does the application software need to address this issue?	Submitted to Review committee 3/8/02. Resolution: pending

References/Bibliography

[Bharadwaj 97] Bharadwaj, R. & Heitmeyer, C. "Verifying SCR Requirements Specifications Using State Exploration." *First ACM SIGPLAN Workshop on Automatic Analysis of Software*, Paris, France, Jan 14, 1997. New York, NY: Association for Computing Machinery, 1997.

[Bharadwaj 99] Bharadwaj, R. & Heitmeyer, C. "Model Checking Complete Requirements Specifications Using Abstraction." *Automated Software Engineering 6*,1 (Jan.1999): 37-68.

[Booch 98] Booch, Grady; Rumbaugh, Jim; & Jacobsen, Ivar. *The Unified Modeling Language Users Guide*, New York, NY: Addison-Wesley, 1998.

[Clancy 94] Clancy, D. & Kuipers, B. "Model Decomposition and Simulation," *Working Papers of the Eighth International Workshop on Qualitative Reasoning of Physical Systems.* Orcas Island, WA, 1993.

[Clarke 95] Clarke, E.M.; Grumberg, O.; Hirashi, H.; Jha, S.; Long, D.E.; McMilland, K.L.; & Ness, L.A. "Verification of the Futurebus+ Cache Coherence Protocol." *Formal Methods in System Design 6*, 2 (March 1995): 217-232.

[Clarke 96] Clarke, E.M. & Wing, Jeannette, "Formal Methods: State of the Art and Future Directions." *ACM Computing Surveys 28*,4 (December, 1966): 626-643. Also, (CMU-CD-96-178) Pittsburgh, PA: Computer Science Department, Carnegie Mellon University, 1996.

[Clarke 98] Clarke, Edmund; Berezin, Sergey; & Campos, Sergio. "Compositional Reasoning in Model Checking," 81-102. COMPOS'97. Bad Malente, Germany, September 7-12, 1997. *Lecture Notes in Computer Science 1536*, New York, NY: Springer Verlag, 1998.

[Comella 01] Comella-Dorda, Santiago; Gluch, D.;J. Hudak, J.; Lewis, G.; & Weinstock, C. *Model-Based Verification: Claim Creation Guidelines.* (CMU/SEI-2001-TN-018 ADA396125). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, Oct., 2001. <http://www.sei.cmu.edu/publications/documents/01.reports/01tn018.html>

[Firesmith 93] Firesmith, Donald, G. *Object Oriented Requirements Analysis and Logical Design.* New York, NY: John Wiley and Sons, 1993.

[Frantz 95] Frantz, Frederick K. "A Taxonomy of Model Abstraction Techniques," 1413-1420. *Proceedings of the 1995 Winter Simulation Conference,* Arlington, VA, Dec. 3-6, 1995. New York, NY: Association for Computing Machinery, 1996.

[Gluch 98] Gluch, D. P. & Weinstock, C. B. *Model-Based Verification: A Technology for Dependable System Upgrade* (CMU/SEI-98-TR-009, ADA 354756). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1998. <http://www.sei.cmu.edu/publications/documents /98.reports/98tr009/98tr009abstract.html>

[Gluch 01] Gluch, D.; Comella-Dorda, S.; Hudak, J; Lewis, G.; &. Weinstock, C. *Model-Based Verification—Scope, Formalism, and Perspective Guidelines* (CMU/SEI-2001-TN-024 ADA396628). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. <http://www.sei.cmu.edu/publications/documents/01.reports/01tn024.html>

[Gluch 02a] Gluch, D.; Comella-Dorda, S.; Gluch, D.; Hudak, J; Lewis, G. & Weinstock, C. *Model-Based Verification—Guidelines for Generating Expected Properties* (CMU/SEI-2002-TN-003 ADA3399228). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. <http://www.sei.cmu.edu/publications/documents/02.reports/02tn003.html>

[Gluch 02b] Gluch. D.; Comella-Dorda, S.; Hudak, J.; Lewis, G; Walker. J.; Weinstock, C.; & Zubrow, D. *Model-Based Verification: An Engineering Practice* (CMU/SEI-2002-TR-021) Pittsburgh, PA: Software Engineering Institute, Carnegie-Mellon University, 2002. <http://www.sei.cmu.edu /publications/documents/02.reports/02tr021.html>

[Heitmeyer 98] Heitmeyer, C.; Kirby, J.; Labaw, B.; Archer, M.; & Bharadwaj, R. "Using Abstraction and Model Checking To Detect Safety Violations in Requirements Specifications." *IEEE Transactions on Software Engineering 24*, 11 (Nov. 1998):932-941.

[Hsieh 98] Hsieh, Yee-Wing & Levitan, Steven P. "Model Abstraction for Formal Verification." *Design, Automation and Test in Europe Conference (1998):* 140-147. Los Alamitos, CA: IEEE Computer Society Press, 1998.

[Jackson 01] Jackson, Michael, *Problem Frames*. New York, NY: Addison-Wesley, 2001.

[Jackson 95] Jackson, Michael. *Software Requirements & Specifications-a Lexicon of Practice, Principles and Prejudices*. New York, NY: Addison-Wesley, 1995.

[Lewis 01] Lewis, G.; Comella-Dorda, S.; Gluch, D.; Hudak, J.; & Weinstock, C. *Model-Based Verification: Analysis Guidelines* (CMU/SEI-2001-TN-028 ADA399318). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, Dec., 2001. <http://www.sei.cmu.edu /publications/documents/01.reports/01tn028.html>

[Pressman 97] Pressman, Roger S., *Software Engineering: A Practitioner's Approach*, 4th ed. New York. NY: McGraw-Hill, 1997.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	5. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	October 2002	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Model-Based Verification: Abstraction Guidelines	F19628-00-C-0003

6. AUTHOR(S)

John Hudak, Santiago Comella-Dorda, David P. Gluch, Grace Lewis, Chuck Weinstock

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2002-TN-011

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

Model-Based Verification (MBV) is a systematic approach to finding defects (errors) in software requirements, designs, or code. The approach judiciously incorporates mathematical formalism, in the form of models, to provide a disciplined and logical analysis practice, rather than a "proof of correctness" strategy.

This technical note presents a number of abstraction techniques that can be used to build essential models of system behavior in the context of MBV and details a methodology for creating state machine models using those techniques. In building essential models, abstraction is used to hide details and expose the entities, variables, states, and transitions needed to construct a state machine model. Through illustrative examples, this technical note identifies the types of simplifications that are useful and effective and highlights the importance of the perspective in determining what are the important elements to include in an abstracted model.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Model-Based Verification, MBV, abstraction, decomposition, variable elimination, enumeration, reduction, non-determinism, grouping	54

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL