Model-Based Verification: An Engineering Practice

David P. Gluch
Santiago Comella-Dorda
John Hudak
Grace Lewis
John Walker
Charles B. Weinstock
David Zubrow

October 2002

TECHNICAL REPORT
CMU/SEI-2002-TR-021
ESC-TR-2002-021

Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

Model-Based Verification: An Engineering Practice

CMU/SEI-2002-TR-021
ESC-TR-2002-021

David P. Gluch
Santiago Comella-Dorda
John Hudak
Grace Lewis
John Walker
Charles B. Weinstock
David Zubrow

August 2002

Performance-Critical Systems

This report was prepared for the

SEI Joint Program Office
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Norton L. Compton

Norton L. Compton, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

Abstract .... **v**

1 Introduction **1**

2 Project-Level Activities **5**
2.1 Project-Level Artifacts 5
2.2 Focusing MBV Activities 6
2.2.1 Scope 7
2.2.2 Formalism 7
2.2.3 Perspective 8
2.3 Team Processes 9
2.4 Programmatic and Technical Considerations 10
2.5 Fine-Tuning Scope, Formalism, and Perspective 11
2.6 Completing the Team Activities 12

3 MBV Engineering Activities **13**
3.1 Output Artifacts of the Engineering Activities 14
3.2 Guidelines for Engineering Activities 14
3.3 Build 15
3.3.1 Essential Models 16
3.3.2 Prepare 16
3.3.3 Build and Refine 17
3.3.4 Translate 19
3.4 Analyze 19
3.4.1 Generate Expected Properties 20
3.4.2 Conduct Analysis 24
3.4.3 Interpret Results 25
3.5 Document 26
3.5.1 Logs 27
3.5.2 Defect Logs 27
3.5.3 Activity Logs 27
3.5.4 Observation Logs 27
3.5.5 Project Logs 28

Appendix: MBV Artifacts 29

References/Bibliography 39

List of Figures

Figure 1: Model-Based Verification Processes and Artifacts 1

Figure 2: Project-Level Activities and Artifacts 6

Figure 3: The Engineering Activities of Model-Based Verification 13

Figure 4: Steps within the Build Activity 15

Figure 5: The Analyze Activity 19

Figure 6: Requirements and Expected Properties Elicitation Processes................20

Figure 7: Expected Property Transformations 23

Figure 8: Model Checking 24

Figure 9: The Artifacts of Model-Based Verification Practices................................29

Figure 10: MBV Input Artifacts...30

Figure 11: Output Artifacts of Model-Based Verification Practices 33

Figure 12: Sample Statement of Scope, Formalism, and Perspective 35

Abstract

Model-Based Verification (MBV) involves building and analyzing formal models of a system as an approach to identifying and guiding the correction of defects in software engineering artifacts. This report summarizes MBV and outlines the responsibilities of engineers engaged in Model-Based Verification. Each of the practices is described together with an initial set of guideline documents. These descriptions include procedural information, technical foundations for the practice, and engineering techniques for an MBV practitioner.

1 Introduction

Model-Based Verification (MBV) is a systematic approach to finding defects (errors) in software requirements, designs, or code. The approach judiciously incorporates mathematical formalism, usually in the form of state machine models,[1] to provide a disciplined and logical analysis practice for reviewing software artifacts [Gluch 98]. MBV involves creating simplified formal models (essential models) of system behavior and analyzing these models (using automated tools) against formal representations of expected properties.

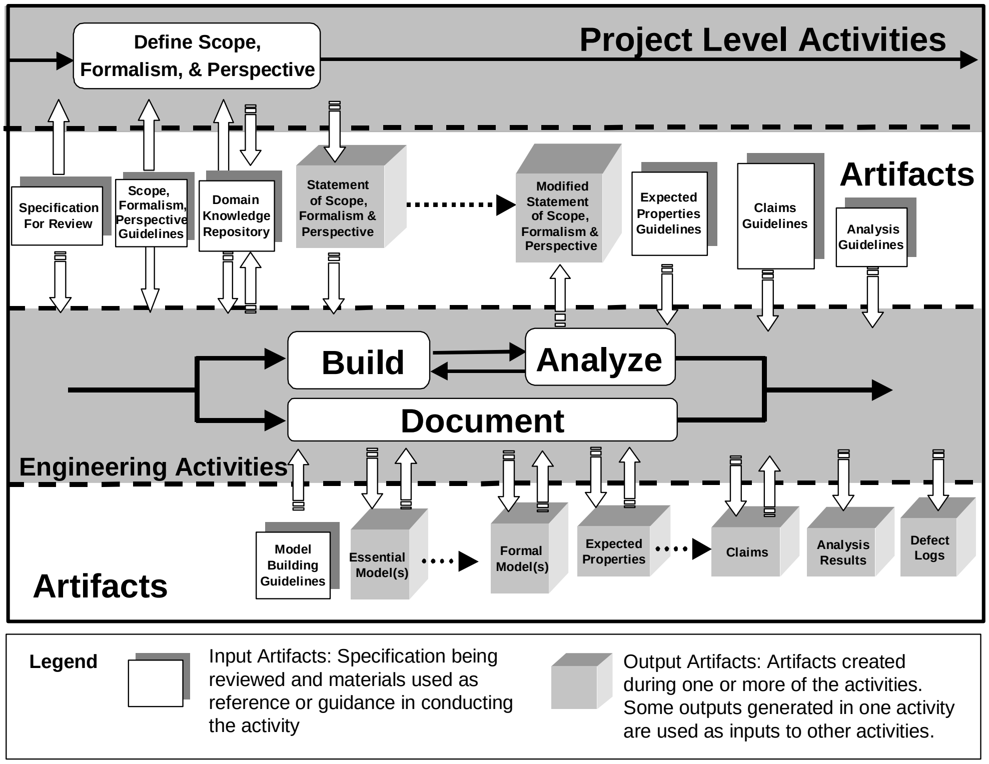

Project Level Activities
Define Scope, Formalism, & Perspective
Artifacts
Specification For Review
Scope, Formalism, Perspective Guidelines
Domain Knowledge Repository
Statement of Scope, Formalism & Perspective
Modified Statement of Scope, Formalism & Perspective
Expected Properties Guidelines
Claims Guidelines
Analysis Guidelines
Build
Analyze
Document
Engineering Activities
Artifacts
Model Building Guidelines
Essential Model(s)
Formal Model(s)
Expected Properties
Claims
Analysis Results
Defect Logs
Legend
Input Artifacts: Specification being reviewed and materials used as reference or guidance in conducting the activity
Output Artifacts: Artifacts created during one or more of the activities. Some outputs generated in one activity are used as inputs to other activities.

Figure 1: Model-Based Verification Processes and Artifacts[2]

[1] While there is a variety of modeling formalisms that are used, the focus of the initial work in model-based verification is finite state machines. Other modeling techniques can be used, for example, the formal approach of the Alloy tool [Jackson 00].

[2] The solid arrows in Figure 1 portray the flow of activities. The arrow into Build and the arrows connecting Build and Analyze indicate that Build occurs first but may occur repeatedly after Analyze.

The artifacts and the key processes used in Model-Based Verification are shown in Figure 1. These can be divided into two distinct categories of activities:

1. Project-level (team) activities
2. Engineering-level (individual) activities

Project-level activities involve both programmatic and technical decisions and engage multiple members of a project team in important MBV technical and process decisions. These decisions relate to defining the scope of the effort (how much of the system is to be modeled and analyzed), the formalism and other techniques to be employed, and the perspective (what aspects of the system are to be considered) of the modeling effort.

Most project-level activities are conducted when Model-Based Verification gets started for a particular project. There are also continuing activities that control the process and update various artifacts throughout all aspects of Model-Based Verification. Generally, these involve interactions and iterations among the project-level and engineering activities. These ongoing activities are shown as the solid arrow extending to the right within the project-level activities of Figure 1. To make the diagram easier to understand, all of the interactions among of the various activities are not shown explicitly.

Individual engineers rather than teams carry out the remaining activities 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 accompanied by a small amount of analysis. A parallel documentation activity[3] gathers detailed information on errors and potential corrective actions.

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 as captured in appropriately abstracted scope and perspectives, a reviewer using model-based techniques focuses the analysis on the most challenging and problematic aspects of the system. Because of the discipline and rigor required to create a formal model, simply building the model, in and of itself, uncovers errors.

Once a formal model is built, it is analyzed (checked) using automated model checking tools. Model checking has been shown to uncover the especially difficult-to-identify errors—the kind of errors that result from the complexity associated with multiple interacting and interdependent components.

While engineers are generally responsible for completing the above tasks, it is often helpful

[3] In earlier descriptions of Model-Based Verification, the Document activity (shown in Figure 1) was referred to as the Compile activity. This is only a change in the name not in the makeup of the activity.

for them to collaborate with other experts. This is especially true during the analysis phase where additional domain expertise can help to guide and facilitate the effort. Throughout all MBV activities, engineers document (log) the defects identified and the results of their analyses.

A variety of formal modeling and analysis techniques are employed within model-based verification [Gluch 98, Clarke 96]. A specific project decision on a technique(s) involves an engineering tradeoff. The considerations involved in these tradeoff decisions include the criticality of and the business issues associated with the system being analyzed (e.g., risks, safety issues, importance, and cost considerations) and the technological foundation of the critical aspects of that system. For the purposes of this report we focus on techniques that involve the use of state-machine-based models and the Symbolic Model Verifier (SMV) tool [McMillan 92].

Model-Based Verification practices must be integrated into an organization's routine verification and validation processes. For example, defect identification results must be provided to the personnel in the organization that can address them. The approach to the integration of MBV practices will vary, depending upon the organization and project involved. What is important in this integration is ensuring that the principles of MBV are maintained throughout the integration activities as well as within the MBV team and engineering activities. Suggestions on this integration have been provided by Gluch [Gluch 99].

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

This report is a step in documenting MBV practices and overcoming barriers to their adoption. It describes the activities and summarizes the techniques involved in the practice of MBV and provides general guidance in the use of these practices and techniques for the analysis of software-based systems. The intended readers are technical managers and software engineers who are planning to use MBV. Over time, an organization or program will most likely develop its own specialized guidelines and repositories of models, model components, and claims.

This report is organized as follows. In the next section we discuss project-level activities in more detail, especially determining scope, formalism, and perspective. Subsequent sections discuss different aspects of the engineering-level activities, including creating abstract models, determining expected properties, generating claims, and analyzing results. The appendix of this report describes the artifacts involved in the practice.

2 Project-Level Activities

Project-level activities focus on Model-Based Verification efforts. These are team activities that require the involvement of personnel who bring varied insights of the system into the activity. The expertise and interests of this group should span user needs, customer perspectives, and the diverse application and technical problems associated with the system's development, implementation, use, and maintenance. The principal outcome of the project-level activity is a statement of scope, formalism, and perspective (SFP). The SFP is used to focus and guide the modeling and analysis activities. The decisions needed to generate this statement are driven by considerations of what is critical (important or risky) relating to technical, programmatic, organizational, and business issues.

2.1 Project-Level Artifacts

The artifacts associated with project-level activities are shown in Figure 2. These include the

- Specification for Review – This represents all of the artifacts that will be scrutinized in the verification activities. This set consists of the specification that is being analyzed and associated supporting documentation. For example, a software requirements specification may reference an interface control document for a detailed description of a data item. In this case the referenced interface control document is needed to understand and analyze the specification.
- Domain Knowledge Repository – The domain knowledge repository encompasses the documents that provide additional information to engineers about the domain of the system being analyzed. This collection may include user documentation, requirements specification, and artifacts developed by the engineers who are modeling and analyzing the system. The documents generated help engineers to gain understanding and insight into the system. Such a document might be a glossary of key terms and acronyms.
- Guidelines for Defining the Scope, Formalism, and Perspective – These guidelines provide support for the project activity of identifying and capturing the critical (important or risky) aspects of the system and its development, ultimately resulting in the statement of scope, formalism, and perspective. The initial version of these guidelines is presented in a report by Gluch [Gluch 01].
- Statement of Scope, Formalism and Perspective – This is the principal output of the project-level activities that focuses and guides the modeling and analysis efforts. It is a description of what area/subsection(s) is to be modeled, what tool(s) and formal languages are to be used for the modeling, and what aspect of the system's behavior is to be investigated (e.g., fault tolerance, safety, communications).

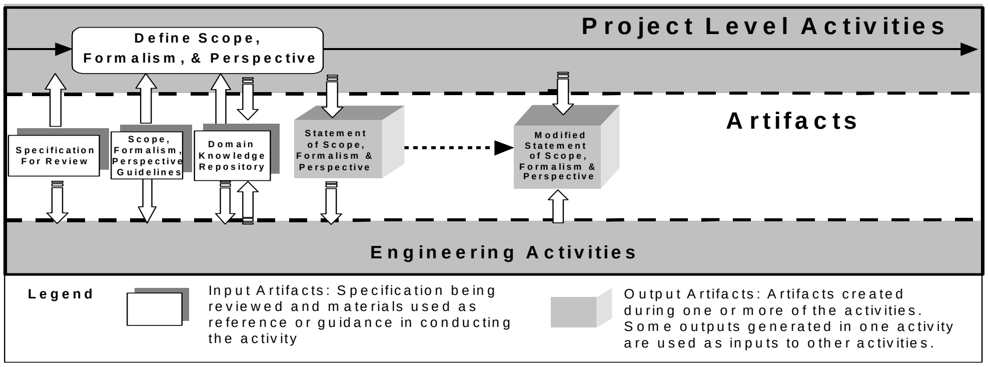

Project Level Activities
Define Scope, Formalism, & Perspective
Artifacts
Specification For Review
Scope, Formalism, Perspective Guidelines
Domain Knowledge Repository
Statement of Scope, Formalism & Perspective
Modified Statement of Scope, Formalism & Perspective
Engineering Activities
Legend
Input Artifacts: Specification being reviewed and materials used as reference or guidance in conducting the activity
Output Artifacts: Artifacts created during one or more of the activities. Some outputs generated in one activity are used as inputs to other activities.

Figure 2: Project-Level Activities and Artifacts

As shown by the dotted arrow in Figure 2, the statement of scope, formalism, and perspective evolves throughout the Model-Based Verification process. It is updated as greater insight into the system develops. These insights re-focus or expand the effort. Similarly, the set of materials within the domain knowledge repository is also modified and extended throughout all of the activities of Model-Based Verification.

While most of the effort at the project level occurs at the beginning of Model-Based Verification activities, ongoing activities are needed to both support the updating of the artifacts and control the process. These may be scheduled events or convened as needed. Often a regular team review is conducted to ensure that the process is proceeding on schedule and to deal with minor problems. In some cases, when a major unexpected problem is encountered, a meeting may be called to address the problem’s impact on the analysis activities. There also may be events that enable the coordination of MBV practices with other project verification and validation efforts.

2.2 Focusing MBV Activities

The initial step in MBV is defining the objectives, approaches, and plans for execution of the MBV activity. The outcome of this planning step is an initial statement of scope, formalism, and perspective. A template and sample statement of scope, formalism, and perspective can be found in Section 4.

Ideally, in a software development effort every subsystem would be verified from every possible angle under the most stringent conditions as early as possible in the development cycle. Practically, this is not possible and the choice of what portions of the system and what artifacts undergo extensive analysis is a tradeoff involving a complex set of management and engineering decisions. In addition to technical complexities, factors like time to market, functionality, and economical viability are important considerations in these choices.

Critical aspects of the system are used as the basis for these tradeoff decisions. The amount of risk involved, as well as the importance of relevant requirements, helps to define which aspects are critical. The choice of these critical aspects requires substantial domain knowledge, and often knowledge about the relevant implementation details of the system. To be effective in these decision processes, it is imperative that the team defining the plan has a broad understanding of the system's requirements, design, and environment. If this is not the case, an individual with this knowledge should be included in the planning activities.

2.2.1 Scope

Scope delineates what parts of the system will or will not be verified. The scope can be physical (a particular subsystem) or logical (a module or sequence). In some situations, it is useful to use different approaches to verify different parts of the system; this would correspond to a number of scopes associated with potentially different perspectives and formalisms. Some examples of scope include:

- a vital system component
- the communications subsystem
- a set of concurrent processes
- switching logic

Scope is mainly concerned with the decomposition of a large problem (the system) into more manageable subproblems. This is extremely important to make MBV viable, as current practices do not scale well and can cope only with limited size problems. After model checking different parts of the system, it may be impossible to use *compositional reasoning techniques* to infer global properties of the entire system [Berezin 98].

2.2.2 Formalism

Formalism defines the modeling technique that will be used to verify the system. Most of the work in Model-Based Verification to date has focused on the use of state machine analysis. There are other approaches that can be used. These include approaches that analyze the timing and scheduling aspects like Rate Monotonic Analysis (RMA) and those that are more object-like and facilitate the analysis of systems with complex relational structures (e.g., Alloy) [Jackson 00]. In MBV, some formalisms are better suited for a particular situation than others. If, for example, the temporal characteristics of a real-time system are being verified, a modeling formalism that supports temporal concepts is needed.

Knowledge of the various modeling techniques and their capabilities is vital to making the right choice. In general, decisions on specific modeling techniques should consider the characteristics of the system being modeled, the efficacy of the modeling technique when applied to those characteristics, the complexity of the model being generated, and the risks associated with the system. In particular, the risks can help to determine the level of formality at which a system should be analyzed (e.g., a highly formalized model would be appropriate for a safety-critical system). High assurance often implies high cost, however, and these types of tradeoffs should be considered when choosing a modeling technique [Gluch 98].

Regardless of the choice of formalism, tools are required to support the modeling and analysis efforts. As we noted earlier, in this report we focus on the use of state machine-based techniques and tools. Most of the currently available state machine analysis tools are the result of research efforts. They require significant expertise on the part of users. However, commercial tools for software model checking are beginning to emerge. These are normally more robust and easier to use. For example, I-Logix has released a version of Statemate Magnum™ that includes state machine model checking capabilities [I-Logix]. Other state machine-based tools that can support MBV include the following: Carnegie Mellon University's version of SMV [McMillan 92], Cadence version of SMV [McMillan 99a], NuSMV [NuSMV], and Bell Labs' Simple PROMELA (Process Meta Language) Indentifier (SPIN) [Holzmann 91].

2.2.3 Perspective

While scope defines the areas of focus for MBV, perspective states what is interesting about those areas. Normally, it is impractical to model the general behavior of a component as this behavior is often defined by an excessive number of states. We recommend a more specific view of the system component. For example, we can focus our effort on studying the control flow or the initialization sequence. The following is a list of possible perspectives for a particular component:

- communications
- flow of control
- redundancy management
- coherency
- emergency procedures/failure modes
- initialization processes
- time-out responses
- message sequencing

Scope can be characterized as a decomposition mechanism, while perspective can be seen as defining a focus for abstraction. Deciding what to consider about a particular component is

™ Statemate Magnum is a trademark of I-Logix.

equivalent to deciding what to ignore. When we decide on a perspective, we are abstracting away all the characteristics and behaviors not related to that perspective.

The perspective not only guides the model-building activity but also the expected property-generation activity. Expected properties are natural-language statements about the desired behavior of a system: behavior that is consistent with user expectations. Expected properties are formalized as claims and are checked against system models. Both general and focused expected properties and their associated formal representations are used to gain confidence in critical aspects of the system. Thus, the perspective provides the view for building models as well as the focus for analysis activities.

Together, the scope and perspective can be used to make the verification process incremental. Initially, a small component (scope) can be verified from a single perspective. Later, more components can be added or more perspectives can be considered. This incremental approach can be followed until satisfactory coverage of the system is reached.

2.3 Team Processes

Making decisions about the scope, formalism, and perspective as a team exercise is important, as different people have different insights into the problem. The process of obtaining a consensus helps to guarantee that every important factor has been considered. Additionally, the people who are involved in such a decision may become more committed to implementing the decision that is eventually made. Commitment is especially important if those who decide on the scope, formalism, and perspective will also be conducting the MBV activities (which is something that we strongly recommend).

The literature listed in the References section discusses several group decision-making techniques including Delphi [Linstone 75] and the Analytic Hierarchy Process (AHP) [Saaty 80]. These techniques are extensively documented, so we are not going to cover them in this report. In fact, the particular technique is not as important as ensuring that the following goals are achieved in the decision-making process:

- The decision must incorporate input from every member of the team. This goal may be threatened by the presence of members with strong personalities and a vested interest in making their opinion prevail over their peers'.
- The team interaction must ensure that the group is more than the sum of its members. The team leader has to foster an environment in which there are no “wrong” inputs and every comment initiates constructive discussion.

- Every assumption has to be validated by the team as a whole. Often, this will reveal any weaknesses and shortcomings in people's assumptions. Team members must be critical and refuse to accept anything *a priori*.
- The final decision must be fully endorsed by every member of the team.

In order to achieve these goals, the team requires members with a diverse set of backgrounds including a good facilitator, system and domain experts, MBV practitioners, and quality assurance advocates.

2.4 Programmatic and Technical Considerations

This section discusses some of the programmatic and technical factors associated with establishing a focus for MBV. We define programmatic factors loosely as those that are interesting from a management point of view and take into consideration nontechnical issues. Among these are economical analyses involving budget, schedule, and Return of Investment (ROI) assessments.

Every quality assurance (QA) technique involves discovering defects and removing them at a cost. Of all the QA activities, software inspections have proven to be one of the most cost-effective techniques, because the early detection of defects results in a large savings [Ebenau 94, Fagan 76]. The cost of fixing defects increases geometrically with the project phase [Basili 01]. MBV is a systematic and formal enhancement to inspections. Although more expensive than traditional reading-based inspections, MBV is capable of finding subtle defects that are difficult and costly to repair when uncovered later. This is especially true for the complex structural and logic aspects of the system, where effective manual inspection is extremely difficult.

The extent of the usage of Model-Based Verification within a project should be assessed based upon the potential value that it can provide. An analysis of value can help to determine the areas and aspects of the system in which it makes economical sense to conduct MBV. It can also help determine the extent and level of formalism at which MBV should be applied.

While a complete set of important factors to consider is yet to be defined some examples include

- costs associated with applying MBV
- number, criticality, and complexity of those defects likely to be found by MBV that might not be found in a traditional inspection

- side benefits derived from MBV including a better understanding of the system behavior, models that can be used later, and claims about global invariants that typically are not written in specifications but are critical when making later changes to systems.
- cost of failure due to a defect that was not found in time

Technical factors, on the other hand, are the day-to-day concern of engineers. These factors cover the viability of the system in terms of its quality attributes including correctness, performance, reliability, and others. Engineers should determine what is critical (important or risky) for the system operation or development. The following are a few examples of characteristics that may make a system more problematic to develop:

- unprecedented systems or parts of systems
- dynamic resource sharing
- redundancy for high reliability
- complex concurrency or distributed functions

The careful consideration of both the critical programmatic and technical aspects will increase the chances of selecting a scope, formalism, and perspective that leverages the capabilities of MBV and makes sensible use of available resources.

2.5 Fine-Tuning Scope, Formalism, and Perspective

Regardless of the methods used to decide on the scope, formalism, and perspective, the decision must be revisited throughout the course of verification and development. Every software project has peculiarities that make it impossible to predetermine the optimal combination of scope, formalism, and perspective. As the verification effort progresses, practitioners will get a better sense of what areas are benefiting most from the modeling activities. The team will discover areas with poor quality or unexpected complexity and find out which formalism fits best, considering the current system and the level of effort required.

The system environment will surely evolve, and this evolution will make many of the elements of the scope, formalism, and perspective obsolete after a while. This dynamic nature of the system makes a tight collaboration between the MBV reviewers and the system analysts/developers critical to an MBV effort. A good way of ensuring this tight collaboration is to present the results from the model checking effort to the development team. This presentation will validate the defects found, in addition to providing valuable feedback for refining the scope, formalism, and perspective statement.

2.6 Completing the Team Activities

When a focus is defined, the work of building and analyzing models and documenting defects can begin. In order to accomplish this, organizational issues must be addressed. These include assigning responsibilities to individual engineers, based upon the partitioning that is established by the statement of scope, formalism, and perspective. These assignments can be made using the project's established processes.

An important element that must be considered in coordinating MBV activities is to recognize that as new insight is gained into the system there will be a need to modify, extend, or refocus these activities. There must be a process to enable these changes to be incorporated into the overall effort, including formally modifying documents and changing the assignments and responsibilities of the team members.

3 MBV Engineering Activities

Model-Based Verification engineering activities involve building and analyzing abstract formal models of the essential properties of a system. These essential models capture what is important or risky for the system and are developed based upon the statement of scope, formalism, and perspective. As shown in Figure 3, the three principal engineering activities of the Model-Based Verification paradigm are as follows:

1. Build: systematically build essential models
2. Analyze: analyze and interpret the essential models by generating formal representations of expected properties and behaviors, analyzing both the models and the properties, and interpreting the results. Generally this will involve the use of automated model checking techniques and tools. Interpreting the results will help to gain insight into defects and to aid in identifying corrective actions.
3. Document: gather detailed information on errors and potential corrective actions

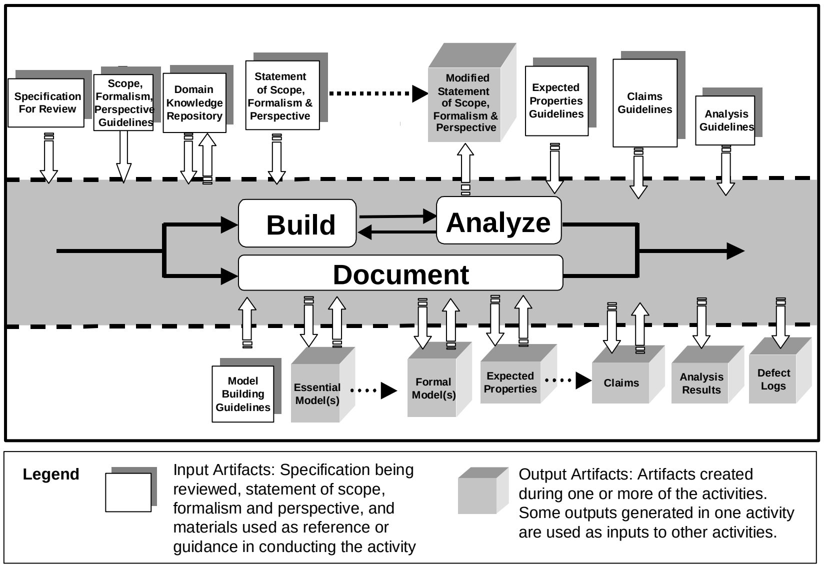

Specification For Review
Scope, Formalism, Perspective Guidelines
Domain Knowledge Repository
Statement of Scope, Formalism & Perspective
Modified Statement of Scope, Formalism & Perspective
Expected Properties Guidelines
Claims Guidelines
Analysis Guidelines
Build
Analyze
Document
Model Building Guidelines
Essential Model(s)
Formal Model(s)
Expected Properties
Claims
Analysis Results
Defect Logs
Legend
Input Artifacts: Specification being reviewed, statement of scope, formalism and perspective, and materials used as reference or guidance in conducting the activity
Output Artifacts: Artifacts created during one or more of the activities. Some outputs generated in one activity are used as inputs to other activities.

Figure 3: The Engineering Activities of Model-Based Verification

The artifacts associated with these activities can be divided into an input and output group. The input artifacts consist of the specification being reviewed; the statement of scope, formalism, and perspective that was developed in the project-level activities; and reference material and guidelines that support the activities. The output artifacts are the products of the engineering activities. They are summarized in the next section. A complete description of all of the artifacts involved in Model-Based Verification practices is included in the Appendix of this document, MBV Artifacts.

3.1 Output Artifacts of the Engineering Activities

As shown in Figure 3, a number of artifacts are created during model building and analysis activities. The output artifacts are summarized below.

- Essential models are abstracted representations of the system being analyzed. These include representations that are written in technique-specific languages (e.g., graphical state diagrams and statecharts). Essential models are the outcomes of the Build activity.
- Formal models are translations of essential models into model checking tool-specific languages. Translated models are not always required. A translated representation is only needed if the initial form of the essential models is not in the appropriate tool-specific language or additional model checking tools are being used for the analysis.
- Expected Properties are natural language statements that represent the desired behavior of the system. These are the outcomes of the initial steps in analyzing essential models.
- Claims are formal statements of the expected properties of the system. These are often expressed as temporal logic formulae (e.g., for the Symbolic Model Verifier Computation Tree Logic notation is used).
- Analysis Results are the outputs of automated model checking tools and their assessment and potential guidance for correcting defects. Model checking tools generate output that confirms the validity of expected properties for the model and in many cases identifies counterexamples of properties that are not confirmed.
- Defect Logs are itemized compilations of the defects identified in the system while building and analyzing essential models of the system. Defect logs are the outcomes of the document activity throughout all phases of Model-Based Verification engineering activities.
- Modified Statement of Scope, Formalism and Perspective results from building and analyzing models. Through these activities greater insight and enhanced knowledge of the system are developed. Often this insight identifies additional parts (scope) or aspects of the system (perspective) that should be analyzed in greater detail. These additional investigations may require alternative modeling and analysis approaches (formalism). The statement of scope, formalism, and perspective is updated to reflect these changes.

3.2 Guidelines for Engineering Activities

There are guidelines that support the engineer in building and creating models. They include the following.

- Model Building Guidelines help software engineers in the modeling activity. They address the use of abstraction techniques in developing essential models of the system, providing guidance to engineers in the creation of accurate and readily analyzable representations of the system. A preliminary version of these guidelines is presented by Hudak [Hudak 02].

- Expected Properties Guidelines facilitate the generation of expected properties in the context of Model-Based Verification. Expected properties are used as the basis for formal claims into the models of the system. These guidelines are presented by Gluch [Gluch 02].
- Claims Guidelines facilitate the claim generation processes by relying on a template-based approach for expressing expected properties in a formal language. They include a set of templates for the most frequent expected properties found in system specifications. These guidelines are presented by Comella-Dorda and are available in the MBV assistant tool [Comella-Dorda 01].
- Analysis Guidelines provide support and guidance to an engineer during the analysis activity, including the interpretation of results produced by model checking tools. They address the problems related to interpreting results and provide strategies to overcome state explosion, analyze results, and provide feedback to the system designers and developers. These guidelines are presented by Lewis [Lewis 01].

3.3 Build

The purpose of the Build activity is to create one or more essential models of the system under analysis. As shown in Figure 4, the Build activity involves three iterative steps:

- Prepare
- Build and Refine
- Translate into Formal Models

The arrows in Figure 4 represent activity control flow. The rounded rectangles represent activities and the steps within activities.

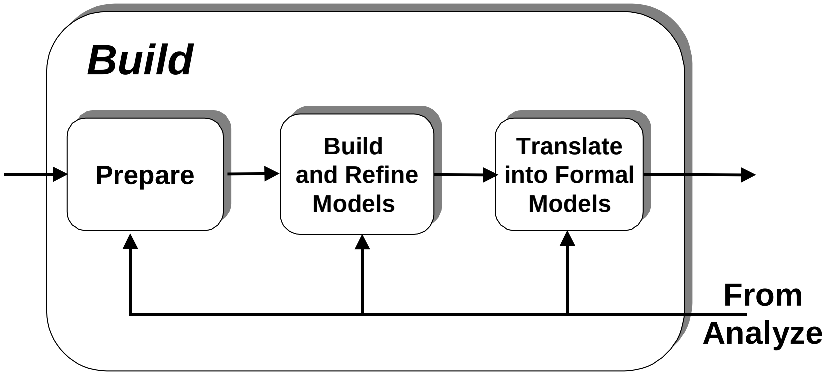


Figure 4: Steps within the Build Activity

The steps within Build are centered on the creation of essential models of the system that are amenable to analysis. The Prepare step is a set of activities that lays the foundation for subsequent steps. The activities include accessing support information and consulting colleagues regarding the system being analyzed, specific modeling strategies, and related technical issues. The heart of the Build activity is the Build and Refine step. It encompasses the activities that are required to create and document essential models. In the translation step, models are prepared for automated analysis. Depending upon the specific analysis tool and the original formalism used for the model, this may be a very simple step[4] or may involve a detailed translation of the model into the formal language required by the analysis tool. More details on implementing a specific build process with examples have been provided by Hudak [Hudak 02].

3.3.1 Essential Models

The principal goal of the Build activity is to develop essential models of the system using abstraction techniques. Abstraction is a process of focusing on the important details and ignoring, or abstracting away, details that are less important. Essential models are important to the practice of MBV because they make it possible to apply formal techniques with reasonable effort. Essential models reduce the complexity of the system being modeled. This reduced complexity requires fewer resources (human and machine) to verify claims about the system than are required for analyzing a completely detailed system model.

While abstract representations are used throughout a development effort, the focus on simplification required for building essential models contrasts with the extensive detail that is needed in the development of requirements, design specifications, or code. In the development of a system, details and redundancy of representations can be useful, especially if they involve different perspectives. In abstracting essential models however, the detailed representations created in the development process must be streamlined into concise, accurate, and simple representations. These are less complex models that faithfully represent a system’s properties and behavior at some level of abstraction. In many ways this is the reverse process of development. Essential modeling can be viewed as distilling a system down to its most important, or core, elements rather than refining the details of its implementation.

3.3.2 Prepare

The first step in Build is to prepare for modeling and analysis. This step is often revisited throughout the process. This preparation involves achieving a working understanding of the system and of what is needed in the Build and analysis activities.

Such preparation may involve

[4] For example, if the I-Logix Statemate tool is used to develop statechart models of the system, no translation is required. The Statemate tool supports model checking of statecharts [I-Logix].

- reviews of the entire specification being analyzed. These reviews can help to provide details on the specific aspects of the portions of the system to be modeled as well as a context for the modeling.
- review of the statement of scope, formalism, and perspective. This review will help to clarify and focus the modeling and analysis efforts.
- dialogues with personnel who have significant knowledge of the specification. These communications will help broaden an engineer's awareness and understanding of the system.
- reviews of summaries of the system. These will help to provide a broadened system context for the engineers in their modeling and analysis efforts.
- reviews of related engineering documentation. This will help to provide engineers with a better understanding of the system and its context.
- production of supplementary representation (e.g., drawings showing interrelations, hierarchy, communications, messages, etc.). These efforts will help to clarify an engineer's understanding of the system and of the specific aspects being investigated.

Building and analyzing models in Model-Based Verification is an iterative process. As shown in Figure 4, Build activities are repeated iteratively throughout the modeling and analysis process. While at each iteration the amount of time and effort devoted to these activities will decrease, an iterative return to the Prepare step is important to help ensure a complete and correct understanding of the system. This is especially important as new areas of investigation are pursued. In this case, iteration in the Prepare step may include reviewing additional documentation to better understand or clarify a new area or problem.

3.3.3 Build and Refine

The core iterative cycles within and between Build and Analyze involve the Build and Refine step. In these iterations incremental versions of a model are developed, assessed, and refined using feedback from critiques and from more formalized analysis activities within the Model-Based Verification process.

In the Build portion of the Build and Refine step, each increment of the complete model is created. In this activity, state diagrams, statecharts, and related representations are developed. Initially, a small portion of the total model is created. Within subsequent iterations, additional details are added and changes are made to the model as part of the refine portion of the Build and Refine step. The statement of scope, formalism, and perspective and the results of critiques and analyses guide this incremental evolution of the model.

Critiquing processes within the Build and Refine step are conducted by individual engineers evaluating their own models or by peers through informal 'reviews' of the models. The more formalized assessments associated with these iterations involve applying MBV analysis methods to intermediate versions of the models. This requires the translation of a partial model into a form amenable to automated model checking, the creation of basic claims, and the checking of the model against these claims. The basic claims are aimed at ensuring that the model is correctly implemented and internally consistent. For example, a basic claim may be one to ensure that all of the states are reachable in the state machine model.

Abstraction is a key element in the Build activity. Through abstraction some elements of a system are included while others are ignored in mapping design characteristics into abstract formal representations. The perspective guides the choices on what and what not to include. There is a variety of techniques that can be employed including elimination, reduction, enumeration, decomposition, non-determinism, and grouping. More details on the abstraction processes and techniques are described by Hudak [Hudak 02].

Throughout the Build and Analyze activities, the distinction between building and refining models is not crisp. For example, a change may be made to the model to fix a small error (Refine), enabling the continued assessment of the model. In other cases, the cumulative results of the critique and assessment activities are incorporated into an incremental build of an additional (new) model.

The need for refinement becomes apparent as the process progresses and may include changes made to increase

- correctness of the model. This involves determining whether a model conforms to the modeling techniques and accurately represents the system as intended. It requires increasingly detailed assessments of the model during the Build and Refine cycle. These include individual critiques by engineers of their models as well as the use of basic automated checks. For example, these simple checks may include syntax checks as well as the execution of simple claims.
- completeness of the model. As a model evolves through successive iterations, a determination of the completeness of the model is needed. This is an engineering judgment based upon assessing the model relative to the statement of scope, formalism, and perspective. Considerations include questions such as: Does the model cover the parts of the system required by the scope? Is the model sufficiently detailed to address the issues described in the perspective statement?
- understandability and verifiability of the model. This is a consideration of whether the model’s structure is readable and readily understood by other engineers, conforms to the restrictions of a particular model analysis approach, and is a tractable abstract representation for automated analysis. For example, it is important that no extraneous variables be included. Extraneous variables make the model harder to understand and increase the possibility that the model is too large for completing an automated analysis in a reasonable time frame.

3.3.4 Translate

Depending upon the choice of formalism and associated tools, the initial (primary) representations for models may require translation into a language that can be used for automated model checking. This translation may be done in parallel with the modeling effort or may be completed as part of a translation sequence, converting the primary model into a formal model.

For example, to use the SMV tool for model checking statecharts[5], the statecharts must be translated into the input language of the tool. Similarly, if both the SMV and SPIN model checkers are being used, an initial SMV representation must be translated into the SPIN input language, PROMELA. These models may be created within the Build activity or may be generated by translating the models developed in the Build activity into a tool-specific formal language. With the evolution of software development tools that support model analyses, this process can be substantially simplified. Support for the translation may range from tool-assisted translation to the capability for statechart model checking (e.g., the Statemate Magnum tool).

3.4 Analyze

As shown in Figure 5, analyzing an essential model involves generating expected properties and behaviors of a system, conducting a systematic analysis (assessment) of its behavior relative to the expected properties, and then interpreting the results. The solid arrows in Figure 5 represent activity control flow and sequencing. The rounded rectangles represent activities and steps within activities.

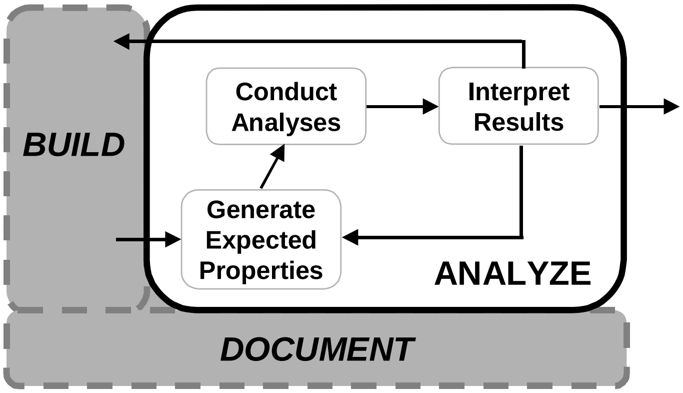


Figure 5: The Analyze Activity

[5] Statecharts are often chosen because of their ease of use and the availability of commercial tools to support their development.

3.4.1 Generate Expected Properties

We have already noted that expected properties are natural language statements about the desired system behavior. As an example of an expected property, consider a system that involves the sharing of resources among different functional components. A requirement may be that only one of these components at a time should access a given resource. A design approach may be to use an algorithm that defines how processes should coordinate their access to a shared resource. In a Model-Based Verification analysis of the design, a model captures the states and state transitions implied by the algorithm. But nowhere in the model is there an explicit statement that (1) only one process should have access to a shared resource at a time (mutual exclusion) or that (2) all processes that desire access will eventually will get access to a shared resource (fairness). Nevertheless, mutual exclusion and fairness are required (i.e., expected) properties of such a system. The model should demonstrate them in its portrayal of the system's behavior.

3.4.1.1 Sources

Expected properties are derived from a variety of artifacts including system requirements, design specifications or code, and user documentation. In addition, it is important to ensure that a variety of views and stakeholders are represented. This can include interviews or the direct participation of users, customers, or domain engineers. A systematic approach similar to requirements elicitation is required. Both requirements elicitation and expected property generation involve multiple stakeholders, can be facilitated by team processes (consensus-based practices), and are directed towards getting the key aspects of the system identified. These similarities are shown in Figure 6.

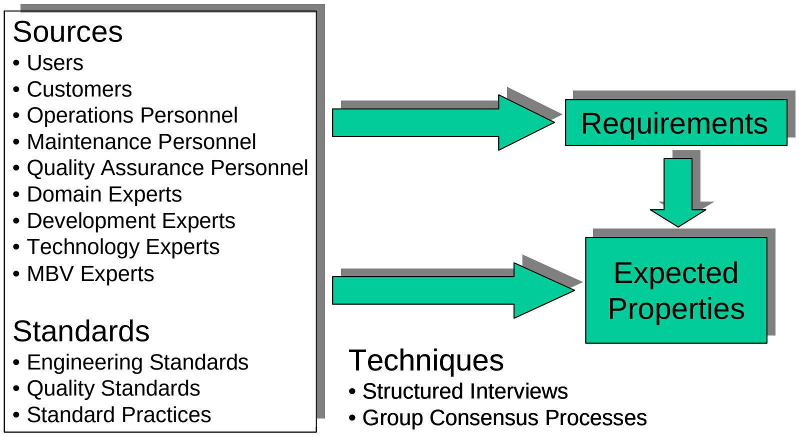


Figure 6: Requirements and Expected Properties Elicitation Processes

The following are potential sources for expected properties.

- Requirements documents: If the project has followed sound engineering practices, requirements should be the most reliable source of expected properties, as they are the consensus of what the system is supposed to do. One important value of generating expected properties from requirements specifications is that it can uncover requirements that are not stated appropriately, are implied, or are omitted.

- Stakeholders: An excellent source of information about what the system is supposed to do is the collective knowledge of stakeholders including users, customers, and operations and maintenance personnel. Members of these groups can provide first-hand insight into behavioral aspects of systems. If sound engineering practices were used in the original development effort, all of these groups would have had a role in defining the requirements documentation.
- Domain experts and quality assurance personnel: We have grouped in this category sources that are not specific to the system. There are expected properties common to individual application domains. For example, every entry in the assets column of a balance sheet should be followed by an entry in the liabilities column. Other expected properties represent quality attributes common to any software system. For example, all internal errors should produce some defined and identifiable external manifestation.
- Developers and implementation technology experts: In theory, technical details of the implementation are not needed to define the expected properties of the system. In practice, however, an understanding of the internals of a potential or actual implementation of the system can be useful in generating expected properties that express invariants that are to be maintained across the design and implementation.
- MBV experts: The activity of translating specifications into models can result in errors. Even if these errors do not affect the quality of the system, they do hamper the ability of models to faithfully reflect the behavior of the system. There are expected properties that address the correctness of the models in contrast to the correctness of the system. For example, every state in a state machine should be reachable under some sequence of valid input. If a state is not reachable, it is cluttering the model without reflecting any feasible system behavior. MBV experts are software engineers who are trained and experienced in applying Model-Based Verification techniques in a variety of application domains. They can help to craft expected properties that are more readily expressed as formal statements, thereby facilitating and reducing errors in the translation of expected properties into claims.
- Standards: There are expected properties induced by the standards with which the system must comply. For example, if the system uses CORBA Messaging 2.4.2 with order policy set up to ORDER_TEMPORAL, the messages from the client should be guaranteed to be processed in the same order in which they were sent.

3.4.1.2 Classifications

When trying to classify expected properties, it is useful to focus on the range in which the property (and the claim) applies. According to this classification, we identify the following types of expected properties.

- invariants (global and local) – conditions that remain unchanged throughout (globally) or conditions that remain unchanged in a particular portion or under specific circumstances (locally).

- assertions – conditions that apply at a single point in an execution[6]
- pre- and post- conditions – condition pairs that apply before and after something occurs in a system. These can be viewed as assertion pairs.
- constraints – statements that apply within and across all or a subset of changes that occur in the system[7]

Invariants, assertions, pre- and post-conditions, and constraints are propositions; they are either true or false when applied to a model. Thinking about the system's behavioral characteristics in the context of propositional forms can help in phrasing natural language statements that are more amenable to direct translation into a formal representation. For example, in a system that controls several traffic lights, it is clear that at each intersection, conflicting traffic flows should not be allowed. In thinking about what should not happen with respect to this property, an invariant can be identified: "In any intersection in the system, it will never be the case that the north-south traffic light and the east-west traffic light are both simultaneously green." This is readily expressed as a formal claim in Computation Tree Logic: AG !(N_S = green & E-W = green).[8]

This classification and the process of generating expected properties are influenced, in part, by the realization that the natural language statements must be translated into formal representations. Consequently, while the generation process should rely on the guidelines outlined earlier, using the four classifications defined above during the identification process can help to tailor the expression of expected properties into forms that are more readily translated into formal expressions.

3.4.1.3 Claims

To be used in automated model checking, the natural language statements of expected properties must be expressed as formal statements—claims. The various translations among requirements and claims are shown in Figure 7. There are three basic paths to formal claims: direct expression in a formal language, natural language statements translated into a formal language, and natural language statements first expressed in a structured limited vocabulary natural language and then translated into a formal language.

In most cases natural language statements of expected properties are generated first. Software engineers who are experts in the relevant formal language translate these statements into formal expressions for model checking [Comella-Dorda 01]. This translation is often not a one-

[6] A local invariant may span more than one state or point of execution (e.g., a loop invariant in a program). Invariants express properties that don't change no matter what actions occur, and assertions express only what is true at a single point in an execution.

[7] An example of a constraint is that no change in the value of a pressure parameter will exceed 5 psi. These contrast with statements about the conditions of a system (e.g., invariants, assertions).

[8] The notation AG means "for every state." A technical note by Lewis provides more details on the CTL notation and claims [Comella-Dorda 01].

to-one mapping of statements. The richness and ambiguities of natural language, in contrast to the precision and constraints of a formal style, often make the translation difficult and error prone. Consequently, the translation process should also involve domain experts and others who helped to develop the expected property statements. Their involvement can be either in direct support of the process or as active reviewers. In an active reviewer role, these individuals interact with software engineers to establish a shared interpretation of the formal statements, one where all parties agree on the consistency of the intent between the natural language statements and their formal expression(s). This collective involvement will help to ensure that subtle differences between the languages do not result in formal statements that misrepresent the expected property statement.

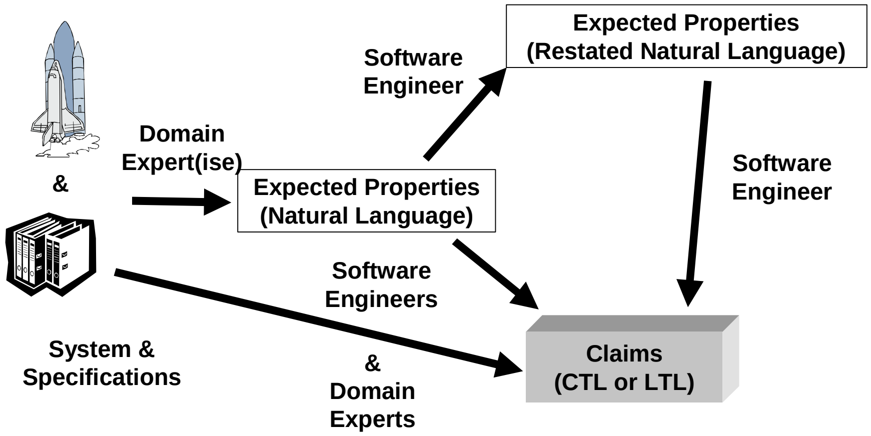


Figure 7: Expected Property Transformations

The language for stating expected properties is dependent upon the particular automated model checker that is used in the analysis. Most state machine model checkers use temporal logic statements. While temporal logic itself is not difficult to understand, temporal logic expressions can be arcane. Working with them requires a sound understanding of the idiosyncrasies of the particular temporal logic representation. For example in the SMV, a property that *any request for service will result in that request eventually being acknowledged* can be expressed as

`AG(request-> AF acknowledged)`[9]

The most direct path is to develop formal claims from the sources used to generate expected properties (e.g., a system description, requirements document). In this case software engineers, expert in the relevant formal language (e.g., Computation Tree Logic (CTL) or Linear Temporal Logic (LTL)), often in cooperation with domain experts, formulate the claims. This approach can work well if the software engineer generating the claims is also an expert in the domain. As is the case with the other paths, it is important that a variety of individuals be in-

[9] The symbol AF is a statement that eventually a state (in this case one where acknowledge is true) will occur. It is either true or not true for a system.

volved either directly or as reviewers of the natural language statements and their formal representations. This involvement will to help to ensure the correctness of the claims.

3.4.2 Conduct Analysis

Model checking is the principal technique used in conducting an analysis. Figure 8 shows the model checking process where an essential model of a system is "checked" against the expected properties. Model checking approaches include a variety of techniques and tools that require an essential model to be expressed as a finite state machine. A model checker "explores" a model's state space to determine the validity of expected properties (expressed as formal claims) relative to the model. The output of a successful run of a model checker is either a confirmation of the validity of each expected property or a statement that the property is not valid for the model. In most cases where the model checker finds the property not valid, a counterexample is provided. Counterexamples show a sequence of steps that negates the expected property.

A model checker can readily generate a counterexample for a statement that declares that something is always true. For example, consider the statement: "The error detection routine is always active." If this were determined to be not valid for a model, a counterexample would show a sequence of states ending where the error detection routine is inactive. In contrast, consider a statement that: "It is possible to get to a state where the alarm is sounding." If this is found to be not valid for a model, a useful counterexample is not produced since a counterexample would require a listing of all paths showing that the alarm is not sounding on any of them.

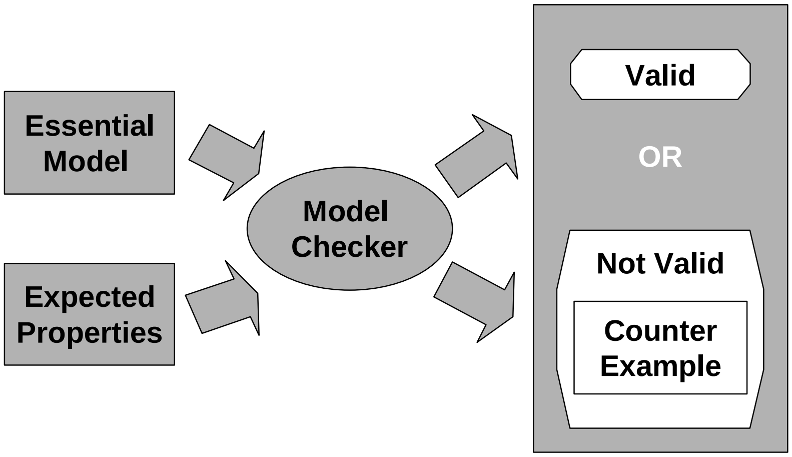


Figure 8: Model Checking

3.4.3 Interpret Results

Obtaining a confirmation or a counterexample from a model checking tool is not the end of the process. It is necessary to interpret the results and assess their impact. Interpreting the results can be as difficult as creating the models or the expected properties. A number of factors must be considered before deciding upon the actions to be taken in response to the results. These considerations are important both when a property is found to be valid and when a property is found to be not valid.

If a model checker confirms a claim, the claim is provably correct for the model. However, this outcome does not assure the correctness or completeness of the system that the model represents and the claims should be confirmed as correct. The claims themselves must be verified for correctness and the model must be verified as an accurate representation of the system. Consequently, the confirmation of the validity of an expected property (claim) can only build confidence in the system. It does not prove that the system or the resulting software possesses that property. Since Model-Based Verification is aimed at identifying defects, positive results from a model checker should be used to bolster confidence much as one would use testing results (i.e., to show the presence of defects not their absence [Dijkstra 69]).

When a claim is found not valid and a counterexample is generated, a detailed review of the counterexample can provide insight into a potential design defect and help to identify the specific aspect of a design that should be corrected. But to fully understand the import of a claim that is found to be not valid, whether or not a counterexample is generated, it should be assessed by considering the following:

- Is the claim stated correctly?
- Does it accurately express a valid expected property?
- Does the model correctly reflect the behavior of the system as specified?
- Does the interpretation of the counterexample reflect a potential defect in the system?

Addressing these questions can help to determine whether there are defects in the system or design, errors in the models, or errors in the claims. An effective approach in considering these questions and distinguishing among the possibilities is to first establish confidence in the correctness of the basic model before probing the correctness of the system. This can be accomplished by generating fundamental claims corresponding to key elements to ensure that the model accurately represents the basic behaviors of the system. These first few claims often reveal mistakes made in building the model rather than flaws in the system. They help to provide confidence in the integrity of the model and its validity as a representation of the system. As greater confidence is gained in the model, more probing claims, aimed at better understanding of the system and uncovering defects in it, can be explored.

Sometimes a check of the model cannot be completed. This often results from the model checkers' inability to handle an excessively large state space. This is known as the state explosion problem, which can occur because the number of states grows very quickly as the complexity of a model increases. For example, even a relatively simple system consisting of four state machines or concurrent processes, each with three state variables, each having five values, results in a total state space of approximately 250 million states.

There are a number of heuristics that can help model developers eliminate or reduce the state explosion problem. They are summarized below. For more details refer to the technical note by Lewis [Lewis 01].

- Abstraction: Make the model's state space smaller by abstracting it to a higher level. The challenge here is making sure that the abstraction correctly models the behavior of the system it is modeling.
- Decomposition: Break big problems into small problems, and then localize the verification of each subproblem to a small part of the overall model [Clarke 89, McMillan 99b]. In decomposition, different parts of the model are reasoned about separately, but the separate results are used to deduce properties of the entire system.
- Tool Idiosyncrasies: Use idiosyncratic characteristics of a tool to reduce the amount of memory or execution time. For example, in SMV this can sometimes be achieved by reordering the variables in a model.

Interpretation activates may prompt a number of different actions. For example, it may be that the interpretation reveals a potential defect. When a potential defect is discovered, it is recorded in a defect log. Alternatively, as a result of the interpretation of the results it may be necessary to revise the model or the claims and repeat the analysis.

3.5 Document

The document activity encompasses the efforts involved in logging and analyzing defect data. These include recording the defects and corroborating data that are identified in the artifact being reviewed and integrating these data into the larger data set associated with the overall verification process. This is an activity that pervades all aspects of Model-Based Verification. Defects are recorded throughout all phases of the practice. During the building and extending of models, defects in the artifact are uncovered and recorded. Throughout analysis activities, the identified defects and related information from counterexamples are included in the set of defect logs for the system.

No specific procedure or format for recording defects is required for the Model-Based Verification paradigm. A general format or one unique to an organization can be employed. What is critical is that complete and accurate data are captured and integrated into the overall verification process for the project and the larger organization.

The remaining parts of this section discuss the specific aspects of one approach to documenting defects and introduce additional data capture logs. Collectively, these data and associated practices enable the efficient application of MBV and provide a basis for process improvement.

3.5.1 Logs

There are four logs that are useful to maintain as part of Model-Based Verification practices. These are the defect log, activity log, observation log, and project log. Each engineer involved with a Model-Based Verification effort should maintain individual versions of the first three of these logs. There is one project log that summarizes the entire effort. At a minimum, some form of defect recording must be done.

3.5.2 Defect Logs

Individual engineers maintain a log detailing the defects (errors) found in the documents being reviewed and a comprehensive listing of these defects may also be recorded. These defect logs are itemized compilations of the defects found in the system artifacts being reviewed.[10] This may be organized as a chronological record of defects uncovered or other organizational structures can be used. No special form is needed. The defect log can be based upon what is already in use within the organization. Specific data included in defect logs may vary.

3.5.3 Activity Logs

The activity log is the place for recording the activities performed and the time spent on an activity (e.g., building models or generating expected properties). Generally this log is maintained as a chronological record of an individual's activities on the project. An entry in this log might consist of the nature of the activity and the start and stop times. Possible activities to track include: system familiarization, learning the modeling approach, discussions with domain experts, creating and revising models, creating and revising claims, analyzing and interpreting the results, overhead, and meetings. The procedures and templates for this activity can be based upon Personal Software Process[SM] approaches [Humphrey, 1995] or may be customized for an organization. Activity logs can be used by an organization to help assess the costs and benefits associated with using MBV.

3.5.4 Observation Logs

It is useful, especially during a first implementation of model-based engineering practices, to have each engineer record his or her observations regarding both the processes and technolo-

[10] It is not necessary to keep a list of errors made during the model-based analysis, although such tracking can lead to improvements in the analysis process.

[SM] Personal Software Process is a service mark of Carnegie Mellon University.

gies associated with MBV practices. These observations may include engineering decisions, rationale, difficulties encountered, insights, and errors made while doing Model-Based Verification. These can be captured in an engineering observation log. The information captured in this repository can be used to identify lessons learned, construct customized organizational implementation guidelines, facilitate transition of the practices into other projects within an organization, and enable improvement of the MBV engineering processes for an organization.

An observation log may contain the following elements:

- observation statement – a description of any insights, anomalies, issues, etc., that the engineer has observed in this particular activity
- date – date of the observation
- activity – the activity that the engineer was engaged in when making the observation
- type – part of the specification or model to which the observation is related.
- comments – recommendations, solutions, and insights relevant to the observation

3.5.5 Project Logs

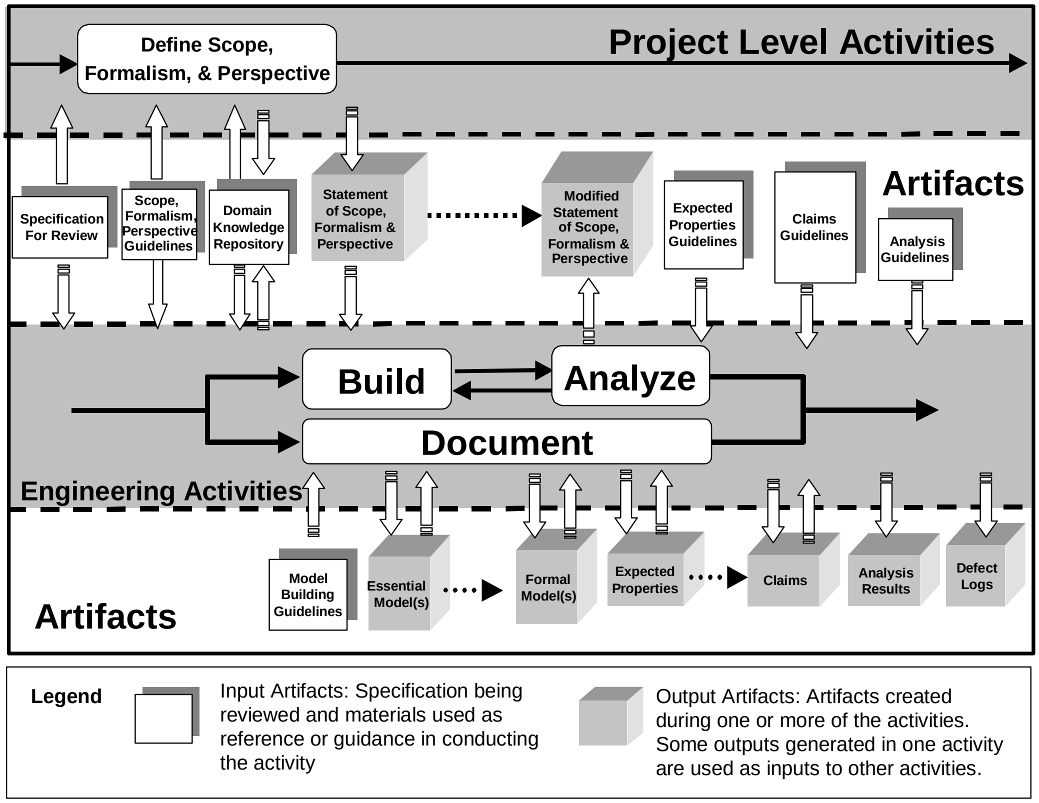
There is one project log that should be developed for each Model-Based Verification effort. This summarizes the context of the investigations and the broader factors and influences on the effort. This log can be one element in an organization’s process improvement efforts. It generally consists of three distinct parts:

1. a description of the salient characteristics of the artifact under review. Useful information in this description includes the size of the artifact(s) being modeled, a measure of the density of information per page, and the development methodology used.
2. a summary of the software development process that was used by the original system designers. The summary of the development process may include minutes from all of the team and joint meetings held. This data can help to track the decision making process.
3. a summary of major results and comments gleaned from the logs kept by the individual participants

Appendix: MBV Artifacts

This appendix summarizes the artifacts involved in Model-Based Verification. Figure 9 presents these artifacts and their relationship to Model-Based Verification activities. These are grouped into two categories:

- Input – These consist of the specification that is being reviewed and materials that are used as reference or guidance in conducting the activity. Input artifacts are white shadowed rectangles in Figure 9.
- Output – These are the artifacts that are created during one or more of the MBV activities. Some outputs generated in one activity are used as inputs to other activities. Output artifacts are shown as gray three-dimensional boxes in Figure 9.



Define Scope, Formalism, & Perspective
Project Level Activities
Artifacts
Specification For Review
Scope, Formalism, Perspective Guidelines
Domain Knowledge Repository
Statement of Scope, Formalism & Perspective
Modified Statement of Scope, Formalism & Perspective
Expected Properties Guidelines
Claims Guidelines
Analysis Guidelines
Build
Analyze
Document
Engineering Activities
Artifacts
Model Building Guidelines
Essential Model(s)
Formal Model(s)
Expected Properties
Claims
Analysis Results
Defect Logs
Legend
Input Artifacts: Specification being reviewed and materials used as reference or guidance in conducting the activity
Output Artifacts: Artifacts created during one or more of the activities. Some outputs generated in one activity are used as inputs to other activities.

Figure 9: The Artifacts of Model-Based Verification Practices

Input Artifacts

The input artifacts involved in Model-Based Verification consist of artifacts that are to be reviewed, guidelines, and other documents that facilitate the MBV process.

- Specification(s) for Review
- Scope, Formalism, and Perspective Guidelines
- Model-Building Guidelines
- Expected Properties Guidelines
- Claims Guidelines
- Analysis Guidelines
- Domain Knowledge Repository

Except for the specification for review, the input artifacts provide guidance or support for the various MBV activities, as shown in Figure 10. Each of these artifacts is discussed in more detail in this section.

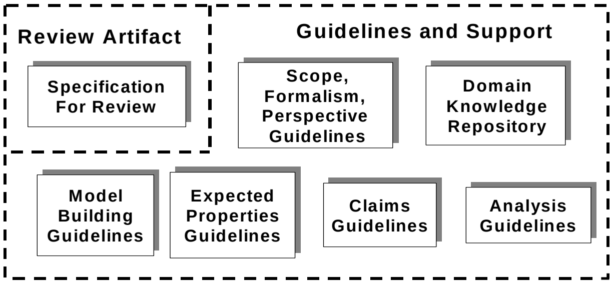

Review Artifact
Guidelines and Support
Specification For Review
Scope, Formalism, Perspective Guidelines
Domain Knowledge Repository
Model Building Guidelines
Expected Properties Guidelines
Claims Guidelines
Analysis Guidelines

Figure 10: MBV Input Artifacts

Specification for Review

The specification for review is the artifact that is to be reviewed using Model-Based Verification techniques. It may also include associated supporting documentation. For example, a software requirements specification may reference an interface control document for a detailed description of a data item. In this case the referenced interface control document is needed to understand and analyze the specification.

The specification for review may be a document or other record from any phase of the software development or maintenance lifecycle. For example the artifact may be a

- software requirements specification
- functional specification

- architectural design specification
- detailed design specification
- piece of source code

Scope, Formalism, and Perspective Guidelines

These documents guide practitioners during the project-level activities of Model-Based Verification. Project-level activities involve identifying and capturing the critical (important or risky) aspects of the system and its development, including both programmatic and technical issues. These activities result in the statement of scope, formalism, and perspective that focuses the efforts of engineers in building and analyzing models. Guidelines for developing the statement of scope, formalism, and perspective are presented by Gluch [Gluch 01].

Model-Building Guidelines

Model-Building Guidelines help software engineers in the modeling activity. They address the use of abstraction techniques in developing essential models of the system, providing guidance to engineers in the creation of accurate and readily analyzable representations of the system. A preliminary version of these guidelines is presented by Hudak [Hudak 02].

Expected Properties Guidelines

These guidelines facilitate the generation of expected properties in the context of Model-Based Verification. Expected properties are natural language statements that express characteristics of the behavior of a system. They are used as the basis for formal queries (claims) into the models of the system. These guidelines are presented by Gluch [Gluch 02].

Claims Guidelines

These guidelines facilitate the claim generation processes by relying on a template-based approach for expressing expected properties in a formal language. They include a set of templates for formalizing the most frequent expected properties found in system specifications. These guidelines are presented by Comella-Dorda [Comella-Dorda 01].

Analysis Guidelines

These guidelines provide support and guidance to an engineer during the analysis activity, including the interpretation of results produced by model checking tools. They address the problems related to interpreting results and provide strategies to overcome state explosion, to analyze results, and to provide feedback to system designers and developers. These guidelines are presented by Lewis [Lewis 01].

Domain Knowledge Repository

The domain knowledge repository encompasses documentation that supplements the artifact under review. These materials provide extra information about the application domain and the system being analyzed. They can be especially helpful to engineers who have not worked in the domain before. This collection may include artifacts that are developed by engineers to gain understanding and insight into the system, such as a glossary of key terms and acronyms. The target audience for the repository need not be limited to these people, however. Documentation and information that may comprise this collection include

- a list of common acronyms used and their definition
- a system context diagram with detailed descriptions of external components and interfaces
- information about how the system is to be used or the expected user environment (e.g., user manuals, operations manuals, maintenance manuals)
- explanations for assumptions made about the system and its operating environment
- a list of relevant standards that apply to the system
- interface control documents or related design information

Output Artifacts

Output artifacts are those that are produced during the MBV process. These include artifacts that are produced for use in other processes outside of MBV and those that are produced for use and modification during MBV activities. The output artifacts include

- statement of scope, formalism, and perspective
- models (essential models and their formal representations required for model checking)
- expected properties
- claims
- analysis results
- defect logs

These artifacts are shown in Figure 11.

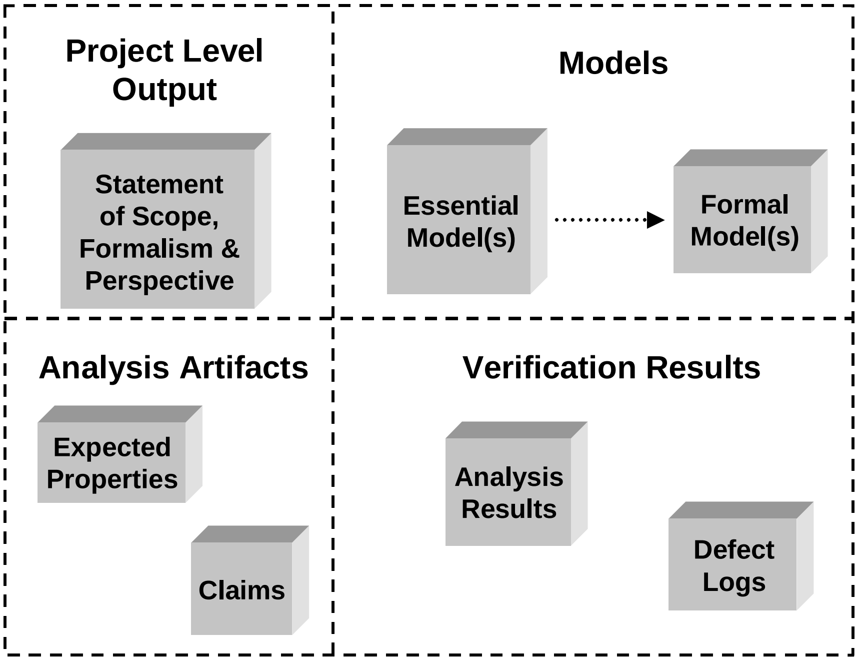


Figure 11: Output Artifacts of Model-Based Verification Practices

Statement of Scope, Formalism, and Perspective

The focus of Model-Based Verification is defined early in the process by developing the statement of scope, formalism, and perspective. This document is a project-level document that guides MBV activities. It is usually modified as a result of the iterative nature of the activity. In particular, the statement of perspective usually evolves as modeling and analysis are performed. The scope and formalism statements are generally more stable since they relate to global process and system issues.

Scope

The *Description* subsection identifies the system or subsystems that will be modeled. The *Rationale* subsection is a short paragraph of why this area is being explored and the reasons for building the model.

Formalism

This section defines the approaches that will be used to model the system, often summarizes how and to what portion of the system each of the tools might be applied, and sometimes suggests other approaches that may be used. The section may also identify specific techniques, tools, notations, and configuration management issues.

Perspective

The statement of perspective contains four subsections: general description, specific guidelines, issues requiring special consideration, and system attributes explicitly ignored.

The *General Description* subsection describes the specific focus of the model building and analysis efforts and includes

1. properties to investigate (e.g., tolerance to a specific set of faults, state completeness, message protocol completeness)
2. subsystems involved as they relate to the scope that is established for the analysis effort (e.g., fire-control computer and interfaces to radar, rockets, or weapon subsystem)
3. operational modes or operational profiles (scenarios) (e.g., automatic acquisition and targeting, making a specific variation of a chemical product)

The *Specific Guidelines* subsection contains statements about critical aspects and issues. These statements should identify specific considerations and provide guidance for investigating them. They can also be seen as precursors to, or explicit statements of, expected properties. These statements may include some or all of the following:

1. initial conditions/assumptions about external entities
2. operational sequences or modes of operation
3. outputs and associated attributes (persistent, transient, periodic, etc.)
4. validation of ‘normal’ modes of operation (what is the intended operation)
5. specific abnormal operations and behavior (certain abnormal conditions should be handled in a specific way)
6. timing issues (e.g., synchronization, deadlock, livelock)
7. protocol/algorithmic validation (when checking protocols or algorithms, what issues are important)

The *Issues Requiring Special Consideration* subsection is a short descriptive paragraph of each issue that is of special concern and should be explored with greater attention. This should include an explanation of the rationale for highlighting this issue (i.e., why this area is being explored, what are the reasons for building the model).

The *System Attributes Explicitly Ignored* subsection states any issues that need not be addressed and the rationale for excluding these.

Sample Statement of Scope, Formalism, and Perspective

A sample statement of scope, formalism, and perspective is shown in Figure 12.

1. Scope

1.1. Description

The Model-Based Verification activities should focus on the pilotless aircraft's flight control computer system, the communications network that connects the sensors, flight control computer, and the actuators.

1.2. Rationale

These systems are part of the flight and safety critical elements of the aircraft.

1. Formalism

State machine modeling and the SMV approach will be used; SPIN may also be used for the analysis of the communications protocol.

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.

3. Perspective

3.1. General Description

The focus of the modeling should be on the redundancy management aspects of the system and associated fault responses. The total set of operational modes and all possible fault scenarios should be considered for the key subsystems identified in the scope.

3.2. Specific Guidelines

- Consider only the basic processing states of each of the redundant components.
- Consider normal and fault responses of the fault detection system.
- Consider consistency of outputs sent to the fault detection system.
- Explore the fault response logic in the fault detection system.
- Address the impact of potential error states identified by the checker.

3.3. Issues Requiring Special Consideration

The synchronization strategy is not explicitly defined in the specification. The implications of this should be explored in detail.

3.4. System Attributes Explicitly Ignored

The algorithms used by the flight data processing units are explicitly ignored. They are being analyzed by another verification team.

Figure 12: Sample Statement of Scope, Formalism, and Perspective

Essential Models

Essential models are the outcomes of the Build activity. They are abstracted representations whose focus (view of the system) is defined by the perspective section of the statement of scope, formalism, and perspective. The type of models and the portions of the system that they model are defined by the formalism and scope sections. In general, an essential model is a semi-formal to formal model of the system that is expressed in one of the established and generally commercially supported software engineering modeling approaches. For example, if a state machine formalism is used, essential models of a system may be represented in statecharts supported by Statemate Magnum [Harel 87, I-Logix]. While a variety of approaches can be used in Model-Based Verification, state machine models have been the principal type used during the emergence of model checking practices.

Formal Models

Currently there is limited tool support for automated model checking. Most of the tools for software model checking [e.g., the SMV] are research tools that support formal modeling forms that are unique to the tool. Consequently it is necessary to convert models expressed in commercially supported forms into a formal model that can be readily analyzed by an automated model checking tool. One exception is the I-Logix Statemate Magnum product [I-Logix]. This tool includes the capability to model check statechart models.

Expected Properties

Model-Based Verification of a system involves analyzing a model of the system for a specific set of properties. It is therefore necessary to identify and create a list of behaviors and behavioral characteristics for the system—expected properties. These include aspects of behavior that both should and should not be exhibited for the system. For example, an elevator control system should always open the passenger doors when a desired floor is reached, and should never allow the cab to move while the doors remain open. Expected properties are expressed in plain, natural language and are converted into more formal expressions later in the process. These formal expressions—claims—are used in the automated model checking of the system.

Claims

Claims are the formal expressions of expected properties. These are generated by translating the natural language statements into the formal notation required by a model checking tool. In some cases they can be formulated directly rather than translated from natural language statements. For many of the specific model checking tools claims are expressed as temporal logic formulae For example, the SMV requires that claims be expressed in computation tree logic (CTL).

Analysis Results

The result of analysis activities is documentation that includes the raw output of automated model checking tools, the assessment of that output, and in some cases guidance for correcting defects. Model checking tools generate output that confirms the validity of expected properties (claims) for the model and in many cases identifies counterexamples of properties that are not confirmed. The counterexamples often require a detailed interpretation to understand the sequence of events that demonstrate the violation of a claim. A detailed review of a counterexample can provide insight into a potential design defect. This insight can also suggest the specific aspect of a design that should be corrected in order to prevent the anomalous behavior identified by the counterexample.

Defect Logs

Defect logs are itemized compilations of the defects (errors) found in the artifacts being reviewed.[11] This defect-capturing process is part of the document activity. The defects identified in the building and analyzing of essential models are interpreted as being potential defects in the artifact and the system that it describes. By analyzing the data in the defect log, the organization can improve its use of MBV as well as demonstrate its return on investment.

Specific data included in defect logs may vary. However, recommended items are

- unique defect ID or identifier
- date the defect was found
- activity that uncovered the defect
- specification and where in that specification the defect was found
- comments about the defect and/or a description of the defect
- classification of the type of defect found
- analysis of what caused the defect
- estimations and actuals of time/effort required to fix the defect
- phase of injection or origin of the defect
- level of certainty that the reported defect is in fact an error
- level of importance that the defect be removed (How critical is it?)

[11] It is not necessary to keep a list of errors made during the model-based analysis, although such tracking can lead to improvements in the analysis process.

References/Bibliography

[Basili 01] Basili, Victor & Boehm, Barry. “Software Defect Reduction Top 10 List.” *Computer 34*, 1 (January 2001): 135-137.

[Berezin 98] Berezin, S.; Campos, S.; & Clarke, E. Compositional Reasoning in Model Checking (CMU-CS-98-106). Pittsburgh, PA: School of Computer Science, Carnegie Mellon University, February 1998. http://reports-archive.adm.cs.cmu.edu/anon/1998/CMU-CS-98-106.ps

[Clarke 85] Clarke, E. M.; Emerson, E. A.; & Sista, A. P. “Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications.” *ACM Transactions on Programming Languages and Systems 8*, 2 (April 1985): 244-263.

[Clarke 89] Clarke, E.; Long, D.; & McMillan, K. “Compositional Model Checking.” 353-362. *Proceedings of the Fourth Annual Symposium on Logic in Computer Science - LICS '89*. Asilomar, CA, June, 1989. New York: IEEE Computer Society Press, 1989.

[Clarke 96] Clarke, E. M. & Wing, J. M. “Formal Methods: State of the Art and Future Directions” *ACM Computing Surveys 28*, 4 (December 1996).

[Comella-Dorda 01] Comella-Dorda, S.; Gluch, D.; Hudak, J.; Lewis, G.; & Weinstock, C. *Model-Based Verification: Claim Creation Guidelines*. (CMU/SEI-2001-TN-018 ADA 396125). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. <http://www.sei.cmu.edu/publications/documents/01.reports/01tn018.html>

[Corbett 00] Corbett, James C.; Dwyer, Matthew B.; Hatcliff, John; & Robby. “A Language Framework For Expressing Checkable Properties of Dynamic Software.” *Lecture Notes in Computer Science.* New York, NY: Springer-Verlag, 2000.

[Dijkstra 69] Dijkstra, E. W. “Structured Programming,” *Report of a Conference sponsored by the NATO Science Committee*. Rome, Italy, Oct. 27-31, 1969. Brussels, Belgium: Scientific Affairs Division, NATO, 1970.

[Ebenau 94] Ebenau, Robert G. & Strauss, Susan H. *Software Inspection Process*. New York, NY: McGraw-Hill, 1994.

[Fagan 76] Fagan, M. “Design and Code Inspections to Reduce Errors in Program Development." *IBM Systems Journal 15,* 3 (1976): 182-211.

[Gluch 98] Gluch, D. & Weinstock, C. *Model-Based Verification: A Technology for Dependable System Upgrade* (CMU/SEI-98-TR-009, ADA354756). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1998. <http://www.sei.cmu.edu/publications/documents /98.reports /98tr009/98tr009abstract.html>

[Gluch 99] Gluch, D. & Brockway, J. *An Introduction to Software Engineering Practices Using Model-Based Verification* (CMU/SEI-99-TR-005 ADA366089). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999. <http://www.sei.cmu.edu/publications/documents/99.reports /99tr005/99tr005abstract.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, 2002. <http://www.sei.cmu.edu/publications/documents/01.reports /01tn024.html>

[Gluch 02] Gluch, D., Comella-Dorda, S.; Hudak, J.; Lewis, G.; & Weinstock. C. *Model-Based Verification—Guidelines for Generating Expected Properties*.(CMU/SEI-2002-TN-003 ADA399228) Pittsburgh, PA : Software Engineering Institute, Carnegie Mellon University, 2002. <http://www.sei.cmu.edu/publications/documents/02.reports /02tn003.html>

[Harel 87] Harel, D. “ Statecharts: A Visual Formalism for Complex Systems,” *Science of Computer Programming 8* (1987) 231-274.

[Holzmann 91] Holzmann, J. *Design and Validation of Computer Protocols*, Upper Saddle River, NJ: Prentice Hall, 1991. SPIN available: <http://netlib.bell-labs.com/netlib/spin/whatispin.html>

[Hudak 02] Hudak, J.; Comella-Dorda, S.; Gluch, D.; Lewis, G.; & Weinstock, C. *Model-Based Verification: Abstraction Guidelines* (CMU/SEI-2002-TN-011). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.

[Humphrey 95] Humphrey, W. *The Personal Process in Software Engineering*. (article) Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1995. <http://www.sei.cmu.edu/publications /documents/95.reports/95.ar.psp.swe.html>

[I-Logix] Statemate Magnum. <http://www.ilogix.com/>

[Jackson 00] Jackson, D.; Schechter, I.; & Shlyakhter, I. “Alcoa: the Alloy Constraint Analyzer” *Proc. International Conference on Software Engineering*, Limerick, Ireland, June 2000. <http://sdg.lcs.mit.edu/~dnj/publications.html>

[Kowalewski 97] Kowalewski, Stefan & Treseler, Heinz. “VERDICT – A Tool for Model-Based Verification of Real-Time Logic Process Controllers.” *Proceedings of the 1997 Joint Workshop on Parallel and Distributed Real-Time Systems*, Geneva, Switzerland, April 1997.

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

[Linstone 75] Linstone, H. A. & Turoff, M. “The Delphi Method: Techniques and Application.” New York, NY: Addison-Wesley, 1975.

[McMillan 93] McMillan, Kenneth L. *Symbolic Model Checking*, Norwell, MA: Kluwer Academic Publishers, 1993.

[McMillan 92] McMillan, Kenneth L. *The SMV System*.
<http://www.cs.cmu.edu /~modelcheck/smv.html>

[McMillan 99a] McMillan, Kenneth L. *Cadence SMV.*
<http://www-cad.eecs.berkeley.edu/~kenmcmil/>

[McMillan 99b] McMillan, K. *Verification of Infinite State Systems by Compositional Model Checking*. Berkeley, CA: Cadence Berkeley Labs, 1999.

[NuSMV] NuSMV: A New Symbolic Model Checker <http://nusmv.irst.itc.it/>.

[Saaty 80] Saaty, T.L. *The Analytic Hierarchy Process*. New York, NY: McGraw-Hill, 1980.

[Ball 01] Ball, Thomas & Rajamani, Sriram K. “Automatically Validating Temporal Safety Properties of Interfaces,” 103-122. SPIN 2001, *Workshop on Model Checking of Software*, LNCS 2057, Toronto, Canada, May 2001.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY
(Leave Blank)

2. REPORT DATE
October 2002

3. REPORT TYPE AND DATES COVERED
Final

4. TITLE AND SUBTITLE
Model-Based Verification: An Engineering Practice

5. FUNDING NUMBERS
F19628-00-C-0003

6. AUTHOR(S)
D. Gluch, S. Comella-Dorda, J. Hudak, G. Lewis, J. Walker, C. Weinstock, D. Zubrow

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER
CMU/SEI-2002-TR-021

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

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ESC-TR-2002-021

11. SUPPLEMENTARY NOTES

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

12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

Model-Based Verification (MBV) involves building and analyzing formal models of a system as an approach to identifying and guiding the correction of defects in software engineering artifacts. This report summarizes MBV and outlines the responsibilities of engineers engaged in Model-Based Verification. Each of the practices is described together with an initial set of guideline documents. These descriptions include procedural information, technical foundations for the practice, and engineering techniques for an MBV practitioner.

14. SUBJECT TERMS
Model-Based Verification, MBV, essential models, expected properties

15. NUMBER OF PAGES
49

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT
Unclassified

20. LIMITATION OF ABSTRACT
UL