Technical Report

CMU/SEI-91-TR-005
ESD-91-TR-005

Evaluation of Process Modeling Improvements

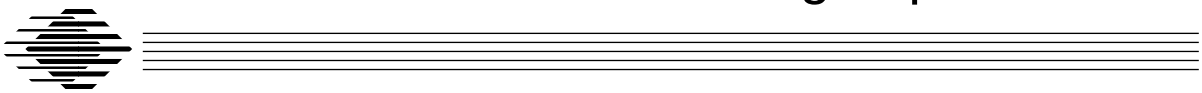
Robert W. Krut, Jr.
David P. Wood

April 1991

Technical Report

CMU/SEI-91-TR-005
ESD-91-TR-005
April 1991

Evaluation of Process Modeling Improvements



Robert W. Krut, Jr.
David P. Wood

Software Process Program

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Evaluation of Process Modeling Improvements

Abstract: The SEI has been involved with the development and analysis of software process models for several years. As part of the ongoing process of technology evolution, a study has been undertaken to experimentally implement a set of proposed improvements to the process modeling techniques used by the SEI, and to evaluate the results of that experimentation. As a result of that study, a number of modifications to our techniques have been identified. These modifications enhance the support of software engineering concepts in the development and use of process models. This report describes the study and elaborates upon the advantages and disadvantages of the proposed technique improvements.

1. Introduction

This report describes the results of the introduction of certain modifications to the process modeling techniques used at the Software Engineering Institute (SEI) as part of the Software Process Modeling Project. The purpose of these modifications, which arose as part of natural technology maturation, was to provide increased clarity and manageability to the resulting models. The specific changes introduced to the techniques evolved over time and were the result of several iterations of proposal, experimentation, and evaluation.

This report details the changes and the rationale behind their introduction; the report provides examples of the revised techniques used in practice.

1.1. Background

In recent years, the software engineering research community has begun to focus significant attention on the processes used to develop and support software. This serves as a complement to the more traditional focus on the products of those processes and the tools utilized by those processes. This attention has led to the development and application of approaches to modeling and analyzing software processes [12, 15, 7, 14]. These models provide a means of reasoning about the organizational processes used to develop and support software. Software Process Modeling is seen as an emerging technique that will contribute to the improvement of software processes and their corresponding products.

Significant experience with process modeling has been gained at the SEI in recent years [15, 14, 13]. Our modeling experience includes major efforts at modeling large-scale software support processes in use by the U.S. Department of Defense. Previous software process models developed at the SEI include a model of the process used by the U.S. Navy to support the operational software for the F-14A aircraft and a model of the process used by the US Air Force to support the operational flight program for the F-16A/B aircraft. These models depict the full Post-Deployment Software Support (PDSS) process, from receipt of a software trouble report, change request, or enhancement request, through to release of the

corresponding software change to the field. The F-14A and F-16A/B models are descriptive of actual practice. The most recent SEI-developed process model is a prescriptive model drawn from an existing specification. It is based upon MIL-HDBK-347, "Mission-Critical Computer Resources Software Support" [16]. This Software Support Handbook (SSH) "presents software support concepts, procedures, and guidance to all managers responsible for Mission-Critical Computer Resources (MCCR) development or support." SEI has modeled and analyzed those portions of the SSH pertaining to PDSS [11].

Currently, the process models developed at the SEI are created using a commercially available automated system, STATEMATE[1] [5, 10, 8, 9], to represent, analyze, and simulate software processes. The SEI software process modeling techniques depict a process in terms of who, what, where, when, and how, using the representations provided by STATEMATE. These representations include three types of graphical diagrams (activity charts, statecharts, and module charts) and supplemental textual "forms." The three types of diagrams are used to represent three different viewpoints of a process (functional, behavioral, and organizational, respectively), while the textual forms are used to describe connections among the three viewpoints, formal definitions, and informal narrative descriptions. The use of these representations is described elsewhere [8] and is not detailed in this report.

Because process modeling is a young and evolving technology, valuable lessons are learned as each new process model is developed. Frequently, schedules do not allow the introduction of changes to the existing techniques as models are being developed; however, it is desirable to examine new concepts as time permits so that modeling techniques can be revised for application to subsequent models.

Several independent efforts are presently underway at the SEI to examine enhancements to the modeling techniques with an eye to possible future use. One of those efforts, the subject of this report, involves the introduction of extensive structure into process models. Other efforts currently underway are examining the introduction of quantitative information and the modeling of aggregate processes.

1.2. Purpose

The purpose of this report is threefold: first, to describe the concepts and reasoning behind the proposed structural improvements to process modeling; second, to describe the mechanisms that were ultimately developed to embody the proposed improvements; and third, to capture the essence of the results of experimentation and evaluation of the technique modifications.

[1] STATEMATE is a trademark of i-Logix, Inc., Burlington, MA.

1.3. Target Audience and Report Organization

The intended audiences for this report are: those interested in the summary results of the described effort, and those interested in the rationale behind the results. To accomodate both audiences, the report is organized into two parts: the main body, and a set of appendices. The body of the report describes the impetus behind the effort and the overall results, while the appendices provide the detailed analysis and sample models.

The body of the report is divided into the following sections:

- Section 2, **Approach**, chronologically describes the various phases of this effort, including implementation of technique modifications and evaluation of results.
- Section 3, **Summary of Findings**, describes, in abstract fashion, the findings of this effort. Results are placed in the larger context of software engineering concepts.
- Section 4, **Conclusions**, provides overall conclusions regarding proposed changes to SEI process modeling techniques.

Detailed technical discussion of the proposed technique modifications, and descriptions of our implementation of those modifications, are provided in several appendices:

- Appendix A: **Revisions to the Modeling Techniques**, details each of the proposed technique revisions.
- Appendix B: **Revision of the Baseline Model**, describes the results of an experimental implementation of the proposed revisions. The revisions were introduced into a pre-existing process model.
- Appendix C: **Analysis of Advantages and Disadvantages**, discusses a comparison of the baseline and revised models described in Appendix B, and provides detailed descriptions of the advantages and disadvantages of introducing the various technique revisions.
- Appendix D: **Baseline Charts of the SSH Model**, provides copies of the activity and statecharts of the baseline model discussed in Appendices B and C.
- Appendix E: **Revised Charts of the SSH Model**, provides copies of the activity and statecharts of the revised model discussed in Appendices B and C.

2. Approach

In recent years, several process models have been developed at the SEI as part of the Software Process Modeling Project. These process models have depicted processes of various sizes, characteristics, and complexity; as such, they provide a valuable basis for critique. For example, the most recent process model depicts approximately 50 functional activities and about as many behavioral states. The activities are related to one another hierarchically and communicate with one another through the flow of information. Similarly, the various behavioral states are related hierarchically and are interrelated based on a multitude of semantic rules embodied in the language of statecharts [8].

The existing SEI process modeling techniques represent the various activities and states within two "monolithic" charts: a single activity chart depicts all activities within the process, while a single statechart depicts all states. These charts are referred to respectively as the "functional view" and the "behavioral view"; these two views relate to one another implicitly by way of certain scoping and visibility rules, or explicitly through flow of information and activity control.[2]

2.1. Identification of Concerns

As part of an ongoing technical review, members of the process modeling team identified a number of concerns related to the existing modeling techniques. At the earliest stages of review, these concerns were identified in general terms based primarily upon their visual manifestation. In particular, it was noted that as the processes being modeled became increasingly complex, the visual representations of those processes became more and more complex as well. As a result, the ability to understand and manage complex models was seriously limited by the existing modeling techniques.

Further review, including the effort described in this report, helped to identify the essence of the deficiency of the modeling techniques: a lack of sufficient structure. Structure is partially embodied in the existing techniques in that hierarchical (compositional) relationships are depicted in both the functional and behavioral views. However, additional structuring is needed to alleviate problems that arise as the complexity of the models increases. Some of the identified problems include:

- **Visual clutter**: as discussed above, increased conceptual complexity of the process results in increased visual complexity of the model. Once the conceptual complexity of a process is beyond a finite limit, resulting models will be difficult or impossible to understand and manage.
- **No change localization**: because monolithic charts are used and access to data is global, all changes to the model are global in nature, complicating development and management of the model. Incremental testing is not possible.

[2]The use of the third type of STATEMATE diagram, module charts, is not addressed by this report.

- **No parallel development**: because monolithic charts are used, it is impossible for multiple process modelers to modify different components of the same model simultaneously.
- **Information overload**: because monolithic charts are used, all process details are visible on a single chart. While this is arguably desirable for simple processes, with complex models this approach can lead to information overload for those who must use the models. Mechanisms for controlling this visibility within the existing modeling techniques are cumbersome.
- **Loose connection between views**: the correlation between behavioral and functional views of the process is largely implied. Although the behavioral view explicitly controls activities, control information from the functional view generally is not depicted to avoid further cluttering of the charts. Instead, the two views are connected using informal guidelines of vague name association. One side-effect of this loose connection is that animations of the model require extensive user inputs to drive them.
- **Poor support for model alternatives**: because monolithic charts are used and access to data is global, building alternative sub-processes to assist in what-if analyses is difficult.

2.2. Proposed Improvements

Based on the concerns identified in the previous section, several potential improvements to the modeling techniques were proposed. Some of the proposed improvements were related to new features and improvements to STATEMATE, while others were based on methodological revisions that were actually supported in previous releases of the tool. These proposed improvements are described in detail in AppendixA, and are summarized below.

Four fundamental changes to the modeling technique were proposed:

1. **Decluttering of functional view**: this modification is based on a new feature of the modeling tool and results in the physical division of the monolithic activity chart into multiple charts based upon levels of composition. Used to the extreme, decluttering results in a set of activity charts identical in nature to a leveled set of data flow diagrams, such as those used by other structured analysis methods [2, 4, 19, 18] and most CASE tools.
2. **Distribution of behavioral view**: this modification results in the localization of behavioral specification. For each activity level, a statechart is created that contains only the behavior relevant to that particular activity level. This scheme is recommended by the tool vendor and is precisely analogous to that used by other real-time structured analysis methods [1, 6, 20], for the development of software systems.
3. **Constraining visibility of elements**: this modification is based on enhancements to the modeling tool and results in the localized visibility of all elements of the model based on implicit (graphic) and explicit (textual) scoping restrictions.
4. **Providing tighter connection of views**: this modification results in the explicit connection of the functional view with the behavioral view through such

things as the detection of reading and writing data. Events generated by these actions are used to drive state changes in the behavioral view.

2.3. Experimental Implementation

In order to provide a basis for determining the value of the proposed modifications to the modeling technique described in the preceding section, the proposed modifications were implemented experimentally. The implementation of the modifications is described fully in Appendix B, and summarized below.

To facilitate a direct comparison of models exhibiting the original baseline modeling techniques as well as the proposed revised techniques, an existing model was chosen for partial redevelopment. The recently completed Software Support Handbook (SSH) model was selected for redevelopment. In this report, the original model is referred to as the **baseline model**, while the model resulting from redevelopment is referred to as the **revised model**.

Although the SSH model was redeveloped to incorporate the proposed technique revisions, care was taken not to alter the functionality or behavior represented by the baseline model. In other words, the processes represented by the baseline and revised models are identical. Although time did not permit the redevelopment of the entire baseline model, one vertical fragment of the baseline was redeveloped to the primitive level. The remainder of the model was redeveloped to a lesser degree.

2.4. Evaluation of Results

The baseline and revised models (described previously in Section 2.3) were compared and contrasted as part of an iterative process of review and critique. Extensive debate among the Process Modeling project members resulted in a comprehensive evaluation of the revised modeling techniques versus the baseline modeling techniques.

The findings of the evaluation process are summarized in Section 3, and are discussed in complete detail in Appendix C.

3. Summary of Findings

Section 2 enumerated several concerns regarding the baseline process modeling techniques used by the SEI. These concerns were identified as part of the normal process of iterative methodology enhancement. In addition, proposed solutions to these concerns were outlined, an experimental implementation of the solutions was conducted, and the results were evaluated. The product of the resulting evaluation is an extensive list of advantages and disadvantages presented in detail in Appendix C.

A broader interpretation of these advantages and disadvantages reveals that the revised techniques embrace the fundamental system design principles of modularity, information hiding, abstraction, and structure.

The revised techniques provide better support for **modularity** in several ways. The physical separation of model components lends itself to parallel development of the model by a team of modelers, and also supports a "divide and conquer" approach to testing the model. The visibility constraints introduced as part of enhanced scoping enforcement make each composition level more of a stand-alone unit.

Similarly, the revised techniques dramatically improve support for **information hiding**. Functional, behavioral, and data details that are not relevant to a particular composition level are hidden from view. Although the baseline modeling approach provides some mechanisms for information hiding, they are primitive and cumbersome and must be implemented manually.

The concepts of **abstraction** are also better supported in the revised techniques. Although conceptual abstraction is supported to a degree in the baseline techniques as a result of the use of leveled composition, the revised techniques provide the basis for the creation of alternative representations for sub-processes. Depending on its use, this feature can be considered analogous to the use of library subroutines or generic components in a programming language. Further, the enhanced scoping rules of the revised techniques support the limited reuse of labels, which also improves abstraction capability.

In a general sense, modularity, information hiding, and abstraction are all part of the notion of **structure**. In this sense, the revised techniques have been shown to enhance model structure. In addition, the revised techniques substantially improve structure by introducing a more natural philosophy of separation of concerns. In the baseline model, function and behavior are forcefully separated on a global scale, resulting in a model that exhibits poor cohesion and thus poor structure. By contrast, the techniques employed in the revised model closely associate the behavior of a function with the function itself, and instead implement separation of concerns by dividing the model based on natural functional distinctions.

In addition to the above improvements, the revised techniques also impact subjective issues such as aesthetics and practicality of the resulting models. One example is the substantially reduced visual clutter evident in a given baseline model chart in contrast to a given revised

model chart. A second example is that the use of the revised techniques makes model documentation and distribution more practical because the resulting charts are more readily printed on standard-size paper using laser printers rather than expensive and inconvenient plotters. Both of these examples are evidenced in the charts shown in Appendices D and E, while these and other issues of aesthetics and practicality are discussed further in Appendix C.

The evaluation of the revised techniques also revealed a number of disadvantages. Most of the identified disadvantages are operational or logistical trade-offs related to the introduction of the structural restrictions discussed above. Some examples of disadvantages of the revised techniques include:

- The need to manage many more charts.
- The tool-imposed limitations on the number of windows that may be opened at one time; under certain circumstances these limitations may be intrusive.
- The introduction of some types of changes into the model may be more time-consuming.
- The inability to view all levels of detail of the model on a single physical chart.
- The inability to globally control activities arbitrarily due to scope enforcement.

These and other disadvantages are detailed in Appendix C.

4. Conclusions

A *system* has been defined as "an organized set of doctrines, ideas, or principles usually intended to explain the arrangement or working of a systematic whole... an organized or established procedure [21]." A *software process* has been defined as "the technical and managerial framework established for applying people, methods, tools, and practices to the development and evolution of software [7]." By these definitions, it is evident that processes are systems.

The design principles of modularity, information hiding, abstraction, and structure are well-established as mechanisms of fundamental importance for the specification and design of systems and software. It is reasonable to assume that these principles are equally valid for specification and design of process models.

The efforts described in this report support this assumption. Indeed, the concerns regarding the baseline modeling techniques, raised in Section 2.1, are merely manifestations of the lack of support in those techniques for these fundamental design principles.

Furthermore, while the baseline techniques are unique as a modeling methodology, the revised techniques are highly consistent with a large number of other methods and tools. In particular, the revised techniques are compatible with commonplace structured analysis and real-time structured analysis techniques. The use of the revised techniques will provide process modelers with greater freedom in the selection of support tools. Even more compelling is the fact that the baseline techniques are no longer supported in their entirety by the lastest version of STATEMATE.

In addition to the support of basic design principles, the revised techniques also enhance the aesthetics and manageability of complex process models, while introducing no serious drawbacks. Most of the identified disadvantages of the revised techniques are attributable to operational or logistical nuisances resulting from the introduction of structure to the models.

While it may be possible or even desirable to employ some of the baseline techniques for simple processes, or for selected subprocesses of more complex processes, the findings of this effort indicate clearly that the proposed revisions to the process modeling techniques should be used for modeling processes of any significant complexity.

References

1. Bruyn, W., Jenson, R., Keskar, D., Ward, P. "ESML: An Extended Systems Modeling Language". *ACM Software Engineering Notes 13*, 1 (January 1988), pp. 58-67.

2. DeMarco, T.. *Structured Analysis and System Specification.* Yourdon, Inc., New York, 1978.

3. Fairley, Richard E.. *Software Engineering Concepts.* McGraw-Hill, Inc., 1985.

4. Gane, C. and Sarson, T.. *Structured Systems Analysis: Tools and Techniques.* Prentice-Hall, 1979.

5. Harel, David, *et al.*. "STATEMATE: A Working Environment for the Development of Complex Reactive Systems". *IEEE Transactions on Software Engineering 16*, 4 (April 1990), 403-414.

6. Hatley, D., and Pirbhai, I.. *Strategies for Real-Time System Specification.* Dorset House, New York, 1987.

7. Humphrey, Watts S. and Kellner, Marc I. Software Process Modeling: Principles of Entity Process Models. *Proceedings of the 11th International Conference on Software Engineering*, IEEE, May 1989, pp. 331-342.

8. *The Languages of STATEMATE.* i-Logix, Inc., 22 Third Avenue, Burlington, MA 01803, July 1990.

9. *STATEMATE; Support for Large Scale Projects; Version 3.0.* i-Logix, Inc., 22 Third Avenue, Burlington, MA 01803, January 1990.

10. *STATEMATE; Kernel; Sun Version 3.0.* i-Logix, Inc., 22 Third Avenue, Burlington, MA 01803, January 1990.

11. Jewart, J. Kelly and Kellner, Marc I. Modeling and Analysis of the Software Support Handbook Process. Unpublished.

12. Kellner, Marc I. Representation Formalisms for Software Process Modeling. *Proceedings of the 4th International Software Process Workshop: Representing and Enacting the Software Process*, Published in *ACM Software Engineering Notes 14*, 4 (June 1989), pp. 93-96.

13. Kellner, Marc I. Experience with Enactable Software Process Models. *Proceedings of the 5th International Software Process Workshop: Experience with Software Process Models*, October 10-13, 1989.

14. Kellner, Marc I. Software Process Modeling: Value and Experience. In *SEI Technical Review*, Software Engineering Institute, Carnegie Mellon University, 1989, pp. 23-54.

15. Kellner, Marc I. and Hansen, Gregory A. Software Process Modeling. Tech. Rept. CMU/SEI-88-TR-9, DTIC: ADA197137, Software Engineering Institute, Carnegie Mellon University, May 1988.

16. *(MIL-HDBK-347) Military Handbook: Mission-Critical Computer Resources Software Support.* U.S. Department of Defense, May 1990.

17. Pressman, Roger S.. *Software Engineering: A Practitioner's Approach.* McGraw-Hill, Inc., 1987.

18. Ross, D. T. "Structured Analysis (SA): A Language for Communicating Ideas". *IEEE Transactions Software Engineering SE-3* (January 1977), pp. 16-34.

19. Wallace, R., Stockenberg, J., and Charette, R.. *A Unified Methodology for Developing Systems.* McGraw-Hill, New York, 1987.

20. Ward, P. and Mellor, S.. *Structured Development for Real-Time Systems.* Yourdon Press, Prentice Hall, New York, 1985.

21. Dictionary. *Webster's Ninth New Collegiate Dictionary.* Merriam-Webster, Inc., Springfield, MA, 1985.

22. Wood, David P. and Wood, William G. Comparative Evaluations of Four Specification Methods for Real-Time Systems. Tech. Rept. CMU/SEI-89-TR-36, DTIC: ADA219187, Software Engineering Institute; Carnegie Mellon University, December 1989.

Appendix A: Revisions to the Modeling Technique

This appendix briefly describes the changes examined in this report. These potential improvements hold promise of making software process models more manageable, as well as easier to develop, analyze, and understand. The changes fall into two categories: those that are the result of tool enhancements and those that are the result of a modification of techniques. Changes of the first category include features to "declutter" the charts and to "scope" the elements. These tool enhancements were included in the recent STATEMATE upgrade to Version 3.0 and are described in [9]. These additions to STATEMATE were created to enhance the support of large scale projects. The second category of changes includes distributed statecharts and alternative methods for controlling the charts of STATEMATE. These modifications were options previously available in STATEMATE and are described in [5, 8, 9]. Specific examples of each of these changes to the SSH model will be discussed in Section B.2.

A.1. Decluttering of Charts

In STATEMATE, the concept of leveling is used to decompose the system under development (SUD) into a set of sub-systems (e.g., functional decomposition when creating an activity chart). In this respect, STATEMATE models are similar to traditional structured analysis/ data flow diagram techniques.

In the previous versions of STATEMATE, the resulting multi-level description was presented on a single diagram (or chart) for each perspective. Three charts would then diagrammatically represent the three perspectives of the SUD. Figure A-1 provides a general example of a multi-level "monolithic" activity chart.

In STATEMATE Version 3.0, charts may be decluttered using a feature referred to as the "box-is-chart" operation. This enables the modeler to present individual levels of decomposition on separate charts (or pages). STATEMATE treats the multiple physical pages as a single logical chart. Figure A-2 provides a decluttered view of the activity chart in Figure A-1. The single monolithic chart is represented by two separate charts. STATEMATE uses a special "@" notation to denote a box whose contents are represented on a separate chart. For example, the box named "ACT_2@B" means that the internal description of activity ACT_2 appears in Chart B.

Figure A-1: A Multi-level Monolithic Activity Chart

Figure A-2: A Decluttered Activity Chart

The STATEMATE vendor, i-Logix, recommends decluttering when the details of a chart are visually complex, when information hiding between the subsystems is desired, or when several alternatives need to be presented by changing the contents of a "black box" (i.e., to enhance interchangeability of system components). Decluttering is not recommended when a system does not lend itself to neat structuring, when a tight inter-relationship exists between low-level elements, or when the resulting charts are found to be too cumbersome to work with. Decluttering may be applied to all STATEMATE charts.

Decomposing a system into multiple charts raises issues of visibility and the scope of elements. These issues (i.e., scoping of elements and distributed statecharts) are addressed in Sections A.2 and A.3.

A.2. Scoping of Elements

A strict enforcement of scoping rules is the second additional feature of STATEMATE Version 3.0. Scoping rules determine where an element is "visible" (i.e., the set of charts in which an element is known and can be used).

Although the SUD was decomposed into multiple levels in STATEMATE Version 2.5, the scoping rules were not enforced within the monolithic chart; all of the elements were considered "global," that is, known throughout the model. Element names were required to be unique and could be referenced from anywhere within the system description.

In STATEMATE Version 3.0, elements may be global (visible to several charts) or "hidden" inside specific charts depending on where the element is "defined" and the set of scoping rules that determine where the element is visible. Graphical elements (activities, data-stores, states, and modules) are defined in the chart in which they are drawn. Textual elements (data_items, information flows, events, conditions, and actions) are defined in the chart that is specified in that element's textual form.

The parent-child-sibling relationship, created by decomposing the system into multiple charts, is the basis for the set of scoping rules that govern the visibility of an element. In STATEMATE Version 3.0, textual elements are visible to the chart in which it belongs and to all its descendant charts. For example, a textual element defined in activity ACT_2 of Figure A-2 is visible to ACT_2, ACT_21, ACT_22, ACT_23, ACT_24, and ACT_25, but is not visible to activities ACT_1 and ACT_3. The textual element must be defined in the highest chart of the structure in which it is to be used. Textual element names must be unique within each chart. However, any other chart may define a different textual element using the same name. STATEMATE refers to redefining a textual element as "masking." Graphical elements are visible only to sibling graphical elements. For example, a statechart may control only activities that are siblings of the control activity described by that statechart. The graphical element names must be unique among its sibling graphical elements. The scope of a graphical element is discussed in greater detail in Section A.3.

A.3. Distributed Statecharts

Under STATEMATE Version 3.0, activity charts are developed by decomposing the SUD into a set of subsystems. These subsystems are represented by multiple pages that are treated as a single logical chart. Statecharts are developed quite differently. The SUD can not be driven by a single logical statechart. Scoping rules restrict the controlling environment of a statechart. A statechart may control only activities that are siblings of the control activity described by this statechart. It cannot be used to control child activities within the siblings or the parent activity in which it is defined. Therefore, statecharts must be distributed, i.e., separate statecharts are required to adequately represent the behavior of each non-primitive activity. Although distributed statecharts were suggested in previous versions of STATEMATE, the scoping restrictions of graphical elements were not sufficiently enforced to require their use.

Developers of the previous SEI process models chose not to employ the distributed statechart technique because it was felt that a monolithic statechart provided a better understanding of the overall behavior of the SUD. In those monolithic models, a single control activity referred to a single statechart describing the entire behavior of all the activities. Therefore, the control activity CONT_1 would refer to a single statechart to control all of the activities in Figure A-3[3]. With distributed statecharts, the statechart referred to by CONT_1 may control ACT_1, ACT_2, and ACT_3. The statechart cannot control ACT_21, ACT_22, ACT_23, ACT_24, or ACT_25. A second controlling activity would have to be included within the ACT_2 activity since the activities within ACT_2 are not visible to the statechart referred to by CONT_1. Figure A-4 demonstrates the use of distributed statecharts. The control activity CONT_2 refers to the statechart that describes the overall behavior of ACT_1, ACT_2, and ACT_3. The control activity CONT_3 refers to the statechart that describes the behavior within ACT_2. The control activities within the monolithic and decluttered activity charts use the "@" notation to refer to the statechart describing the behavior of the activities.

A.4. Controlling of Charts

Statecharts are responsible for controlling the activation and deactivation of activities, and the timing of flow of information between them in activity charts. SEI process models developed using previous STATEMATE versions employed the following techniques to control the charts.

- Events and conditions to trigger transitions within the charts.
- A "throughout" command to activate and deactivate activities.
- A "history" connector to re-enter a group of states.

[3]Figures A-3 and A-4 are similar to Figures A-1 and A-2 with the addition of the control activities.

Figure A-3: A Multi-level Monolithic Chart

Figure A-4: A Decluttered Activity Chart with Distributed Control

The "throughout" command was defined within the textual form associated with the statechart; the "history" connector was a re-entry connector to a system (or sub-system) that returns control to the most recently visited state within a group.

Although these modeling techniques are valid ways to control the charts, alternative methods of control may provide improvements to the model in the following areas:

- Minimizing the interaction with the user during animation.[4]
- Reducing the visual complexity of the activity charts by reducing the required number of events and conditions appearing on the charts.
- Creating a better visual link between an information flow and the transitioning event associated with it.
- Designing the models to better represent the actual processes.

The techniques used are not new to the current languages of STATEMATE, but rather are a different way of developing the models. These techniques are considered to be extensions to the current modeling techniques.

STATEMATE provides multiple techniques to develop the control within the charts. Several of these techniques are examined in this report to determine if they enhance the model in the ways just described. This examination consists of substituting alternative methods of control into the process model. These alternatives are:

- Using the "write_data/written" action/event pair to trigger transitions.
- Associating an action with the event of "entering" or "exiting" a state.
- Using the "start/stop" actions to activate and deactivate activities.
- Using the "suspend/resume" actions to temporarily exit and return to a state.

The "write_data/written" action/event pair could alternatively be replaced by the "read_data/read" action/event pair for triggering transitions within the charts.

In all STATEMATE versions, events and conditions are used to trigger transitions within the charts. By using the "write_data/written" action/event pair (abbreviated wr! and wr, respectively), an attempt is being made to better automate the transitioning process between states. The "write_data" action creates an event "written" that is immediately sensed and the transition to the next state is initiated. This combination creates a sequence of steps that automate the transition between states and reduces the need for externally inputting events during animation by self generating the event.

The "entering" command associates an action with the event of having entered a particular state, and it is usually defined within the "Reactions" fields in the associated textual form for a state. In this report, the "entering" event is used in conjunction with the "write_data/written"

[4]Animation is the continuous display of the current status of a simulation by visually highlighting the active graphical elements in the developed model [22].

action/event pair. This combination creates a sequence that, upon entering a particular state, generates the action to write (or assign) information and transition to the next state once the information has been written (or assigned).

Figure A-5 provides an example of the "entering" event used in conjunction with the "write_data/written" action/event pair. When event E_1 occurs, the statechart transitions from state STA_A to state STA_B. Within the textual form of state STA_B, the action to assign a value to a data-item (which is referred to as "X" in the following discussion)[5] is associated with the event of "entering" the state. Therefore, upon entering state STA_B, the action wr!(X), "write_data(X)," occurs. If event E_2 is defined as wr(X), "written(X)," the transition to state STA_C occurs the instant the assignment takes place since X has been assigned a value. This example demonstrates the ability to create transitions on a statechart without externally entering an event. The "start/stop" actions (abbreviated st! and sp!, respectively) are not much different from the "throughout" command. Both activate and deactivate activities. However, the "start/stop" explicitly starts and stops an activity, while the throughout implicitly activates an activity when the associated state is entered and deactivates the activity when the state is exited.

The "suspend/resume" (abbreviated sd! and rs!, respectively) must be used in conjunction with "start/stop" actions. The "suspend/resume" actions cause an activity to "freeze" or suspend, and resume from where it left off. Everything about the previous status of the state is "remembered." The "history" entrance returns to the state from which it exited but as a new entry into that state. All actions that are performed upon entry are repeated.

Figure A-6 represents a portion of a statechart where the "start/stop" and "suspend/resume" actions may be employed. The actions to "start," "stop," "suspend," or "resume" may occur within the textual forms of a statechart or be labeled with the event which triggers a transition. For example, if an activity is to be active during state STA_B, it may be explicitly activated by one of the following:

- Defining the "start" action with the event of "exiting" state STA_A in the textual form of STA_A.
- Labeling the transition from state STA_A to state STA_B with the trigger that causes it and the action to be taken.

In the second example, the event E_1 would appear as E_1/st!(activity). The action to start the activity occurs instantaneously when the transition is taken.

[5]Data-item X would appear as the data flow label exiting the current activity within the activity chart.

Figure A-5: A Self-Generating Transition

Figure A-6: Behavioral View of an Activity (ACT)

As an example of the use of the "start," "stop," "suspend," and "resume" actions, let Figure A-6 represent the behavioral view of an activity (ACT), that is active during state STA_B, suspended during states STA_D or STA_E, and idle for states STA_A and STA_C. For this example, assume that activity ACT is active when a suspension occurs. Reasons for suspensions may include requests for additional information, approval from an external source, etc.. In this example, the actions will be defined within the textual forms of the states. The instructional layout of the statechart for activity ACT would be as follows:

- Within the textual form of state STA_A, the "exiting" event is associated with the action to "start" activity ACT, st!(ACT).
- Within the textual forms of both STA_D and STA_E, the "entering" event is associated with the action to "suspend" ACT, sd!(ACT); the "exiting" event is associated with the action to "resume" ACT, rs!(ACT).
- Within the textual form of STA_C, the "entering" event is associated with the action to "stop" activity ACT, sp!(ACT).

When event E_1 occurs, STA_A is exited, activity ACT is activated, and the statechart transitions to state STA_B. Activity ACT remains active unless events E_3, E_5, or E_2 occur. If the control activity senses the event E_3, the statechart transitions to STA_D, and activity ACT is suspended. The activity remains suspended until the event E_4 occurs. At this time, STA_D is exited, activity ACT is resumed and the statechart transitions back to STA_B. The event E_5 from STA_B would yield the same sequence of steps. The suspension of activity ACT by states STA_D and STA_E could represent a need for two different activities to be active when ACT is suspended. Finally, the event E_2 will transition the statechart from STA_B to STA_C. Upon entering STA_C, activity ACT is explicitly stopped.

Please refer to the "Languages of STATEMATE" [8] for a more in-depth explanation of these events and actions.

Appendix B: Revision of the Baseline Model

This appendix discusses specific examples of the changes to the Software Support Handbook (SSH) model for this investigation.

B.1. Modeling Context

The recently completed SSH Process Model was the starting point for this investigation. This process model was redeveloped with the potential modeling improvements provided by the upgrade to STATEMATE Version 3.0 and the alternative methods to control the charts as discussed in Appendix A. The re-development emphasized transforming the handbook drawings into a second set of drawings without changing the functionality or behavior of the model. To enable a comparison of the models, no additional system information was incorporated into the basic model. The information that was not available at the time the SSH process model was developed was not incorporated into the basic model before this investigation started and is therefore carried over into the redeveloped model.

For this analysis, only the activity charts were decomposed via the box-is-chart operation provided in STATEMATE because when decluttering was applied to the activity charts, the resulting distributed statecharts were not complex enough to warrant their own decluttering. In addition, since Version 3.0 does not require a complete separation of the leveled charts (i.e., decluttering to the primitive levels), the activity charts were decluttered to a "comfortable level of complexity." This reduced the time required to generate the additional charts, yet yielded charts that could be easily viewed when printed within this report.

Statecharts were not developed for each non-primitive activity. This investigation focused on developing a set of statecharts at each level of decomposition. This focus enabled an understanding of the development of a statechart for a particular activity within each level of the decomposition, while minimizing the number of statecharts developed during this investigation. Consequently, portions of the original SSH model's "lowest levels" behavior was omitted.

Module charts were not included in this examination.

B.2. Model Descriptions

The activity chart and statechart for the baseline SSH Process Model are presented in Appendix D. The redeveloped SSH process model charts are located in Appendix E. The chart sets are referred to as the baseline and revised charts, respectively. Enlarged, color copies of the baseline and revised charts are available for inspection in the SEI library.

The baseline charts define the system description with a single activity chart and statechart.

Both charts are multi-leveled charts. The entire activity chart is controlled by the single statechart. The transitions on the statechart are event- and condition-driven. For example, transitions are taken when an activity is "done." For completeness, all of these transitioning events should appear on the activity chart as control flows. However, they were previously omitted for clarity of the chart. The history connector "H" was used in the baseline statechart to return to the SUD after the Configuration Control Board (CCB) review.

The revised charts comprise multiple activity charts and statecharts. In this study, each chart was not decomposed to its primitive level. Decluttering was applied only to the top level of the original baseline charts. The advantages and disadvantages of decluttering were clearly evident without the additional effort of completely separating all non-primitive activities into separate charts. Activity charts, included in Appendix E, were created for the overall system description and the initial analysis, software development, system integration testing, and product logistics sub-systems. As in the baseline charts, the control flows for the transitioning events were omitted in the activity charts. Appendix E includes the five associated controlling statecharts for these activities. These individual statecharts are far less complex than those of the baseline statechart.

Because this investigation examined all levels of decomposition of the SSH process, the initial analysis sub-system was chosen as the example for further study. Notice that the three non-primitive activities within the INITIAL_ANALYSIS activity (STATUS_ACCTG, IMPACT_ANALYSES, CONFIG_CTRL_DECIS) are not presented on separate pages. This relates to the amount of decluttering the developer deems appropriate for activity chart complexity. Appendix E also contains the controlling statecharts for these sub-activities. Table B-1 provides the location of each of the charts referenced within Appendix E. Time did not permit the development of a complete set of charts for this model. If the SUD had been decluttered to its primitive levels, an estimated 34 charts (activity charts and statecharts) would have been developed. Most of these charts would have been conceptually trivial. However, the scoping restrictions posed their own unique complexities when developing the statecharts for the lowest level activities.

Table B-1: Figure Numbers of the Revised Model Charts

Figure B-1 provides the hierarchy of the individual charts developed for the revised model. For example, the details of the parent activity chart, PMT_ACT_OVW, are provided in the statechart, SSA_STATE, and in the activity charts INITIAL_ANALYSIS, SW_DEVELOPMENT, SYS_INTEG_TESTG, and PROD_LOGISTICS. Since, the non-primitive sub-activities within INITIAL_ANALYSIS were not decluttered, they do not appear on this hierarchy chart. Their associated statecharts are represented by: STATUS_ACCTG_STATE, IMPACT_ANAL_STATE, and CCD_STATE. Table B-2 lists the controlling statecharts for each non-primitive activity examined in this analysis.

The revised charts are functionally equivalent to the baseline model, and do not incorporate any additional process information (i.e., unresolved issues identified by the baseline modeling effort and resulting from limitations in the handbook being modeled). The unresolved issues that are seen in the baseline charts reappear in the revised charts, since it was the intent of this report to redevelop the baseline charts "as is." Examples of the unresolved issues include lack of:

- Destinations for information flows or transitions.
- Determination of how the conditions are actually set in an activity.
- Details of an information feedback.

These unresolved issues are detailed in [11].

A comparison of the baseline and revised activity charts shows little difference in the contents of the activity description. Activities in the revised charts are extracted from the baseline chart and placed in separate charts. This decluttering reduces the visual complexity of the charts but does not alter them functionally. By contrast, the most drastic difference is in the development of the statecharts. Statecharts no longer have the ability to globally control all the activities within the SUD. A statechart may control only its siblings within an activity. For example, the statechart INIT_ANAL_STATE is the controlling statechart for the activity INITIAL_ANALYSIS. The only activities it may control are: STATUS_ACCTG, INIT_CONFIG_CTRL, IMPACT_ANALYSES, and CONFIG_CTRL_DECIS. It may not control the sub-activities of STATUS_ACCTG, IMPACT_ANALYSES, or CONFIG_CTRL_DECIS. Additional statecharts must be developed to control those sub-activities. Therefore, for this report, the STATUS_ACCTG_STATE, IMPACT_ANAL_STATE, and CCD_STATE statecharts were developed. However, an exception to this rule exists. For nonbasic activities that have no control activity, STATEMATE provides a special default behavior. This default behavior starts and stops all subactivities with the parent activity. An example of this is found in the IMPACT_ANALYSES activity. Located within IMPACT_ANALYSES are three sub-activities: MGMT_ANALYSIS, TECH_ANALYSIS, SUPPORT_ANALYSIS. If no control activity is defined within IMPACT_ANALYSES, the three sub-activities are assumed to be active as long as IMPACT_ANALYSES is active. The statechart, IMPACT_ANAL_STATE, represents this type of default control. Therefore, it was not necessary to develop this statechart. If an activity must exhibit behavior other than this default, this behavior must be modeled explicitly in a controlling statechart.

Figure B-1: Hierarchy of the Revised Model Charts

Table B-2: Controlling Statecharts for the Revised Model Activities

The statecharts of the revised model were developed using the alternative techniques to control the charts. Examples of these alternatives are shown in the statecharts that control the activities within INITIAL_ANALYSIS. Examining the states controlling this chart provides specific examples of each of the controlling mechanisms being investigated. As stated previously, these alternatives were an attempt to simplify the interactive animation of the process model.

In the revised model, several examples demonstrate the technique employed to step through a basic state.[6] Examples of the basic states controlling the activities within INITIAL_ANALYSIS are: IMPACT_ANALYSES_ACTIV in Figure E-4 (or IMPACT_ANAL_STATE in Figure E-6), INIT_ACCTG_ACTIV and FINAL_ACCTG_ACTIV in Figure E-5, and PROD_DELIV_PLAN_ACTIV in Figure E-7. Within the textual form of each of these states, the event of entering the state triggers the action to assign a value to a data-item. By associating the "write_data" action with the event of entering the state, the state itself sets up the transitioning event. The instant a value is assigned to the data-item, the "written" event transitions the statechart to the next state. In the baseline model, transitions occurred based on manually entering data indicating that an event had completed or was "done." This technique of "writing" the information flow allows the developer to create a hands-off sequence of events for animation. In addition, this technique can be expanded to traverse a sequence of basic states.

The following is an in-depth example, from Figures E-3 through E-5, of the technique employed to step through a basic state. Listed are the steps that occur from the initial activation of activity chart INITIAL_ANALYSIS to activity INIT_CONFIG_CTRL. These steps are:

- Activity chart INITIAL_ANALYSIS is activated.
- Control is transferred to statechart INIT_ANAL_STATE.
- The default is transitioned to the state STATUS_ACCTG_ACTIV.
- The activity STATUS_ACCTG is activated.
- Control is transferred to statechart STATUS_ACCTG_STATE.
- Control is transferred to state INIT_ACCTG_ACTIV since the default condition of ANALYSIS_PASSED is false.
- The activity INIT_STATUS_ACCTG is activated.
- Within the textual form of state INIT_ACCTG_ACTIV, a definition exists that upon "entering" the state, the write_data action writes the information flow ID_PROB_CHG_REPT.

Once ID_PROB_CHG_REPT is "written":

- The activity INIT_STATUS_ACCTG terminates.
- A transition from the state INIT_ACCTG_ACTIV occurs.

---

[6] A basic state, a state with a single input and a single output, controls a primative level activity. No decisions are to be made within the control of an activity.

- The statechart STATUS_ACCTG_STATE terminates.
- The activity STATUS_ACCTG terminates.
- The activity INIT_CONFIG_CTRL is explicitly activated (see following discussion).
- The statechart INIT_ANAL_STATE transitions from STATUS_ACCTG_ACTIV to the state ICC_ACTIV.

The key to this sequence of steps is the definition within the textual form of state INIT_ACCTG_ACTIV. The above example demonstrates the ability to traverse basic states without any external input for animation.

The INIT_CONFIG_CTRL activity within INITIAL_ANALYSIS provides an example of an activity that is controlled by the alternative techniques used to explicitly "start," "stop," "suspend," and "resume" activities in the revised model. This activity must be suspended when requesting additional information or a CCB review. The portion of statechart INIT_ANAL_STATE which controls activity INIT_CONFIG_CTRL was developed similar to FIGURE A-6 discussed in Section A with activity A representing INIT_CONFIG_CTRL and state STA_B representing state ICC_ACTIV.

During animation, activity INIT_CONFIG_CTRL is explicitly started by the event "exiting" within the textual form of STATUS_ACCTG_ACTIV before transitioning to state ICC_ACTIV. Once control is transitioned to ICC_ACTIV, three possible decisions are available: to request additional information, to request a CCB review or to implement the software change. Since ICC_ACTIV is a multi-decision state, the decision must be entered externally. The "suspend/resume" actions are used when requesting additional information or a CCB review. For example, when a request for information is made, ICC_ACTIV generates a WR!(INFO_REQUEST) action which creates a transition to the WAIT_FOR_INFO state via the WR(INFO_REQUEST) event. Within the textual form of WAIT_FOR_INFO:

- The "entering" event suspends the activity INIT_CONFIG_CTRL.
- The "entering" event generates a WR!(ADDL_INFO) action.
- The "exiting" event resumes the activity INIT_CONFIG_CTRL.

Therefore, when state WAIT_FOR_INFO is entered, INIT_CONFIG_CTRL is suspended, a WR!(ADDL_INFO) action is generated, the state is exited via the WR(ADDL_INFO) event, activity INIT_CONFIG_CTRL is resumed, and the statechart transitions back to state ICC_ACTIV.

Two points are apparent in this sequences of steps. First, the activity, INIT_CONFIG_CTRL, is never completely stopped. It is "frozen" while the request is being made. Second, once the decision to make a request is made, no external inputs are required to transition through the "loop."

Finally, once the decision to implement the software change is made, the statechart transitions to state IMPACT_ANALYSES_ACTIV, where the activity INIT_CONFIG_CTRL is explicitly stopped.

Appendix C: Analysis of Advantages and Disadvantages

This appendix provides an in-depth comparison of each of the techniques used in developing the baseline model versus the revised model. The comparisons consist of the following subsections:

- Advantages: where the development technique of the revised model resulted in a clear advantage over the development technique of the baseline model.
- Disadvantages: where the development technique of the revised model resulted in a clear disadvantage over the development technique of the baseline model.
- Comments: where there is no clear advantage or disadvantage; a set of consequences or personal preferences.

The discussion of these issues is based on their applicability to SPM. The conclusions drawn may or may not apply to other types of modeling.

C.1. Decluttered vs. Monolithic Charts

The activity charts of the revised model represent a decluttered version of the baseline monolithic activity chart. The box-is-chart operation within STATEMATE Version 3.0 enables the developer to create a multiple chart configuration that presents the individual levels of decomposition on separate charts. In this example, multiple activity charts were developed to describe the functional view of the SUD.

C.1.1. Advantages of Decluttering

The clear advantage of using STATEMATE Version 3.0 is the reduced visual complexity of the individual charts. (This is easily demonstrated by a brief examination and comparison of the charts in Appendices D and E.) The overall conceptual complexity of the functional view (activity charts) is identical in the revised and baseline models because they represent the same conceptual picture. However, the individual charts of the revised model are less visually complex and less cluttered than those generated in the baseline model.

The visual complexity of the revised charts is independent of the conceptual complexity of the system. As the system conceptual complexity increases, only the number of charts defining that system increases. The individual charts do not increase in their visual complexity. With the baseline charts, however, each perspective is presented on a single chart and the conceptual complexity of the SUD drives the visual complexity of the chart. Therefore, as the conceptual complexity of the system increases, so does the visual complexity of the monolithic charts.

The visual complexity issue becomes evident when the revised and baseline charts of the

SSH model are plotted. This visual complexity is extremely important when documenting and distributing the results of a SPM effort. As demonstrated in Appendix E, the revised model charts can be easily inspected and understood when printed on 8 1/2 by 11 inch paper. In contrast, the baseline charts of Appendix D are cluttered and visually complex.[7] In a situation where the conceptual complexity of the SUD is significantly increased, the techniques used to create the revised model would yield a larger set of charts similar in visual complexity to those presented in Appendix E, whereas the techniques used to develop the baseline model would become increasingly cluttered and eventually yield an unreadable set of charts. Expanding this discussion to include the 24 by 36 inch plotter paper only extends the problem further (i.e., the monolithic approach places a finite upper limit on the conceptual complexity of the SUD, beyond which the resulting models become difficult or impossible to read and understand). The decluttered approach poses no such limit.

In contrast to previous STATEMATE versions, the multiple chart configuration of Version 3.0 provides more flexibility when developing a process model. To provide configuration management, STATEMATE limits access to a specific chart to a single user at any given time. Since the baseline model represents the SUD by one chart per perspective, it is awkward for multiple users to develop a chart. A user must "release" a chart before another user may access it. The multiple charts configuration of the revised model enables multiple users to simultaneously work on the SUD. Thus the development of the charts can be spread out over multiple users. For example, because the activity chart is represented by multiple pages, each user can access his or her particular portion (or sub-activity) without interfering with the other users.

Process understanding and training of personnel are important aspects of modeling a software process. The multiple charts configuration used in the revised model enhances the ability of a model developer (or demonstrator) to focus on specific details of the system without being bogged down with the non-relevant information. For example, each chart could represent a task to be completed by a specific individual or group. Only information relevant to the task would be provided with the necessary sources and sinks of the external information. (A monolithic chart provides all of the information of the entire process.) On line "viewing" commands can reduce the scope of information presented; however, the resulting charts are visually inferior to the charts created by decluttering.

The decluttered approach of the revised model enhances the ability to manage a set of alternative representations for a variety of chart components, for example, three alternatives for INITIAL_ANALYSIS and four alternatives for SW_DEVELOPMENT. Each component is treated as a separate chart, and multiple versions of each chart can be managed. As long as the interfaces are consistent and only the internal details differ, alternative versions can be easily interchanged. This issue originates from the long term goal of process modeling to

[7] STATEMATE provides an option to plot (or view) a specific level of a chart by hiding the lower levels of detail. However, the resulting charts are plotted with flows (or transitions) to and from unknown destinations. This may create more confusion for users examining the charts.

perform "what-if" analyses. (Managing a set of alternatives becomes increasingly difficult with a monolithic chart as the number of alternatives increase.)

The multiple chart configuration created by decluttering enabled us to use a "divide and conquer" approach to testing the charts. Therefore, the advantage of testing the revised model was that each chart could be tested individually, as a combination of activity chart and its controlling statechart, or as an entire package (or model). Each page of an activity chart could be tested separately before combining all pages into one logical chart and then retesting. On the other hand, the baseline model could not be tested piecemeal. For example, the activity chart could not be properly tested until it had been completed.

C.1.2. Disadvantages of Decluttering

The SEI platform limitations have posed problems in running animated simulations. The X-window version of STATEMATE is currently running on our Sun 3/80 machines with the 4.0.3 operating system. The maximum amount of real memory is 16 MB and a maximum of 40 MB of swap. If approximately 6 or more windows are open in addition to the simulator, the platform runs out of memory, and the simulation aborts. This count of approximately 6 windows includes both activity charts and statecharts, meaning that only a fraction of the model can be open for simulation at any one time. No memory limitation problems have occurred during non-simulator operations.

Changes to the charts are more time consuming to make in cases where a change affects interface elements that appear on both a child and parent chart (e.g., labels on flows, adding/deleting flows, changing box names for flow source/targets) because the change must be made consistently on two charts.

It is not possible to automatically generate a plot showing the entire conceptual (logical) chart. This type of plot is beneficial when inspecting a desired view that spans more than a single chart. For example, suppose that the IMPACT_ANALYSIS activity in the revised model was decomposed into its own separate chart. The decluttering approach would not provide a detailed inspection of INITIAL_ANALYSIS in one chart. The information would be spread out over two charts whereas, all of INITIAL_ANALYSIS could be seen at once with a monolithic chart.

With the decluttered approach of the revised model, a manually produced overview chart may be required to help reviewers follow through the model. An example of this type of overview is shown in Figure B-1.

C.1.3. Comments on Decluttering

The largest consequence of decluttering the charts is the necessity of dealing with the resulting number of charts. The SSH process model charts could conceivably have grown from 2 charts (one activity chart and one statechart) in the baseline model to as many as 34 charts in the revised model. In addition to their creation, all 34 charts would have required maintenance and testing. Furthermore, during simulation (animation), STATEMATE provides

no guidance in identifying the appropriate charts required or the hierarchy of the charts for traversing the SUD. This burden is placed on the user.

Clear and firm interfaces must be established between portions of the model. This may be difficult when developing a process model because such processes may not possess clear interfaces between components. This type of situation is more likely to occur when modeling descriptive models rather than prescriptive models.

The degree to which an activity chart is decluttered is strictly a personal preference. STATEMATE does not require a certain amount of decluttering on charts. An individual may chose to represent the SUD by a single monolithic activity chart, a set of basic activity charts, or anything in between.

If presented poorly, a multiple charts configuration could hinder a discussion of the entire SUD because all the system information is no longer provided by a single source (or chart). Viewers of a monolithic activity chart view the "whole picture" or whole process from a single chart, whereas when viewing the entire system from multiple charts, the information is distributed and viewers may experience confusion due to "flip flopping" between charts. This type of situation may occur during animation of the revised SSH process model.[8] When animating the revised model, the user has to swap windows in and out to follow the animation. This may create a seemingly unorganized traversal of the system. The presentation will be beneficial only if the individual presenting it is organized. However, it must be noted that animating any system of extreme complexity may be confusing to a viewer.

When decluttering a chart, no automated assistance is available to duplicate required information across charts. Additional effort and time is required to ensure that the duplicated information is consistent across the decomposition.

Bi-directional browsing of charts is not adequately supported by STATEMATE Version 3.0. This issue is not a disadvantage to decluttering; however, the desirability of such a feature is made apparent by the use of decluttered charts. The bi-directional browsing feature should maintain a record of existing charts and a record of the charts' location within the hierarchy of the charts. A "logical dive" operation is currently provided in the graphic editor that will start up another graphic editor for the child chart of a box-is-chart box. This is conceptually convenient for top-down browsing. However, if the child chart is not loaded in the work area but does exist, the tool assumes that the user wished to create a new chart rather than access the chart not loaded. Furthermore, no convenient support is provided for bottom-up browsing, i.e., readily accessing the parent chart. If the name of the parent chart is unknown, the user must identify the chart name from the textual form of the child chart before invoking a new graphic editor from the main menu. If the desired parent chart is not loaded into the work area, the tool again assumes that the user wished to create a new chart rather than identify whether the chart existed but was not loaded.

---

[8] Animation is used for demonstrating the model. Animating a model can be compared to walking an individual through the model.

C.2. Scoping of Elements

The charts of the revised model were developed under a strict set of visibility rules that govern the scope of textual and graphical elements. STATEMATE Version 3.0 enforces these visibility rules whereas, within previous versions of STATEMATE, all elements were accessed globally. The baseline model takes advantage of this global nature of elements by employing monolithic charts.

C.2.1. Advantages of Scoping of Elements

The primary advantage to element scoping is the establishment of a well-defined set of visibility rules. These rules enhance the ability of the developer to create a well-structured,[9] modular[10] process model. Structure and modularity have been shown to be effective techniques for the specification and design of software systems. It is reasonable to assume that such techniques would be equally effective in developing process models. The development of the revised model appears to support this assumption.

Two aspects of the new STATEMATE scoping rules provide the additional capability to generate modularity within a model: visibility rules for textual elements and visibility rules for graphical elements. The visibility rules for textual elements enable the developer to govern the scope of a textual element. This scope may range from a globally defined textual element to a locally defined element for information hiding.[11] In this respect, the visibility rules for a textual element resemble those employed in modern programming languages supporting nesting and block structure.

The visibility rules for graphical elements are the second aspect of the new STATEMATE scoping rules. The scope of a graphical element is limited to sibling graphical elements. This restricted visibility is the mechanism by which the modularity is enhanced. The scope of the graphical element requires the control activities to be localized (i.e., the behavior of a non-primitive activity is defined by its own control activity).

Modularity within the model is initially established by the decomposition (abstract leveling) of the SUD. The addition of the visibility rules for a textual element provides the ability to hide information and define the interfaces between the modules of the model. The visibility rules for graphical elements complete the ability to develop stand-alone, modular units within the model by requiring the control activity to be defined at a local level. The result is a modular system consisting of well-defined modules with well-defined interfaces among them.

The result of modularity in the revised model is a set of manageable subsystems, where

---

[9]The use of structuring permits decomposition of a large system into smaller, more manageable units with well-defined relationships to the other units in the system [3].

[10]Modularity is the single attribute of software that allows a program to be intellectually manageable [17].

[11]Hiding implies that effective modularity can be achieved by defining a set of independent modules that communicate with one another only that information necessary [17].

each subsystem within the SSH model hides its internal details (function and control) and communicates only through well-defined interfaces. The modular model:

- Reduces the perceived complexity of the model by breaking it into manageable pieces.
- Facilitates change (critical to maintainability of the model).
- Enables subsystems to be developed in parallel [17].

By contrast, the baseline model did not exhibit modularity. All elements were global. There was no information hiding and control activities were not defined within a subsystem. The lack of visibility rules on graphical elements enabled the development of statecharts which were analogous to the use of "go to" commands in a programming language.

Scoping of elements permits the reuse of variable names within different areas of the model. Therefore, independent developers may name their local elements as they wish regardless of the possibility of an identical name elsewhere in the model. This advantage becomes increasingly more important as the system being modeled increases in complexity. In addition, each independent module developed may be tested individually by the developer. The freedom to distribute the modeling effort was not practical in developing a model with the techniques of the baseline model.

C.2.2. Disadvantages of Scoping of Elements

Introducing restrictions always entails trade-offs. The approach to modeling a system must be altered to accommodate these restrictions. A trade-off may be the increased effort to develop a portion of the model for the sake of creating modularity. An example of this type of a trade-off involves the development of statecharts under STATEMATE Version 3.0. With the visibility rules, multiple statecharts must be used to describe the behavior of the SUD. The disadvantage is not necessarily the effort in creating the additional statecharts but rather the loss of freedom in developing the behavioral view.

C.2.3. Comments on Scoping of Elements

The scoping restriction had a strong impact on the model development, especially on the scope of a statechart. The set of visibility rules that govern the scope of a graphical element defined the controlling environment of a statechart. Distributed statecharts are necessary since STATEMATE Version 3.0 strongly enforces element scoping. Monolithic statecharts no longer can be used to describe the behavior of the entire SUD because the scoping restrictions force the behavior of each non-primitive activity to be defined by its own controlling statechart.

C.3. Distributed vs. Monolithic Statecharts

Creation of the charts for the revised model was guided by a strong set of restrictions. Distributed statecharts were necessary to adequately represent the behavior of each non-primitive activity. The following is a list of advantages, disadvantages, and comments on distributed statecharts.

C.3.1. Advantages of Distributed Statecharts

Distribution of statecharts offers many of the same advantages as decluttering. With distributed statecharts, the behavioral view is distributed over several statecharts rather than a monolithic chart. By reducing the amount of control described in a statechart, the statechart is less visually complex, the behavior being described is easier to understand, easier to test, and less cluttered to plot and view on a computer screen. By distributing the behavior of the SUD over several statecharts, each statechart is conceptually less complex, development of the statecharts can be spread out over multiple users, the ability to manage a set of alternative representations is enhanced, and the ability of a model developer to focus in on specific details of the system (behavior) is simplified.

By distributing the statecharts, the structure and modularity of the model is increased. The enhanced structure minimizes the amount of inter-chart control flows; the enhanced modularity makes it easier to introduce localized changes since we are now dealing with small, self-contained diagrams that describe the behavior of small, self-contained activities. By distributing the statecharts, the emphasis is placed on separating the functional concerns of the model.

Distributed statecharts were supported during the development of the baseline and revised models. STATEMATE Version 2.5 incorporated a lax set of visibility rules, thus enabling the developer to employ a monolithic approach to modeling the baseline model. However, STATEMATE Version 3.0 strengthened the visibility restrictions. STATEMATE no longer supports the connection of a monolithic statechart to control multiple levels of activities. Therefore, the advantage of distributed statecharts is that it provides the only acceptable means of creating control within the current version of STATEMATE.

Distributed statecharts are more consistent with virtually all other structured analysis techniques [1, 2, 4, 6, 20, 19]. The approach of distributing the control information enables the modeling to be more generic (i.e., tool independent). Therefore, more emphasis is placed on the concepts and techniques rather than on the tool being employed to model the process.

With distributed statecharts, each module is described by the activity of the module and the statechart that describes its control. The module may be considered a portable package. The module may be exported to other STATEMATE modeling efforts as a black box, provided the interfaces are consistent. This type of reuse would not be available if the behavior of the module (or non-primitive activity) were buried within a monolithic statechart.

C.3.2. Disadvantages of Distributed Statecharts

Monolithic statecharts provided the developer the ability to create a transition that activated an activity within any arbitrary level of the activity chart. This freedom reduced the effort in creating control for a set of activities. Distributed statecharts for the revised model were more difficult to develop because the statecharts had to implicitly exchange information among themselves in order to recreate the behavior of the baseline model. Only through the use of conditions can this exchange be accomplished. However, this solution raises a flag during the basic testing of the model.

The dependency between the functional and behavioral perspectives created by distributed statecharts may be a hindrance during model development and modification. Small changes in the functional view may cause a large number of changes in the behavioral view. For example, the addition or removal of encapsulating activity boxes may yield substantial changes in the statecharts, even though no process behavior is modified by the presence or absence of the encapsulating activity box.

Often, it is impossible to obtain a unified view of the behavioral perspective on the process being modeled. With decluttering, there is a conceptually unified single chart, which has been decomposed into multiple levels (charts). However, with distributed statecharts, there is no unified conceptual view of the behavioral perspective. Therefore, the ability to present the entire behavior of the SUD is extremely difficult. With a monolithic statechart, the SUD could be discussed purely from the behavioral view. By distributing the statecharts, the behavioral view must be discussed through the functional view.

The use of distributed statecharts reduces the understandability of the behavioral perspective at the interfaces between the several distributed statecharts. With a monolithic statechart, the graphical construct of a state transition line is used to illustrate the sequencing of one step after another. However, a transition cannot cross the interface between distributed statecharts. Instead, one must use separate transitions on the separate charts.

C.3.3. Comments on Distributed Statecharts

A consequence of distributed statecharts is the necessity of dealing with the resulting number of charts. As discussed in Section C.1.3, each of these charts must be created, maintained, and tested. Distributed statecharts increase the extent of dependency between the functional and behavioral perspectives in software process modeling. This violates one of the basic premises upon which the previous SEI process modeling approach was designed: the modeling perspectives were to be made as separate, distinct, and orthogonal as feasible.

For example, following the previous SEI premise allows one to modify the functional organization (e.g., grouping by encapsulation) without changing the other modeling perspectives (charts), or to change the sequencing behavior of process tasks without modifying other modeling perspectives. However, in the revised modeling approach, functional concerns are separated from each other. This may be a far more natural distinction than would separating functional from behavioral.

The difficulty in recreating the behavior of the baseline model using distributed statecharts may lie in the fact that the baseline model was not structured with the concept of distributing control. The need to exchange information between statecharts could be minimized if the model were originally developed with distributed statecharts.

C.4. Controlling of Charts

STATEMATE provides multiple techniques to control the activation and deactivation of activities and the timing of flow of information between them in activity charts. Several previously unused techniques were incorporated into the statecharts of the revised model. These alternative methods of control were examined to determine their impact on modeling and the animation capability.

C.4.1. Advantages of the Alternative Methods of Control

The alternative methods of control applied in the revised model significantly improved the animation process, enabling the modeler to animate the process with minimal external input of information. This type of animation is not a batch driven, but rather an interactive walk through of the model with the modeler entering data as required. By tying a state transition to an information flow with the "write_data/written" action/event pair, and associating an action with the event of "entering" a state, sequential activities could be traversed without external input until decision points of the process were reached. (In a non-batch animation of the baseline model, each event was externally entered individually as the sequential activities were traversed. This hindered any demonstration of the process model.)

By tying a state transition to an information flow with the "write_data/written" action/event pair, and associating an action with the event of "entering" a state, fewer control flows cluttered the activity charts because these methods of control appear on the statecharts and statechart textual forms. Any other event, X, would have to appear as an event flow to the control activity on the activity chart.

By tying a state transition to an information flow with the "write_data/written" action/event pair, a visual link is established between an information flow in the activity chart and the associated transitioning event in the controlling statechart. This link simplifies the examination process of the charts. This advantage would be more apparent if the process model did not have the one-to-one mapping of states to activities that the SSH process model exhibits.

The "suspend/resume" actions better represent the process being modeled. This concept is demonstrated when an activity is temporarily exited (e.g., a request for additional information). Theoretically, the activity would suspend upon requesting the information and resume when the information is provided, rather than stop when the information is requested and start over again when the information is received. This is exactly the difference between the "suspend/resume" actions and the "history" connector of the baseline model. In addition, the "suspend/resume" technique visually highlights activities that are suspended during animation. The technique of using the history connector in the baseline model displays only whether the activity is active or not.

Explicitly starting and stopping an activity was an advantage only when the "suspend/resume" actions were to be employed. Otherwise, the throughout command employed in the baseline model was the better choice.

C.4.2. Disadvantages of the Alternative Methods of Control

A disadvantage of the alternative methods of control is that too much information may be hidden within the textual forms. The events causing the transitions are no longer obvious by examining the charts.

Another, more subtle impact of the control method used in the revised model versus that used in the baseline model is that the modeler must be more aware of the model execution and animation semantics. As previously discussed, in the baseline model, activities are triggered by the arrival of events (usually named with a "done" suffix). These events are supplied interactively by entering event names at the appropriate times. Because these events are not generated internally within the model, the modeler need not be concerned about the logic or timing of their generation. By comparison, activity triggers in the revised model are generated internally within the model itself. As a result, execution of the model is more stand-alone, and animations can be carried out with much less user interaction than necessary with the baseline model approach. One side-effect of this revised model approach is that the modeler may spend considerably more time creating modeling alternatives that make animation possible, smoother, or more interesting, whereas modeling decisions made using the baseline approach can be more arbitrary in nature. It is possible that the time spent on animation considerations can cause the modeler to lose focus on what may be a more important concern: describing the target process as clearly and succinctly as possible.

C.4.3. Comments on the Alternative Methods of Control

A tradeoff occurs between the amount of information hidden within a statechart textual form and the length of the label provided on a statechart. If the information is to be "up front," then that information must be included with a transition label on the statechart. In the revised model, the actions to "start," "stop," "suspend," "resume," and "write_data" are hidden within the statechart textual forms.

Appendix D: Baseline Charts of the SSH Model

This appendix presents the baseline charts of the SSH model. These baseline drawings are slightly restructured versions of the original SSH model drawings. The changes aid in the visual clarity of the drawings and are strictly cosmetic.

Figure D-1: Baseline Monolithic Activity Chart

Figure D-2: Baseline Monolithic Statechart

Appendix E: Revised Charts of the SSH Model

This appendix presents the revised charts of the SSH model. The layout of the revised activity drawings was created with clarity being of prime importance. Therefore, when examining the drawings, pay close attention to the names within the "external" boxes rather than to the direction in which a flow exits the activity. Figure B-1 provides the hierarchy of the charts of the revised model.

The revised charts comprise multiple activity charts and statecharts. Ideally, each level of decomposition would be represented on a separate page. However, in this study, each chart was not decomposed to its primitive level. Decluttering was applied only to the top level of the original baseline charts. The resulting activity charts were felt to be of a comfortable level of visual complexity. Time did not permit the development of a complete set of charts for this model. The redevelopment of the SSH model primarily focused on the initial analysis portion of the model. Therefore, a complete set of charts describing the activities and behavior were generated for the initial analysis subsystem, and portions of the remaining original SSH model's lowest levels of behavior were omitted.

If decluttering was applied to a chart (denoted by a "@" symbol in an activity box), the internal information of that box is presented in a separate chart. For example, the box name, INITIAL_ANALYSIS@PHASE_I in chart PMT_ACT_OVW, means that the internal information for box INITIAL_ANALYSIS is located in chart PHASE_I. In addition, the control of each non-primitive activity is represented in a statechart referred to by the control activity. Therefore, the control activity @SSA_STATE refers to statechart SSA_STATE for the control information of the chart PMT_ACT_OVW.

Section B.2 provides an example of walking through a portion of the revised model.

Figure E-1: Revised Model PMT_ACT_OVW Activity chart

Figure E-2: Revised Model SSA_STATE Statechart

Figure E-3: Revised Model INITIAL_ANALYSIS Activity Chart

Figure E-4: Revised Model INIT_ANAL_STATE Statechart

Figure E-5: Revised Model STATUS_ACCTG_STATE Statechart

Figure E-6: Revised Model IMPACT_ANAL_STATE Statechart

Figure E-7: Revised Model CCD_STATE Statechart

Figure E-8: Revised Model SW_DEVELOPMENT Activity Chart

Figure E-9: Revised Model SW_DEV_STATE Statechart

Figure E-10: Revised Model SYS_INTEG_TESTG Activity Chart

Figure E-11: Revised Model SYS_INTEG_STATE Statechart

Figure E-12: Revised Model PROD_LOGISTICS Activity Chart

Figure E-13: Revised Model PROD_LOG_STATE Statechart

Table of Contents

List of Figures

List of Tables