On Deriving Statecharts Supervision Models from SDL Specifications Using SSM

Antonio Mendes da Silva Filho
DIN/CTC/UEM
Zip: 87020-900
Maringá - PR, Brazil
amendes@din.uem.br

Abstract

This paper presents a discussion of using *software supervision* as a means of improving the software reliability of reactive systems during the in-use phase. Supervision software consists of monitoring both the inputs and outputs of a target system and checking them against the target system's specification. All discrepancies between observed sequences of signals and the target system's specification are reported as failures. The emphasis of this paper is on showing the suitability of using Statecharts as a formal technique to specify the reactive system supervisor. The target reactive system is assumed to be specified in SDL (Specification and Description Language). *Statecharts-based Supervisor Modeling* (SSM) is presented by using examples. As well, benefits of this approach are discussed.

KEY WORDS: reliability, specification, supervision models , Statecharts.

1.0 Introduction

Several approaches exist which improve the reliability of software during the in-use phase. They can be broadly classified as software-based or hardware based techniques. Two of the major software techniques are N-version programming and recovery blocks [1, 2, 11, 12]. Each of these relies on extra software to provide redundancy that attempts to inhibit errors before they manifest themselves as failures. Experience shows that the additional software required by these approaches is not only expensive and hard to maintain, but has not proven to be fully effective in eliminating failures.

A paradigm for the indirect improvement of software reliability called *software supervision* has been investigated [4, 6]. Software supervision consists of monitoring both the inputs and outputs of a target system and checking them against the target system's specification. All discrepancies between observed sequences of signals and the target system's specification are reported as failures. Software supervision is carried out for the improvement of software reliability during the in-use phase, since most research efforts on improving software reliability have focused on the first four stages of software development.

The emphasis of this paper is on presenting Statecharts-based Supervisor Modeling (SSM). SSM is a way to derive supervision models from SDL-specified reactive systems [20]. Examples of fragments of SDL-specified systems are given, along with their corresponding Statecharts supervision models, to support this presentation.

Given the preceeding, Section 2 discusses several issues regarding the software supervision paradigm, and Section 3 presents Statecharts supervision models for fragments of SDL specifications, as well as Statecharts-based Supervisor Modeling. In Section 4, a discussion of the main benefits of this approach is given. Finally, Section 5 presents the concluding remarks.

2.0 Background Issues

Since this paper addresses the use of software supervision as a means to improve software reliability of reactive systems, the issues underlying are provided.

2.1 Rationale

Developing a reliable software system is a major requirement today as it is mainly used for critical applications. Applications such as automatic flight control, banking, and telecommunications systems demand reliability and real-time features. In such an environment, the occurrence of a failure may result in damage to the company's reputation, and even distressing economic consequences [3].

This paper discusses software supervision as a technique for improving the software reliability of reactive systems. Specificaly, the category of telecommunication systems is considered. The software supervision approach is based on the *belief* method [15]. In such an approach, the inputs and outputs of a reactive system are monitored and compared to the system's specification. Currently, software supervision has been studied for the SDL-specified target system. SDL is a specification formalism that has been recommended by ITU (International Telecommuncation Union) for behavioral description and specification of telecommunication systems [20]. Since SDL allows the existence of nondeterminism in the target system specification, its supervisor needs to hold hypotheses or beliefs about the system's possible behaviors.

Experiences addressing the supervision paradigm has been worked out as reported in [4, 5, 18, 19]. This paper concentrates on modeling aspects for the supervisor of reactive systems. This proposal suggests the use of Statecharts as a specification technique for supervision models. Benefits of this approach are presented as well. Following, examples of techniques to improve software reliability are discussed.

2.2 Paradigms of Software Reliability Improvement

Several approaches have been proposed to improve the system's software reliability as well as to provide a fault-tolerance feature. Two of the majors techniques are N-version programming and recovery blocks [1, 2, 11, 12]. Although both approaches have been shown to achieve a reduction in software failures, they have not proved to be cost-effective [22, 23]. Another approach is software audits [9]. In this approach, software data errors are detected and possibly corrected by means of audit programs. Audit programs consist of additional software which has access to the main program's data structures. Typically, audit programs execute at a lower priority than the main program, and only periodically check data structures for errors. As well, this technique is only able to reset the software to a safe state rather than retract the error [8]. Another approach attempts to improve software reliability by using a Real Time Supervisor [4, 18]. The supervision-based approach includes the following features: behavioral monitoring, failure detection and reporting. Note this technique focuses on improving software reliability at the operational stage. The supervision approach is based on the theory of beliefs discussed below.

2.3 Theory of Beliefs

The theory of beliefs provides a method for creating a model to explicitly represent nondeterminisms permissible in the system specifications [15]. At any point in a specification where nondeterminism causes multiple valid paths to exist, the belief method creates a *belief* to represent each possible path. These beliefs evolve concurrently and are represented as threads. Each thread can be thought of as a process which is associated with a belief. When an output arrives which invalidate the belief represented by a thread, that thread is terminated. If all the threads in a belief set are terminated, a failure has occurred, since no belief remains to explain the behavior.

Where multiple processes are involved, threads in different processes, which represent the same belief, are connected by links. If any thread is terminated, all the remaining threads in the link will be terminated provided that they are not involved in any other links.

2.4 Software Supervision

Software supervision is a means of improving the software reliability of a target system. This approach underlies a *Real-Time Supervisor* whose major aim is to monitor the system inputs and outputs as well as to report failures upon their occurrence. Such a method utilizes a *black box* view of the system. Figure 1 illustrates the scenario involving both the target system and the RTS.

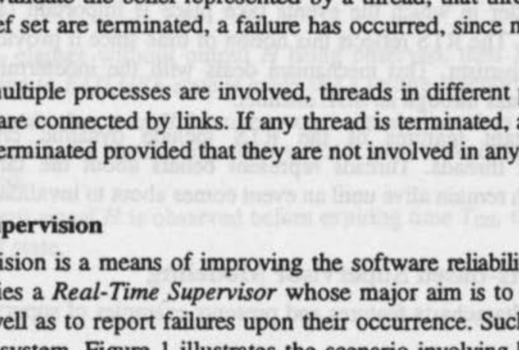


Figure 1: Scenario of the Supervisor.

As Figure 1 shows, the RTS monitors only the external signals of the target system. To do this, the RTS creates beliefs about the target system's behavior based on the signals observed at the target system boundary. After an input is sensed, the RTS checks for the output signal to verify if any non-legitimate behavior (or failure) has occurred. Note that the RTS is a passive component, since it only senses signals either from the input or output of the target system. Below are the RTS characteristics.

- The RTS does not know the internal states of the target system. Consequently, it creates beliefs which stand for the evolving behavior of the system being supervised. This occurs for every external signal monitored by the RTS.
- Assuming that the target system is specified in SDL, the RTS cannot simply be a replica of the target system and perform a comparison to the target system. The RTS needs to accommodate all possible behaviors of the target system due to the nondeterminisms permissible under SDL.
- Another issue observed is that most real-time systems are *event-driven*. Hence, the order in which the events take place is important (and not the event timings). The RTS reflects this notion of time since it provides a *signal in-transit* mechanism. This mechanism deals with the indeterminate delays when signals pass through an SDL channel.
- Other important features of the RTS include dynamic creation and termination of threads. Threads represent beliefs about the target system behavior which remain alive until an event comes about to invalidade them.

3.0 SSM: Statecharts-based Supervisor Modeling

This section highlights Statecharts features and presents examples of supervision models using SSM.

3.1 Statecharts

A natural technique for describing the dynamic behavior of a system is using a state diagram. The described system or function is always in one of a finite set of states. When an event occurs, the system reacts by performing some action, such as generating a signal, changing a variable value and/or making a transition to another state. Statecharts extend the classical state diagrams in several ways. They enhance their visual appearance. As well, they cater to hierarchical descriptions, multi-levels of concurrency and communication. Statecharts were introduced about a decade ago as a visual formalism for specifying the behavior of complex reactive systems [7, 13, 14]. Herein, Statecharts sematics is used as given in [16, 17].

3.2 Statecharts Supervision Models

The SDL-specified system may contain nondeterminisms, which makes the system specification simpler. Examples of nondeterminisms are given below.

- The first example of nondeterminism involves the SDL nondeterministic constructs *none* and *any*.
- The second example of nondeterminism refers to *indeterminate delay* experienced by the signals when they travel through SDL channels.

Statecharts supervision models are used to cope with these nondeterminisms permissible in SDL specifications. The reason for the use of these supervision models is to accommodate all possible legitimate behaviors. The supervision models for the former type of nondeterminism are given in [4, 5]. This paper concentrates specifically on the later.

To support this discussion, fragments of SDL specifications and its corresponding Statecharts models are presented. The following are some scenarios which the RTS may face. Initially, Figure 2a. shows a simple fragment of an SDL system specification *X*. Figure 2b presents the finite-state machine for SDL process *Z*. Furthermore, consider that system *X* must react to stimulus *G* with T_{GH} units of time.

The supervisor model is given in Figure 2c. Now, considering Figure 2c, note that for the input *G*, a timer T_{GH} is set up and output *H* is waited. Consequently, any of the following situations may occur:

1. if T_{GH} elapses without output *H* being observed, then T_{GH} is consumed, a transition to *FAILURE* state is made and a failure is reported;
2. if a signal other than *H* (an unexpected signal other than *H*), i.e., U_H, comes in, timer T_{GH} is reset, a transition to *FAILURE* state is made and a failure is reported;
3. if output signal *H* is observed before expiring time T_{GH}, then process *Z* goes into *S2* state.

It is worth observing that after a failure is reported, the supervisor can be led back to its recently visited state. For instance, if an unexpected signal (U_G) had been sensed while the supervisor was in *S1* state, a transition to a state of reporting failure would be made. After reporting failure, a transition leading the supervisor back to *S1* state would be made. In the experiments carried out, an event, named *RE_SYNC*, was introduced which is periodically generated, leading the supervisor to its initial state.

As well, since Statecharts conditions were used, it is mandatory to clear these conditions after they are consumed. In the example shown in Figure 2, only the triggering elements were explicitly shown. Note that a *FAILURE* state is added. Moreover, transitions from states which wait for observation of external signals are also added, either capturing unexpected signals or detecting violations of timing constraints.

Another interesting issue is the added *WAIT_H* state which aims to keep the supervisor in a temporary state while signals travel through channels. All these steps are given in [4, 6] through a mapping procedure which allows the designer to generate Statecharts supervision models from SDL-specified systems.

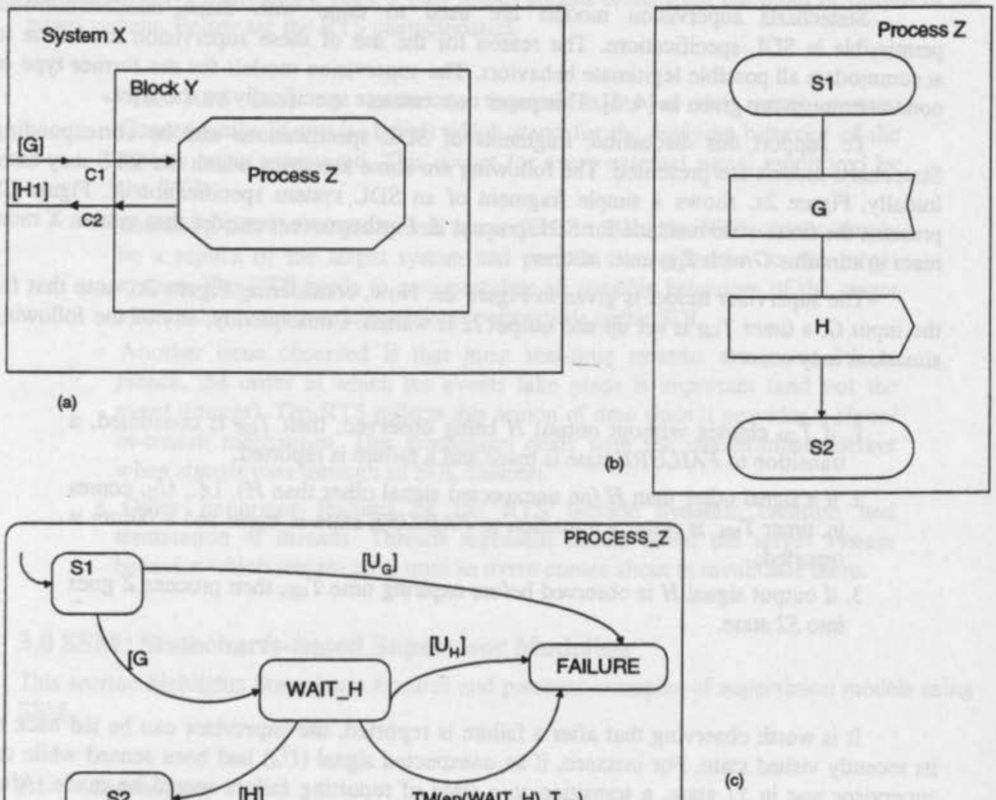
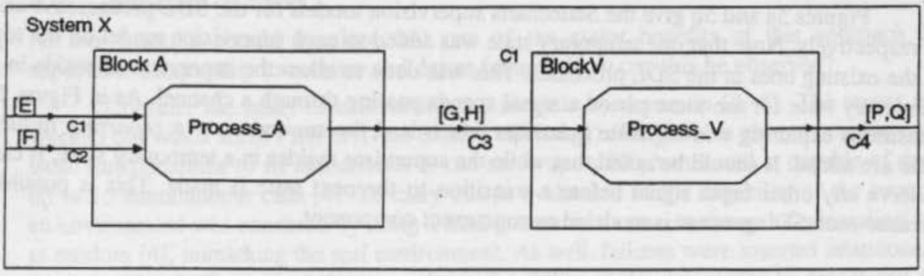


Figure 2: Supervision Model - Example 1.

The previous example was simple, and the nondeterministic channel delay was not dealt with. The following example looks at this issue specifically. The SDL channel delay is one of the major sources of nondeterminism. It motivates multiple, but legitimate behaviors likely to occur. How does it come about? Consider Figure 3a which shows an SDL system *S*, consisting of two blocks, *A* and *V*. The corresponding high-level Statecharts supervision model is presented in Figure 3b.



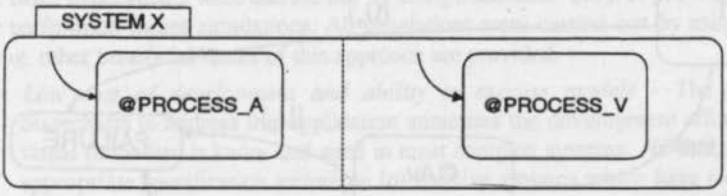


Figure 3: Supervision Model - Example 2.

The finite-state machines for SDL processes *A* and *V* are given in Figures 4a and 4b. An initial observation to be made here is that there are two channels which receive signals *E* and *F* from the environment. This gives rise to nondeterminism due to the indeterminate channel delays experienced by these signals. Note that signals *G* and *H* are internal to the system under analysis, and signals *P* and *Q* external.

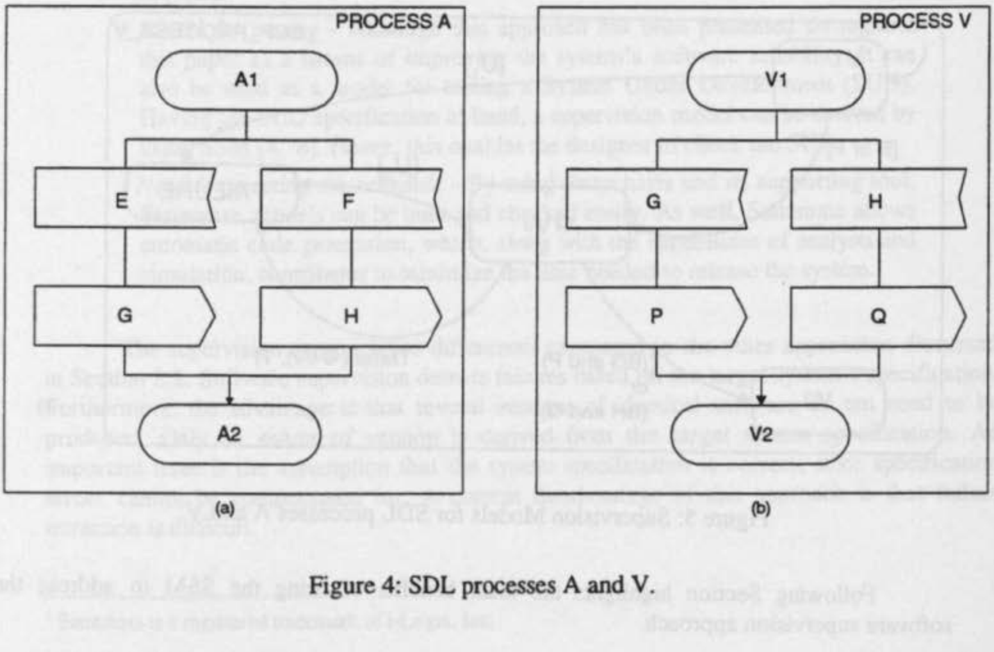
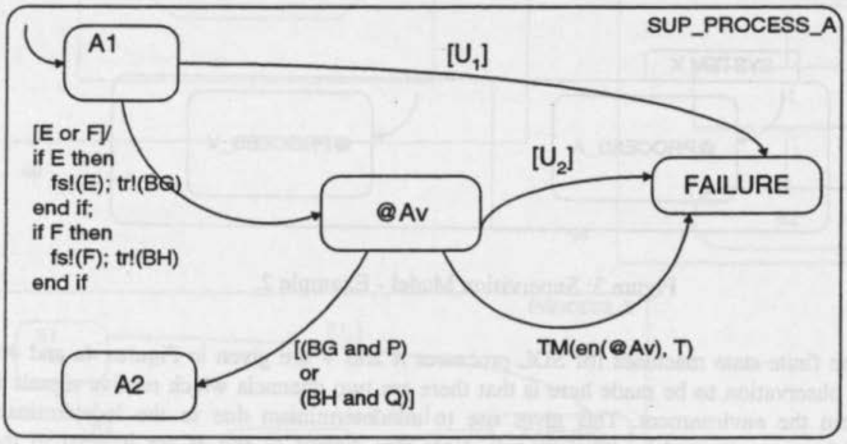


Figure 4: SDL processes A and V.

Figures 5a and 5b give the Statecharts supervision models for the SDL processes *A* and *V*, respectively. Note that one temporary state was added to each supervision model on the top of the existing ones in the SDL processes. This was done to allow the supervisor to reside in a temporary state for the same period a signal spends passing through a channel. As in Figure 2, transitions capturing non-legimate behaviors which lead the supervisor to a *reporting failure* state are added. It should be noted that while the supervisor resides in a temporary state, it can observe any other input signal before a transition to the next state is made. This is possible because each SDL process is modeled as concurrent component.



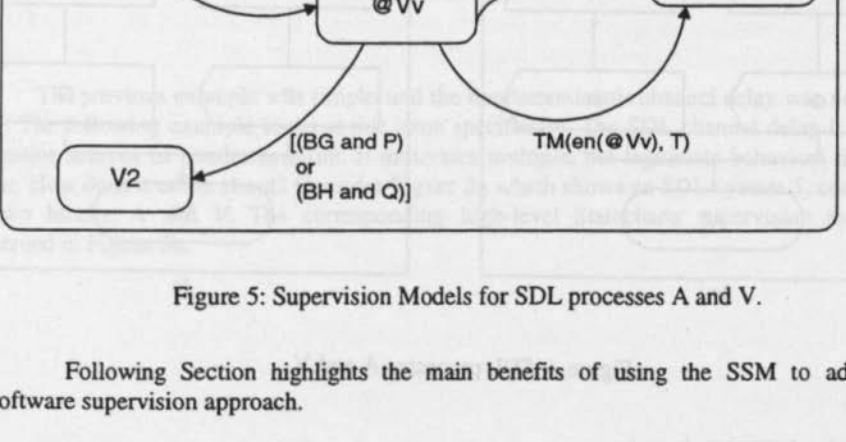


Figure 5: Supervision Models for SDL processes A and V.

Following Section highlights the main benefits of using the SSM to address the software supervision approach.

4.0 Benefits of SSM

At this point, it should be clear that one of the major benefits of this approach is the supervisor's capability to monitor and detect failures. How can this be observed?

Consider the study case involving the scenario as shown in [4, 5]. The target system worked out was a small PBX (Private Branch eXchange). The PBX is able to provide only the basic functionalities to its subscribers. It can serve up to 60 phones and is capable of carrying up to 15 simultaneous calls [4]. To carry out an evaluation of the capability of the supervisor, an environment was emulated by using a load generator. It was in charge of generating stimuli at random [4], mimicking the real environment. As well, failures were inserted intentionally, at nondeterministic times, in order to evaluate the ability of the supervisor to detect and report failures. Such experiments were carried out by using Statemate[1] [17]. It proved to be a helpful tool for performing theses simulations. All simulations were carried out by using workstation. Following, other beneficial issues of this approach are provided.

- *Low cost of development and ability to execute models* - The use of Statecharts to address this application minimizes the development effort. This visual formalism is know and used in most complex systems. As well, it is an appropriate specification technique for reactive systems which have real-time requirement [10, 21]. Moreover, since this work used the Statecharts semantics as supported in Statemate [16], all models could be analyzed and executed for validation purposes.
- *Software reliability improvement* - This approach is beneficial for indirectly improving the system's software reliability. As mentioned earlier, the supervisor works at the operational stage after a system has been released. Thus, its ability to detect and report failures earlier allows, e.g., a telecom company to correct faults and/or replace faulty devices before a distressing situation takes place. The use of a supervisor for a telecom system is reported in [4, 5].
- *Model for testing* - Although this approach has been presented throughout this paper as a means of improving the system's software reliability, it can also be used as a model for testing a System Under Development (SUD). Having the SUD specification at hand, a supervision model can be derived by using SSM [4, 6]. Hence, this enables the designer to check the SUD.
- *Need for meeting the schedule* - By using Statecharts and its supporting tool, Statemate, models can be built and checked easily. As well, Statemate allows automatic code generation, which, along with the capabilities of analysis and simulation, contributes to minimize the time needed to release the system.

The supervision approach has differences compared to the other approaches discussed in Section 2.2. Software supervision detects failures based on the target system's specification. Furthermore, the advantage is that several versions of identical software do not need to be produced. Only an *enhanced* version is derived from the target system specification. An important issue is the assumption that the system specification is correct, since specification errors cannot be compensated for. A current disadvantage of this approach is that failure retraction is difficult.

[1] Statemate is a registered trademark of i-Logix, Inc.

5.0 Concluding Remarks

This paper was focused on the supervision of SDL-specified reactive systems and on the use of Statecharts as a specification technique for supervision models. The software supervision paradigm and its capability to improve software reliability were discussed. Statecharts-based Supervisor Modeling (SSM) was illustraded by using examples. SSM allows the designer to derive supervision models from SDL-specified reactive systems. This paper was concentrated on the modeling aspects, and simple examples were presented. A practical example, using a PBX as a target (reactive) system was worked out and reported in [4]. Therein, practical issues regarding this approach are provided. Benefits of this approach are also discussed.

Currently, Software Supervision and SSM are being investigated to evaluate how supervisor *resynchronization* (after reporting failure) can better be addressed. Finally, a comparative study between this technique of improving the system's software reliability and other ones which provide redundant software is being carried out.

Acknowledgements

The author would like to thank the i-Logix, Inc. for their technical support.

References

[1] R. J. Abbott. *Resourceful Systems for Fault Tolerance, Reliability and Safety*. ACM Computing Surveys, pp. 35-68, March 1990.

[2] A. Avizenis. *The N-Version Approach to Fault-Tolerant Software*. IEEE Transactions on Software Engineering, pp. 1491-1501, December 1985.

[3] A. M. da Silva Filho. *On Using Logic Programming in Real-Time Systems*. Proc. of ILPS'94 Post-Conference Workshop on Design and Implementation of Parallel Logic Programming Systems, New York, USA, pp. 110-119, November, 1994.

[4] A. M. da Silva Filho. *Evaluation of Suitability of Statecharts to Real-Time Supervision for the Improvement of Telecom System Reliability*. Thesis, Dept. of Electrical and Computer Engineering, University of Waterloo, Canada, April 1995.

[5] A. M. da Silva Filho. *On Mapping SDL-specified Systems into Statecharts Supervision Models*. Technical Report (forthcoming).

[6] A. M. da Silva Filho. *Statecharts Supervision Models for Reactive Systems*, Proc. of CASCON - Computer Advanced Studies Conference'95, Toronto, Canada, November 1995.

[7] D. Harel et al. *On the Formal Semantics of Statecharts*. Proc. of the 2nd IEEE Symposium on Logic in Computer Science, pp. 54-64, 1987.

[8] J. R. Connet et al. *Software Defenses in Real-Time Control Systems*. Fault-Tolerant Computing, pp. 94-99, June 1972.

[9] M. N. Meyers et al. *ESS: Maintenance Software*. The Bell System Technical Journal, pp. 1139-1167, September 1977.

[10] S. G. Cohen et al. *Applications of Feature-Oriented Domain Analysis to the Army Movement Control*. CMU/SEI-91-TR-28, Software Engineering Institute, December 1992.

[11] J. J. Horning et al. *A Program Structure for Error Detection and Recovery*. Lecture Notes in Computer Science. pp. 171-187, Berlin, Springer-Verlag, 1974.

[12] B. Randell. *System Structure for Software Fault Tolerance*. IEEE Transactions on Software Engineering. SE-1(2), pp. 220-232, 1975.

[13] D. Harel. *Statecharts: A Visual Approach to Complex Systems*. CS84-05, Dept. of Applied Mathematics, The Weizmann Institute of Science, 1984.

[14] D. Harel. *Statecharts: A Visual Approach to Complex Systems*. Sci. Comput. Program. 8, pp. 231-274, June 1987.

[15] D. B. Hay. *A Belief Method for Detecting Operational Failures in Soft Real-Time Systems*. Thesis, Dept. of Electrical and Computer Engineering, University of Waterloo, Canada, 1991.

[16] i-Logix, Inc. *The Semantics of Statecharts*. Technical Report, 1991.

[17] i-Logix, Inc. *Statemate User and Reference Manuals - Vols. I and II*. Statemate Documentation, 1993.

[18] J. Li and R. E. Seviora. *A Real-Time Supervisor with Reduced Space and Time Requirements*. Proc. of 1993 IEE System Engineering for Real-Time Applications, pp. 90-95, 1993.

[19] R. E. Seviora. *Models for Real-Time Supervision*. Proc. of 5th Euromicro Workshop on Real-Time Systems, 1993.

[20] International Telegraph and Telephone Consultative Committee ITU (International Telecommunication Union). *SDL Formal Definition, Dynamic Semantics*. Annex F.3 to Recommendation Z.100 (Blue Book), 1988.

[21] D. P. Wood and W. G. Wood. *Comparative Evaluations of Four Specification Methods for Real-Time Systems*. CMU/SEI-89-TR-36, Software Engineering Institute, December 1989.

[22] J. C. Knight and N. G. Leveson. *An Experiment Evaluation of the Assumption of Independences in Multiversion Programming*. IEEE Transactions on Software Engineering. pp. 96-109, January, 1986.

[23] P. G. Bishop et al. *PODS - A Project on Diverse Software*. IEEE Transactions on Software Engineering. pp. 929-940, September 1986.