Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

Search and rescue: A case study of design flexibility

Kenneth N. McKay
WATMIMS Research Group
Department of Management Sciences
University of Waterloo
Waterloo, Ontario N2L 3G1

Jan Laube
Bureau of Management Consulting
Supply and Services Canada
Ottawa, Ontario
K1A 0S5

ABSTRACT

This paper is a case study description of the major techniques used in the design of a Search and Rescue (SAR) model, how the methods contribute to flexibility, and how these software engineering principles relate to a formal methodology (Zeigler 1987) that has been proposed specifically for simulation development. The techniques described in this paper can be used with any of the common simulation languages (e.g., SIMAN, GPSSH, SLAM II, and SIMSCRIPT II.5).

1. INTRODUCTION

This paper deals with the practical design of flexibility in simulations. Designed-in flexibility contributes to two aspects of model usage: first, increased latitude in selecting different scenarios for experimentation without changing the simulation code; and second, a decrease in the magnitude of the code change when a concept or function does change.

Flexibility is very important in creating good simulation programs:

- During the analysis phase, what is stated to be constant is not and what is claimed to vary does not; one cannot blindly believe what one is told.
- Model validation normally occupies a large portion of the development cycle and significant savings can result in reducing the overhead -- cost and time.
- The model will be used for many additional applications beyond that initially envisaged -- both data and algorithmic variations.
- A percentage of the initial model will undoubtedly be wrong and will have to be re-developed.

There have been a number of general discussions about software engineering as applied to simulation (e.g., McKay, Buzacott, Moore, and Strang 1986; Golden 1985; Sheppard 1983; and Ryan 1979). These and other similar papers provide overviews of the general principles, but do not present sufficiently rich examples that illustrate the power of the techniques. Although software engineering and simulation has been written about for at least a decade, it still appears to be the perception of many simulation programmers that software engineering principles are not suited to the real-world problems they face. It is one of the goals of this paper to illustrate that software design theories can in fact be used, and that semi-formal and formal methodologies can work together in practical situations.

The formal and semi-formal design concepts used in the SAR model are:

- experimental frames
- module and data coupling
- finite state automata

To assist in understanding the impact of the design concepts, the following sections provide a brief introduction to the SAR domain, model requirements, and overall development approach. Following the general background discussion, the three design concepts will be described.

2. SEARCH AND RESCUE

The study performed for the National Search and Rescue Secretariat (NSS) of Canada, addresses operational planning issues regarding SAR operations across Canada. The domain of the model is the response to land or sea incidents which involve a probable threat to human life. In essence, SAR is a situation that involves incidents that result in rescue missions, and the corresponding assignment of the appropriate interacting resources at certain times. There are many stages and events in a rescue mission, some of which are:

- notifying the appropriate authorities of the problem
- confirming the situation
- locating the incident
- determining what help is required
- performing the on-scene assistance

Many confounding if-then-and-or-buts constraints can affect what happens during a rescue mission. Wind speed, icing conditions, terrain, visibility, sea state, time of the year, day of the week, time of the day, type of resource, etc. can all interact and affect the outcome. Requirements may change and resources may be pre-empted as a mission takes place.

3. MODEL REQUIREMENTS

Why simulate SAR? SAR is a large and complex activity that involves significant funds and resources to maintain. Long term planning is necessary for determining location, quantity, and characteristics of SAR resources (rotary and fixed wing aircraft, and rescue vessels) for reacting to incidents when they occur. Analysis is also needed for determining policies and standards for public aircraft or ships so that incidents can be reduced at their source.

The dynamic relationships in the model are complex and simulation can be used to gain insight into the

factors that can influence SAR operations. It is not the only analysis tool that will be used; the model will be used in conjunction with other statistical methods.

As an analysis tool, the model is intended to provide information for scenarios which would reflect different resource placement, resource capabilities, and incident characteristics. The model picks up the SAR incident after it has happened and the regional authority has been notified.

4. DEVELOPMENT LIFE CYCLE

The model has been developed as part of a major analysis of SAR operations in Canada. The problem characteristics were studied and model requirements established. The information available for analysis was analysed for strengths (e.g., completeness, accuracy) and weaknesses (e.g., incomplete, invalid, and inaccurate data). Once the preliminary requirements were established, and it was determined that simulation was a suitable tool and that sufficient data existed to feed the model, work began on data enhancements and model architecture.

The project life cycle has involved:

1. documentation of the requirements, assumptions, functionality from a software systems viewpoint
2. design of the model architecture using formal and semi-formal methods that satisfies the stated requirements
3. translation of the model architecture to a detailed design level (pseudo-code)
4. selection of the appropriate tools for implementation (database and simulation language)
5. design and implementation of a user interface for model management
6. implementation of prototype and completed versions of the model

The above activities provided major milestones, task assignments, and concrete deliverables that assisted with project management. The activities were structured to illustrate early feasibility of the design; prototyping was crucial and was consciously considered and planned for throughout the life cycle. In general, prototyping should be considered for any non-trivial undertaking as it will bring to light technical difficulties with the model development environment, and problems of data integrity, collection, and analysis.

The major tasks remaining are verification (fall of 1988) and validation (mid 1989).

5. MODEL OVERVIEW

This section briefly describes the simulation structure. While many details have been necessarily omitted, it is hoped that the high-lights indicated will assist in understanding how certain software engineering concepts have contributed to the model's flexibility.

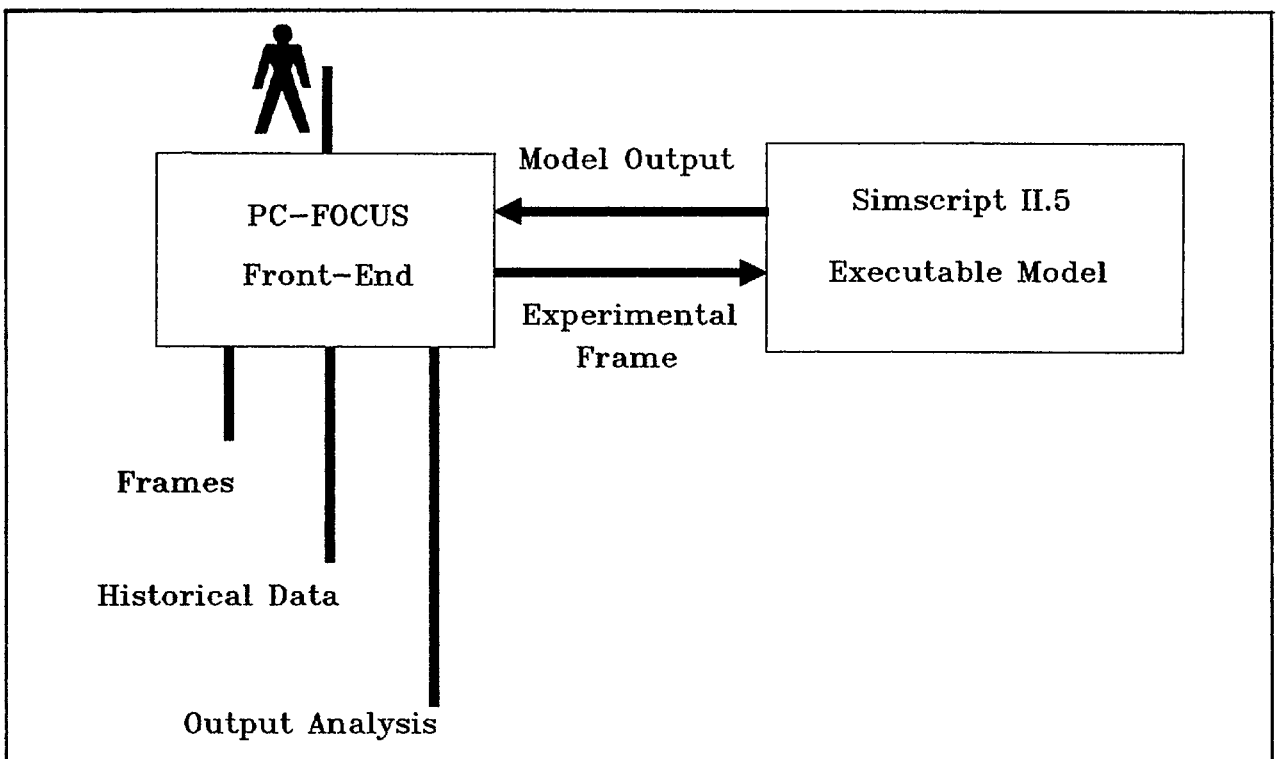

Figure 1 – System Architecture

The planning model (Figure 1) is a seamless integration of a database front-end implemented in PC-FOCUS and a SIMSCRIPT II.5 simulation model. The target computer system is a personal computer (80386) configuration running MS-DOS. The user front-end is responsible for database maintenance, scenario configuration, and scenario analysis. Data files describing the scenario, resources, applicable incidents, operating rules, etc. are passed between PC-FOCUS and SIMSCRIPT II.5. The front-end triggers the simulation model and then control returns to the front-end. The front-end provides tools to assist with the problems of data management and keeping track of what was simulated in different runs.

PC-FOCUS was selected due to its micro/mainframe interface and extensive facilities for the manipulation of data. SIMSCRIPT II.5 was chosen primarily because of its ability to handle the model design which contains extensive use of logic and advanced data structure concepts.

The executable simulation model (Figure 2) is divided into four major sections:

1. the centralized logic that corresponds to the functions of the Rescue Co-ordination Centre (RCC) which provides the supervisory control for SAR operations within a region (there are four across Canada)

2. the remote or distributed logic that corresponds to the functions of the resources while they are out on missions -- transitting, refueling, searching, assisting, towing, etc.

3. de-centralized logic that simulates the functions of on-the-scene co-ordination for searches and assists

4. an experimental frame structure that describes the current modelling domain: type and location of resources, resource characteristics, how to handle certain types of incidents, how to task the resources, and so on

In the centralized logic, the RCC is notified of the incident and decides what should be done, the order of doing it, what resources should be initially tasked, which ones will be mobilized later, and which resources are initially needed for the on-scene assistance. The individual resources are tasked (possibly pre-empted from current activities), and perform the actions necessary to locate and process the incident.

The distributed logic is implemented as a finite state machine with very clear transitions along a timeline. As resources proceed through the mission, they communicate to the RCC logic through messages advising the RCC about what is happening and possibly waiting for instructions about what to do next. This has been designed to closely match the flow of control and information that exists during live missions. The resources also communicate with the de-centralized logic (Search/Assist Master) for controlling searches and on-scene assistance.

The de-centralized logic decides when the search is over, and co-ordinates the on-scene resources rendering assistance. The de-centralized logic is also responsible for requesting additional resources to be tasked by the RCC when the on-scene situation is discovered to be different from the initial problem description.

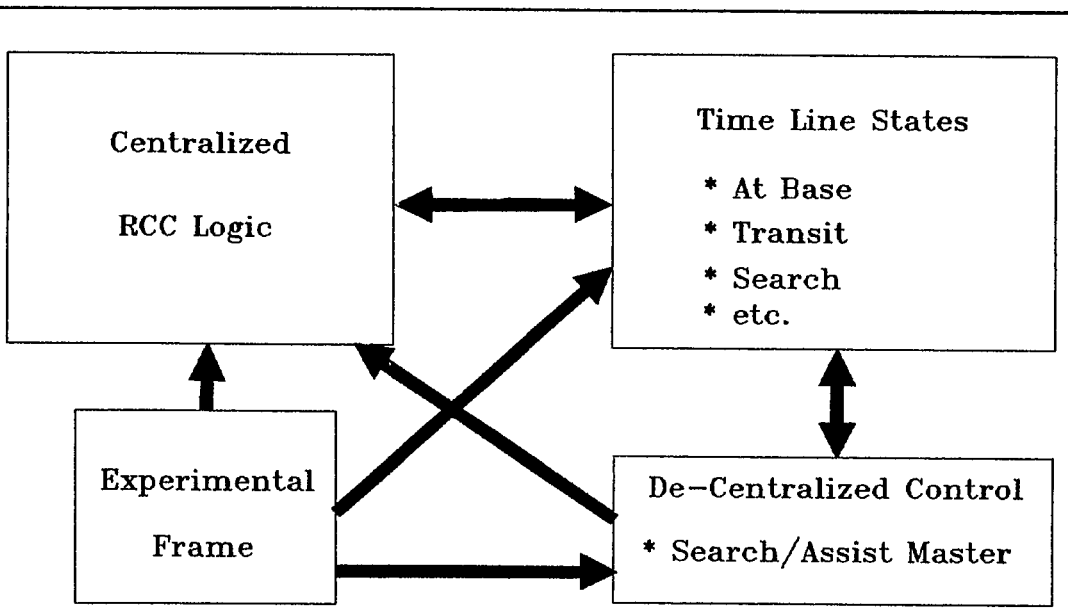

Figure 2 - Executable Model Architecture

6. SOFTWARE ENGINEERING CONCEPTS

The remainder of this paper focusses on several practical software engineering concepts that can be used in day to day simulation construction. General discussions on software engineering (e.g., Yau and Tsai 1986, Bergland 1981, Parnas and Clements 1986, and Goldberg 1986), or specific texts (e.g., Pressman 1987; Zelkowitz, Shaw and Gannon 1979; and Yourdon and Constantine 1979) should be referred to for comprehensive descriptions of different techniques and concepts. A programmer requires a large toolbox in which the right tool can be found: there is not one tool or concept that solves every problem.

The following sections explain some of the software engineering concepts used in the SAR model and how they contribute to flexibility. In most cases, if a designer or developer is not familiar with a tool or concept, they should experiment with it and not expect immediate results as Larkin, McDermott, Simon and Simon (1980) point out in a discussion on expertise in physics: individuals have to solve a problem first to discover how the problem should have been solved properly.

It should be pointed out that the concepts are interrelated and together form a philosophy of design for the SAR model. Different models would likely emphasize different concepts -- dogmatic application of any tool or set of tools should be avoided.

6.1 Experimental Frames

Programs can be described as being algorithms and data; and simulation programs are no different. There are two ways to exploit this statement in the context of program flexibility. First, there can be flexibility in the data that is being used. Second, flexibility can exist in selecting what logic to execute.

A common software engineering principle is to use constants, or variables and not hardwire data in the executable code. The grouping and isolation of input, output, and control variables is commonly called an experimental frame in simulation literature and is thoroughly described in Zeigler (1976). As Zeigler (1987) points out, the experimental frame concept shares a resemblance with the frame structure in Artificial Intelligence and objects in object-oriented programming. The frame can identify properties, values, ranges, and object specific routines that should be invoked while using the object. All of the relevant information required to perform a simulation run or a series of runs exists within the frame. Some simulation languages provide this separation naturally (i.e., SIMAN with experimental frames). Other languages such as SIMSCRIPT II.5, GPSSH, and SLAM II require the simulation programmer to create their own frame structures.

Working with the experimental frame concept can create data management problems. Typically, there is a collection of data (possibly in a database) from which the various experimental frame components are selected. The multiple experimental frames must also be matched to the output for analysis and audit purposes. Some simulation environments such as TESS (Standridge and Pritsker 1987) provide a data management system which can be manipulated to synthesize the frame concept. Others, such as SIMAN (Pegden 1986) provide the experimental frame directly, but leave the data management problem in the hands of the programmer.

The approach chosen for the SAR model was to use a PC-based database system and design a system that integrated the database describing the domain, provided a selection mechanism with which to create experimental frames, and also provided the data management tools for manipulating simulation scenarios. The PC database system also provided the file hand-off mechanism to the executable model.

In the SAR SIMSCRIPT II.5 model, all of the static data structures that define what resources are available, what they can do, etc. are loaded from separate files on disk; the logical grouping of files represents the experimental frame. The SIMSCRIPT model processes the files in a pre-specified order and dynamically creates all of the necessary structures and variables for executing the model.

The front-end system provides the necessary model management functions that exploit the concept of experimental frames. Without the integrated system, model verification, validation and subsequent use would be difficult to perform.

The other part of software is of course the algorithms. The algorithms form the structure of the system and control what happens in what order in a simulation. Changing this level in the simulation usually requires a great deal of work before the simulation works again. Two things have been done in the SAR model to enhance the experimental frame and subsequently minimize the effect of changing the algorithms.

First, whenever there are multiple choices, there are flags and codes in the experimental frame that indicate what option is active. In the executable code, a CASE structure is typically used to invoke the appropriate logic based on the current control blocks being used. For example, this allows local overrides on a mission by mission basis, as well as, global option selection for all missions. Clues for deciding what should be implemented in this fashion came from the requirements analysis. Whenever maybe, sometimes, or yes but is expressed often enough, a flexible structure was considered. Instead of making a decision to do one or another, it is better to support both concepts (within reason) and allow the validation phase the power to eliminate meaningless options. Discrimination is needed and it is through experience that developers will learn what to include or not.

If the problem is complex enough that simulation is warranted, then it is probably complex enough that many decisions cannot be decided upon in the early stages of model development: it is far cheaper and easier to build in the options from the beginning than later. The use of built-in options should be a statement of policy and once adopted, it is easy to implement.

The second concept added to the experimental frame specifically for flexibility in algorithms was a special table. During a mission there are many events that occur which can be sequence dependent and interrelated. For example, dropping survival supplies should be done before the people are evacuated, or the service should be forgotten. Or, a fixed wing aircraft may remain in the area after it has done its own duty to support a helicopter until it has also finished. There are many combinations and possible operation guidelines that can affect what a resource does. The requirements analysis phase hi-lighted this area as one that many what-if questions will be directed

toward. Thus, the ACTIVITY CONTROL table was created that describes the time line, stages along the time line, relationship of activities with the time line and with each other. The model has been designed with a general purpose time line manager (co-ordinator) that processes the ACTIVITY CONTROL table to obtain information about what resources do next, how resources are to react to each other along the time line, etc. The model user can alter this table which is analogous to an expert system rule-base and thus alter a major algorithmic component in the model from the experimental frame.

Placing the ACTIVITY CONTROL logic in the experimental frame is expected to eliminate many what-if changes to the executable code. Simulation code changes should be avoided whenever possible since they require programming staff plus the model user and could possibly jeopardize the verification and validation status of the model.

6.2 Module and Data Coupling

A common problem with any kind of software design is the level of coupling within the software. By coupling we mean both logical and physical relationships:

- how much does one subroutine know about the syntax of the data compared to the semantics
- what one subroutine assumes about the subroutines that call it and about the subroutines it calls in turn
- what is the nature of the interface between two routines -- implicit and explicit data sharing

Understanding the coupling present in a simulation is very important and it can assist in establishing appropriate test-beds -- where and what do they have to emulate (Zeigler 1987). Coupling is a significant issue when software has to be debugged, verified, and changed on a continual basis.

While there are many aspects of coupling that can be discussed, two stand out in affecting long-term flexibility of the model: data awareness and internal assumptions.

One set of routines should be aware of the physical representation of a specific data structure and these routines provide the interface point for all access -- inquiry and manipulation. This prevents dozens of locations in the code from being sensitive to the actual data. For example, it is very common in the SAR model to use distances as part of decision making processes. Distance calculations can use absolute or relative origins, and can be specified using cartesian or latitude/longitude co-ordinates. Should every module in the software be aware of these issues? If they were, it would be very hard to experiment with and determine the strength and weaknesses of the different approaches, or it would be hard to correct a bug in the distance algorithm -- in how many places does it occur? Therefore, there is only one place in the simulation that knows what the data structure fields mean and what to do with them when it comes to distance calculations. Anything to do with distance must come from this module.

When subroutines and modules are designed, there should not be assumptions made between callers and called subprograms. It should be assumed that if the interface is satisfied (e.g., provision of data, instructions, and results), the subroutine will perform a specific task. It should not make use of any other information about who is using the interface and infer values of data or control options to execute. The modules should be decoupled and if every routine is written like a subroutine library member, then maintenance will experience a lower risk of failure. If routines are too smart and make inferences, the task of changing the code is significantly greater: all software must be checked to see what is being assumed where.

The SAR model has been designed with both types of coupling in mind. The overall structure is relatively flat and resembles a subroutine library. The modules do not make assumptions about callers and callees -- they work strictly with the data in the passed control blocks. Information that is used in decision making is hid from global view and library utilities are provided that permit the syntax to change without changing the semantics.

6.3 Finite State Automata

The experimental frames and coupling approaches used in the SAR model are not specified in a formal specification language such as DEVS (Zeigler 1987). The experimental frames are described as data structures: fields, contents, purposes. The coupling is a direct artifact of the finite state automata and is documented as inputs and outputs to the modules in the design's pseudo-code. As such, the frames and coupling can be considered semi-formal. However, the actual design is described as a formal deterministic finite automata state structure:

- an initial state
- set of possible states
- set of external stimuli to a state
- set of internal stimuli generated within a state
- set of possible states following a state
- set of transforms taking a given state, a stimulus, and mapping this to the next state

The finite state structure has been used for many years in designing complex software systems (e.g., communication protocols, operating systems, etc.) and a concise readable explanation can be found in Zelkowitz et al. (1979). As a formal definition, the finite state automata provides a mechanism for documenting and studying the system. The method hi-lights what logic should be in a state, and how states are coupled.

Figure 3 illustrates how the automata concept can describe a resource transitting to a search site taking fuel consumption into account (note: this is a small and incomplete subset of the SAR states and stimuli required for the transit phase of a mission). When in transit, a ship or plane may run short of fuel and require refueling. The TRANSIT state is limited in knowledge and simply moves a resource from point A to point B. The required fuel is initially determined and if the resource cannot make the trip without refueling, a phased trip is planned. The resource will travel from fuel depot to fuel depot until the search area is reached. Arriving at a fuel depot triggers the internal stimulus that causes the REFUEL state to be activated. After refueling, the trip continues. When the resource reaches its final destination, the search phase may commence.

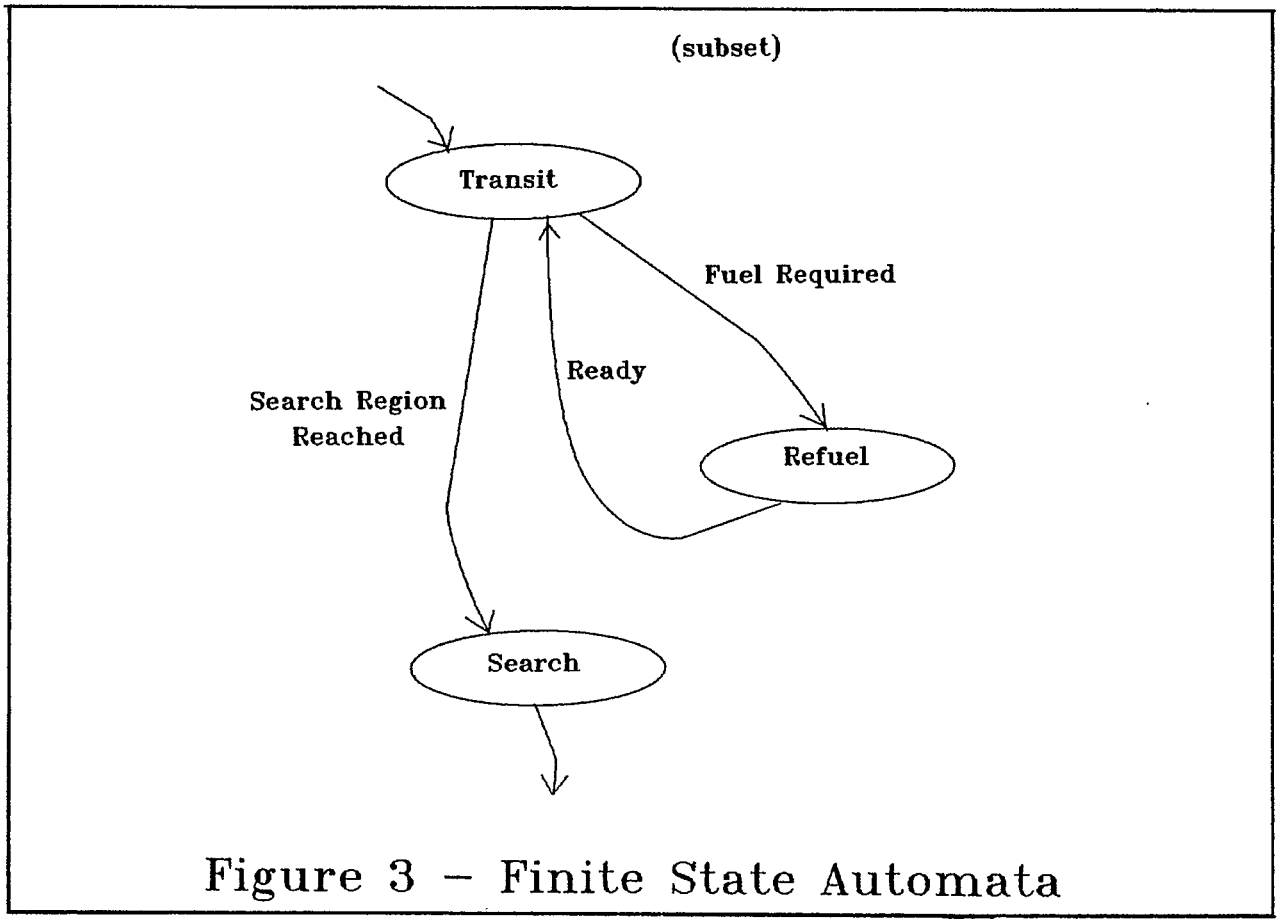

Figure 3 – Finite State Automata

In the SAR model, there are states, stimuli, and transforms. The automata model has been used to validate the operational requirements of the model during design reviews and has provided the coupling for the major model components. Being finite and rigorous permitted the thorough analysis of the problem before pseudo-code and detailed design started. The detailed design was then created using the finite state automata as the structure. An informal pseudo-code was then used to describe the detailed design which allowed the review of the design by experts in the SAR domain.

The general structure of a state in the SAR model is shown in figure 4. There are three sources of stimuli to the state. First, prior states can send a resource to the state as a stimulus. Second, the co-ordinating software can send an external stimulus (e.g., pre-emption, or search has finished). Third, there can be internal stimulus created within the state.

Within a state, the logic is normally structured to execute general housekeeping tasks first (independent of stimulus, etc.), followed by establishing any state-wide timers (such as sunset), and then special processing for each stimulus coming into the state. The type of resource/entity will cause a number of internal stimuli to be investigated. If the internal stimulus is time based, the minimum elapsed time will be used as the over-riding stimulus. That is, the earliest event out of a number of possibilities will be used: running out of fuel, reaching destination, etc. The state is configured with a single WAIT point that will be triggered by the state-wide, external, or internal stimulus. Following the wait point is general housekeeping logic again and then depending on the trigger, the appropriate next state is activated. The logic within the state makes no assumptions regarding the prior states, or the states following.

The use of finite state design can be applied to any of the common simulation languages assuming that the implementers have adequate programming experience and training. That is, it is doubtful if a scientific programmer enlisted to create a simulation would have the required skills to design and program the finite state structures from scratch. Special training can occur, or special environments can be used that support the concepts of state analysis, experimental frames, and controlled coupling. One environment that has these concepts is the DEVS-Scheme (Zeigler 1987). DEVS is a model specification scheme that formalizes the complete process and integrates the specification with a model development environment.

To summarize, the reasons for using finite state automata are many:

- first and foremost, it is a design tool with which the designer can organize the dynamics of a situation and play mental what-if games: is anything missing, or what else can happen that was not forseen

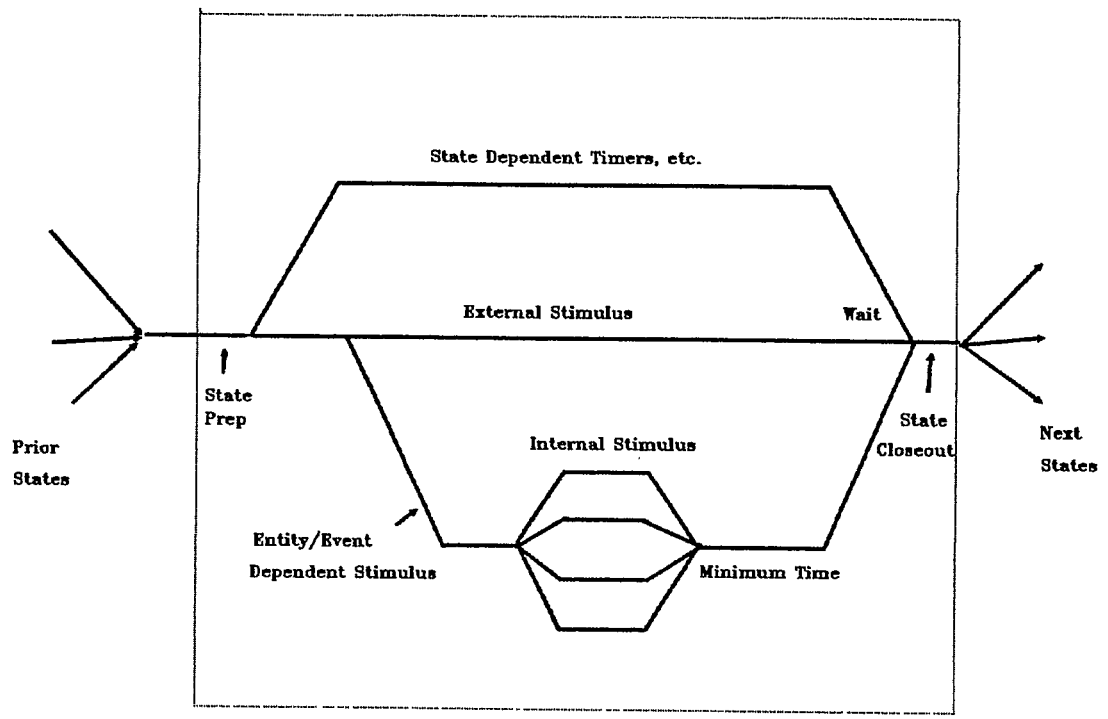

Figure 4 – General State Structure

- describing and documenting the results of the data flow analysis
- provision of a thorough analysis to ensure that major events and flows have indeed been captured
- clear identification of how the simulation code can be structured and what the interface is between each process
- assistance in specifying what the normal execution path is compared to exception processing, which can also be used as the basis for determining what should be in a prototype

7. CONCLUSION

We have indicated how three major software engineering principles have been applied to a complex simulation problem and how formal and semi-formal approaches can be used together in a practical situation. We feel that the resulting model exhibits operational and structural traits that will allow the model to be used for many experiments (data and algorithmic) without modifying the simulation code and when changes to the code must be made, the impact is minimized and risk reduced.

BIBLIOGRAPHY

Bergland, G.D. (October 1981). "A guided tour of program methodologies," Computer.

Goldberg, A.T. (July 1986). "Knowledge-based programming: A survey of program design and construction techniques," IEEE Transactions on Software Engineering, Vol. 12, No. 7.

Golden, D.G. (October 1985). "Software engineering consideration for the design of simulation languages," Simulation.

Larkin, J. McDermott, J., Simon, D.P., and Simon, H.A. (1980). "Expert and novice performance in solving physics problems," Science, Vol. 208, June 1980.

McKay, K.N., Buzacott, J.A., Moore, J.B., and Strang, C. (1986). "Software engineering applied to discrete-event simulation," Proceedings of Winter Simulation Conference.

Parnas, D.L. and Clements, P.C. (February 1986). "A rational design process: how and why to fake it," IEEE Transactions on Software Engineering, Vol. 12, No. 2.

Pegden, C.D. (1986). Introduction to SIMAN, Systems Modeling Corporation, State College Pennsylvania.

Pressman, R.S. (1987). Software Engineering: A Practitioner's Approach, McGraw-Hill, New York.

Ryan, K.T. (1979). "Software engineering and simulation," Proceedings of Winter Simulation Conference.

Sheppard, S. (January 1981). "Applying software engineering to simulation," Simulation.

Standridge, C.R. and Pritsker, A.A.B. (1987). TESS: The Extended Simulation Support System, Halsted Press, New York.

Yau, S.S. and Tsai, J.J.P. (June 1986). "A survey of software design techniques," IEEE Transactions on Software Engineering, Vol. 12, No. 6.

Yourdon, E. and Constantine, L.L. (1979). Structured Design, Prentice-Hall, Englewood Cliffs.

Zeigler, B.P. (1976). Theory of Modelling and Simulation, Wiley, New York.

Zeigler, B.P. (November 1987). "Hierarchical, modular discrete-event modelling in an object-oriented environment," Simulation.

Zelkowitz, M.V., Shaw, A.C., and Gannon, J.D. (1979). Principles of Software Engineering and Design, Prentice-Hall, Englewood Cliffs.

AUTHORS' BIOGRAPHIES

KENNETH N. McKAY (BMath, MASc) is Associate Director of WATMIMS, the University of Waterloo Management of Integrated Manufacturing Systems Research Group and a software consultant. He has R&D experience in the computer industry and has contributed to a large number of software projects, including topics such as: computer performance monitoring, relational databases, accounting, real-time control, simulations, and job-shop scheduling software. Mr. McKay's expertise includes: design evaluation, software engineering, software quality, systems architecture, and man-computer interfaces.

Kenneth N. McKay
Associate Director, WATMIMS Research Group
Department of Management Sciences
University of Waterloo
Waterloo, Ontario, CANADA N2L 3G1
(519) 888-4519

JAN LAUBE (BA, MA) of the Bureau of Management Consulting, is the project leader for the Search and Rescue Simulation Project. He has a background in economics and is interested in the modelling of social and physical systems. Mr. Laube has used techniques such as micro-economics, simulation, and statistical risk analysis in his projects. Some of the projects he has undertaken in the past are: risk analysis of oil and LNG/LPG spills, determination of the rate of deterioration of the housing stock for capital cost allowances, and a study of the impact of foreign investment in various industry sectors.

Jan Laube
Bureau of Management Consulting
Supply and Services Canada
365 Laurier West
Ottawa, Ontario, CANADA K1A 0S5
(613) 995-7144