CONCISE REPRESENTATION OF AUTONOMOUS INTELLIGENT PLATFORMS IN A SIMULATION THROUGH THE USE OF SCRIPTS

Avelino J. Gonzalez
University of Central Florida
ajg@engr.ucf.edu

Robert H. Ahlers
Naval Training Systems Center
ahlers@ntsc-rd.navy.mil

ABSTRACT

The focus of the investigation described in this paper is the development of a concise, yet rich knowledge representation paradigm that could be effectively and efficiently used to model the intelligent behavior of simulated agents in a simulator-based tactical trainer. The behavior of these agents would be similar to that of an adversary who would react to a student's action in a manner representative of enemy tactics. The availability of this feature would be of significant utility to the training process for two reasons: 1) the student would face a realistic enemy who is knowledgeable about tactics in the domain of interest and, 2) the instructor would not have to be burdened with playing the part of the enemy in those training systems where this is commonly done.

The hypothesis presented is that a combination of a script-like structure and pattern-matching rules in an object-oriented environment could serve as the desired representation and reasoning paradigm. This hypothesis was tested through the development of a prototype system that implemented the knowledge of a submarine tactical officer on a patrol mission. The prototype was implemented in CLIPS 5.1, a rule and object-based expert system shell developed by NASA. The results of the prototype show that the combination of scripts and rules in an object-oriented environment promises to meet the requirements described above.

1.0 INTRODUCTION

The use of intelligent simulated agents in a training simulation can provide a more realistic training experience to the students than would be presented otherwise. This is especially true for tactical trainers where an intelligent agent representing an adversary is able to react and counter the student's action in a realistic fashion.

In many training simulators, the instructor typically controls the simulated adversaries, providing them with intelligent behavior. But this can be quite burdensome to the instructor. The presence of intelligent agents in a training simulation that can autonomously react to the student's action in a realistic fashion would not only serve to increase the effectiveness of the training process, but also make it more efficient by off-loading some of the tasks from the instructor. Intelligent simulated agents will be henceforth referred to as *Autonomous Intelligent Platforms* (or *AIP*s).

But creating truly intelligent AIP's is a difficult task due to the many potential variations of any scenario, and the multiple actions that can be taken for each scenario. A rule-based paradigm appears to be a natural way to represent this knowledge, but the numerous conditions resulting from the many variations can translate into a large number of rules for even relatively simple tasks. Alternative representation and reasoning paradigms such as model-based, constraint-based, or case-based reasoning, although promising in some respects, (see Borning, 1977, and Castillo, 1991) are not "natural" for this form of knowledge.

Thus, a significant obstacle to the full-scale development of AIP's for training simulators is the lack of a concise, yet rich, representation paradigm and reasoning scheme for the knowledge and behavior involved in such a task. Before describing the approach, however, a look at the type of knowledge to be represented may be warranted.

1.1 Tactical Knowledge

Tactical knowledge is typically general in nature. An example of general tactical knowledge is:

When facing an inferior enemy who has a tactical disadvantage, attack as soon as possible.

While the rule is seemingly simple and straight-forward, it may be quite difficult for a non-expert to identify what constitutes an "inferior enemy" or a "tactical disadvantage". Thus, the inherent difficulty experienced by non-experts with such general axioms is the classification of the situation in a form abstract enough for the general tactical knowledge to be applicable. An expert, on the other hand, would look for key features such as the enemy's numbers, their heavy weaponry, their defenses, the surrounding terrain, the weather and the enemy's ability to fight in it, the element of surprise, etc. These

would allow him to identify the enemy's strength and evaluate his advantage or disadvantage.

Thus, tactical experts are proficient at their task by recognizing and treating only the key features of the situation, and abstracting these for use as the premises for the general knowledge. Thus, they only use a small portion of the available inputs.

An example of this is the tactical exercise of driving an automobile: An expert driver is generally bombarded with a multitude of inputs when driving: audio inputs such as engine noise, road noise, traffic noise, a blaring radio, conversation with passengers, etc.; visual inputs such as the instruments, other automobiles, the surrounding scenery, pedestrians, etc.; tactile inputs such as vibrations of the car, the position of the steering wheel, the gear shifter, the clutch, etc. These inputs are handled easily, almost subconsciously, when they are all in the normal or expected range. However, if one of these should become abnormal, such as the noise and vibrations that result from a tire blowout, the driver will immediately focus on these in order to recognize the present situation as a blowout, while ignoring all the other ones. This is referred to as *situational awareness*.

Most tactical actions in the military consist of a pre-defined set of actions which are embarked upon after a certain situation has been recognized. The situation could be a mission, a set of orders, or merely a reflection of a specific set of battle conditions at the moment. The problem faced by military tacticians, therefore, is two-fold: 1) how to recognize the present situation (situational awareness), and 2) what to do when the situation is recognized (referred to as *actionable information*).

1.2 Description of the Approach

The approach described here to address the above problem is based on the following hypotheses:

1) There is only a limited number of things that can take place in any situation. Using the example of the automobile driver, it would not be normally expected that a tire blowout take place while waiting at a stop light. This can be used to advantage to prune the search space of the problem, since there is no need to consider a blowout while waiting at a stoplight. Getting rear-ended, on the other hand, is a much more likely proposition.

2) The presence of a new situation will generally alter the present course of action to some degree. For example, the recognition of a blowout at highway speeds will cause the driver to attempt to coast to a stop while maintaining a firm grip on the steering wheel, and directing the car towards the shoulder of the road. Thus, the context changed from one of "normal driving", to one of "blowout", with its attendant actionable information. This context remains in effect until the car comes to a complete stop, at which point another situation will be recognized and acted upon (e.g., get out of car, inspect tire, change tire).

By associating the potential situations and corresponding actions to specific contexts, the identification of a situation can be simplified because only a subset of all possible situations is applicable under the active context. This context-based approach also easily addresses what actionable information to use when a situation is recognized.

One approach to implementing the approach described above lies partly in the use of a script-like concept. A *script* is a knowledge representation paradigm developed by Roger Shank at Yale University [Schank and Abelson, 1977] which attempts to capture the actions, objects, persons, and concepts that may be related within a given context. For example, a restaurant script will be composed of all the actions which are typically part of going to a restaurant, such as reading the menu, ordering the meal, eating it, paying the bill, etc. A restaurant script also contains props, objects which are typical to a restaurant scene from the customer's standpoint, (e.g., tables, chairs, menus, food, eating utensils, napkins, salad bars) as well as actors (e.g., waiters, hostesses, chefs, busboys). The actions involved are only those typical of the restaurant experience. It would not be normally expected, therefore, that the customer wash his car at the restaurant.

This concept can be easily extended to military tactics. A script can be used in this application to express the set of steps (at either a high or low level) that are necessary to carry out the action required by the present situation. Within the context of a mission, there is a limited number of things that are generally expected in terms of actions to carry out and the expectations in regards to the possible situations. It would be quite difficult to represent all this knowledge using rules alone. Thus, the basis for the work described here is the use of a script-like structure combined with a minimal number of rules as the knowledge representation paradigm for a set of AIP's. For lack of a better name, this representation and reasoning paradigm will be referred to as *Script-Rules-Objects*, or *SRO*. The next section describes in greater detail how SRO can be implemented to achieve the AIP objectives.

2.0 GENERAL DESCRIPTION OF SRO PARADIGM

Scripts are used to represent specific contexts or situations. A script can be likened to a situation that has been recognized, and which has a prescribed set of procedures that must be carried out. Additionally, any situation, by its very nature, will limit the number of other situations that can take place. The behavior of the objects in the simulation are controlled by the script that is active at the time. Exactly one script must be active at any one time. A script is composed of:

- message-handlers that initialize the appropriate objects in the simulation at the point of initial activation of the script.
- message-handlers that execute certain actions during the course of the script.

- rules which are applicable only when the script to which they belong is active. These rules assess the situation, prescribe action to be taken within the context of its script, or determine when a transition to another script is called for.

The AIP is controlled by a *General Script* (*GS*), a high level script that defines the overall mission to be undertaken. The GS is an instance of a class representing the mission to be executed and it is composed of *Acts*, intermediate-level scripts that define certain maneuvers, situations, or tactics relevant to that mission. Acts can be "installed" in a sequential nature, or in response to a developing situation. The Acts, in turn, can have *Sub-acts* which are scripts describing lower-level tactics or maneuvers to be undertaken within the context of the Acts.

Each script (GS, Act, or Sub-act) will have a set of rules and/or procedures attached which will implement some action and/or detect when a transition to another script is called for. The rules, in particular, form the basis of situational awareness. If a change in the parameters of the simulation calls for a change in the situation, the rules will recognize the change and execute the appropriate change of script. The use of scripts, in addition to prescribing actionable information as a block of procedures, can help in the situational awareness process by limiting the types of situations that can be expected. This can be referred to as the context feature of a script. For example, in a context of peacetime and within territorial waters, a submarine would not expect to be attacked by an enemy.

Explaining the concept of SRO would be easier if the explanation is tied to an example. Therefore, the representation of submarine tactical knowledge during a routine patrol mission will be used as an illustration. It is also the basis for the prototype described in section 3.0 below.

The AIP which is the recipient of the tactical knowledge will be referred to as *ownsub*. While this might cause confusion with the traditional naval custom of using this nomenclature to identify the student's own platform, this AIP is the focus of this investigation and should be recognized as such.

Ownsub is represented as an object (instance of class SUBMARINE) in an object-oriented environment. Its static slots (defined as those slots whose values will not change during the simulation) define its capabilities such as its maximum speed, quiet speed, maximum depth, periscope depth, weapon systems, (e.g., number and ranges of torpedoes, missiles), sensors (e.g., range and types of passive sonar, active sonar, radar, towed arrays), and Electronic Warfare capabilities (e.g., sonar decoys). Additionally, ownsub's dynamic slots (those whose values will be updated at least once during the simulation) describe its actual position (i.e., x-coordinate and y-coordinate), depth, heading, and speed, in the course of the simulation, as well as whether the sensing equipment is on or off. Damage assessment as well as the conditions of the stores (weapons, food supply, etc.) is also contained in dynamic slots

A local database containing all of the external information that is relevant to the mission is also necessary. This includes all mission-dependent static inputs which define the task to be undertaken, as well as the environment that will be experienced by ownsub. Some of these are the 1) mission, 2) geographical information, 3) presence of friendly forces in the sector or in the route, 4) basic assumptions about the state of affairs (e.g., peacetime, tensions, war), as well as others, such as coordination with other friendly forces.

This local database should also contain all the situation-dependent dynamic inputs which ownsub would see from its sensors. Such information partially consists of 1) sonar input, 2) radar input, 3) communications with the command or with other friendly ships/submarines.

This type of information should be placed on the database by a central "manager" function that can access all the data in the simulation, yet knows what data is to be made visible to ownsub. It is recommended that a blackboard architecture with hierarchically-arranged blackboards be used to implement this feature. (This, however, was not employed in the prototype.)

The general description of a script (without the inputs) could be stored in memory or on disk. A script can be installed onto an AIP object as may be called for by the situation, overwriting the old one when the change represents mutually-exclusive scripts. If the new script is not mutually-exclusive with the existing one, then it needs to be overlaid onto the old script so that at its completion the original script regains control.

Each script contains procedural attachments to implement procedural control over the simulation, such as dictating the speed, depth and bearing of ownsub. Moreover, the script would also contain a set of rules together with its own "mini" inference engine consisting of a pattern matcher, a Rete net and an agenda, as well as the capability to assert and retract facts from the local factbase, to call procedures, and to change scripts.

A script is always in control of the situation. This is indicated by a fact asserted into the factbase which identifies the script in charge, such as ***(general-script SEARCH-AND-TRACK)***. General Scripts are, by definition, always mutually exclusive with one another. If a change of GS is indicated by the external inputs (i.e., new orders from fleet commander), or by internal reasoning about the situation at hand, then the new GS replaces the old one. The fact that "advertises" the old GS is removed from the factbase and one indicating the new one is asserted.

The Acts of a GS are generally mutually-exclusive with one another, but each co-exists with its parent GS. When a particular script (Act or Sub-act) is in effect, it is considered to be the active-script, as in ***(active-script SECTOR-SEARCH)***. Only one script can be active at any time, other than the GS, which is active as long as it is valid. If the GS is replaced, however, its active-script must also be deactivated. When a mutually-exclusive script (Act or Sub-act) is to be installed, the presently active script is deactivated and considered to be the previous-script i.e., ***(previous-script TRANSIT-TO-SECTOR)***. This

introduces a rudimentary ability to reason temporally when it is important to know what ownsub was doing previously.

Sub-Acts are treated as Acts, except they are not necessarily considered to be mutually-exclusive. They are simply overlaid on top of the active Act. If a non-mutually-exclusive script is overlaid on an active script, the presently-active script is considered to be the background-script ***(background-script TRANSIT-HOME)*** while the new one assumes the role of active script. Upon deactivation of the latter, the background script is re-installed as active if it is still valid. The scripts themselves contain the knowledge of which other script they are or are not compatible with.

The actions to be taken as prescribed by a script is done by sending messages to the appropriate objects. For example, if the SECTOR-SEARCH Act becomes active, then a message is sent to ownsub which sets its speed equal to the quiet speed, turns off the active radar, etc., in order to quietly search the sector. Each script has one initialization message that is sent to ownsub to initialize the script.

Rules will have a pattern in their premises that indicates the active-script to which they are applicable. Only when there is a fact in the factbase indicating the active status of the appropriate script will these rules be "active" and capable of being executed. There are basically two types of rules involved in the intelligent decision-making: *Sentinel rules* continually monitor the simulation data in order to recognize the factors that can lead to a change in situation. These rules are typically tied into a particular script. Sentinel rules form the basis of situational awareness, since it is these that infer the new situation from raw data. *Transition rules*, on the other hand, react to the situation identified by the sentinel rules and determine which script should be activated as a response to the changing situation.

An example of a sentinel rule is one where, when the TRANSIT-TO-SECTOR (full-speed travel to the sector to be patrolled) script is active, a rule belonging to that script will monitor the position of ownsub so that arrival at the designated sector is recognized. This rule requires the fact ***(active-script TRANSIT-TO-SECTOR)*** be present in the factbase. The action of this rule may designate the situation to be ***(situation arrived-in-sector ownsub)***. A transition rule will recognize this posted fact, and react by activating the SECTOR-SEARCH script, and initializing all appropriate elements. Once the new script is activated, the sentinel rule mentioned above is no longer applicable, since the parent script has been de-activated.

Some sentinel rules, however, will always be "active" regardless of which General-Script/Act/Sub-act is active. These are general rules that oversee the entire simulation and are not tied to any one script. For example, a rule that searches for enemy torpedoes needs to be always in an active status whenever ownsub is out of port in a wartime context, whether there are enemy submarines detected or not. If an under-attack situation is identified, then a transition rule will immediately activate the UNDER-ATTACK script. Such rules do not require any active-script facts to be present in the factbase in order to execute.

Please note that the above section generally describes how the SRO paradigm should be implemented. It does not describe exactly how the SRO paradigm was actually implemented in the prototype described in Section 3.0 of this report. The prototype represents some simplifications of the description contained above.

3.0 IMPLEMENTATION AND VERIFICATION OF SRO PARADIGM

The SRO paradigm was implemented in a prototype in order to verify that 1) it can be used to suitably represent tactical knowledge, and that 2) it can do so concisely. The more specific objective of the prototype was to implement a simple mission of searching a pre-determined sector for the presence of enemy submarines, and to track one when found. Such a mission was labeled *SEARCH-AND-TRACK*. The knowledge implemented in the prototype is described in detail in Gonzalez [1992].

Object-oriented languages excel in applications to simulations. Objects can be defined and assigned a certain behavior, which can be controlled by sending messages to it. Since the prototype is, in its most basic terms, a simulation, an object-oriented language was chosen as the implementation tool. Since the endowment of intelligence to the object ownsub was the primary goal of the prototype, an expert system tool which could manipulate rules was also desirable. Many commercially-available expert system shells incorporate these features. CLIPS 5.1 was chosen for the prototype due to its powerful pattern-matching capabilities, its newly-available Clips Object-Oriented Language (COOL), its procedural capability, its availability in a PC platform, and its low cost.

The basis of this prototype is the instantiation of the class SUBMARINE, called "ownsub", which represents the AIP. Ownsub traverses a Cartesian coordinate plane of unlimited size in three dimensions (x-coordinate, y-coordinate and depth). It's actions are controlled by a set of scripts that are based on the General Script SEARCH-AND-TRACK. The Acts that compose the SEARCH-AND-TRACK GS are called: TRANSIT-TO-SECTOR, SECTOR-SEARCH, COVERT-TRACKING, and TRANSIT-HOME.

These four Acts are mutually-exclusive, and are described in detail in Gonzalez [1992]. Whenever any one of them is installed, a message is sent to ownsub to initialize its parameters for operation under this script. This involves setting its speed, heading and depth, activating or deactivating certain sensors, deploying its weapons, etc. A script is installed by asserting a fact to the factbase that advertises it as the active script. These facts were discussed in Section 2.0 above, and are of the form:

(active-script <script>)

The prototype is centered around the actions of ownsub. Objects representing up to 5 enemy submarines can be created, and they are referred to as *opsub1*, *opsub2*, etc., through *opsub5*. Additionally, other objects in the simulation can be created to represent torpedoes (up to 3 at one time) and sonar decoys (up to 5). Only the ownsub and opsub*n* objects contained any attributes that changed during the course of the simulation.

The performance objective of the prototype was to indirectly control the actions of ownsub by directly controlling those of opsub. The only exceptions to these was when specific orders were given to ownsub.

Situational awareness can be thought of as the reasoning necessary in order determine when to effect a transition to another script. The sentinel rules described in section 2.0 form the basis of this process. For example, the situation where an enemy is detected is determined by a calculation of the distance between the positions of ownsub and that of all existing opsubs. If the active sonars of all submarines are off, then ownsub's passive sonar will detect the presence of the enemy at 3000 meters. If ownsub's active sonar is on, then that distance increases to 4000 meters. Likewise, if the enemy's active sonar is on, not only is the range then 5000 meters, but it is assumed that the enemy is aware of ownsub's presence, something that was not assumed in the previous two cases. Losing track of an enemy happens when the distance is greater than 500 meters above the applicable range. The range of the torpedoes is roughly 7 Km, which is implemented as a run duration of four minutes at a speed of 100 Km/hr.

It should be noted here that while the above is admittedly a mis-representation of the capabilities of submarines, their weapons, and their sensing equipment, (the ranges and speeds are known to be much different than this), the relatively slow speeds of the submarines involved require that short distances be used in order to limit the duration of the simulation to a reasonable one, while at the same time being able to exhibit the many features of the prototype.

A sentinel rule also searches for the presence of torpedoes, and recognizes them as such when the objects representing torpedoes come within the appropriate range as described above. Other sentinel rules monitor simulation for the end of an enemy attack, check to see whether a torpedo has hit the target, and when to end an evasion maneuver, among many other things.

In some cases, rules were written such that they skipped the intermediate step of explicitly asserting the situation and integrated the functions of the sentinel and the transition rules into one. While this has the effect of making the system more concise, which was one of the objectives of this prototype, it tends to de-modularize the knowledge, something that could be disadvantageous when carrying out the knowledge engineering for a significantly larger system.

4.0 EVALUATION AND DISCUSSION OF SRO PARADIGM AND PROTOTYPE

The purpose of this section is to describe the findings that were made during the process of developing the prototype, discuss the lessons learned, and mention where enhancements to the SRO paradigm could be made as a result of further research. It is of particular interest to analyze the models described above for their conciseness, since that is considered a critical issue.

4.1 Model Conciseness

The issue of concise representation is a critical one which merits careful evaluation. This section begins by looking at the CLIPS elements used in the prototype and examines their nature. One way to measure conciseness is to simply count all the CLIPS elements used in the prototype. However, a difference must be made between elements used for the purpose of carrying out the simulation, versus those used for intelligent decision-making. Therefore, all the elements for each of the prototypes will be counted, but only those used for decision-making will be used in the analysis of conciseness. The prototype used the following types of CLIPS elements: classes, global variables, functions (time-related, general functions, and main functions), message-handlers, and rules.

The classes represent SUBMARINE, SECTOR, GENERAL-SCRIPT, SEARCH-AND-TRACK, TORPEDO, and SONAR-DECOY. Of the global variables defined, 10 were used to represent time points, while the rest represent ranges (of torpedoes, sonar-decoys, etc.). Of the 13 general functions, only 11 were used for intelligent decision-making.

Of the 27 message handlers, 9 were used as part of the simulation (to move the objects in their prescribed direction.) Thus, only 18 were used for decision-making.

In the rules, 16 of the total of 50 were specifically used for the process of asserting and retracting facts in the factbase which describe data contained within the objects (e.g., position, active-scripts). These can be considered to be for purposes other than decision-making. Thus, only 34 were used for decision-making. The final tally of elements used by the prototype for intelligent decision-making was 6 classes, 15 global variables, 13 time-related functions, 11 general functions, 2 main functions (one for initialization, and the other one for cycling), 18 message-handlers, and 34 rules.

Ideally, it would be of great benefit to develop a purely rule-based version of the knowledge and capabilities exhibited by the prototype, so that an objective comparison of efficiency and conciseness could be made. Nevertheless, a rough comparison can be made with another situational knowledge-based system prototype developed at the first author's institution. The latter prototype is a performance monotoring system that evaluates the actions of a student in an automobile driving simulator. Both systems are similar in that they represent situational awareness knowledge and both were developed with

CLIPS, but the performance monitoring prototype represents its knowledge in rules alone. While the submarine AIP prototype employing SRO required 34 rules and 18 message handlers, the performance monitor required nearly 50 rules and numerous other functions to perform a much simpler situational recognition mission. This comparison, rough as it may be, supports the hypothesis that the SRO paradigm is capable of representing the knowledge involved in the SEARCH-AND-TRACK mission in a concise manner. It is also likely that additional refinements in the system could lead to an even more succinct representation.

5.0 SUMMARY AND RECOMMENDATIONS

This research attempted to determine whether the use of a combination of scripts, objects and rules would represent a concise, yet rich paradigm for modeling AIP's. The results of the research indicate that the SRO paradigm can be used to concisely represent the knowledge required of an AIP in a training simulation. Nevertheless, additional work remains in order to make the concept a "user-hardened" technology.

Most importantly, a formalization of the SRO concept is necessary. The resulting formal knowledge representation paradigm should incorporate features that address temporal reasoning as well as dealing with uncertainties. Additionally, guidelines should be developed to assist a user in creating a knowledge base from human tactical knowledge. These guidelines could be incorporated in a highly interactive graphical authoring system with a look and feel similar to that of the VISTA system [Ahlers, 1990]. Lastly, the burden of carrying out the simulation should be placed on a more generalized simulation environment, with the CLIPS prototype merely providing the intelligence and the control of the simulated agents. This would make the system more widely applicable and possibly capable of being retrofitted to existing simulators.

Furthermore, additional testing in a more complex threat environment is clearly warranted so as to confirm or deny its general applicability to the problem. Lastly, an implementation in an existing training simulator should be undertaken to determine the feasibility of retrofitting the latter with AIP's.

6.0 ACKNOWLEDGEMENTS

The investigation described in this paper was carried out by the author under the sponsorship of the Naval Training Systems Center, Human Factors Division, Orlando, FL, and under the auspices of the ASEE Summer Faculty Program, in the Summer of 1992.

7.0 REFERENCES

[Ahlers, 1990] Ahlers, R. and Schnitzius, M., "Human Engineering the Knowledge Acquisition Interface: The Evolution of VISTA", Proceedings of the The Third Annual Florida Artificial Intelligence Research Symposium, Cocoa Beach, FL, April, 1990.

[Borning, 1977] Borning, A., ThingLab - An Object-oriented System for Building Simulations Using Constraints", ACM Transactions on Programming, Languages and Systems, 3 (4) October, 1977, pp 353 - 387.

[Castillo, 1991] Castillo, D., "Toward a Paradigm for Modelling and Simulating Intelligent Agents", Doctoral Dissertation, University of Central Florida, December, 1991.

[Gonzalez, 1992] Gonzalez, A. J., "Concise Representation of Autonomous Intelligent Plattforms in a Simulation through the Use of Scripts", Technical Report TR92-019, Naval Training Systems Center, Orlando, FL, (in preparation).

[Khboshch, 1990] Khboshch, Vladimir Alekseyevich, "Submarine Tactics", Voyenizdat, 1989, translation in 1990, by the Foreign Broadcast information Service.

[Labayle-Couhat and Prezelin, 1988] Labayle-Couhat, J. and Prezelin, B., "Combat Fleets of the World 1988/89', Translated by A.D. Baker III, Naval Institute Press, Annapolis, MD 1988.

[Shank and Abelson, 1977] Shank, R. C., and Abelson, R. P., 1977, "Scripts, Plans, Goals and Understanding", Erlbaum, Hillsdale, NJ, 1977.