The 9th International Symposium on Automation and Robotics in Construction June 3-5, 1992 Tokyo, Japan

VISUAL SIMULATION PROGRAMMING FOR CONSTRUCTION OPERATIONS

By

Amr A. Oloufa, Ph.D., P.E.
Department of Civil Engineering
University of Hawaii
2540 Dole Street, Holmes Hall 383
Honolulu, HI 96822, U S A

ABSTRACT

Discrete event simulation is an extremely valuable tool for construction operations. The application of simulation in construction has been limited due to the extreme hardship in developing valid simulation models and coding them in a simulation language. With the advent of object-oriented modeling and programming, simulation entities may be described in a format that encapsulates their characteristics and capabilities. This paper will detail on-going research aimed at the design of object-oriented entities that embody their performance characteristics and methods. The construction team, using a graphical user interface, may select one or more entities and specify the interaction of these entities in an intuitive format. The user will be capable of changing the number and/or mix of equipment with little or no change to the underlying simulation code. This research will also utilize the latest advances in the creation of graphical user interfaces.

INTRODUCTION

Simulation is a tool to study the dynamics of complex systems. Shanon[1] defines it as "*Simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system.*"

The simulation process starts with the development of the simulation model. A variety of models exists such as logical, mathematical, physical and so on. In this paper, our attention is devoted to the computer model. A model is designed to be a representation of a system. The representation should be as accurate as possible so as to serve as a replacement the system being evaluated. However, the incorporation of every possible feature of the system in its representative model is extremely hard if not impossible. For this reason, the selected model should incorporate in its definition as many elements as possible that represent the primary effects related to the system under consideration.

The simulation approach may be used to study almost any problem. This is provided that the simulation model identifies the elements involved in the simulation along with the major attributes of the interactions between these elements. Each simulation on the model becomes an experiment on the system that can be observed and controlled.

Contrary to mathematical-based techniques for the study of systems, simulation-based analyses do not yield an optimum solution to a situation affecting a system. Simulation can be used to evaluate BETWEEN a set of predefined alternatives. It is clear then that the application of analytical techniques is preferable to simulation. The main reason for using simulation is that the majority of systems cannot be adequately analyzed by mathematical techniques. This is especially true when random activities are inherent in the system such as waiting for a resource to become available. Although some mathematical solutions are available for queueing problems, such solutions are available only for limited cases.

The state of a system is a set of variables needed to describe the system at a specified point in time. The states of critical entities govern the system's behavior. A change in these sets of variables represents movement of the system from one condition (state) to another. State variables change at specified points in time referred to as event times. In discrete-event simulation, the values of state variables do not change between event times. If the state variables of a system change between event times, the system is said to be continuous. Discrete-event simulation studies reflect accurately the behavior of most construction systems and are easier to implement. For this reason, our attention will be devoted solely to discrete-event simulation.

The modeling process generates a simulation model. This model is then used in the simulation experiments. For the simulation exercise to be meaningful, output from the simulation model has to be trusted. To ensure the accuracy of the output, two distinct procedures are involved, namely verification and validation. Verification is involved with the determination that a computer program functions as it is intended, a process similar to the debugging of conventional computer programs. Validation is the process of determining if the conceptual model (as opposed to the computer program) is an accurate representation of the simulated system. Law and Kelton[2] mention that "*If a model is "valid", then the decisions made with the model should be similar to those that would be made by physically experimenting with the system (if this were possible)*".

The success of a simulation study is largely a factor of user interactions with the simulation model. If the simulation model is not trusted, or if its output is "overwhelming" or hard to analyze, the end result would be a failure. In the past few years, the incorporation of animation in simulation models (Oloufa[3, 4]) has lead to a major improvement in both the verification and validation of computer simulation models. Here the users can interact with the simulation model and the results of these "experiments" are displayed in an animated format showing the system entities and their interactions. Animation highlights potential bottlenecks in the system and creates an intimate and trustful environment with the users of the simulation model. Animation also explains the modeled system in terms of the physical interacting entities, thus enhancing the simulation credibility in addition to displaying the system's critical characteristics in a graphical format.

Advantages of using simulation are:

1- May serve as the only resort to study a system that is too complex to be described in an analytical model.

2- Enables the investigator to study an existing or planned system when subjected to any set of operating conditions.

3- Alternate solutions to a problem can be studied and compared.

4- Gives the investigator an insight and enhances the comprehensibility of the system considered. This benefit is similar to that derived from the application of network models (e.g. CPM) in project planning and scheduling.

The following are disadvantages of using simulation:

1- Reliable simulation models are often expensive and time consuming to develop.

2- Simulation generally portrays the system's performance for a given set of conditions. A large number of simulation runs may be required.

3- Simulation aids in selecting the best alternative from the standpoint of the model and its constraints, which is not necessarily the optimum one.

4- The large amount of data produced after simulation may give misleading information about the simulation model's performance and validity.

SIMULATION IN CONSTRUCTION

Perhaps the most commonly cited construction simulation language in the literature is CYCLONE, which was developed by Halpin[5]. CYCLONE is an extremely easy to use simulation language based on the Q-GERT family of simulation programs. The language utilizes a set of five modeling elements and has good data collection capabilities. CYCLONE models are developed graphically using familiar block diagrams. The building entities are then numbered and the simulation code is developed.

In a brief historical review, the GPSS simulation language (Greenberg[6]) was developed by IBM in the early 1960s'. The novelty about GPSS was its process-oriented approach where the operations (i.e. processes) involved in the system were the focus of the simulation. The simulation language also included a "block" building approach, however it was not necessary to represent the system in terms of blocks before the simulation code can be developed. There are currently several commercial simulation programming languages (SCS[7]) that offer a lot more features and programming power than GPSS and CYCLONE.

In the author's opinion, the languages mentioned above suffer from a major weakness. This is because these languages emulate the conventional programming languages such as C and FORTRAN in terms of their emphasis on algorithmic decomposition (i.e. processes). Booch[8] mentions two methods for system decomposition, the algorithmic and object-oriented decompositions. In the algorithmic decomposition, the system is explained in the conventional top-down structured approach where the system processes are identified. Object-oriented decomposition on the other hand is the definition of the system in terms of its key abstractions or components. The first object-oriented programming language SIMULA, a simulation language, was developed in 1967. While this language has failed to become a commercial success, its modeling emphasis is currently regarded as the state-of-the-art.

The recent emergence of object-oriented simulation languages for manufacturing applications has received a lot of attention (Cammarata et al[9], Knapp[10, 11], Roberts et al[12], Rothenberg[13] and Ulgen[14]). The object-oriented view is that the system is composed of interacting physical objects. These objects are typically the central focus of the simulation studies. In the modeling process, it is essential to decompose the system into its component parts. We also declare the valid operations these components engage in and specify how this engagement affects their states before, during and after these operations.

Each object represents a physical component of the system being modeled. The set of objects that represent the same kind of system component is called a class. A class definition is used to define a particular abstract data type. An abstract data type specifies its own operations in addition to its own characteristics. A class has individual objects called instances. A class describes the form of its instances and how they carry out their operations. Objects communicate with other objects using messages. A message is a request that an object carry out one of its operations. A message specifies the operation desired but not how it is done. The receiving object, using its embedded methods, determines how to carry out the operation requested by the message. The set of messages to which an object can respond is called its interface with the rest of the system. The object's operations can be invoked only by the messages its receives via its interface. When an object receives a message, an operation is invoked. This operation is controlled by a specific method stored within the object's definition. An object's response to the messages it receives is governed by its methods. Each method stores information on how to respond to a specific message received by the object. An object with its instance variables and methods is used to model any piece of equipment used in the project where instance variables and methods correspond to performance parameters and functions respectively.

One of the most powerful features of object-oriented languages is inheritance through hierarchical system description. The principle feature of class hierarchy is that any class inherits all the properties from its superclass but may also have properties that are unique to it. Some object-oriented languages also permit the ability to inherit properties and methods from other classes (i.e. multiple inheritance). Inheritance facilitates programming since new classes need only be specified by their difference from an existing class rather than having to be defined from scratch (also refer to Oloufa[15]).

VISUAL PROGRAMMING AND MODEL GENERATION

Shu[16] defines visual programming as "*The use of meaningful graphic representations in the process of programming*", and "*the application of graphical techniques and pointing devices to provide visual environments for program construction and execution; for information retrieval and presentation and for software design and understanding*". O'Keefe[17] mentions that there are three benefits to "Visual Simulation", namely Selling (i.e. presentation power), Gaming (i.e. what-if analyses) and Learning. Several researchers (Gordon and McNair[18], Davis *et al*[19], Glicksman[20], Thomasma and Ulgen[21], Tseng *et al*[22] and Ozden[23]) and indeed a few software packages such as XCELL+ (Conway and Maxwel[24]) and SIMFACTORY (Tumay[25]) have attempted to automate the model building process through visual programming. All these implementations however were designed specifically for manufacturing environments and are incapable of **direct** representation of some common situations in construction simulation. In the simulation of manufacturing operations, units of a *product* "enter" the simulation, undergo some *processing* and then leave the simulation environment. However in construction, the situation is mostly a set of interacting resources doing certain jobs, and sometimes alternating the roles of customers and servers. There is usually no "*product*" leaving the simulation.

So far, the author is only familiar with the work of Riggs[26] and Odeh, Tommelein and Carr[27] to create a graphical user interface for CYCLONE. However, these approaches used "*generic*" icons to create a model. Also the user had to be familiar with the simulation language CYCLONE.

The author proposes in this work the development of a graphical interactive environment for simulation model generation. Here the user will build the model through the selection of resources and the specification of the interaction/s between them. These resources may be represented as icons that resemble actual construction project entities, and will reside in a form-based environment. The proposed work to be done can be summarized in the following steps:

1- Identify the entities that need to be represented in a typical model along with their attributes and capabilities each entity *may* engage in.

2- Develop the nature of the interaction between the various entities.

3- Proceed to generate the simulation code.

The proposed system will function as two layers between the user and the simulation language being used. The first layer will be language-independent. An additional layer will change the output from the first layer to the simulation language employed. The components of the proposed system will function as shown in Fig. (1).

While the simulation language utilized does not have to be object-oriented, the mapping of resources and interactions from an object-oriented model to an object-oriented simulation language is much easier. It should be emphasized here that the success of the proposed approach requires the availability of a true object-orient simulation language. Some object-oriented simulation languages that change their native code to the C language before compilation will cause several problems regarding the implementation of the proposed modeling approach.

It also should be noted that the proposed approach does not handle conditional branching situations and some initialization mechanisms. Such situations will be added in future extensions of the proposed system.

RESOURCE SPECIFICATION MODULE [*RESPEC*]:

The resource specification module is used to define the various resources used in the construction project. This definition is a complete representation of the resource that includes its attributes, capabilities, type, and the number of units available.

Resources represent equipment, material and labor. Equipment resources belong to families of specialized equipment types. For example, both trucks and loaders may belong to the resource type "Vehicle" which is capable of translational motion. However each type has its own attributes and specialized capabilities. Trucks are used to carry

soil whereas loaders are used for loading the trucks. Also trucks come in various types in terms of capacity, power, on- and off-road and so on. The object-oriented approach offers a very elegant mechanism for the definition of resources due to the hierarchical nature of their description,. As shown in Fig. (2), the user defines a superclass "Loader" that has the capability "Load Truck". The user may further specialize the Loader by adding Loader1 and Loader2. Each loader has the same function (i.e. Load Truck), but with different attributes. Inherited resource types may also have other specialized capabilities. Therefore the addition of other resource types requires specification of their unique attributes and capabilities only, since common capabilities are inherited from their superclass.

Shown in Fig. (3), is the data input form for the REsource SPECification Module [RESPEC]. Here the user enters the resource name and the number of units available. The "Type" is a combo box with the various families of construction resources. The "Type" field provides the information necessary for the inheritance mechanism. The user then enters the various attributes for the resource. For example, for a loader, these attributes may be load capacity, fuel consumption and so on. The user then proceeds to enter the various capabilities of this resource. For example, a loader can load trucks, lift bucket, push soil, etc. Each resource has the generic capabilities "Create"; to create its own type, "Dispose"; to get rid of itself and "Wait Duration"; to wait for a fixed period of time defined in the module REACT explained below. The duration of each capability may be represented in terms of a statistical distribution of random or deterministic nature, or in terms of a function. Up to four parameters of a statistical distribution or function may be entered as shown. A completely defined Capability is one whose duration is independent of any attribute belonging to other resources in the simulation. For example the "Travel To" capability of a Truck may be dependant only on its normal speed. If however, the truck will be using a variety of roads with different surface conditions, then the duration of the Capability "Travel To" is also dependent on the attribute of the resource "Roadway" and therefore is known only at run-time. Input for functions belonging to these capabilities will be explained in the following section.

RESOURCE ACTION MODULE [*RE*ACT]:

The Resource Action Module (shown in Fig. (4)) is used to define the interactions between the entities in the simulation. In contrast to a Customer-Server approach, here we use the term "Actor" to refer to an active object. An actor is an object that captures one or more resources and then proceeds to implement one or more of its own capabilities or capabilities belonging to the "captured" resources. For every captured resource, the user can ask several capabilities to be invoked by entering the "owner" resource of these capabilities. Each of these capabilities may be dependent capabilities as explained above. In this case, the user enters the attribute name/s and the owner of the attributes. The attributes may belong to the actor, captured resources or may be global variables. For example, a loader (actor) may capture a truck. After this, the loader will invoke its capability "Load Truck" which needs the attribute "Truck capacity" belonging to the captured resource "Truck", to determine the time required for loading the truck.

Each capability invoked may be one of two "types", concurrent or single. A concurrent capability runs in parallel with other capabilities that are invoked. In contrast, a single capability has to be completed before any other capability can be invoked. At the completion of a capability, the specified resource may be released. After the resource is released, it can have one of two options. It may move back and join the pool of idle resources, or it may become an actor and invoke other capabilities. The "Move" cell is used to select this option. For example, after the loading operation explained above is completed, the released actor "Loader" will join the pool of idle loaders. The released truck will become an actor and invoke its capability "Travel To" and proceed to the dump site.

The combo boxes shown in Fig. (4) speed up the data entry. Once an actor is selected, the capabilities of this actor in addition to the capabilities of its superclass may be selected. The attributes of the selected actor are also retrieved. This works in conjunction with the Resource Specification Module explained above.

This proposed system may be used to create fairly complex simulation models. However, the description as it stands, lacks conditional branching, initialization and stopping mechanisms. These will be added in future extensions of the proposed system.

ACKNOWLEDGEMENTS

The author would like to thank CACI Corp. of La Jolla, CA for their support. Special thanks to Gary Scillian for his help and comments.

REFERENCES

1- Shannon, R.E.: "*System Simulation: The Art and Science*", Prentice Hall, Englewood Cliffs, N.J. 1975.
2- Law, A. M. and Kelton, W. D.: "*Simulation Modeling and Analysis*", Second Edition, McGraw Hill Inc., New York, NY, 1991.
3- Oloufa. A. A.: *"Feedback Mechanisms For Operational Simulation"*, To appear in the ASCE Journal of Computing In Civil Engineering.
4- Oloufa, A. A.: "*Object-Oriented Simulation of Earthmoving Operations*", Proceedings of the 7th International Symposium on Automation and Robotics In Construction, Bristol, England, May 1990.
5- Halpin D. W. and Woodhead, R. W.: *"Design of Construction and Process Operations"*, John Wiley & Sons, New York, 1976.
6- Greenberg, S.: "*GPSS Primer*", Wiley-Interscience, New York, 1972.
7- SCS, The Society For Computer Simulation: "*Directory of Simulation Software*", 1991.
8- Booch, G.: "*Object Oriented Design With Applications*", The Benjamin/Cummings Publishing Company, Redwood City, CA, 1991.
9- Cammarata, S., Gates, B. and Rothenberg, J.: *"Dependencies and Graphical Interfaces In Object-oriented Simulation Languages"*, Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA.
10- Knapp, V. : *"The Smalltalk Simulation Environment"*, Proceedings of the 1986 Winter Simulation Conference, Washington, D.C.
11- Knapp, V. : *"The Smalltalk Simulation Environment, Part II"*, Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA.
12- Roberts, S. D. and Heim, J.: *"A perspective on Object-oriented Simulation"*, Proceedings of the 1988 Winter Simulation Conference.
13- Rothenberg, J.: *"Object-Oriented Simulation: Where Do I Go From Here?"*, Proceedings of the 1986 Winter Simulation Conference, Washington, D.C.
14- Ulgen, O. J.: *"Simulation Modeling in an Object-oriented Environment using Smalltalk-80"*, Proceedings of the 1986 Winter Simulation Conference, Washington, D.C.
15- Oloufa, A. A.: "*Intuitive Simulation Modeling Using Object-Oriented Constructs*", Proceedings of the 8th International Symposium on Automation and Robotics In Construction, Stuttgart, Germany, June 1991.
16- Shu, N. C.: "*Visual Programming*", Van Nostrand Reinhold, New York, 1988.
17- O'Keefe, R. M.: "*What is Visual Interactive Simulation? (And is There a Methodology For Doing it Right)*", Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA.
18- Gordon R.F. and McNair, E. A.: "*A Visual Programming Approach To Manufacturing Modeling*", Proceedings of the 1987 Winter Simulation Conference.
19- Davis, C. K., Sheppard S. V. and Lively, W. M.: "*Automatic Development of Parallel Simulation Models In ADA*", Proceedings of the 1988 Winter Simulation Conference.
20- Glicksman, J.: "*A Simulator Environment For An Autonomous Land Vehicle*", Proceedings of the 1986 Winter Simulation Conference.
21- Thomasma T. and Ulgen, O.: "*Hierarchical, Modular Simulation Modeling In Icon-Based Simulation Program Generators For manufacturing*", Proceedings of the 1988 Winter Simulation Conference.
22- Tseng, F. T., Zhang, S. X. and Wolfsberger, J. W.: "*Automatic Programming Assistant For Network Simulation Models*", Proceedings of the 1988 Winter Simulation Conference.
23- Ozden, M. H.: "*Graphical Programming of Simulation Models In an Object-Oriented Environment*", Simulation, February 1991.
24- Conway, R., Maxwell, W. L., and Worona, S. L.: "*User's Guide to Xcell Factory Modeling System*", Scientific Press, Palo Alto, CA, 1985.
25- Tumay, K.: "*Factory Simulation With Animation: The no Programming Approach*", Proceedings of the 1987 Winter Simulation Conference, IEEE, Atlanta, GA.
26- Riggs, L. S.: "*Interactive Graphing of Simulation Networks*", ASCE Journal of Computing, Vol. 3, No. 2, April 1991.
27- Odeh, A. M., Tommelein, I. D. and Carr, R. I.: "*Using Design Drawings And Project Plans To Construct Discrete-Event Simulation Networks*", Proceedings of the 8th International Symposium on Automation and Robotics In Construction, Stuttgart, Germany, June 1991.

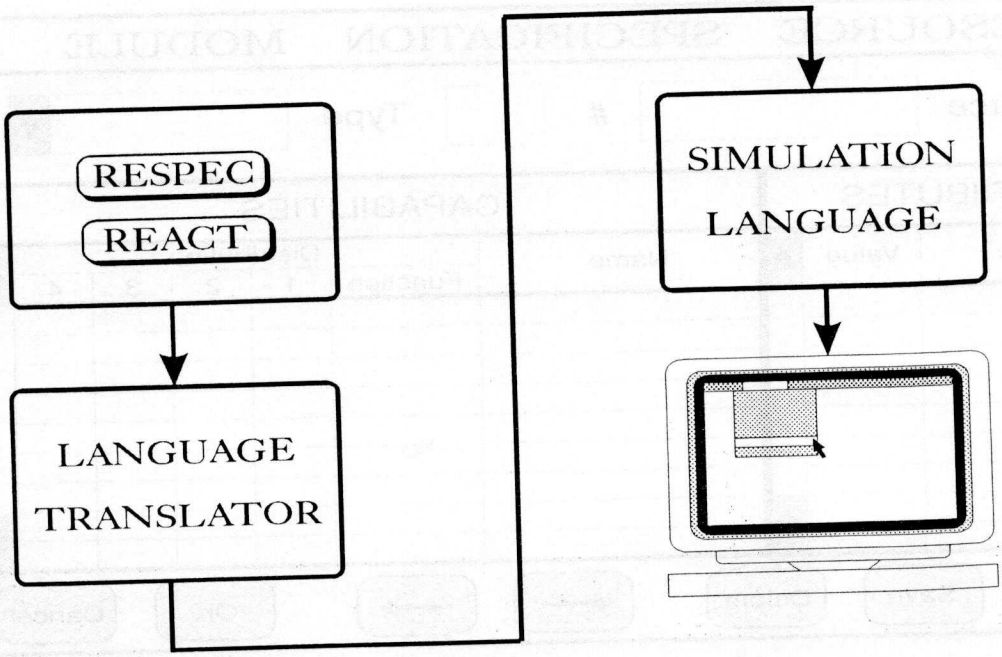

Fig. 1 Proposed System

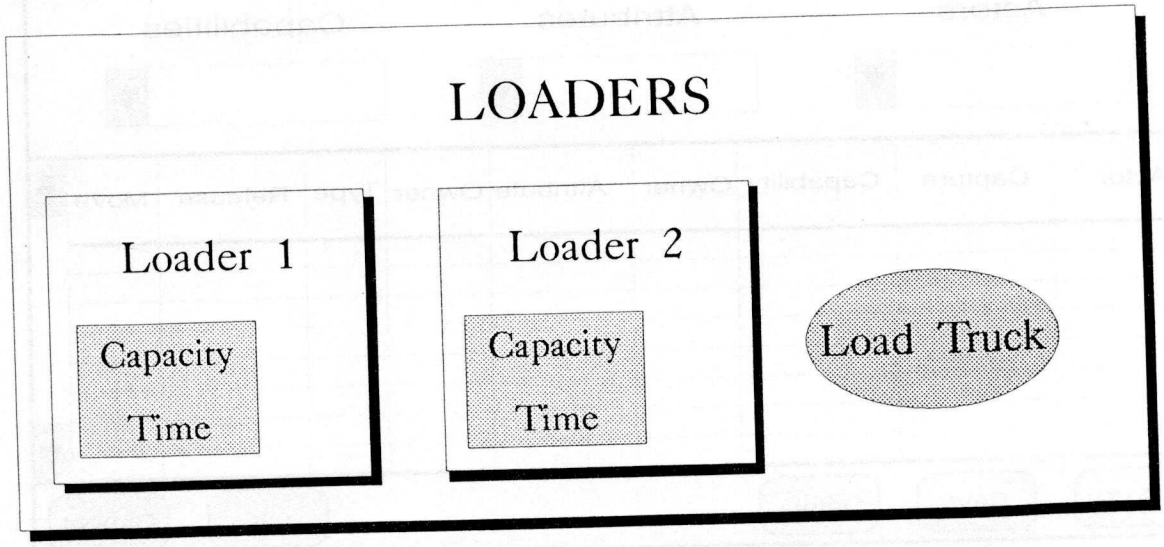

Fig. 2 Inheritance In Object-Oriented Approaches

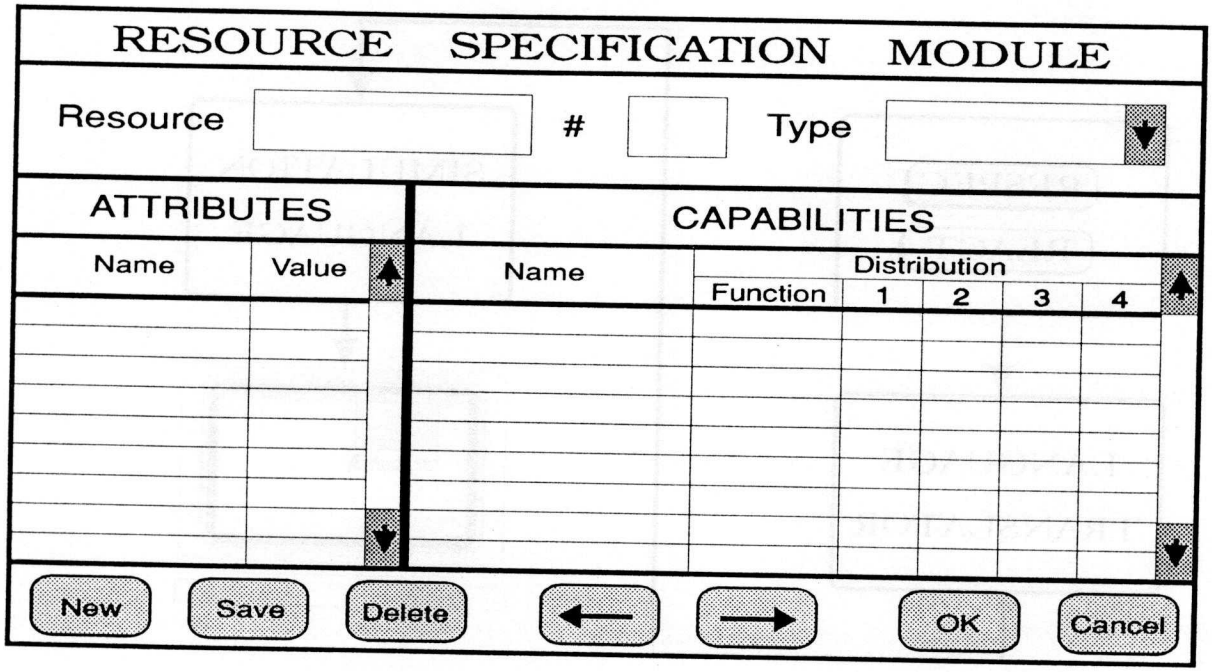

Fig. 3 Resource Specification Module

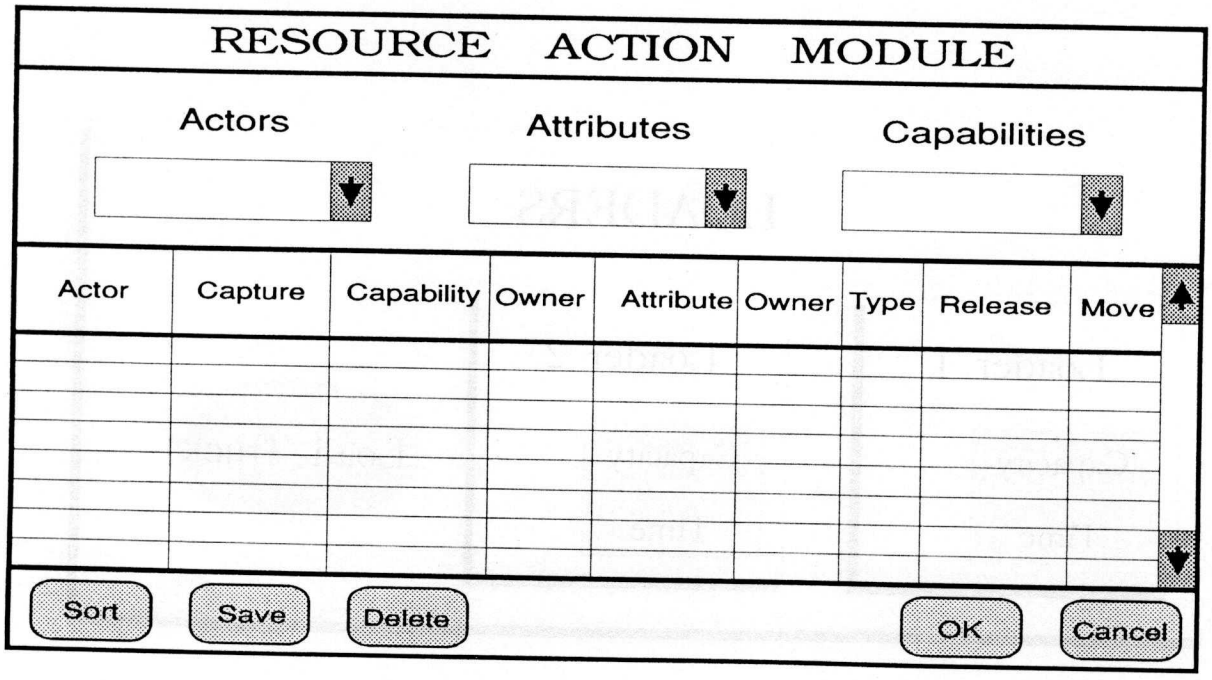

Fig. 4 Resource Action Module