INTUITIVE SIMULATION MODELING USING OBJECT ORIENTED CONSTRUCTS

By

Amr A. Oloufa, Ph.D., P.E.
Assistant Professor
University of Hawaii, Department of Civil Engineering
2540 Dole Street, Holmes 383, Honolulu, HI 96822, USA

ABSTRACT

The development and computer coding of an accurate simulation model is probably the most important and time consuming activity in the modeling of systems and the operations they are involved in. This hardship has confined the use of simulation modeling to the planning stages of the project. The object oriented view is that the system is composed of interacting physical objects. Work to date in interactive simulation modeling has focused on the automation of network coding through the specification of interactions between generic icons that represent transactions, queues, resources, etc. This paper will build on a previous paper by the author for the creation and coding of the simulation model using a more intuitive approach through the utilization of object oriented constructs.

INTRODUCTION

Discrete-Event simulation offers an extremely powerful tool for the planning and execution of operations. After the creation of a model representing the problem domain, simulation is used to compare different scenarios and to check the feasibility of alternative construction methods. A typical simulation study starts by studying the simulated system through identifying its components and how they interact with one another. The success of operational simulation is largely a function of the accuracy of the simulation model used in the study and a large proportion of time invested in simulation studies is spent on formulating, coding and debugging the simulation model.

In the process of model building and verification, many specialists are involved. These include the domain expert who is conversant with the modeled system, the modeler and the simulation programmer. After the simulation model is developed, the decision maker has to be confident of its validity. To reduce the investment in time required for the development of a successful model, the decision maker would have to be trained in simulation methodology and techniques. This is clearly not practical and has been the biggest impeding factor against successful use of discrete event simulation in construction operations.

An alternative approach is to provide the manager with an integrated tool capable of handling the various stages of simulation model development. This integrated environment should allow the decision maker to develop discrete event simulation models with no programming. Major requirements for such a system (adapted from Wales and Luker 1986) are as follows:

1- Allow users to define the system in terms familiar to them not in terms of the simulation language used.

2- Guide users in the steps required for model development.

3- Allow the development of a simulation-model-by-example similar to query-by-example in database systems.

4- Check users responses for consistency and completeness.

5- Creation of new models by the adaptation of existing ones.

6- Provide a visual representation of the system in a form understandable by decision makers.

7- Generate the simulation model code automatically.

Due to the special nature of construction operations and the likelihood of the presence of unforeseen conditions where a quick response is needed, there is a definite need for a utility that "assembles" the simulation model and codes it without programming.

In developing a modeling methodology for construction applications, several issues in addition to the ones mentioned above have also to be considered. The first is the incorporation of site specific information in modeling. Most current modeling methodologies fail to consider the fact that the construction site is always changing with time thereby affecting performance and durations of the various work tasks. Also since communication is a cornerstone for a meaningful simulation, the model and its output should be comprehensible to the end-users.

It is therefore desirable to develop a modeling methodology for construction operations that is capable of accurate representation, easy and intuitive to build models with, and generates simulation code.

Late in the 1960's, a new approach was proposed with the introduction of the GPSS simulation language. The novelty of GPSS was its emphasis on a modeling structure that hides from the user the mechanics of the simulation. GPSS was built upon a predefined class of active entities called "transactions" which flowed through a flowchart of selected operations; similar to the programming flowcharts of procedural languages.

In the 1970's and 80's, several languages like SIMSCRIPT, GPSS/H, SLAM and SIMAN, were introduced and/or modified. These languages have tried to satisfy the dual goals of generality (found in general programming languages) and convenience (provided by specialized simulation languages) by providing a set of predefined concepts for direct modeling.

In construction applications, following the early work at Stanford University by Fondahl, Teicholz and Gaarslev [1960, 1963, 1969], Halpin [1976] developed CYCLONE (CYClic Operation NEtwork). At Stanford University, CYCLONE modeling capability was extended. Kavanagh [1985] at the University of Missouri-Rolla has proposed a computer model for repetitive construction operations by using a personal computer to prepare the code for use by a remote mainframe running GPSS. All above simulation analyses employed in construction operations lacked a feed-back mechanism that adjusts the model's performance due to site changes. Because of the nature of construction operations, Oloufa [1988] proposed an integrated simulation approach as shown in fig.(1) that includes a feed-back mechanism.

USING OBJECTS IN SIMULATION

Dijkstra [1979] mentions that "The technique of mastering complexity has been known since ancient times: divide *et impera* (divide and rule)". In the modeling process, it is essential to decompose the system into its component parts. We then proceed to refine these parts further in an attempt to capture the system's essence. Booch [1991] 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. Booch [1991] also mentions that complex systems can not be modeled by both approaches simultaneously since the above mentioned decompositions are orthogonal. He mentions that experience leads us to apply the object-oriented view first and then use the resulting framework for expressing the algorithmic decomposition perspective.

In describing a system using the Object-Oriented decomposition, we define its components and how these components interact together [Rothenberg (1986), Cammarata *et al* (1987) and Roberts *et al* (1988)]. We also declare the valid operations these components engage in and specify how this engagement affects their states before, during and after these operations. The states of these equipment along with their respective performance parameters are described before they engage in their respective operations, during, and after the operations are completed.

In the process of system modeling, the components of the system, how they interact and the possible outcomes of their interaction are identified. The next step is the mapping of these components to the respective terminology in the modeling environment. We are faced with a situation that what used to be comprehensible physical components have now to be represented in terms of transactions, queues, resources, attributes, etc. in a network-oriented simulation language (see fig.(2)).

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. The simulation problem is one of finding convenient means of modeling these objects in an effort to simulate their behavior.

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. 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 as shown in fig.(3).

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. Each subclass has the properties of its super class and may also have properties that are unique to it. 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.

Oloufa [1990] used a customer-server approach to demonstrate the application of both object-oriented [Knapp (SimTalk 1986, 1987), Bezivin (1987, 1988) and MODSIM II (CACI 1990)] and traditional simulation languages. Shown in fig.(4) is the model used to simulate an earthmoving operation using SimTalk. Here we have class SimTalkEarthMovingSite which is the simulation environment. Classes SimTalkTrucks and SimTalkLoaders represent the equipment involved in the simulation. Class SimTalkLoaders is the superclass of classes SimTalkLoader1 and SimTalkLoader2. The former classes (SimTalkLoader1 & 2) inherit the same function (i.e. methods) from their superclass SimTalkLoaders however they have their own performance parameters (i.e instance variables) as shown in fig.(5). In our case, the loading operation is the method loadTruckWith: however each of the subclasses has its own performance parameters defined in the method initialize. In the case of adding new loaders to the simulation, the user need not specify their methods but rather only what is "different" about the newly added element (subclass) since the "similarities" are inherited from the superclass. Inheritance is a major advantage of object oriented programming.

In object oriented programming, the simulation environment need not know "how" the equipment execute their operations since the equipment objects themselves execute the methods included "within" their definition constrained by and subject to their instance variables. This means that the same message actions can be executed differently depending on the receiving object. Therefore the addition of new classes of equipment in the model need not change and code in the former program but rather to respond to the same message actions. This is known as polymorphism and is an important advantage of object oriented programming. Another example is for a scraper and a backhoe. Both pieces of equipment are used for excavation. However, each performs the excavation differently. The method "startExcavation" can be sent by an object but is implemented differently in the scrapers and backhoes. A user can therefore change the machinery used in the simulation without having to change the program code in the calling object.

AUTOMATED MODEL GENERATION

Most work to date in the discrete event simulation of construction operations has focused on the adaptation of simulation in manufacturing applications. We have shown that a separate methodology is needed in the simulation of construction operations. So far, the author is only familiar with the work of Leland Riggs of Georgia Tech to create a graphical user interface for CYCLONE (for an example of a CYCLONE network, see fig.(6)). The research done by Riggs however used "generic" icons to create a model. The user had to be familiar with the simulation language which was not object oriented and also had no animation capabilities. 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 icons and the specification of the interaction/s between these icons. These icons will resemble actual construction project entities and will function as "Customer" or "Server" roles depending on the modeled situation.

The proposed work to be done can be summarized in the following steps:

1- Identify all the entities that need to be represented in a typical model along with their operational attributes.

2- Identify the different operations that each entity *may* engage in.

3- Categorize the *potential* interactions between entities and categorize entities as customers and/or servers.

4- Design the graphical interface with these entities as "building blocks".

5- Develop the dialogue with the user where the specification of the interaction between the different entities is defined.

6- When the model "blocks" are specified, proceed to develop the code.

O'Keefe [1987] 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 (1987), Davis *et al* (1988), Glicksman (1988), Thomasma and Ulgen (1988), Tseng *et al* (1988) and Ozden (1991)) and indeed a few software packages such as XCELL+ (Conway and Maxwell 1987) and SIMFACTORY (Tumay 1987) have attempted to automate the model building process. All these implementations however were designed specifically for manufacturing environments and suffer from most of the shortcomings listed in the previous sections.

The component objects of our prototype system are one of four types: Labor, Equipment, Material, and Workspace. Each of these objects forms an abstract class (i.e. a class that has no instances). Each abstract class will have subclasses that will add to its content and behavior and augment its methods. For example the abstract class Equipment may have a subclass called Loaders containing methods such as Load_Truck or Dump_Soil. The subclass may in turn contain other subclasses representing the various type of loaders, each with its own performance parameters and specialized methods.

The process starts by the user identifying the "objects" involved in the simulation. Such objects are mapped into the four abstract classes mentioned before. The user then proceeds in "*naming*" all the possible operations these objects MAY engage in, depending on their capabilities. For example a bulldozer may be used in "moving brush" or "pushing a scraper" and so on. The user would then select any "real" object (e.g. a crane) from the menu. The program then asks the user whether this object will play a customer role or a server role. Based on the user's response, the program will ask the user to identify the objects that will serve the complementary server/customer roles. Establishing a customer/server relationship enables the program to create a queue where resources (customers) will reside if a server is busy or not available. A server may serve customers from different abstract classes and the same customer maybe seized by one or multiple servers. However, if a server is not available, each customer will wait in the queue, with different queues created for each abstract object type. After the customer/server roles are identified, the system then asks the user for the method/methods that will be executed after the resource is captured. This is followed by the methods executed after the resource is released by the server.

The proposed prototype will have two types of methods, Independent and Dependent. Independent methods do not rely on the instance variables of other objects in the simulation, in contrast to dependent methods. An example of an independent method is the truck's velocity between two given points. If however, this velocity was dependent on the rolling resistance of the haul road, then this method would belong to the dependent type. If the users specifies that a method is dependent, the system then asks about the instance variables needed and the objects they belong to.

After the model is created, the system would create the model code in an object oriented simulation language. This approach facilitates model assembly and greatly aids model validation and verification and also creates the simulation code with no programming. Users familiar with the simulation language may enhance the developed simulation code if the need arises.

The proposed prototype is by no means capable of modeling every conceivable situation in the construction project. Currently the specification does not support conditional branching mechanisms. Such mechanisms will be included in the future extensions of the system.

PROPOSED TOOLS

The choice of programming tools is of extreme importance in this research. Although it is possible to develop object oriented applications using a procedural language, the produced code will suffer from the disadvantages of procedural programming languages. It was decided to use an object-oriented programming tool that interfaces with the user and helps them to interactively specify and assemble the model. This tool should be capable of the following:

1- Use graphical icons to represent objects.

2- Provide for user responses through common Graphical User Interface (GUI) objects (e.g. buttons, radio buttons, check boxes, list boxes, edit boxes and hypertext).

3- Provide for simple rule-based reasoning.

This research utilizes KnowledgePro from Knowledge Garden Inc. [1991]. KnowledgePro is a rich object-oriented programming language that was designed mainly as a hypertext or expert system tool. Through a dialog with the user, the program requests information about resources and their interactions. The program then asks the user to specify the types of methods invoked and requests any extra parameters. For example, for a Travel_To method contained in a truck object, the program may ask about the distance, slope and rolling resistance etc. After the model specification is complete, the program generates the simulation code using the MODSIM II object oriented simulation language [CACI 1990]. MODSIM II is a powerful language that also allows an object to do more than one operation in the same time. An object in the middle of an operation may receive a message to do a different conflicting operation. In response the object may ignore, defer or execute the conflicting request. For example a scraper may be required to move, excavate and monitor road surface conditions at the same time. If it is confronted with large boulders, it may continue moving but should cease to excavate to save the blade. Although this is a fairly common situation, it has been traditionally difficult to model especially when these activities interact.

Limiting the choice of development tools to object oriented ones facilitates mapping from one program syntax to another and greatly aides debugging of the code. It also minimizes problems related to tracking the source of some run-time errors.

EXTENSIONS AND FUTURE RESEARCH

"Operation Libraries" may be developed for a variety of construction resources where all the potential interactions between these resources are identified. When a problem arises in the construction site, the project team may update the model parameters and/or alter the model with no programming.

The proposed approach may also be used as a teaching tool where students may assemble the simulation model without programming and observe the impact on productivity and/or performance through changing the model parameters. If some programming is needed, the proposed system may be used to develop an "initial" model that serves as a template for further model enhancements.

ACKNOWLEDGEMENTS

The author would like to thank CACI Products Company of San Diego, CA and Knowledge Garden Inc. of Nassau, NY for supporting this research.

REFERENCES

1- Bezivin, J.: "*Design And Implementation Issues In Object Oriented Simulation*", Simuletter, 1988.

2- Bezivin, J.: "*Timelock: A Concurrent Simulation Technique and Its Description in Smalltalk-80*", Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA.

3- Booch, G.: "*Object Oriented Design With Applications*", The Benjamin/Cummings Publishing Company, Redwood City, CA, 1919

4- CACI Products Company: "*MODSIM II User's Manual*", San Diego, CA, 1990

5- 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.

6- Conway, R., Maxwell, W. L., and Worona, S. L.: "*User's Guide to Xcell Factory Modeling System*", Scientific Press, Palo Alto, CA, 1985.

7- 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.

8- Dijkstra, E.: "*Programming Considered as a Human Activity*", Classics In Software Engineering, New York, NY, Yourdon Press, 1979.

9- Fondahl, J. W.: "*Photographic Analysis for Construction Operations*", Journal of the Construction Division, ASCE, Vol.86, No. C02, May 1960.

10- Gaarslev, A.: "*Stochastic Models to Estimate the Production of Material Handling Systems in the Construction Industry*", Technical Report No. III, The Construction Institute, Stanford University, Aug. 1969.

11- Glicksman, J.: "*A Simulator Environment For An Autonomous Land Vehicle*", Proceedings of the 1986 Winter Simulation Conference.

12- Gordon R.F. and McNair, E. A.: "*A Visual Programming Approach To Manufacturing Modeling*", Proceedings of the 1987 Winter Simulation Conference.

13- Halpin D. W. and Woodhead, R. W.: "*Design of Construction and Process Operations*", John Wiley & Sons, New York, 1976.

14- Kavanagh, D.: "*SIREN: A Repetitive Construction Simulation Model*", Journal of Engineering and Management, ASCE, Vol.III, Sep. 1985.

15- Knapp, V.: "*The Smalltalk Simulation Environment*", Proceedings of the 1986 Winter Simulation Conference, Washington, D.C.

16- Knapp, V.: "*The Smalltalk Simulation Environment, Part II*", Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA.

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- Oloufa, A. A.: "*A Framework for the Operational Simulation of Construction Projects*", Department of Civil Engineering, University of California, Berkeley, Ph.D. Dissertation, May 1988.

19- Oloufa, A. A.: "*Modeling Operational Activities In Object Oriented Simulation*", Submitted to the ASCE Journal of Computing, Sep. 1990.

20- Ozden, M. H.: "*Graphical Programming of Simulation Models In an Object-Oriented Environment*", Simulation, February 1991.

21- Roberts, S. D. and Heim, J.: "*A Perspective on Object Oriented Simulation*", Proceedings of the 1988 Winter Simulation Conference.

22- Rothenberg, J.: "*Object-Oriented Simulation: Where Do We Go From Here?*", Proceedings of the 1986 Winter Simulation Conference, Washington, D.C.

23- SimTalk User's Manual, Tektronix Corp., Beaverton, OR, 1988.

24- Teicholz, P.: "*A Simulation Approach to the Selection of Construction Equipment*", Technical Report #26, The Construction Institute, Stanford University, 1963.

25- Thomasma T. and Ulgen, O.: "*Hierarchical, Modular Simulation Modeling In Icon-Based Simulation Program Generators For manufacturing*", Proceedings of the 1988 Winter Simulation Conference.

26- Tseng, F. T., Zhang, S. X. and Wolfsberger, J. W.: "*Automatic Programming Assistant For Network Simulation Models*", Proceedings of the 1988 Winter Simulation Conference.

27- Tumay, K.: "*Factory Simulation With Animation: The no Programming Approach*", Proceedings of the 1987 Winter Simulation Conference, IEEE, Atlanta, GA.

28- Ulgen, O. J.: "*Simulation Modeling in an Object Oriented Environment using Smalltalk-80*", Proceedings of the 1986 Winter Simulation Conference, Washington, D.C.

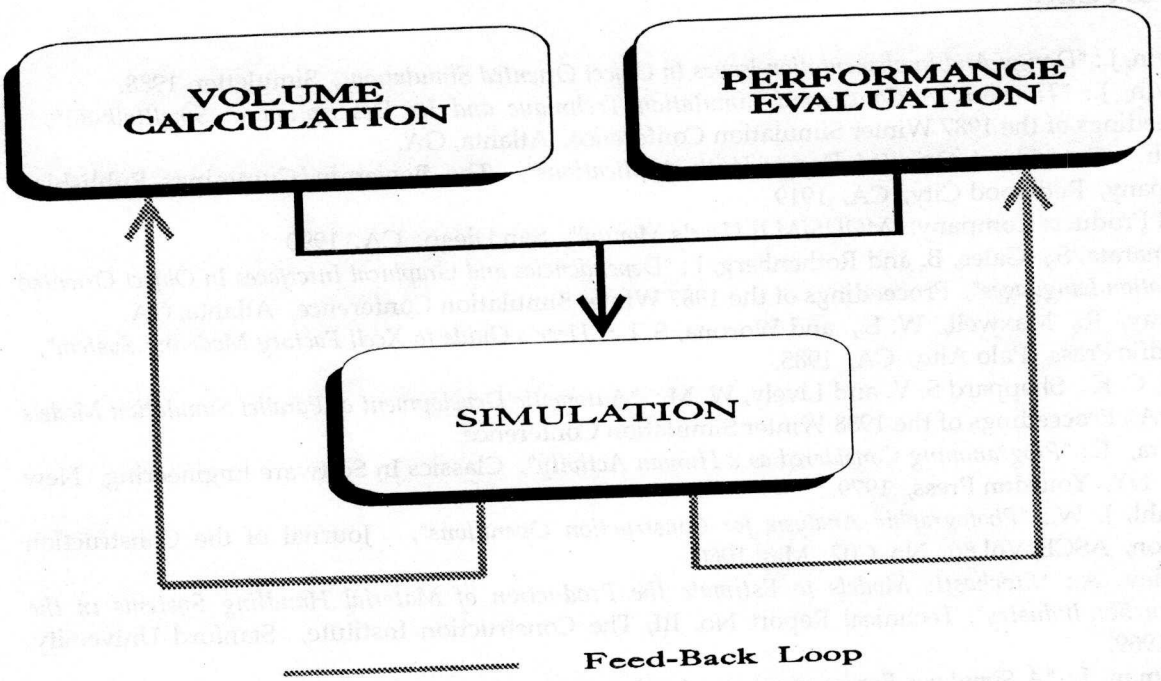
VOLUME CALCULATION
PERFORMANCE EVALUATION
SIMULATION
Feed-Back Loop

Fig. [1]

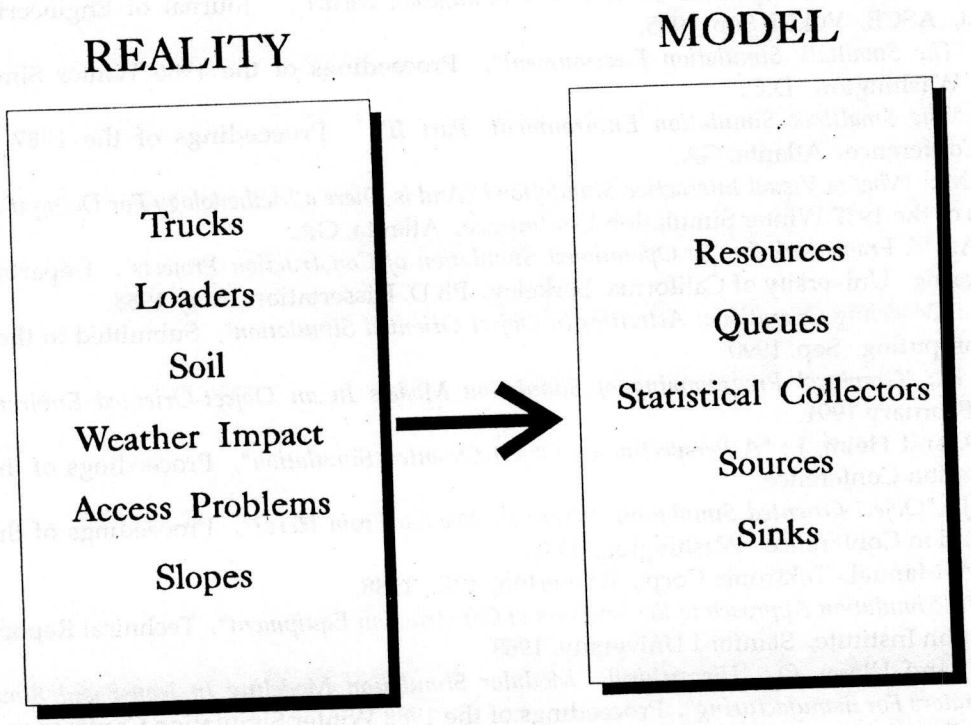
REALITY
MODEL
Trucks
Loaders
Soil
Weather Impact
Access Problems
Slopes
Resources
Queues
Statistical Collectors
Sources
Sinks

Fig. [2]

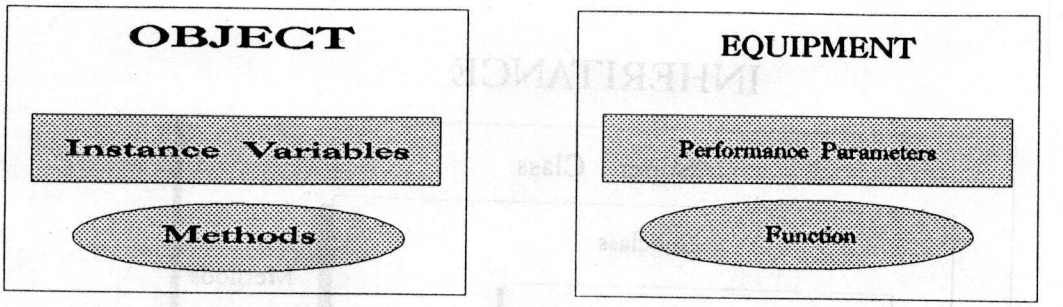
OBJECT
Instance Variables
Methods
EQUIPMENT
Performance Parameters
Function

Fig. [3]

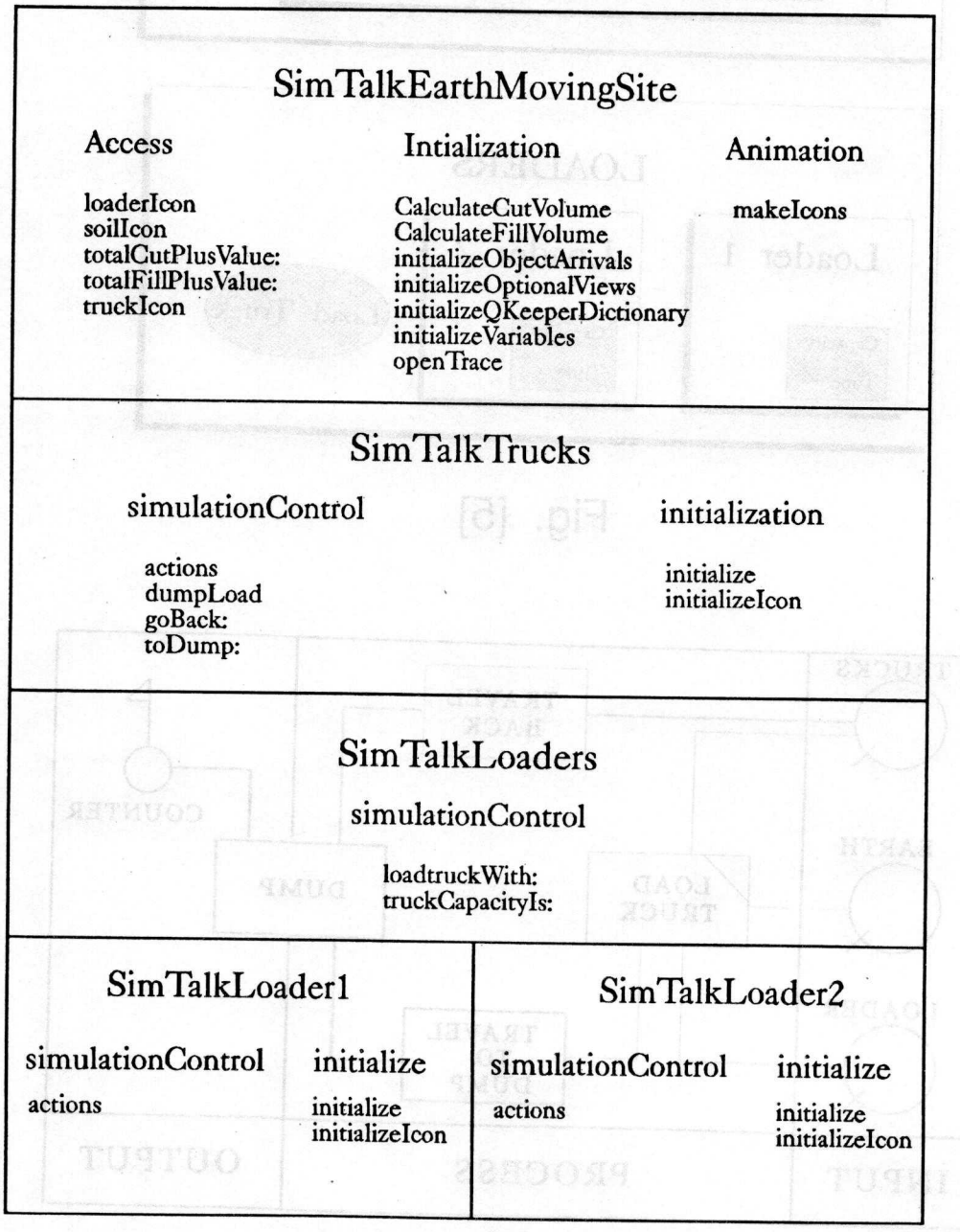
SimTalkEarthMovingSite
Access
loaderIcon
soilIcon
totalCutPlusValue:
totalFillPlusValue:
truckIcon
Intialization
CalculateCutVolume
CalculateFillVolume
initializeObjectArrivals
initializeOptionalViews
initializeQKeeperDictionary
initializeVariables
openTrace
Animation
makeIcons
SimTalkTrucks
simulationControl
actions
dumpLoad
goBack:
toDump:
initialization
initialize
initializeIcon
SimTalkLoaders
simulationControl
loadtruckWith:
truckCapacityIs:
SimTalkLoader1
simulationControl
actions
initialize
initialize
initializeIcon
SimTalkLoader2
simulationControl
actions
initialize
initialize
initializeIcon

Fig. [4]

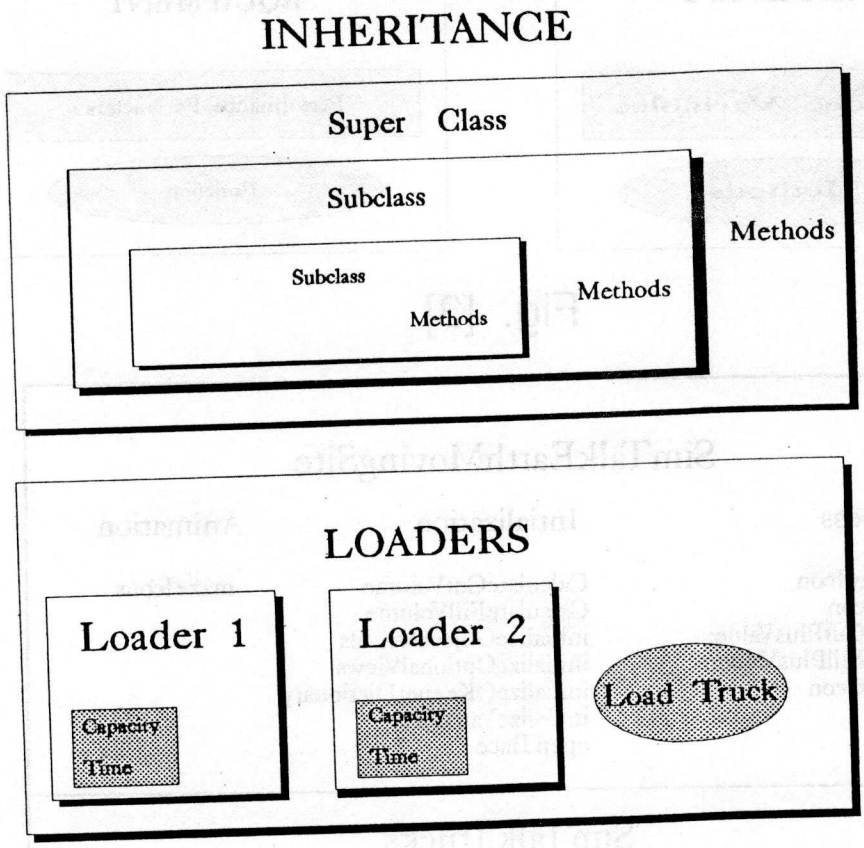
INHERITANCE
Super Class
Subclass
Subclass
Methods
Methods
Methods
LOADERS
Loader 1
Capacity
Time
Loader 2
Capacity
Time
Load Truck

Fig. [5]

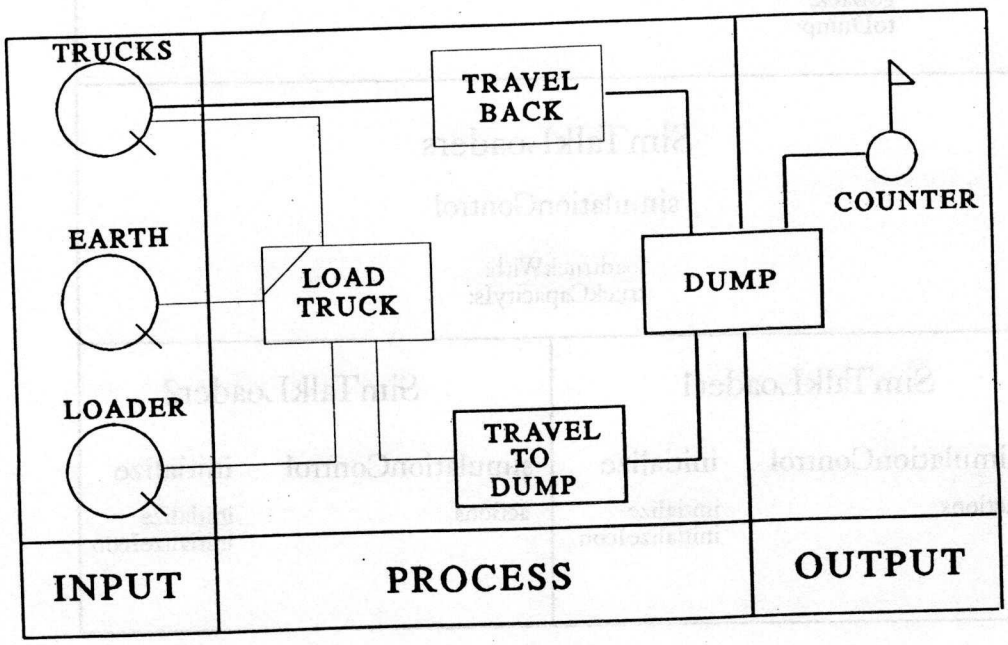
TRUCKS
EARTH
LOADER
TRAVEL BACK
LOAD TRUCK
TRAVEL TO DUMP
DUMP
COUNTER
INPUT
PROCESS
OUTPUT

Fig. [6]