Proceedings of the 1991 Winter Simulation Conference
Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)

EXTENDING COMMON LISP OBJECT SYSTEM FOR DISCRETE EVENT MODELING AND SIMULATION

Suleyman Sevinc

Department of Computer Science
University of Sydney
NSW 2006 AUSTRALIA

ABSTRACT

We described DEVS-CLOS modelling and simulation environment. The environment combines Zeigler's DEVS formalism with powerful constructs of CLOS. DEVS-CLOS allows dynamic model creation and modification and is capable of supporting advanced simulation research, as well as applications.

1 INTRODUCTION

Common LISP Object System (CLOS) is an extension of Common LISP to support object oriented programming (Steele 1990). There are various reasons behind our choice for using CLOS to do modelling and simulation. To start with, simulation environments do not require any programming constructs that can not be supported by a general purpose programming language such as CLOS. Moreover, CLOS consists of some attractive feaures that reduce model development time considerably. Most importantly, the resulting environment is entirely open to modifications, therefore capable of being used in research as well as applications. The extension is named DEVS-CLOS after the formalism it is based on (DEVS, Zeigler(1976)). We will briefly review relevant parts of CLOS in the next section. This will be followed by a summary of novel simulation concepts supported by DEVS-CLOS.

2 CLOS

Basic CLOS facilities include those for defining classes, generic functions and methods. A class definition consists of a set of slots, generic functions to access these slots and other classes (called direct superclasses) whose definitions are inherited. Generic functions are constructs with a set of methods and a dispatching mechanism. Methods are related to the classes via a relation called applicability. A method is said to be applicable only if its parameter specializers are satisfied by the current set of arguments. For multilevel inheritance, a class precedence algorithm is included in CLOS definition which produces a unique ordering of superclasses for a class. The output of the algorithm is used to order all applicable methods associated with the generic function call and the dispatching mechanism selects and applies a subset of these, in the order specified. For more information, the reader is advised to refer to Steele (1990).

3 DEVS-CLOS

DEVS-CLOS is an advanced implementation of DEVS concepts which have been developed by B.P. Zeigler in his seminal works Zeigler (1976, 1984). It is closely related to DEVS-SCHEME described in Zeigler (1990) in some detail (see also Cellier, Zeigler and Wang (1990) and Zeigler, Hu and Rozenblit (1989) It is implemented with two goals in mind; We used DEVS concepts and abstract simulator code to ensure correctness of our simulator and we tried giving our modellers a flexible basis to do applications or research. A typical simulation study in DEVS-CLOS starts with a series of decompositions. The decomposition process results in a hierarchical representation of the system being studied (see Figure 1 for an example supermarket model). The leaves of the system decomposition tree are called the **ATOMIC-MODELS** whereas intermediate nodes are **COUPLED-MODELS.**

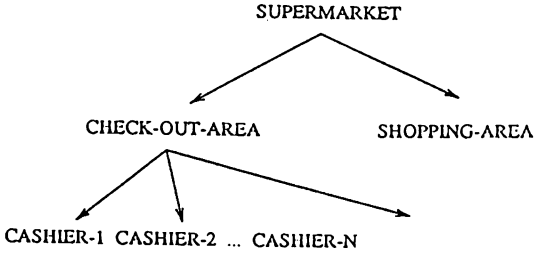

Figure 1: Hierarchical Model Representation

In contrast to the analysis process, the synthesis starts at the bottom. **ATOMIC-MODELS** are implemented first. **COUPLED-MODELS** are formed using coupling specifications which are mappings between input/output ports of other models. To completely define an atomic model, we have to define 3 functions; external transition function to process input events, internal transition function to process events scheduled by the model itself and output function whose return value constitutes an output which is forwarded to the proper destination using coupling specifications. Figure 2 shows all the related definitions for the CASHIER model of Figure 1. We will now provide an example to illustrate dynamic model creation controlled by logic statements in DEVS-CLOS.

```
(defclass CASHIER (atomic-models)
 ((queue-length :initform 0 :initarg :ql
   :accessor queue-length)))

(defext CASHIER
 (self (queue-length model)
     (+ 1 (queue-length model)))
 (if (> (queue-length model) 1)
   (continue)
   (hold-in WORKING (random 10))
))

(defint CASHIER
 (self (queue-length model)
     (- (queue-length model) 1))
 (if (> (queue-length model) 0)
   (hold-in WORKING (random 10))
   (passivate)
))

(defout CASHIER
 (send (make-instance 'data-message
         :port 'OUT
         :content t)
))  )
```

Figure 2. Definition of model CASHIER in DEVS-CLOS

3.1 Novel Model Oriented Operations in DEVS-CLOS

Suppose we are trying to simulate the three is a crowd policy of a supermarket in which a new cashier is promised to be made available whenever all the queues contain 3 or more customers (Figure 3).

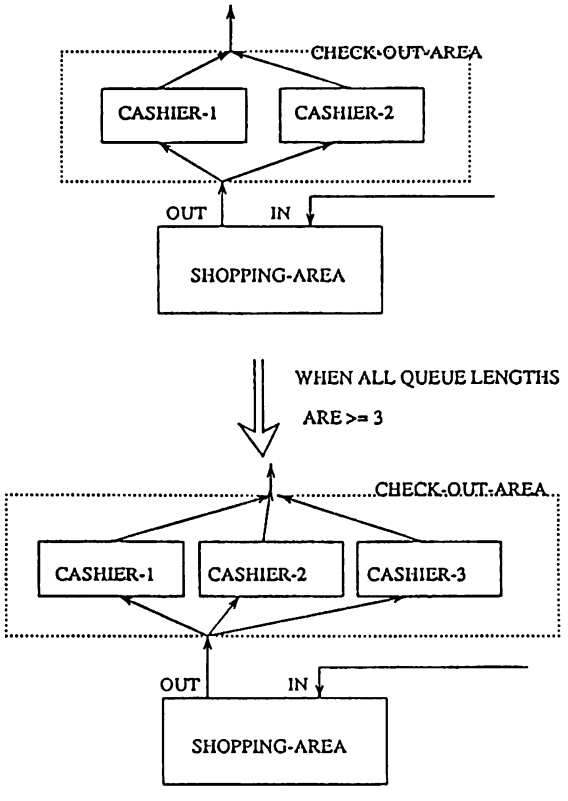

Figure 3: Creation and Integration of a Model

More formally, we would like to create a cashier model whenever the statement $\forall\ m \in$ **MODELS m.queue-length** ≥ 3 becomes true. In DEVS-CLOS, models can be referred to by name and can be dynamically created or destroyed. Suppose the cashier models are named in some consistent manner such as CASHIER-1, CASHIER-2, etc. The statement above can be expressed by (**conditional-expand** CASHIER-~a (elements **MODELS**)) which returns a list of symbols (|CASHIER-1| |CASHIER-2| ...). The macro *conditional-expand* returns all the models in set **MODELS** whose names match the form CASHIER-~a where ~a is don't care. There are many ways in CLOS to test a specific condition on list elements;

```
(mapcar #'(lambda (x)
  (>= (queue-length x) 3))
```

```
(conditional-expand CASHIER-~a
        (elements MODELS)))
```

will return a list with all non-nil elements when all the queues have 3 or more customers. This list may be reduced to a single truth value using **reduce** operation of Common LISP (CL). The following is a complete coding of the statement *whenever* $\forall$ *m* $\in$ MODELS m.queue-length $\geq$ 3 make a new model in DEVS-CLOS.

```
(if (reduce #'log-and
    (mapcar #'(lambda (x)
      (>= (queue-length x) 3))
     (conditional-expand CASHIER-~a
           (elements MODELS)))
  ;;when ∀ m ∈ MODELS m.queue-length ≥ 3
  (let
    ((new-name (car (expand CASHIER-~a range))))
  (defmodel new-name
        (CASHIER atomic-models)
        :couplings ((SHOPPING-AREA OUT)
              (new-model IN))
) ))
```

We would like to note that the above code is edited for clarity where necessary. (**expand string list**) is a macro that generates a list containing all combinations of **string** substituted by the elements of **list** in a manner similar to the **format** statement of CL. **defmodel** is an extended model definition way in which couplings can be specified in terms of (**model port**) pairs. *log-and* is a function which performs logical and.

Similarly, $\exists$ *m* $\in$ **MODELS** such that m.queue-length $\geq$ 3 may be coded in the following way;

```
(reduce #'log-or
 (mapcar #'(lambda (x)
   (>= (queue-length x) 3))
  (conditional-expand CASHIER-~a
        (elements MODELS))))
```

4 CONCLUSIONS

DEVS-CLOS, based on DEVS, is a flexible simulation environment. The environment is model oriented and can fully support both behavioural and structural simulation. The structural changes are triggered by first order statements and involve making new models and integrating them into larger models, destroying existing models when they are not needed and changes in the model structure when necessary. Current research looks into ways of using first order logic to reason about the simulation behaviour.

5 REFERENCES

F. Cellier, B.P. Zeigler and Q. Wang, "A Five Level Hierarchy for the Management of Simulation Models", Proc. Winter Sim. Conf., Dec. 1990

Steele, G.L. Jr., *COMMON LISP: THE LANGUAGE*, second edition, Digital Press, 1990.

Zeigler, B.P., *Theory of Modelling and Simulation*, Krieger Publications, 1984.

Zeigler, B.P., *Multifacetted Modelling and Discrete Event Simulation*, Academic Press, 1984.

Zeigler, B.P., *Object-Oriented Simulation with Hierarchical, Modular Models*, Academic Press, 1990.

Zeigler, B.P., J. Hu and J.W. Rozenblit, Hierarchical Modular Modelling in DEVS-Scheme, *Proc. Winter Sim. Conf.*, 1989.

6 AUTHOR BIOGRAPHY

SULEYMAN SEVINC is a lecturer in the Department of Computer Science at the University of Sydney, Australia. His research interests are model abstraction and theory-based modelling environments.