Proceedings of the 1993 Winter Simulation Conference
G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)

APPLYING ACTIVE DATABASE MODELS FOR SIMULATION

A. Cornelio

Bellcore, PY4-4L211
33, Knightsbridge Road,
Piscataway, NJ 08854, U.S.A.
e-mail: acorn@cc.bellcore.com

Shamkant B. Navathe

College of Computing,
Georgia Institute of Technology
Atlanta, GA-30332-0280, U.S.A.
e-mail: sham@cc.gatech.edu

ABSTRACT

Complex physical systems require the support of data models to define and manipulate the data generated during simulation. These models should have the expressive power to represent the static and temporal relationships among data and they should have the capability to initiate actions to enforce these relationships when the application state is changed. Most commercial databases are passive, i.e., they do not take action or enforce a constraint when the data undergoes a state change. Object oriented database systems have taken the technology one step further by encapsulating operations with the data. In this paper we propose a data model for simulation which builds on object oriented and active database principles to represent physical systems in terms of its structure and function. This model is called the structure-function paradigm, or SF-paradigm for short.

1 SURVEY AND MOTIVATION

Traditional database techniques handle large quantities of uniformly formatted data as described in Astrahan et al. (1976) and Stonebraker, Wong, and Kreps (1976). Here the data manipulation languages (Elmasri and Navathe 1989) and consistency rules (Stonebraker 1975) operate on the database by inserting, deleting, or modifying data. These systems do not adequately model the behavior of the application domain, which is critically needed for the analysis and the simulation of designs.

Object-oriented techniques found in Ketabchi (1985), Kemper, Lockemann, and Wallrath (1987), Andrews and Harris (1987), and Banerjee, Kim, and Kim (1988) and recent enhancements to database models store operations (Ong, Fogg, and Stonebraker 1984), procedures (Stonebraker, Anton, and Hanson 1987b), rules (Stonebraker, Hanson, and Hong 1987a), and methods along with the structured data. These enhancements improve the modeling power of databases by encapsulating the operational behavior with the data object to define the object's semantics. However, as discussed below, encapsulation of application functions within structure is not always natural in simulating engineering systems.

All natural and man made systems that are inherently complex, for example, biological systems or engineering systems do not necessarily have behaviors that are encapsulated within the boundaries of a structural object. On the other hand, they have complex, interdependent functional components that relate to the system structures. An obvious example is the human body, the circulatory and the respiratory systems are complex functional systems that have their own models and abstractions to represent their behavior in terms of functions. In the SF-paradigm, these functions are represented as hierarchies of complex functional objects. The heart and lungs are complex structures which are contained in the ribcage, another complex structure which is part of the skeletal framework. In the SF-paradigm, these organs are represented as hierarchies of complex structural objects. Complex functions (i.e., the circulatory and the respiratory systems) relate to different complex structures (i.e., the heart and the lungs); for example, the lungs belong to both the functional systems, whereas, the heart belongs to the circulatory system. Here, it is easy to see structural abstraction (aggregation of parts) does not follow abstraction of functions, i.e., the abstraction hierarchies in the structural and the functional domains are *not isomorphic*.

When functions are encapsulated within structures, the abstraction of functions follows the abstraction of structures, which implies that the structural and functional hierarchies are isomorphic. Encapsulation of functions within a structure is well suited for generalization hierarchies, where the operations on abstract objects are refined in more specialized objects. This technique however severely limited in simulation applications where modeling aggregation abstractions of physical and functional systems are predominant. Here,

either the functional schema is fragmented to fit within the boundaries of the structure, or the structures are fragmented to fit within functional objects. The S-F paradigm, on the other hand, extends the concept of complex objects as shown in Kim, Chou, and Banerjee (1987) or the concept of molecular objects as shown in Batory and Kim (1985) to include complex behavior without distorting either the structural model or the functional model of the application.

The explicit relationship between structures and functions provides a framework for the database system to process structural and functional queries. Traditional databases can process a limited set of structural queries. An example of a structural query is "Retrieve the dimension of the spindle"; an example of a functional query is "What does the Valve do when the Spindle is turned to the right by 90 degrees?" The combination of structural and functional queries provides a powerful data platform to support simulation. The other advantage of the SF paradigm is to modularly expand the functional model, the structural model, and the relationships between structures and functions as the design and simulation environment changes.

The S-F paradigm states that: to represent simulation applications, a physical component is stored as one structural object, a functional characteristic is stored as one functional object, and a relationship between physical components and functional characteristics is stored as an interaction object. Abstractions apply to both the structural objects and the functional objects (Navathe and Cornelio 1990) thus enabling simulation at different levels of abstraction on the structures or function. The SF-paradigm is general enough (a) to represent engineering designs as shown in Cornelio, Navathe, and Doty (1990), (b) to support the monitoring of industrial processes as shown in Cornelio and Navathe (1993), and (c) to support simulation of engineering designs. Simulating engineering designs by measuring their performance against the expected performance or input specification (Zeigler 1985, and Fishwick 1988) prevents costly mistakes during manufacture. Athough, this paper will focus on data models for the simulation of engineering systems, these modeling techniques are general enough to be applied to any environment that needs to flexibly manipulate structures by one or more functional model; examples of these environments are software prototyping and debugging applications and virtual reality simulation applications.

The paper is organized as follows. The next section describes the constructs of the SF-paradigm, section 3 describes the execution cycle and its application to simulation, section 4 shows through examples how the execution cycle is used for simulation, and section 5 concludes the paper.

2 DATA MODEL

2.1 Formal Specification

A *Structural Object*, S, is a two-place tuple of features, S: <Es, Is> where Es are the *external features* of S and Is are the *internal features* of S. The external features are the properties such as surfaces, terminal pins, leads, connectors, etc., that potentially connect the structural object to other structural objects and to the external world. The internal features are denoted as Is: <Ps, Cs, Rs> where Ps is a set of properties or attributes of the structural object which include the state variables, Cs is a set of sub-structural object declarations and Rs is a set of assembly relations that connect the external features of elements of Cs.

A *Functional Object*, F, is a two-place tuple, F:<Ef, If> where Ef and If are the *external features* and *internal features* of F. The external features refer to a set of input variables, INf, and a set of output variables, OUTf, and are denoted as Ef: <INf, OUTf>. The internal features consist of the state_output function(s), Pf, which consists of the state transfer function, Pfs, and the output function, Pfo, where Pf = Pfo • Pfs; a set of sub-functional object declarations, Cf; and a set of control relations Rf that contain the relationships between elements in Cf. The internal features are denoted as If:<Pf, Cf, Rf>.

The state transfer function generates the new state of the structural object from the current state and the input variable values. The output function generates the outputs of the functional object from the current system state. These outputs are then fed to other functional objects, Cf, by the control relations, Rf. The state transfer function and the output function are implemented by rules, constraints, and/or procedures.

An *Interaction Object*, D, is a three-place tuple, D:<S, F, I>, where S is a set of structural object declarations, F is a set of functional object declarations and I is a set of mappings between structural and functional objects.

The *Interface*, I, relates the structural objects and the functional objects. It consists of three types of mappings, I:<M1, M2, M3>; where, (a) M1 is a set of tuples which map the functional object names, Fi, to the structural object names, Sj,

M1: {<Fi, Sj>}

(b) M2 is a set of tuples which map the state transfer function, Pfs, of the functional objects to some feature of the structural object, Cv,

M2: {<Pfs, Cv>}

(c) M3 is a set of tuples which map the events defined on the state variables, Ps, of the structural objects to a functional object, Fi,

M3: {<Ps, Fi>}

The M1 mappings associate structures and functions at the object level. They determine the types of structures associated with a function, and the types of functions associated with a structure. The M2 mapping relates the state transfer function to the state variables; this mapping updates the state of the system when there is an input to one or more functional objects. The M3 mapping relates events on the state variables to the functional object. The M3 mappings are used for constraint specification and satisfaction where a state variable alerts the functional object by generating events. In an implementation, the M1 and M2 mappings are straightforward and can be represented as tuples as shown in section 2.2, the M3 mapping on the other hand correlates events and is represented by rules, as shown in section 2.3.

2.2 Interaction Types

This section formalizes the M1 and M2 mappings between structures and functions. The M1 mapping describes interaction at the object level, whereas the M2 mapping describes interaction at the state variable and state transfer function level. The interaction types for M1 and M2 are similar, so only M1 will be discussed (for M2 types replace structures with state variables, and replace functions with state transfer functions).

The M1 mapping has four basic association types:

(1) 1:1 structure-function correspondence. The structural description is isomorphic to the functional description. Here, a functional object updates the state variables of only one structural object and the state variables of a structural object are updated by only one functional object. The M1 mapping is denoted as:

M1 = {m1i | m1i = <Fi, Si>}

where Fi and Si are singletons.

(2) 1:n structure-function correspondence. Here, the state variables of one structural object are updated by more than one functional object.

(i) The structure supports all the functions simultaneously. For example, the lungs are used for respiration and circulation. The M1 mapping is denoted as:

M1 = {m1i | m1i = <AND(F1, F2, ..., Fj), Si>}

where AND associates a set of functions F1, ..., Fj to the structure Si.

(ii) The structure supports alternative functions and the appropriate function depends on the context within which the function is active. A good example from Kleer (1985) is a resistor (structure) that can function as a load, voltage sensor, current-to-voltage coverter, voltage-to-current converter, etc. The M1 mapping is denoted as:

M1 = {m1i | m1i = <OR(F1, F2, ..., Fj), Si>}

where OR lists the set of alternative functions F1, ..., Fj associated with the structure Si. The appropriate function is determined from functional context information, described in Cornelio (1989) and in Cornelio, Navathe, and Doty (1990).

(3) n:1 structure-function correspondence. Many structures are associated with a function.

(i) Here, the functional object updates the state of many structural objects. All structural objects are affected by the input to the functional object. For example, in order to see we require our eyes, and brain. The M1 mapping is denoted as:

M1 = {m1i | m1i = <Fi, AND(S1, S2, ..., Sj)>

where AND specifies the set of structures, S1, ..., Sj, associated with the function Fi.

(ii) The same function has alternative structural implementations. For example, the function of lifting can be done by a robot's arm, fork lift, chain pulley, or a crane; the choice of the structure depends on the requirements of the applications. The M1 mapping is denoted as:

M1 = {m1i | m1i = <Fi, OR(S1, S2, ..., Sj)>}

where OR lists the set of alternate implementations S1, S2, ..., Sj of the function Fi. The correct structure is determined from appropriate structural context information, described in Cornelio (1989) and in Cornelio, Navathe, and Doty (1990).

(4) n:m structural-functional correspondence. The state variables of many structural objects are updated by many functional objects. This is the most general and commonly occurring case, and is expressed as a combination of the above two cases.

By inserting or deleting elements from these mappings, the correspondence between the design structure and the design behavior is easily changed. New structures and functions are included in the simulation with minimum effort and prototypes of complex systems are rapidly tested. Also, alternate functional views of a structure, and alternate structural views of a function are easily represented thus facilitating the storage of several simulation options.

2.3 Events

To present the role of the M3 mappings, we have to examine the role of events in a database system. Events are essential for simulation databases that have to monitor their own states and take actions based on (a) these states and (b) the stimulus from the outside world. Events should be detectable and well-defined to be captured by data-modeling constructs. This is done

through conditions which detect (a) when a feature reaches a specified state or exhibits a specified behavior pattern over time; or (b) when the functional object needs data. In the first case, the events will activate or deactivate a functional object; in the latter case the events will query the structural object for an existing state or a past state of the structural object.

The interaction between these events and the functional object is modeled by the M3 mappings.

There are three types of events: the data event, D_event; the function event, F_event; and the virtual event, V_event.

D_events are issued by a structural object when a structural feature reaches a predefined state. These events trigger on the value(s) and/or on the trend of values of a structural feature. Examples of D_events are (a) the temperature crosses a threshold; (b) the temperature increases; or, (c) the time is 8:00 a.m.
The syntax for a D_event is

```
feature_name: domain,
  EVENT: Vi
    {
       ALERT: Pointer to M3 mapping;
       ON: s_value
          WHERE statement;
          WHEN statement;
    }
```

Feature_name is the name of the structural feature on which the D_event is defined. The keyword EVENT is followed by the name of the event, Vi. The keyword ALERT defines the pointer to the M3 mapping in the interaction object. The keyword ON defines the *triggering condition* for the D_event. This condition defines an envelope of states and/or a sequence of states for the feature on which the D_event will signal the functional object. The ON condition is composed of a signal value, s_value, the WHERE statement, and the WHEN statement. The signal value, s_value, consists of the event id and the current state value (or set of state values) which is sent to the interaction object when the event triggers. The WHERE statement contains the monitoring conditions on the value of the feature, whereas the WHEN statement contains the monitoring conditions on the temporal trend of the feature. Each feature can have many events with distinct monitoring conditions that point to different sets of functional objects.

An example of a D_event that triggers if the temperature crosses the threshold, max_c, is shown below.

```
temperature, c: real,
  EVENT: D1
    {
      ALERT: fan_control.map_name;
      ON: (c, t)
        WHERE (insert(c, t) OR update(c, t))
          AND (c, t) > max_c;
        WHEN ((c, t-1) < (c, t));
    }
```

The time stamped tuple for temperature is denoted as (c, t). WHEN is a temporal statement defined on the time-stamped value of the feature. A temporal statement is satisfied if the temporal conditions are met by all values that satisfy the WHERE statement. In the above case, the temperature increases and crosses the threshold, max_c. A more complete presentation on temporal semantics is given in Rafi (1989).

The WHERE statement operates on the first part of the time stamped tuple (c, t); for example (c, t) > max_c in the above example means that c > max_c with t having a don't care value; also (c, T) > max_c means that the value of c is greater than max_c, at a fixed time T. Similarly, the WHEN statement operates on the second part of the time stamped tuple (c, t); for example (c, t) > T means that t > T with c having a don't care value. Therefore, (c, t) > T means any value of c *after* time T, and (c, t) < T means any value of c *before* time T. The complete syntax for the WHERE, and WHEN statements is given in Cornelio (1989).

The *F_event* detects the initiating and terminating conditions for a functional object and passes the values of the features from the structural objects to the input variables of the functional objects. The F_event construct defines the M3 mappings between D_events and the functional objects. The F_event correlates a set of asynchronous low level D_events to define composite higher level F_events.

A F_event is either an F_start event or a F_end event. A F_start event correlates the D_events (or user inputs) needed to activate a functional object. For example, the D_events on position and velocity vector are to be present before a F_start event can be issued to start a trajectory correction procedure for a spacecraft. A F_end event correlates (a) the D_events needed to deactivate the functional object; or (b) the outputs from the functional object that terminate an action. Examples of F_end events include the time running out on an operation in a real time system, a signal that indicates two processes have successfully established contact between two remote hosts, or a sensory signal that stops the movement of a robot's arm.

The syntax for a F_event is

```
EVENT: name,
  {
    PRIORITY: absolute value or time constraint;
    IF: D1_events, or user specified control;
    ACTIVATE: functional object;
    DISABLE: yes/no;
  }
```

The keyword EVENT is followed by the name of the activation rule; by default, it is the name of the functional object if there is only one rule for the object. The keyword IF is the activation condition (or correlating function) to trigger the F_start event. It consists of conjunction, disjunction, or aggregation of D_events or user inputs. The ACTIVATE keyword specifies the functional object that will be activated when the F_start event is triggered. DISABLE is a manual override by the user to temporarily disable the function. The keyword PRIORITY specifies either an absolute priority, for example, a number from a scale of 1 to 5; or a time based priority which specifies the maximum time the F_event can remain in the event server. The F_end event is similarly defined, except that the ACTIVATE keyword is substituted with DEACTIVATE.

V_events are phantom events that are forced by the functional object to retrieve the system state stored in the structural object. The functional object signals a V_event to check the state of a feature that did not report a D_event but whose (i.e., feature's) information is critical in deciding the action to be taken. Therefore, V_events are not correlated by the interaction object and these events are treated as high priority events in the system.

The syntax for a V_event is

```
EVENT: Vi
  {
    ALERT: Pointer to M2 mapping;
    ON: s_value
      WHERE statement;
      WHEN statement;
  }
```

The keyword ALERT points to the M2 mapping(s); the s_value defines a query (or function call) that retrieves the state values from the structural object. This query or function call is executed in the structural object. The WHERE statement detects the missing values needed by the functional object; the temporal conditions in the WHEN statement is a time constraint which specifies the maximum time a functional object will wait for its input variables before issuing a V_event for the missing data value.

3 EXECUTION CYCLE AS UNIT OF SIMULATION

3.1 Introduction

This section describes the simulation schema and the execution cycle. The simulation schema is a general framework to describe the simulation model (functional data), the physical configuration or layout of the design, the data generated during simulation, and the execution policy of the environment. The execution cycle is the basic unit of execution in the simulation schema and will be discussed in detail later in this section.

3.2 Simulation Schema

A simulation schema, SIM, is a five place tuple, denoted as

$$SIM = <S, F, D, DB(t), IC>$$

Where, S is the set containing structural information. F is the set containing functional information, D is the set of interaction objects, DB(t) is the set of values for the state variables (of the structural objects), and the input-output variables (of the functional objects). These variable values are the run-time extension of the simulation schema and they are a function of time and the simulation run number. IC is the execution cycle that relates S, F, D, and DB(t) in a simulation. The next subsection discusses the role of IC, i.e., the execution cycle, in simulation.

3.3 Execution Cycle

The execution cycle is a mechanism to coordinate the exchange of information between the structures and functions to (a) update the system state when inputs are applied to the system or when new state values are computed, or (b) to initiate actions when the system reaches a critical state.

There are two kinds of execution cycles: (a) the function driven execution cycle where a functional object is activated by an external input or by another functional object (i.e., not by events from the structural object); the functional object then computes and updates the state values of one or more structural objects; and (b) the structure driven (or data driven) execution cycle where a structural object due to a state update (or an external input) issues an event which activates a functional object; this functional object computes and updates the state values for one or more structural objects.

These cycles are atomic and are the basic units of execution. A transaction will consist of one or more of these cycles.

3.3.1 Function Driven Execution Cycle

The function driven execution cycle is illustrated for the simple case where the M1 mapping is of association type 1 (see section 2.2), i.e., the topologies of the structural and functional hierarchies are isomorphic. When an input stimulus (i.e., a functional query) is applied to the system, a functional object(s), F1 (say), receives these inputs. These inputs activate the functional object and generate an activation instance for F1, which is a process or activity that is defined by the functional object F1.

The activation instance of F1 *binds* with the structural objects that are supplied by the M1 mappings. These structural objects are now updatable only by F1. The state variables of bound structural objects defined in the M2 mappings are *read* by the activation instance of F1. The state transfer function, Pfs, of F1 *computes* the new values for these state variables. The updated state variables are *written* from the activation instance of F1 to the relevant structural objects. The structural objects are *released* from the activation instance of F1. This completes on function driven execution cycle.

The output function of F1 computes the output variable values from the current state variables. These output variables are transferred to F1's parent functional object which uses the control relations to activate other functional objects thus starting another function driven execution cycle. The updated state variable values in the structural object, on the other hand, can start their own structure driven execution cycle by issuing D_events. Therefore, a function driven execution cycle can give rise to other function driven execution cycles and structure driven execution cycles.

3.3.2 Structure Driven Execution Cycle

We now describe the structure (or data) driven execution cycle. When state variables of the structural object are modified, and if one (or more) triggering condition(s) is met, a D_event(s) is signaled. The interaction object uses the F_start rule to correlate the D_events. If the correlation is successful, the system generates a F_start event. A token (i.e., a pair consisting of event id and priority) for the F_start event is placed on the event-ready queue. The event scheduler uses the token's priority to determine the sequence of triggering for the F_start events. By issuing a F_start event the system has made a commitment to functionally analyze the data that caused the D_events.

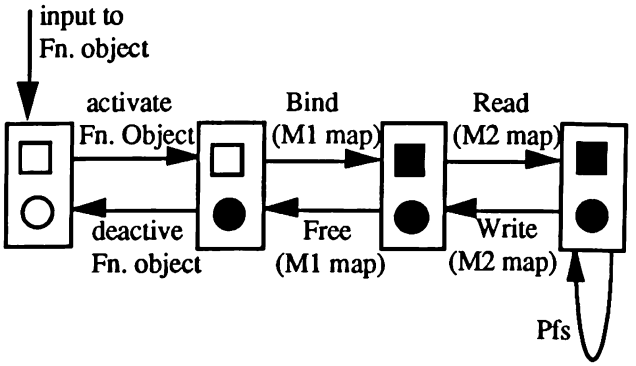

Figure1a Function Driven Execution Cycle

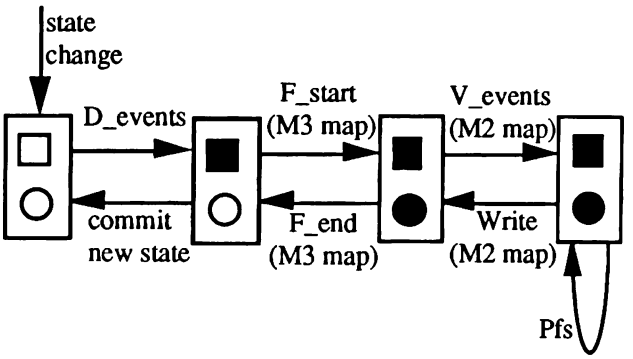

Figure 1b Structure Driven Execution Cycle

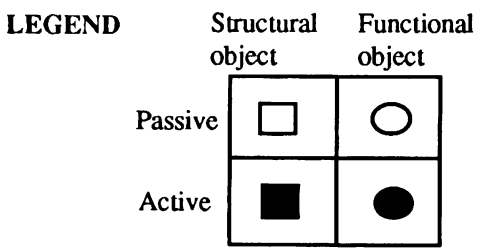

Figure 1 Execution Cycle

The F_start event creates an activation instance for the functional object. This activation instance binds to structural objects which are defined in the M1 mapping. Binding (as in the functional driven execution cycle above) ensures that no other functional object can update the state of the structural object during this execution cycle. The s_values of the D_event (i.e., the signal data) is passed to the functional object. The data not present in the functional object are read from the structural objects by issuing V_events. The V_events use the coupling information held in the M2 mappings to get to the relevant features in the structural objects.

When all the relevant data are read, the functional object executes the set of actions specified in the state transfer function, Pfs. The computed values are committed to the structural object, or the initial state update that caused the D_events is rolled back and the structural objects are released from the functional objects. At this point the F_end event destroys the activation instance of the functional objects. This completes one structure driven execution cycle.

The structure driven execution cycle propagates to other features of the system when (a) there is at least one feature in the current cycle that was updated by the functional object, and this updated feature has a set of events which can generate D_events (excluding the one for the current cycle); or (b) the function delegates the testing to sub-functions. In case (a), the execution cycle repeats for the updated features as a structure driven execution cycle; in case (b), the newly activated sub-functions initiate a functionally driven execution cycle.

Figure 1 illustrates the function driven execution cycle and the structure driven execution cycle. The figure shows how the events and the mappings are used and when the structures and functions are active and passive.

4 EXAMPLE

The example to illustrate the execution cycle will be done in two parts for clarity. The first part will illustrate a structural driven execution cycle by simulating a temperature control system; the second part will illustrate a functional driven execution cycle by simulating the flow of fluids through a valve.

4.1 Temperature Control System

This section uses a temperature monitoring and control system example to describe the operation of the active system. This example first constructs a model for the temperature control system and then applies the structure driven execution cycle described above.

Define a model in terms of structural objects, functional objects, interaction objects, D_events, F_events, and V_events such that

(1) There are two structural objects, FAN and ENGINE, with state variables fan speed and temperature, respectively; there is one functional object, temperature_change, TC.

(2) An event defined on the variable temperature signals a D_event, D1, if the engine's temperature crosses a threshold, max_c.

(3) An event defined on the variable fan speed signals a D_event, D2, if the fan speed changes by 5 percent or more of its current value.

(4) The interaction object, fan_control, has a M1 mapping which relates the structures, FAN and ENGINE to the function, TC. The interaction object also has (a) the M2 mapping which relates the state variables speed and temperature to the state transfer function of the functional object, TC; and (b) the M3 mapping which stores the activation rule between the current temperature, current fan speed, optimal temperature and desired fan speed. On the successful firing of the activation rule, the interaction object generates a F_start event. A F_start event activates a functional object, TC, if either D_event occurs.

(5) The state transfer function in the functional object, TC, updates the fan speed as the temperature fluctuates or warns the user that the fan cannot cool the system.

(6) A F_end event that deactivates the functional object if the actions in (5) are taken or if the user manually disables the warning system.

When the temperature crosses the threshold, a D_event is generated. This D_event, D1, uses the interaction object's M3 mapping to signal the functional object, TC. The activation rule is satisfied due to statement 4 above and a token (i.e., the id and priority value of the F_start event) is placed on the event-ready queue. When the token moves to the head of the queue, the F_start event is triggered. This event creates an activation instance for TC. The activation instance binds with the structural objects ENGINE and FAN by using the M1 mapping. The temperature of the engine is available due to the s_value returned by D1, but the fan_speed is unknown. Therefore, the activation instance of TC issues a V_event on fan_speed and reads it from the structural object FAN by using the M2 mapping. The functional object, TC, executes actions to change the fan_speed or to alert the user. On completing these actions, the new value of fan_speed is committed to the structural object, FAN. The structural objects FAN and ENGINE are released from the functional object, TC, and the F_end event is triggered.

4.2 Flow Control Valve Example

The function driven execution cycle will be illustrated by a liquid flow control example. The function, F1, of the valve is to Transmit_Liquid between two points, and it does this by channeling liquid through the Valve, F11, varying the passage dimensions, F12, and controlling the flow rate, F13. The structures of the Flow_Control_Valve, S1, are: Housing, Spindle, Sealing bush, and various bearing bushes. For clarity, this example will show an interaction between the structural object, Flow_Control_Valve, S1, and the functional object, Transmit_Liquid, F1, illustrating the role of the M1 and M2 mappings in simulation. We start by defining a structural object for the

Flow_Control_Valve, S1, and its corresponding functional object, Transmitting_Liquid, F1, and an interaction object, D1, between S1 and F1.

```
DEFINE STRUCTURAL OBJECT: S1
   EXTERNAL FEATURES
        (not shown)
    INTERNAL FEATURES
    Ps: /* properties */
        Status: {ON, OFF};
    Cs:   /* components (not shown) */
    Rs:   /* assembly relations (not shown) */
END STRUCTURAL OBJECT: S1.

DEFINE FUNCTIONAL OBJECT: F1
   EXTERNAL FEATURES
    INf:   Torque, Ti:real;
    OUTf: out_flow, Fo:real;
   INTERNAL FEATURES
   Pf:  f_transmit   /* state_output function */
      INPUT: Fi;
        BODY:       Status = f_t(Fi, Ti, Status)
        /* state transfer function */
                    Fo = f_out(Status)
        /* output function */
      OUTPUT:   Fo;
   Cs:  F1, F2, F3;  /* components */
   Rs:  /*control relations (not shown) */
END FUNCTIONAL OBJECT: F1.

DEFINE INTERACTION OBJECT: D1
   SCOPE
      STRUCTURAL OBJECTS: S1;
      FUNCTIONAL OBJECTS: F1;
   INTERFACE
      M1:  m11: <F1, S1>;
      M2:  m21: <F1.f_t, S1.Status>
      M3:  /* not shown */
END INTERACTION OBJECT: D1.
```

Assume that the Valve is shut off. On receiving a functional query: "What happens to the Flow_Control_Valve's Status when an input torque, Ti, and an input flow, Fi, is applied," the functional object, F1, generates an activation instance (i.e., a process for F1). The M1 mappings identify that the structural object, Spindle, S1, is affected by the functional object F1, and S1 is bound to the activation instance of F1. The M2 mapping identifies that the state variable (i.e., Status) of S1 will be changed by F1, and the current state variable value "Status = OFF" is copied to the activation instance of F1. The state transfer function, f_t, of F1, processes the input variable, Torque, Ti, the input flow, Fi, and the current state variable, Status = OFF, to generate the new state, Status = ON. The new value of Status is written back to S1 and S1 is released. The functional value then returns the value of Status as the result of the query. The output function, f_out, computes the output variable value, i.e., the output flow rate, Fo, from the current state variable value, Status = ON. This output variable value is sent to the parent of F1 which then sends it to other functional objects.

Through events on the updated state and by the use of the M3 mappings (discussed in the above example), a new structure driven execution cycle may be initiated. Examples can be constructed by combining together a series of structure driven and function driven execution cycles or by performing the execution at different levels of abstraction.

5 CONCLUSION

This paper presented a technique for representing an engineering simulation application in an object-oriented database environment. The contribution here is represent, manage, and query the structures and functions in the application as data. This technique can represent complex structures and complex functions without distorting either representation and at the same time provides for modular extensibility of both structures and functions. Future work will include a control mechanism for structure driven execution cycles and function driven execution cycles.

REFERENCES

Ahmed, R. 1989. Version Control and Management in Design Databases, Ph.D. Dissertation in Department of Computer and Information Sciences, University of Florida, Gainesville, Florida.

Andrews, T., and Harris, C. 1987. Combining Language and Database Advances in an Object-Oriented Development Environment. In the *Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications, OOPSLA*, ACM, Orlando, Florida, pp. 430-440.

Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D., Eswaran, K. P., Gray, J. N., Griffiths, P. P., King, W. F., Lorie, R. A., McJones, P. R., Mehl, J. W., Putzolu, G. R., Traiger, I. L., Wade, B. W., and Watson, V. 1976. System R: Relational Approach to Database Management. In *ACM Transactions on Database Systems*, Vol. 1, No. 6, pp. 97-137.

Banerjee, J., Kim, W., and Kim, K. C. 1988. Queries in Object-Oriented Databases. In *Proceedings of the Fourth International Conference on Data Engineering*, Los Angeles, California, pp. 31-38.

Batory, D. S., and Kim, W., Modeling Concepts for VLSI CAD Objects. 1985. In *ACM Transactions on Database Systems*, Vol. 10, No. 3, pp. 289-321.

Cornelio, A., 1989. A Structure-Function Specification System For Engineering Designs and Simulation, Ph.D. thesis, University of Florida, Gainesville, Florida.

Cornelio, A., Navathe, S. B., and Doty, K. L. 1990. Extending Object-Oriented Concepts for Engineering Design and Simulation. In *Proceedings of sixth International Conference on Data Engineering*, IEEE, Los Angeles, California.

Cornelio, A., and Navathe, S. B. 1993. Using Active Database Techniques for Real-Time Engineering Applications. In *Proceedings of the ninth International Conference on Data Engineering*, IEEE, Vienna, Austria.

Elmasri, R., and Navathe, S. B. 1989 *Fundamentals of Database Systems*. Benjamin Cummings, Redwood City, California.

Fishwick, P. A. 1988. Role of Process Abstraction in Simulation. In *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 18, No. 1, pp. 18-39.

Kemper, A., Lockemann, P., C., and Wallrath, M. 1987. An Object-Oriented Database System for Engineering Applications, *Proceedings of International Conference on Management of Data*, ACM SIGMOD, San Francisco, CA, pp. 299-310.

Ketabchi, M. A., On the Management of Computer Aided Design Databases. 1985. Ph.D. dissertation, Department of Information and Computer Science, University of Minnesota.

Kim, W., Chou, H. T., and Banerjee, J. 1987a. Operations and Implementations of Composite Objects. In Proceedings of the Third International Conference on Data Engineering, Los Angeles, California, pp. 626-633.

Kleer, de J. 1985. How Circuits Work, in *Qualitative Reasoning About Physical Systems*, Bobrow D. G., editor, pp. 205-280.

Navathe, S. B., and Cornelio, A. 1990. Modeling Engineering Data By Complex Structural Objects and Complex Functional Objects. In *Proceedings of the International Conference on Extending Database Technology*, Venice, Italy.

Ong, J., Fogg, D., and Stonebraker, M. 1984. Implementation of Data Abstraction in the Relational Database System INGRES. In *ACM-SIGMOD Record*, Vol.14, No. 1, pp. 1-4.

Stonebraker, M. 1975. Implementation of Integrity Constraints and Views by Query Modification. In *Proceedings of the International Conference on the Management of Data*, ACM SIGMOD, San Jose, California, pp. 65-78.

Stonebraker, M., Wong, E., and Kreps, P. 1976. The Design and Implementation of INGRES. In ACM Transactions on Database Systems, Vol. 1, No. 3, pp. 189-222.

Stonebraker, M., Hanson, E., and Hong, C. H. 1987a. The Design of the POSTGRES Rules System. In *Proceedings of the Third International Conference on Data Engineering*, Los Angeles, California, pp. 365-374.

Stonebraker, M., Anton, J., and Hanson, E. 1987b. Extending a Database System With Procedures. In *ACM Transactions on Database Systems*, Vol. 12, No. 3, pp. 350-367.

Zeigler, B. 1985. *Theory of Modelling and Simulation*, Robert E. Kreiger Publishing Company.

AUTHOR BIOGRAPHIES

ALOYSIUS CORNELIO received his Ph.D. ('89) from the University of Florida and is now a member of technical staff at Bell Communication Research (Bellcore). His research interests include data modeling for business and engineering applications. He is a member of IEEE and ACM.

SHAMKANT B. NAVATHE is a Professor in the College of Computing at Georgia Institute of Technology. He is well-known for his work on database conversion, database design, database modeling, distributed database allocation, and database integration. He has worked with IBM and Siemens in their research divisions and has been a consultant to various companies, including Honeywell, Nixdorf, CCA, ADR, Digital, MCC, Equifax, and Harris corporations. He is an associate editor of ACM Computing Surveys and Data Knowledge Engineering (North Holland) and is on the editorial board of Information Systems and Distributed and Parallel Databases. He is an author of the book, Fundamentals of Database Systems, with R. Elmasri (Benjamin Cummings, 1989). He has published the book, "Conceptural Database Design: An Entity Relationship Approach", with Carlo Batini and Stefano Ceri in 1992. His current work addresses database integration, heterogeneous federated databases, object-oriented databases, the design of intelligent interfaces to databases, multi-media database management, and engineering databases. He is the editor of the series: "Database Systems and Applications for Benjamin Cummings, and has published the book, "Conceptual Database Design: An Entity-Relationship Approach", with Carlo Batini and Stefano Ceri in 1992.