An Object Behavior Modeling Augmented with Modeling Integrity Constraints

Hirotaka Sakai
Deptartment of Industrial and Systems Engineering, Chuo University
1-13-27, Kasuga, Bunkyo-Ku Tokyo 112, JAPAN
sakai@indsys.chuo-u.ac.jp

Abstract

The behavior modeling is a critical task in object oriented database design. The behavior model of objects is defined as a set of states and events which represents a script of object life cycles. This model is structurally refined through the analysis of abstraction structures of the state class and the event class.

The behavior model is further augmented with the assertion class which describes integrity constraints of objects. A method for designing objects based on this enhanced version of the behavior model is also discussed.

1. Introduction

Recently, with the advance of object oriented approaches to database design, the behavior modeling of objects has grown to central concerns in data engineering [1,2,5,9,12,14,18,20,21].

As an object is a coherent concept in the real world, any characteristics of an object should be encapsulated to constitute a totality of a thing. As also discussed in many data models, these characteristics are generally recognized to be structural properties, operational properties, and integrity constraints on objects[3,5,7,10].

Though the word 'behavior' is used to stress dynamic characteristics referring to operations or transactions on objects, it essentially refers to the totality of structural, dynamic, and integrity constraints aspects of objects[4]. Recent remarkable progresses in behavior modeling could be seen in [1,10,11,22,23], where behavior modeling methods based on the object abstraction concepts are systematically developed together with some design tools using diagrammatic techniques.

On the other hand, modeling general integrity constraints of objects are proposed in [6,8,24]. In these approaches, integrity constraints are modeled as assertions, invariants in entity classes, or pre- and post-conditions of transactions, and relationships among entities, transactions, and integrity constraints are investigated.

In this paper, an object behavior modeling method developed in [22] is enhanced to treat integrity constraints in the form of assertions. The behavior of objects is defined as a set of states and events which expresses scripts of object life cycles. States and events associated with a certain object(entity) class are also defined as objects and make up a state class and an event class respectively. Though the behavior model by itself reflects an aspect of integrity constraints, general constraints are described in the form of assertions. An assertion class is defined as a set of assertions associated with a certain object class giving an explicit integrity constraints structure. From coherent view of things, both of the behavior model and the assertion class could be encapsulated in the associated object class.

Each of three kinds of classes constructs abstraction structures such as generalization, aggregation, and association. In this behavior model, we concentrate on the aggregation concept and discuss the design method on the behavior model augmented with assertion classes.

2. The Object Behavior Model

2.1 Aggregation of Objects
As a basis of modeling integrity constraints, we use the general notions of objects and objects abstraction mainly concentrating on the aggregation.

An object is a thing that is identifiable. Similar objects are classified into a class and each element of a class is called its instance. Every instance of a class $E$ has a common set of properties the values of which are in turn instances of their component(or domain) classes $E_1, E_2, \ldots, E_n$. An instance of a class $E$ is called an aggregate of the values of its properties which are instances of the component classes $\{E_i\}$, or we say simply $E$ is an aggregate class having component classes $\{E_i\}$ denoting $E = \text{aggregate}(E_1, E_2, \ldots, E_n)$. The aggregate and the component relationships among classes are not restricted to be hierarchical. They may be reflexive (aggregate to each other) or cyclic ($A$ is an aggregate of $B$, $B$ is an aggregate of $C$, and $C$ is in turn an aggregate of $A$).

The grouping (or association) is an abstraction to construct an object that is a set of objects obeying a certain membership rule\[1,13\]. If the membership rule is satisfied, any instances of the same or different classes may become members of a group object.

### 2.2 The Behavior Model of Objects

Every object has its life cycle in a given environment. It comes into existence, changes its state with time, and finally goes out of the environment it lived. We call this dynamic aspects behavior of objects. An object may have different life cycles in different environments.

We use the notation $[a_1, a_2, \ldots, a_k]$ for an ordered set of symbols with $a_i < a_j$ ($i < j$).

Let $E$ be a class. A life cycle of $E$ is an ordered set of symbols called states and is denoted $S_i = [s_{in}, s_{1}, s_{2}, \ldots, s_{n}, s_{fin}]$. Among the states, $s_{init}$ called the initial state and $s_{fin}$ called the final state stand for "(the object) is coming into existence" and "has gone out" respectively. $E$ may have one or more life cycles. A state class of $E$, denoted $SE$, is a union of a finite number of life cycles $S_i$'s of $E$ where each $S_i$ has the initial and final states in common.

For a life cycle $S_i$ of $E$, an ordered set of symbols called events $T_i = [t_{in} = s_{init}, t_2, \ldots, t_{n} = s_{fin}]$ is defined. With each event $t_j$ the states $s_{j-1}$ and $s_j$ are associated as the prestate and the poststate respectively. This fact is denoted $t_j(s_{j-1} \rightarrow s_j)$. Or we can think of the consecutive states $s_{j-1}$ and $s_j$ define the event $t_j$ denoting $s_{j-1} = \text{prestate}(t_j)$ and $s_j = \text{poststate}(t_j)$. In particular, we call $t_{init}$ having $s_{init}$ as the prestate the 'producer' and $t_{fin}$ having $s_{fin}$ as the poststate the 'consumer'. An event class of $E$, denoted $TE$, is defined as a union of $T_i$'s. The behavior model of a class $E$ is defined as a tuple $(SE, TE)$.

Semantically, a state $s$ is interpreted as a milestone in the functional processes in a life cycle of an object, and an event $(s \rightarrow s')$ stands for an action of the state transition from $s$ to $s'$. In this definition, all instances of $E$ have a common life cycle in a certain environment. Namely, every instance of $E$ changes its state obeying the defined order, though each state of $SE$ may be taken by different instances at different time points.

For an arbitrary instance $e$ of $E$, if $e$ is at a state $s$, we then recognize that the object $s$ exists as an instance of $SE$. Similarly, if $e$ is changing its states by an event $t$, we recognize that the object $t$ exists as an instance of $TE$.

The behavior model of objects is expressed in a graphic form using Petri net like graph\[17\] with places and transitions representing states and events respectively. The behavior diagram is a combined diagram of a rectangular box representing a class and the Petri net representing the behavior as will be shown in the later examples.

### 2.3 Structuring State Classes

1. **Grouping and Ungrouping of States and Events**
   
   Let $E$ be a class having a behavior model $(SE, TE)$. We make a group of some states of $SE$ which satisfy a certain membership rule and call it a group state. In particular, $s_{init}$ is defined as a group state consisting of all the states of $SE$ except for $s_{init}$ and $s_{fin}$. Similarly, a group event is defined as a set of some events of $TE$ satisfying a certain membership rule. In particular, for all $i$, we have $t_{init} = t_{fin} = t_{init}(s_{init} \rightarrow s_{fin})$ and $t_{fin} = t_{fin}(s_{init} \rightarrow s_{fin})$.

Conversely, the granularity of states and events are explained using the grouping concept.

1. **Ungrouping of a state**

   A state $s$ may be ungrouped (i.e. refined) into a set of sequences $\{s_i\}$ with $s_i = [s_{init}, s_{1}, s_{2}, \ldots, s_{k(i)}]$.

2. **Ungrouping of an event**

   An event $t$ may be ungrouped into a set of sequences $\{t_i\}$ with $t_i = [t_{init}, t_{2}, \ldots, t_{n(i)}]$. In particular, we define the iterated event denoted $t^*(u \rightarrow v)$ which is equivalent to $t(u \rightarrow v), t(v \rightarrow v), \ldots, t(v \rightarrow v)$ with the repetition of indefinite number of $t(v \rightarrow v)$'s. As suggested by this, the same state can be taken repeatedly by the same object.
Figure 2.1: Ungrouping of States

Through the grouping and the ungrouping of states, the state class \( S^E \) is organized to a grouping hierarchy as illustrated in Figure 2.1. Corresponding to the ungrouping of states, events representing state transitions are also orderly ungrouped into finer granules of events.

[2] Decompositions of Behavior Models

Suppose we have a class \( E \) which is an aggregate of classes \( E_1, E_2, \ldots, \) and \( E_k \). This means that \( E \) has properties \( p_1, p_2, \ldots, p_k \) where each \( p_i \) takes one or more instances of \( E_i \) as its value. In general, in a given environment, the class \( E \) has a certain task to perform a certain function utilizing the component classes \( E_1, E_2, \ldots, E_k \) as its resources or servers. Some examples are 'Personnel Management' which is an aggregate of 'Employee' and 'Project', 'Product Management' which is an aggregate of 'Order' and 'Inventory', and 'Order' which is in turn an aggregate of 'Customer' and 'Product'. In many cases, for one instance of \( E \), the cardinality of values each property \( p_i \) takes is not restricted to one as could be seen in 'Personnel Management' where one personnel manager manages many employees and projects. In such a situation, the behavior model of the aggregate class \( E \) is thought of to be broken down into parts of behavior models of the component classes \( E_1, E_2, \ldots, E_k \). Or conversely, parts of behavior models of the component classes are aggregated to the behavior model of the aggregate class.

The decomposition of behavior models is explained vertically and horizontally as follows.

Let \( E \) be an aggregate set \( E = \text{aggregate}(E_1, E_2, \ldots, E_k) \), and \( t(u \mid v) \) be an event of \( T^E \) with the prestate \( u \) and the poststate \( v \) of \( S^E \). The states \( u \) and \( v \) are naturally considered to be aggregates of some states \( u_1, u_2, \ldots, u_n \) and \( v_1, v_2, \ldots, v_k \) respectively with \( u_i, v_i \in S^{E_i} \). Corresponding to the states, the event \( t \) is also decomposed to the component events \( t_i(u_i \mid v_i), t_2(u_2 \mid v_2), \ldots, t_k(u_k \mid v_k) \) with \( t_i \in T^{E_i} \). Note that, for each \( E_i \), the pare \( u_i \) and \( v_i \) denoted \( u_i \mid v_i \) may be taken by more than one instance of \( E_i \) which participates in the aggregate of an instance of \( E \). We call this decomposition of states and events the vertical decomposition of a behavior model.

On the other hand, the aggregate object set \( E \) plays the role of an 'actor' which requests certain services from its 'servers' \( E_1, E_2, \ldots, E_k \). Namely, an instance of \( E \) as an actor sends messages to appropriate instances of \( E_i \)'s which are values taken through the properties of \( E \). The message sending and receiving mechanism is explained by ungrouping the states and events as follows.

1. To do something, the aggregate event \( t(u \mid v) \) of \( T^E \) and each component event \( t_i(u_i \mid v_i) \) of \( T^{E_i} \) are ungrouped to \( t = [t_1, t_2, \ldots, t_n] \) and \( t_i = [t_i_1, t_i_2, \ldots, t_i n(i)] \) respectively. Corresponding to the event decomposition, the prestates and poststates are also naturally ungrouped.

2. Each event \( t_j \) acts as a message sender to an instance of a certain \( E_i \) to invoke an appropriate event \( t_j(i) \).

Note that the aggregate object of \( E \) is responsible for the scheduling of message sending such as orders, selections, and iterations of invoking the component events. We call this ungrouping of states and events the horizontal decomposition of a behavior model.

Figure 2.2 shows the vertical and horizontal decompositions of the behavior model of an object set \( E = \text{aggregate}(E_1, E_2) \). The asterisks means that each of the properties of \( E \) may take multiple objects of \( E_1 \) and \( E_2 \) as its values.

Figure 2.2: Vertical and Horizontal Decompositions of a Behavior Model
3. Assertion Classes

The expressiveness of objects increases by introducing the concept of assertions. Though the property and the behavior model of objects are by themselves thought of as static and dynamic integrity constraints on objects, the assertion is an explicit means to express the integrity constraints in the form of predicates. In this paper, the integrity constraint refers to the assertion. With each life cycle $S_i$ in the behavior model of a class $E$, we associate a set of assertions $C_i = \{c_{ij}\}$ which every instance of that class should obey in the life cycle $S_i$. The sets of assertions are not necessarily disjoint to each other. We call a union of a finite number of $C_i$'s for all life cycles $S_i$'s of $E$ the assertion class associated with $E$ and denote $C_E$. For an arbitrary instance $e$ of $E$, if an assertion $c$ of $C_E$ is true at a certain state $s$ of $S_E$, then the object $e$ is recognized to exist as an instance of the assertion class $C_E$. In accordance with the grouping of life cycles, the assertions are also grouped.

We quote a slightly modified version of the example of classes from [24,25]. Consider the following environment where three classes Employee (abbreviated $E$), Project (abbreviated $J$), and Personnel Management (abbreviated $PM$) exist.

1. $E$ has properties $e$-name, salary, and works-in. The value of works-in is an instance of $J$.
2. $J$ has properties $j$-name, budget, status, leader, and member.
   Both of the values of the properties leader and member are instances of $E$. (Two classes $E$ and $J$ are the aggregate of each other.)
3. $PM$ is the aggregate of $E$ and $J$, and has properties employees and projects taking multiple number of instances of $E$ and $J$ as values respectively.

Figure 3.1 illustrates the aggregation relationships among three classes.

![Figure 3.1: Aggregation between Classes](image)

The followings are the descriptions on the behavior of objects in the environment together with integrity constraints.

1. An employee is hired with at least the predefined minimum-salary. He is dismissed only if he does not work in any project.
2. A new project is initiated to be active by being assigned a leader. A project may be suspended being left with no leader. No employee may continue to work in a project that is suspended. A suspended project may be permanently terminated, or it may be restarted by being assigned a new leader.
3. The total budget of projects should not exceed a given limit of-budget.
4. An employee is assigned to an active project.
5. The salary of an employee should not exceed that of the leader of the project he works in.

In this environment, the behavior models representing the life cycles of the server classes $E$ and $J$ are defined below.

(B1) The behavior model of $E$ $(S_E, T_E)$
(B1s) The state class $S_E$
   $E$ has two life cycles $S_{SALARY}$ and $S_{JOB}$ standing for the salary and the job histories respectively.
   $S_{SALARY} = \{s_{EINT}, s_{E3} : "initial salary", s_{E2} : "salary changed", s_{EFIN}\}$
   $S_{JOB} = \{s_{EINT}, s_{E3} : "unassigned", s_{E4} : "assignment changed", s_{EFIN}\}$
(B1t) The event class $T_E$
   [producer]
   $t_{EINT}(s_{EEXIST}) : "Hire an employee e giving an initial salary."
   [consumer]
   $t_{EFIN}(s_{EEXIST} \mid s_{EFIN}) : "Dismiss an employee e."
   [modifier]
   $T_{SALARY} = \{t_{E,1} \ast (s_{E1} \mid s_{E2}) : "Change the salary of an employee e."
   $T_{JOB} = \{t_{E,2} \ast (s_{E3} \mid s_{E4}) : "Change assignment of an employee e from a project j(possibly nil) to a project j'(possibly nil),"\}$

(B2) The behavior model of $J$ $(S_J, T_J)$
(B2s) The state class $S_J$
   $J$ has three life cycles $S_{STATUS}$, $S_{LEADER}$, and $S_{MEMBER}$ standing for the status, the leader, and the member histories respectively.
The event class $T^j$:

- **Producer**
  - $t_{\text{INIT}}(s_{\text{INIT}} \land \neg s_{\text{EXIST}})$: "Create a project $j$ with the assignment of the leader and the member."

- **Consumer**
  - $t_{\text{FIN}}(s_{\text{EXIST}} \land s_{\text{FIN}})$: "Terminate a project $j".

- **Modifier**
  - $T_{\text{STATUS}} = \{t_{s2} \times (s_{s1} \land s_{s2}) : \text{"Change the status of a project } j \text{ from 'active' to 'suspended' or vice versa."}\}$
  - $T_{\text{LEADER}} = \{t_{s3} \times (s_{s3} \land s_{s4}) : \text{"Change the leader of a project } j \text{ from an employee } e (\text{possibly nil}) \text{ to an employee } e' (\text{possibly nil})."\}$
  - $T_{\text{MEMBER}} = \{t_{s5} \times (s_{s5} \land s_{s6}) : \text{"Change the member of a project } j \text{ from a set of employees } \{e\} (\text{possibly nil}) \text{ to a set of employees } \{e'\} (\text{possibly nil})."\}$

For more elaborate descriptions, these events should be broken down into more atomic forms like "appoint an employee to a project" and "relieve an employee of a project" of $T^E$, and "make a project active" and "make a project suspended" of $T^J$.

We next describe the integrity constraints in the form of the following assertions making up assertion classes associated with $E$ and $J$ respectively. The notation in the form $<\text{object}> <\text{property}>$ represents the value which the $<\text{property}>$ of $<\text{object}>$ takes.

1. **Assertions on each instance $e$ of $E$**
   - $c_{E1}: e \text{ salary } \geq \text{ minimum-salary}$
   - $c_{E2}: e \text{ works-in } \neq \text{ nil } \rightarrow$
     - $(e \text{ works-in status } = \text{ 'active'} \text{ and } (e \text{ salary } < (e \text{ works-in leader salary})) \text{ or }$
     - $(e \in (e \text{ works-in member}) \text{ or } e = (e \text{ works-in leader}))$
   - $c_{E3}: (e \text{ works-in } = \text{ nil }) \text{ at prestate}(t_{E,\text{FIN}})$

2. **Integrity constraints on each instance $j$ of $J$**
   - $c_{J1}: \text{ The total budget of all projects should not exceed the limit-of-budget, namely}$
     $\sum_{j \in J} (\text{budget}) \leq \text{ limit-of-budget}$
   - $c_{J2}: j \text{ status } = \text{ 'active'} \rightarrow (j = (j \text{ leader works-in})) \text{ and }$
     $(j \text{ member } = \text{ nil }) \text{ and } (j \text{ budget } > 0)$
   - $c_{J3}: j \text{ status } = \text{ 'suspended'} \rightarrow (j \text{ leader } = \text{ nil }) \text{ and } (j \text{ member } = \text{ nil }) \text{ and } (j \text{ budget } = 0)$
   - $c_{J4}: (j \text{ status } = \text{ 'suspended'}) \text{ at prestate}(t_{J,\text{FIN}})$

The assertions listed above are then classified into categories: $C_{\text{SALARY}}, C_{\text{JOB}}, C_{\text{STATUS}}, C_{\text{LEADER}},$ and $C_{\text{MEMBER}}$ which are associated with life cycles: $S_{\text{SALARY}}, S_{\text{JOB}}, S_{\text{STATUS}}, S_{\text{LEADER}},$ and $S_{\text{MEMBER}}$ respectively. From these categories, we define the assertion classes $CE$ and $CJ$ as follows.

(A1) The assertion class $CE$ as a union of $C_{\text{SALARY}}$ and $C_{\text{JOB}}$ with
- $C_{\text{SALARY}} = \{c_{E1}, c_{E2}\}$
- $C_{\text{JOB}} = \{c_{E1}, c_{E3}\}$

(A2) The assertion class $CJ$ as a union of $C_{\text{STATUS}}, C_{\text{LEADER}},$ and $C_{\text{MEMBER}}$ with
- $C_{\text{STATUS}} = \{c_{J1}, c_{J2}, c_{J3}, c_{J4}\}$
- $C_{\text{LEADER}} = \{c_{J1}, c_{J2}, c_{J3}, c_{J4}\}$
- $C_{\text{MEMBER}} = \{c_{J1}, c_{J2}, c_{J3}, c_{J4}\}$
In the assertion class $C_J$, three sets of assertions $C_{\text{STATUS}}$, $C_{\text{LEADER}}$, and $C_{\text{MEMBER}}$ are identical. This suggests that the associated life cycles $S_{\text{STATUS}}$, $S_{\text{LEADER}}$, and $S_{\text{MEMBER}}$ should obey one common control mechanism concerning the validation of integrity constraints. We could therefore group these three life cycles into an aggregate life cycle $S_{\text{PROJECT}}$. The followings are the new version of the behavior model and the assertion class of $J$.

(B2*) The behavior model of $J$ ($S^J$, $T^J$)

(B2s*) The state class $S^J$

$J$ has a life cycle $S_{\text{PROJECT}}$

$S_{\text{PROJECT}} = \{s_{\text{INIT}}, s_{\text{EXIST}} : \text{"initial project"},

s_{\text{EXIST}} : \text{"project changed"}, s_{\text{FIN}}\}$

where $s_{\text{INIT}}$ and $s_{\text{EXIST}}$ are the group states with

$s_{\text{INIT}} = \{s_{\text{INIT}} : \text{"initial active status"}, s_{\text{INIT}} : \text{"initial leader assigned"}, s_{\text{INIT}} : \text{"initial member assigned"}\}$

$s_{\text{EXIST}} = \{s_{\text{EXIST}} : \text{"status changed"}, s_{\text{EXIST}} : \text{"leader changed"}, s_{\text{EXIST}} : \text{"member changed"}\}$

(B2t*) The event class $T^J$

[producer]

t_{\text{INIT}}(s_{\text{INIT}} \mid s_{\text{EXIST}}): \text{"Create a project $j$ with the assignment of the leader and the member."}

[consumer]

t_{\text{EXIST}}(s_{\text{EXIST}} \mid s_{\text{INIT}}): \text{"Terminate a project $j$"}

[modifier]

$T_{\text{PROJECT}} = \{t_{\text{ Chang if e nil} : \text{"Change the status, the leader, and the member of a project $j$"}}$

where $t_{\text{change}}$ is the group event with

$t_{\text{change}} = \{t_{\text{ change status} \mid s_{\text{INIT}}}, t_{\text{ change leader} \mid s_{\text{INIT}}}, t_{\text{ change member} \mid s_{\text{INIT}}} : \text{"Change the status of a project $j$ from 'active' to 'suspended' or vice versa.\"},

\text{"Change the leader of a project $j$ from an employee $e$ (possibly nil) to an employee $e'$ (possibly nil)."},

\text{"Change the member of a project $j$ from a set of employees $\{e\}$ (possibly nil) to a set of employees $\{e'\}$ (possibly nil)."}\}$

(A2*) The assertion class $C_J = C_{\text{PROJECT}}$

$C_{\text{PROJECT}} = \{c_{\text{INIT}}, c_{\text{EXIST}}, c_{\text{FIN}}\}$

The assertions of two assertion classes $C_E$ and $C_J$ are not independent, but they are often related to each other as could be seen in the relationship (e works-in = nil, $c_E$

\begin{align*}
\text{(e works-in status) = 'active'}
\end{align*}

or

\begin{align*}
\text{j status = 'active', } c_J
\end{align*}

\begin{align*}
\text{(j leader works-in) = } j
\end{align*}

The relationships among assertions require the event propagation at the time of invoking a related event.

4. Aggregation of Behavior Models and Assertions

In a specific environment such as the personnel management, events defined in the behavior models of $E$ and $J$ are scheduled by an aggregate object of $PM$. Namely, an instance of the class $PM$ behaves as an actor scheduling the behavior of the component objects. Though we treat here only a simple case, if $E$ and $J$ instead participated in more than one functional environments, their behavior should be scheduled by more than one objects of different aggregate classes that share $E$ and $J$ as component classes.

The behavior modeling method of the aggregate class is generally defined as follows.

Let $E$ be a class having component classes $E_1, E_2, ..., E_n$ such that $E$ performs some tasks utilizing its component classes as resources. We assume that, with each component class $E_i$, the behavior model $(S_{E_i}, T_{E_i})$ and the assertion class $C_{E_i}$ in a refined form is associated.

(1) Design the behavior model $(S^E, T^E)$.

(a) Analyze the tasks of $E$. Corresponding to a given task of $E$, define a life cycle $S_{E_X}$. The life cycle $S_{E_X}$ is vertically decomposed into the related component life cycles $S_{E_X,i}$ of the component state classes $S_{E_i}$ of the component state classes $S_{E_i}s$. $S_{E_X}$ is then horizontally decomposed into a set of life cycles of the form $[s_{\text{INIT}}, s_{\text{EXIST}}, s_{\text{FIN}}]$. The state $s_{\text{exist}}$ is a group state consisting of the state standing for "waiting for a service of a task" and the aggregate state having component states of some $S_{E_X,i}$s. This ungrouping of life cycles may repeatedly applied making up a hierarchical structure.

Define the state class $S^E$ as a set of all the states of $S_{E_X}$.

(b) For each life cycle $S_{E_X}$ and for each state $s_{ak}$ in $S_{E_X}$, define an event $t_{ak}(s_{ak} \mid s_{ak})$ and a set of events $T_{E_X} = \{t_{ak}^*\}$. $T_{E_X}$ represents the set of events performing a task of $E$. Define the event class $T^E$ as a set of all the events of
The behavior model is defined as a tuple \((SE, TE)\).

(2) Design the assertion class \(CE\).

For each life cycle \(S_E\) of \(E\) which corresponds to a given task of \(E\), define the set of assertions \(C_{E_X}\). Each \(C_{E_X}\), in addition to generic assertions of \(E\), includes the aggregate object having some assertions of \(C_{E_X}\)'s as components, where each \(C_{E_X}\) is the associated set of assertions with the component life cycle \(S_{E_X}\) of \(E\). The assertion class \(CE\) is defined as a set of all the assertions of \(C_{E_X}\)'s.

The behavior model of \(PM\) is determined by analyzing the tasks of \(PM\) as the personnel management scheduler. We assume that these tasks are the salary management, the job management, and the project management.

Let \((SPM, TPM)\) be a behavior model of \(PM\). In designing first the state class \(SPM\), we analyze the tasks of \(PM\) and classify them into three categories: the salary, the job, and the project management. From these categories, life cycles \(SPMSALARY, SPMBUS\) and \(SPMPROJECT\) are defined. Each of these life cycles are further ungrouped into granules of the form \([SPMINIT, sa, SPMPIN]\). The state \(s_a\) is a group state consisting of the state standing for "waiting for a service of a given task of \(PM\)" and the aggregate state having component states which are some states of \(SE\) and \(SJ\) concerning the task of \(PM\). The state \(s_a\) is repeatedly brought about as long as the task of \(PM\) exists. These states are categorized as follows.

(1) Salary management with \(SPMSALARY\) having a component \(S_{SALARY}\):

\(s_a1\): "hiring employees"
\(s_a2\): "changing salaries of employees"

(2) Job management with \(SPMBUS\) having components \(SPM\) and \(SPMPROJECT\):

\(s_a3\): "changing project assignments of employees"
\(s_a4\): "dismissing employees"

(3) Project management with \(SPMPROJECT\) having components \(SPM\) and \(SPM\):

\(s_a5\): "creating projects"
\(s_a6\): "changing projects"
\(s_a7\): "terminating projects"

The state \(s_a6\) is in turn ungrouped into \([s_a61, s_a62, s_a63]\)
\(s_a61\): "changing statuses of projects"
\(s_a62\): "changing leaders of projects"
\(s_a63\): "changing members of projects"

We then have the behavior model of \(PM\) in the following descriptions.

(B3) The behavior model of \(PM\): \((SPM, TPM)\)

(B3s) The state class \(SPM\)

\(SPM\) is a set of states of three life cycles \(SPMSALARY, SPMBUS, \) and \(SPMPROJECT\):

\[SPMSALARY = \{[SPMINIT, s_a1, SPMPIN]\}\]
\[SPMBUS = \{[SPMINIT, s_a2, SPMPIN]\}\]
\[SPMPROJECT = \{[SPMINIT, s_a3, SPMPIN], [SPMINIT, s_a4, SPMPIN]\}\]

(B3t) The event class \(TPM\)

[producer]
\(SPMINIT([SPMINIT] SPMLEXIST): "Create a personnel manager.")

[consumer]
\(SPMSALARY([SPMLEXIST] SPMPIN) : "Dismiss a personnel manager.")

[modifier]
\(SPMBUS([SPMLEXIST] SPMPIN) : "Hire an employee e by giving an initial salary",
\(SPMBUS([SPMLEXIST] SPMPIN) : "Change the salary of an employee e."
\(SPMPROJECT([SPMLEXIST] SPMPIN) : "Create a project j assigning an employee e to its leader and a set of employees \{e'\} to its member."
\(SPMPROJECT([SPMLEXIST] SPMPIN) : "Change a project.";
\(SPMPROJECT([SPMLEXIST] SPMPIN) : "Terminate a project j."

The event \(TPM\) is in turn ungrouped into \(TPM1, TPM2, TPM3\).

\(TPM1([SPM1] SPMLEXIST) : "Change the status of a project j from 'active' to 'suspended' or vice versa, together with changing the leader from an employee e to nil or from nil to e, and the member from a set of employees \{e'\} to \{e'\}."
\(TPM2([SPM2] SPMLEXIST) : "Change the leader of a project j from an employee e (possibly nil) to an employee e'(possibly nil)."
\(TPM3([SPM3] SPMLEXIST) : "Change the member of a project j from a set of employees \{e\} (possibly nil) to a set of employees \{e'\} (possibly nil)."
We have sets of assertions $C_{PM/SALARY}$, $C_{PM/JOB}$, and $C_{PM/PROJECT}$ associated with each of the life cycles of PM. Each of these sets includes the aggregate object having component objects which are instances of $C_E$ and $C_J$. As for the aggregate assertions, PM is responsible for the validation of them as an actor which causes state changes of the component classes $E$ and $J$. The assertion class of PM is defined below.

(C3) The assertion class $C_{PM}$

$C_{PM}$ consists of the sets of assertions associated life cycles of PM: $C_{PM/SALARY}$, $C_{PM/JOB}$, and $C_{PM/PROJECT}$.

$C_{PM/SALARY} = \text{aggregate}(cE_1, cE_2)$

$C_{PM/JOB} = \text{aggregate}(cE_2, cE_3)$

$C_{PM/PROJECT} = \text{aggregate}(cE_2, cJ_1, cJ_2, cJ_3)$

The events of $TPM$ is instantiated (i.e., executed) under the assumptions of $C_{PM}$. As the state class $SPM$ has three life cycles $SPM/SALARY$, $SPM/JOB$, and $SPM/PROJECT$ with the associated sets of assertions $C_{PM/SALARY}$, $C_{PM/JOB}$, and $C_{PM/PROJECT}$ respectively, the sets of events $T_{PM/SALARY}$, $T_{PM/JOB}$, and $T_{PM/PROJECT}$ should obey these respective sets of assertions. If any assertions of $C_{PM}$ do not hold, any events of $T_{PM}$ can not be instantiated. Under this assumption, events of $TPM$ is vertically decomposed into events of $T_E$ and $T_J$. The followings are descriptions of event actions in the form of message sending to the component objects.

(1) Actions of $T_{PM/SALARY}$ obeying $C_{PM/SALARY}$

$t_{PM,1}$ : Send a message to $E$ to invoke $t_{E,1}$.$t_{PM,2}$ : Send a message to $e$ to invoke $t_{E,3}$.

(2) Actions of $T_{PM/JOB}$ obeying $C_{PM/JOB}$

$t_{PM,4}$ : Send a message to $e$ to invoke $t_{E,5}$, and send messages to $j$ and $j'$, if any, to invoke $t_{J,3}$ or $t_{J,4}$.

$t_{PM,6}$ : Send a message to $e$ to invoke $t_{E,6}$.

(3) Actions of $T_{PM/PROJECT}$ obeying $C_{PM/PROJECT}$

$t_{PM,7}$ : Send a message to $J$ to invoke $t_{J,7}$, and send messages to $e$ and $(e')$ to invoke $t_{E,2}$.

The action of $t_{PM,6}$ is further ungrouped into actions of $t_{PM,61}$, $t_{PM,62}$, and $t_{PM,63}$.

$t_{PM,61}$ : Send messages to $j$ to invoke $t_{J,2}$, $t_{J,3}$, and $t_{J,4}$, and send messages to $e$ and $(e')$ to invoke $t_{E,2}$.

$t_{PM,62}$ : Send a message to $j$ to invoke $t_{J,3}$, and send messages to $e$ and $(e')$ to invoke $t_{E,2}$.

5. Conclusion

An object behavior modeling method augmented with modeling integrity constraints in the form of assertion classes was discussed. This modeling method is expected to grow to the foundation for general methods for object oriented database design. Three kinds of classes proposed in this paper ought to be equally qualified to culminate to abstraction structures. Namely, abstraction structures of these classes will enable to develop more elaborate design methodologies of objects.

The problem areas that need more investigation are, for example, the following items.

(1) Formalize the more refined structure of behavior models.

(2) Study the abstraction structure of behavior models as well as assertion classes, and investigate relationships between them in detail.

(3) Describe assertions using the temporal logic to treat more sophisticated environments concerning integrity constraints.

References


