Object-oriented Framework for Requirement Specification

^{*}Stephen Liao, ^{*}Jian Ma and [†]Louis Lasoudris

^{*}Department of Information Systems, City Polytechnic of HongKong, HongKong

[†]DSI, Institut National des Telecoms, Every, France

Abstract

Object-oriented (OO) approach has been used in the whole life-cycle of software development. Several methods have been proposed for the requirement specification of an application system. However the emphasis of these methods is not put on the specification of dynamic behaviour of classes. This paper presents an OO framework for the specification of an application system at the stage of system analysis. The framework emphasises the description of roles, events and dynamic constraints of classes. It provides a powerful tool to study the roles played by an object at the different stages of its life.

Keyword Codes: D.2.1; D.1.5.

Keywords: Requirement/Specification; Object-Oriented Approach

1. INTRODUCTION

The concepts of object-orientation were originated from a programming language SIMULA [5]. Recently they are being widely used in the whole life cycle of a software system development [1], [2], [3]. Requirement specification at the stage of system analysis is also influenced by the OO concepts [1] [2]. Current OO methods for system analysis, e.g. OOA[2] and O* [4], make use of the concepts in OO programming languages, where a same concept can be used from the requirement specification to the system implementation. OO methods also provide rich semantics in modeling an application system. However the emphasis of these methods is not put on the specification of dynamic behaviour of classes. Thus they lack the mechanisms in describing the event sequences and role evolution of a class of objects.

This paper proposes an OO framework for the requirement specification at the stage of system analysis. The framework is object-oriented in the sense that it emphasises the concept of object, class of objects and that it uses the perspective concept in the class of objects to find the subclasses or roles of the class. The framework also emphasises the role description and dynamic behaviour of an application system. It provides a clean and an overall picture of the system.

2. OBJECTS AND CLASSES OF OBJECTS

In the paper, objects are treated as a kind of information resource to an organisation. The state of an object changes constantly and the object operations can be triggered by events. Formally an object is defined as:

Object = <Name, {characteristics}, {context}>

where **Name** is an identifier of an object which is unique in an class. It plays a role of identifying objects at the analysis level.

Characteristics are values which determine the states of objects.

Context is composed of real links about an object.

There are many variations for the class concept. Classification seems to be a trivial process in identifying a class. However a set of rules need to be searched in order to find out classes of objects at the stage of the system analysis.

3. PERSPECTIVES IN CLASSES OF OBJECTS

In OO system analysis, an object may play several roles. These roles may be different during an object life, e.g. a person may have several roles such as student, employee, retired employee at the different stage of his or her life, he or she may also play roles as a student and a soccer player at the same time. Perspective is a means to classify the subclasses of classes of objects. The results of classification help to find the roles of the class and study the role evolution. Perspectives are composed of a set of criteria. These criteria refer to structural and behavioural properties of a class. Results of the perspective application are a set of new classes. Every element of the characteristics or the context of a class is a possible criterion, e.g. the class "Person" may be observed by sex perspective:

Perspective_{sex} Person = ({men}, {women}).

There can be non-evaluative link or evaluative link between two classes of objects. The non-evaluative link implies that an object belongs to a class in a perspective p, but does not belong to another class in the same perspective during all its life, e.g.

Perspective_{sex} Person = ({men},{women}), men(t1)->women(t2)

where a person is a man and can never become a woman.

The evaluative link implies that an object, which belongs to a class at time t1, can belong to another class at time t2 in the same perspective, e.g.

Perspective_{profession} Person = ({student},{employee}) student(t1)-->employee(t2)

where a student can become an employee in the future.

If two classes of objects originating from the same perspective are linked by an evaluative link, they represent roles of the original class on which the perspective is applied. In this case, there is at least one criterion whose value is changeable.

If two classes originating from the same perspective are linked by a non-evaluative link, they represent categories of the original class on which perspective is applied. In this case, the values of all criteria are fixed.

4. SPECIFICATION OF CLASSES OF OBJECTS

Perspective application in classes of objects provides a good mechanism to examine the set of objects and to constitute the new coherent set of objects at both structural and behavioural levels. This leads us to examine the description of a class in a refined way at object model level.

In this research, a specification of classes describes a structure shared by all instances of one class with its associated roles. Formally,

Specification of classes = <name, static part, dynamic part, [{role description}]>

Where name is the identifier of classes at the user level. Static part and dynamic part are defined as follows:

Static part = <{attribute}, {attribute constraints}, {static links}, {static link constraints}>

Where **attribute** is to describe attributes of all objects associated with the class. **Attribute constraints** are constraints on attribute values, i.e. domain constraints, values ranges. **Static links** describe the authorized links between classes.

An event class means that a fact will arrive in a class because something happens in the exterior of the class or there is a significant internal state change in the class. Significant state change may also trigger other events.

The consequence of the arrival of events in a class can be triggered by one or a set of other events which are either in the same class or in other classes. The consequence or arrival condition of events are fixed by the event rules. Thus the dynamic part is defined as:

Dynamic part = <{event name}, {event rules}>.

Where event name is associated to one specification of classes. Event rules give the precondition which is necessary for the arrival of an event E_i ; They also give new attribute values after the arrival of an event. In general:

Precondition $\rightarrow$ **Event**

Where an event on a class_i can not happen if the precondition is not met.

If an event will inevitably occur when a precondition is met, this precondition is a *strong* precondition. Otherwise if an event will not absolutely occur when a precondition is met, this precondition is a *weak* precondition.

A precondition is formed by an expression consisting of events or attribute values.

{Attributes(class_i) = value, Event(class_j)} $\rightarrow$ Event(class_i)

The consequence of events in a class consists of modification of attributes in the class and the triggering of events in the same class or in other classes, i.e.,

Event $\rightarrow$ **Consequence**

An event in a class may modify one or several attributes in the same class. Some modifications are not significant for triggering an event, i.e.,

Event(class_i) $\rightarrow$ Change{attributes(class_i)}

An event in class_i may or may not inevitably trigger one or several events in the same or other classes. Similar to the preconditions, an event can have a strong or a weak consequence on another event. Formally,

Event(class_i) $\rightarrow$ {Event(class_j)}

5. LINKS BETWEEN TWO EVENTS

It has been indicated that an event can be conditionally triggered by another event. Now the links between two events with the temporal logic are studied in detail.

If there are occurrences of event e_1 at the time t_1 and of event e_2 at the time t_2 , e_1 and e_2 can occur at the same time or one after another. The links between two events is expressed by one of the abbreviations: NP, SP, AP, NPS, SPS, APS, NF, SF, AF.

The following formula is used to explain the above elements:

Event1(class,[time])&{[condition]}

--Time formula--> [Time operator].Event2(class,[time])

Time operator= {Always A, Sometimes S, Never N}

Time Formula = (Past P, Future F, Present S) which defined the time of occurrence of the right part of the expression.

6. ROLE DESCRIPTION

The concept of role represents an evolution of characteristics and context of classes. The evolution has to be described as the values and the structures of an class.

Role=<role name, static part, dynamic part>

Where static part is defined as:

static part=<{role attributes}, {role attribute constraints},{static links to the role}>

These components are written in the same manner as those of specification of classes. In fact, they belong to the set of characteristics of the specification of classes, but they define the specific characteristics of each different view on the specification of classes. The specific attributes of the roles will enrich the static part of the specification of classes.

dynamic part=<{role event}, {role event rules}>

The concepts of role event and role event rules are exactly the same as those in a specification of classes that we have already introduced. The dynamic part of role is the evolution between roles.

Role is an interesting concept because it allows us to analyze evolution of an object. With the concept of role, an object can create different events and have different reactions facing same events at different stages of evolution.

SUMMARY

This paper presents an OO framework for the requirement specification at the stage of system analysis. Within the framework, a class is the realization of the concepts of data abstraction and encapsulation. A class can be classified into subclasses and roles using the concept "perspective". Subclasses and roles can inherit properties from the original class.

The proposed framework emphasizes the specification of dynamic behaviors of the application system. Some important concepts in the framework, such as perspective, role, event, and dynamic constraint enable the designers to produce a good presentation about the evolution of roles of objects of the application systems

REFERENCES

1. Booch G., "Object-Oriented Analysis and Design", 2nd ed Benjamin /Cummings 1994.
2. Coad P. and Yourdon E., "Object-Oriented Analysis", 2nd ed., Prentice-Hall, 1991.
3. Pernici B., "Requirements specifications for object-oriented systems", INFORSID 90, Eyrolles, June, 1990 pp41-64.
4. Brunet J., "Modeling the world with semantic objects", Proc. of IFPS object-oriented approach in information systems, North Holland, 1991.
5. Dahl O. "Simula" Norwegian Computer Center, Oslo Technical Report, 1965