International Journal of Industrial Engineering & Production Research (2019) 30: 341-352
DOI: 10.22068/ijiepr.30.3.341

RESEARCH PAPER

On Attributes of Objects in Object-Oriented Software Analysis

Hassan Rashidi[1*] & Fereshteh Azadi Parand[2]

Received 24 December 2018; Revised 07 August 2019; Accepted 13 August 2019; Published online 16 September 2019

ABSTRACT
One of the modern paradigms to develop software is object-oriented analysis and design. In this paradigm, there are several objects in the software, and each object plays some specific roles. There is a sequence of activities to develop an analysis model. In the first step, this study works to develop an initial use case model. Then, in the second step, a number of concepts are identified, and a glossary of participating objects is built. Identifying attributes of objects (and classes) is one of the most important steps in the object-oriented paradigm. This paper proposes a method to identify the attributes of objects and verify them. The method is also concerned with classifying and eliminating the incorrect attributes of objects. Then, the method is evaluated through a large application, a Control Command Police System. After that, several guidelines concerning attributes of objects based on the practical experience obtained from the evaluation are provided.

KEYWORDS: *Object-Oriented, Analysis Model, Object, Attributes, Use Case.*

1. Introduction

One of the modern paradigms of developing software is Object-Orientation (OO). In this paradigm, our world is defined using the object categories (classes) or object types (pure abstract class or Java interface) (see [10], [12], [13], [16], [17], [23]). Every class/object in the software plays a specific role, each of which is programmed in Object-Oriented languages such as C++ and Java. A number of attributes (data variables) and services (operations/functions/methods) are assigned to these classes. Then, the behavior of the software is modeled as a sequence of messages that are sent between various objects. In OO models, a number of relationships such as inheritance, association, and aggregation (see [10], [17], [24]) are identified between the classes/objects. Moreover, there are many popular design modeling processes and guidelines such as GRASP [21] and ICONIX [20] for assigning responsibility to classes and objects in an object-oriented design.

Figure 1 depicts a typical sequence of activities to develop an analysis model. In the first step, the users, developers, and client are involved in developing an initial use case model. These activities occur in a tight loop until much of the functionality of the system has been identified as use cases with names and brief descriptions. Then, the developers construct sequence diagrams to identify any missing objects. When all entity objects have been named and briefly described, the analysis model should remain fairly stable as it is refined.

In the second step, a number of concepts are identified, and a glossary of participating objects is built. In this step, we are not really finding out objects. In fact, we are actually finding categories and types (analysis concepts) that will be implemented using classes and pure abstract classes. The developers usually classify the participating objects into entity, boundary, and control objects [4].

The third step is to identify the attributes of objects on which we focus. In this step, we must work on identifying association and nontrivial behavior of objects. The results of the preceding steps include an analysis model for reviewing. (a) This model organizes the data into objects and classes and structure the data via the relationships of inheritance, aggregation, and association; (b) it specifies the local functional behaviors and defines their external interfaces; (c) it captures the control or global behavior; and (d) it captures

* Corresponding author: *Hassan Rashidi*
hrashi@atu.ac.ir

1. *Department of Mathematics and Computer Science, Allameh Tabataba'i University*
2. *Department of Mathematics and Computer Science, Allameh Tabataba'i University.*

constraints (limits and rules). The interaction between objects helps refine the participating objects. We should review the model with experts and, sometimes, the model needs revisions, as shown in the figure.

The main motivation of this paper is to survey the approaches for identifying attributes of objects. The structure of the remaining sections is as follows: In Section 2, the literature review in the steps depicted in Figure 1 is presented. In Section 3, a method to identify and verify the attributes of objects is proposed. In Section 4, the practical experience and guidelines are presented. Finally, Section 5 concludes the paper along with a summary of the main points.

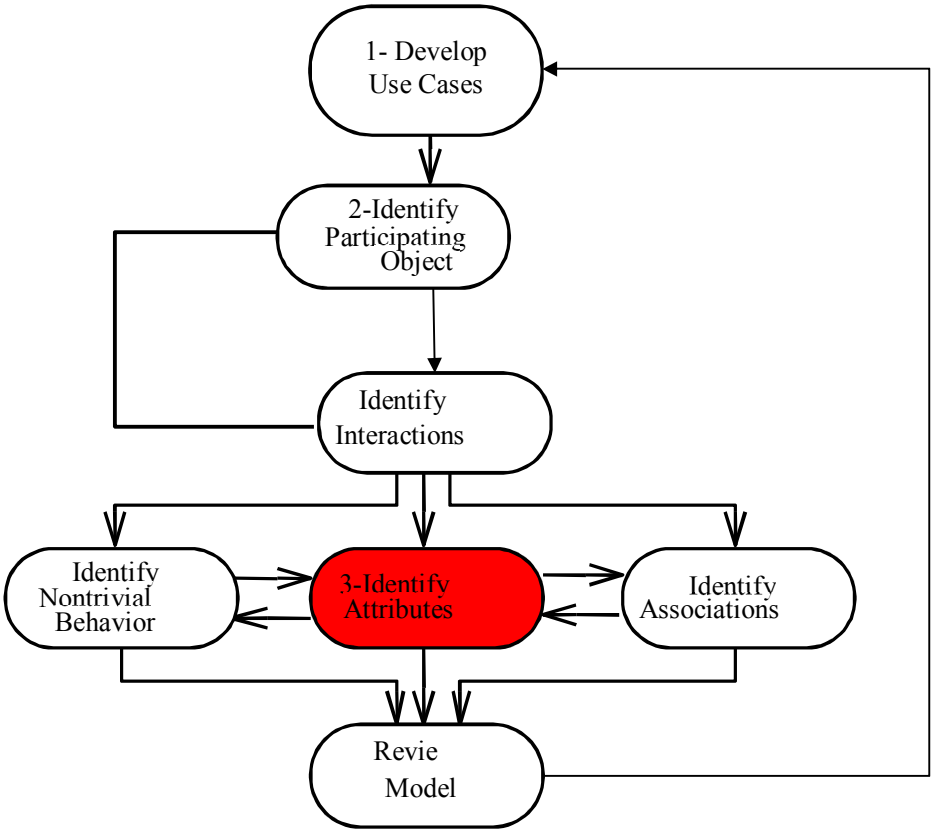

Fig. 1. A typical sequence of the analysis activities

2. Literature Review On the Steps

Identifying use cases is one of the most important steps in software requirement analysis. Rashidi et al. (2017) conducted a literature review on use cases and, then, presented six taxonomies for them [30]. The first taxonomy is based on the level of functionality of a system in a domain. The second taxonomy is based on the primacy of functionality, and the third one relies on the essentialness of functionality of the system. The fourth taxonomy is concerned with supporting functionality. The fifth taxonomy is based on the boundary of functionality, and the sixth one is related to generalization/specialization relation. Then, the use cases are evaluated in a case study in a control command police system. Several guidelines are recommended for developing use cases and their refinement based on some practical experience obtained from the evaluation.

Rashidi (2015) did a literature review of techniques to identify objects [23]. These techniques include (a) *Using Nouns, (b) Using Traditional Data Flow Diagrams, (c) Using object-oriented domain analysis, (d) Reusing an Application Framework, (e) Reusing Class Hierarchies, (f) Reusing Individual Objects and Classes, (g) Using Generalization, (h) Using Subclasses, (i) Using Subassemblies, (j) Using Object Decomposition, (k) Using Personal Experience*, (l) *Using Class-Responsibility-Collaboration Cards, (m) Using the definitions of objects and classes,* and (n) *Using things to be modeled*. This research presented six taxonomies for these techniques. The first taxonomy is based on the documents existing for a domain. The

second taxonomy is concerned with the reusable previous knowledge, and the third one relies on commonalities in a domain. The fourth taxonomy is based on decomposing a domain. The fifth taxonomy is grounded on experience view, and the sixth one is related to using abstraction in a domain. In this research, the constraints, strengths, and weaknesses of the techniques in each taxonomy are described. This research reviewed the techniques to find objects in object-oriented software development and made six taxonomies for them. To get some experience in practice, the techniques were applied to four projects, including two system software products and two applications [22]. The most important findings include the application of a mixture of the techniques and employment of the experts to implement and get the best software products in practice.

After identifying objects, the relationships among objects must be identified. Rashidi (2015) did a review on the relationships among objects in object-oriented development and made five taxonomies for their properties 0. Mainly, the relationships are three basic types (inheritance, association, and aggregation). This research presented five taxonomies for properties of the generalization/specialization, association, and aggregation relationships in the software. The first taxonomy is concerned with a temporal view, and the second one is based on structure. The third taxonomy relies on behavioral view, and the fourth one is specified from a mathematical view. Finally, the fifth taxonomy is related to the interfaces between objects. Moreover, in this research, the relationships are evaluated in a case study and, then, several recommendations are proposed. The main conclusion of this research is that the relationships must capture some concepts that are applied to the problem domain or some sub-domain.

Bavota et al. (2014) proposed an approach for automating the extract class refactoring [1]. This approach analyzes structural and semantic relationships between the methods in each class to identify chains of strongly related methods. The identified method chains are used to define new classes with higher cohesion than the original class while preserving the overall coupling between the new classes and the classes interacting with the original class.

In an application, we must distinguish between the procedural semantics and declarative semantics for its implementation in programming languages. In the procedural semantics, a set of instructions that must be executed sequentially can be written, whereas the declarative semantics specify a set of facts and rules. These semantics do not specify the sequence of steps for doing the processing. Rashidi (2016) presented four taxonomies for the rules in the object-oriented paradigm and discussed how this paradigm could be extended to be used in supporting declarative semantic of applications [25]. Then, the author evaluated the rules in the taxonomies in four case studies. After that, an approach was suggested for the determination and implementation of declarative semantics based on some practical experience obtained from the evaluation.

3. The Proposed Method

To identify the attributes of objects, it is difficult to decide what data are entities and what data are attributes. For example, in an application, should the 'lock' be an entity or attribute? There are arguments for and against each of these choices. The primary criteria for making decisions are whether a choice results in a more precise description and whether a choice unnecessarily constrains design decisions. Fig. 2 proposes a method to determine and verify the attributes of objects. For each object, attributes of objects must be identified. Then, the attributes are classified and the incorrect ones are eliminated. After that, an activity of verification of attributes must be performed. The outcome of this method is an analysis model that is given to the design and implementation phase.

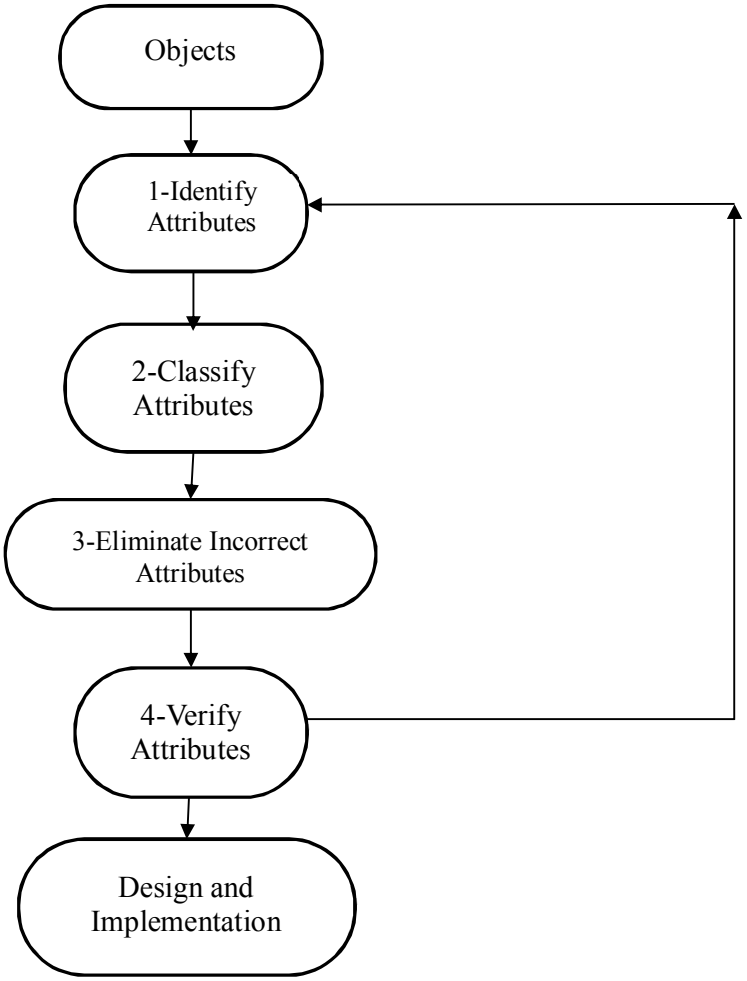

Fig. 2. The proposed method to determine and verify the attributes of objects

Everything in the real world has several characteristics. For example, a person can be described by her or his height: weight, hair color, eye color, and so on. Each characteristic that is common to all instances of the object/ class is abstracted as a separate attribute. For example, Hassan is 6' tall, weighs 175 pounds, and has red hair and brown eyes, while Hussein is 5'10" tall, weighs 160 pounds, and has black hair and green eyes. For a person, potential attributes are height, weight, hair color, and eye color. Note that the characteristics that are abstracted into attributes are highly problem-dependent. Consider the "person" object.

Most of us can come up with a large number of characteristics for a "person" conceptually. When the abstraction of a person is limited to a specific problem domain or to a specific problem, the number of applicable characteristics is reduced. Thus, for the purposes of analysis, an attribute is an abstraction of a single characteristic that is applicable to the business domain and is possessed by all of the entities that were themselves abstracted as objects. From a technical perspective, an attribute is a variable (data item or state information) for which each object (instance) has its own value. Each attribute must be provided with a name that is unique within the object/class. Because each attribute may take on values, the range of legal values allowed for an attribute should also be captured.

3-1. Approaches to identifying attributes

To identify the attributes of objects, the key issue is the following questions: what data do we believe the object is responsible for knowing and owning? The following questions must be asked about each potential object: (1) How is an object described in general? (2) What parts of the general description are applicable to this problem domain? and (3) What is the minimal description needed for this application?

The first approach is to take the **Eastern** or **Taoist approach.** In this approach, analysts take a view on object-oriented analysis such that they will design the system by asking only the first

two questions. Analysts will not be concerned with the specific application that they are implementing. We have found that, with this approach, there is a tendency to produce a more flexible and robust model from a business perspective. Analysts will then be able to respond to changes in the marketplace more quickly. This flexibility is usually at the expense of performance and space utilization. If you are trying to produce flexible and reusable software, you should apply an Eastern or Taoist philosophy to problem-solving and object-oriented modeling.

The second approach is to take the **Western approach.** In this approach, analysts ask all of the three questions and look only at the present application so that they tend to produce a fine-tuned, high-performance system with good space utilization that makes more effective use of the hardware. However, it will be at the expense of having less reusable classes/objects and having less flexibility to respond to the marketplace. In Eastern philosophy, we would not be asking Question (3). We expect that the proper modeling of the problem domain would automatically contain our business solution to our application. This kind of philosophy is based on experimentation versus our classical Western thinking, which is based on planning.

To aid the analyst in determining the attributes of objects, a **Heuristic approach**, to begin with, uses the adjectives and possessive phrases in the requirements document. In particular, a noun phrase followed by a possessive phrase or an adjective phrase should be examined. Then, after identifying a few attributes, the analyst should ask three questions to identify more attributes. In the case of entity objects, any belongings that must be stored by the system can be a candidate attribute, e.g., *red* car, the *40-year-old* man, the *color* of the truck, and the *position* of the cursor. Attributes of objects can be identified using the heuristic rules as follows: ([4], [15], [10]):

- Scan the possessive phrases in the analysis document.
- Represent the stored state as an attribute of the entity object.
- Describe each attribute.
- Do not represent an attribute as an object; use an association instead.
- Do not spend time describing fine details before the object structure is stable.

3-2. Classification of attributes

According to some research on object-oriented analysis, there are usually four types of attributes ([2], [7], [10]): (a) descriptive, (b) naming, (c) state information, and (d) referential:

- **Descriptive Attributes:** Descriptive attributes are facts that are intrinsic to each entity. If the value of a descriptive attribute changes, it only means that some aspects of an entity (instance) have changed. From a problem-domain perspective, it is still the same entity. For example, if Hassan gains one pound, from nearly all problem-domain perspectives, Hassan is still a person. More importantly, Hassan is still the same person as before when he gained one pound.
- **Naming Attributes:** These are used to name or label an entity/object. Typically, they are somewhat arbitrary and frequently used as identifiers or as part of an identifier. If the value of a naming attribute changes, it only means that a new name has been given to the same entity/object. In fact, naming attributes do not have to be unique. For example, if Hassan changes his name to Ali, all that changes; yet, his weight, height, etc. are still the same.
- **State-Information Attributes**: They are used to keep a history of the entity. These are usually needed to capture the states of the finite state machines used to implement the dynamic aspect of the behavior of objects. For example, the attribute 'speed' of a "Car" is used to control the different states of the object. The State-Information attributes are important to build simulation software [26].
- **Referential-Information Attributes**: They are some facts that connect one object to another so as to capture relationships. For example, assume that "Driver" and "Car" object. Then, we can define the attribute 'driver' for "Car" object.

Coad and Yourdon (1991) put it very well when they said [5]: "Make each attribute capture an atomic concept". The atomic concept means that an attribute will contain a single value or a tightly-related grouping of values that some applications treat as a whole. In this respect, attributes of objects are divided into individual attributes and composite attributes:

- **Single-Value Attribute**: They are

individual attributes such as 'age', 'salary', and 'weight' of an object "Person".

- **Group-Values Attribute**: They are composite data items such as legal 'name', 'address', and 'birth date' for an object "Person".

Some attributes of an object are dependent on other attributes. In fact, the attributes can be calculated from other attributes and are generally used to increase the performance of an application. In this aspect, attributes can be classified into the following categories:

- **Basic Attribute**: It is a simple attribute that adheres to an object such as 'birthday', 'salary', and 'weight' of a "Person".
- **Performance Attribute**: It is calculable based on the basic attributes such as 'age' that can be calculated from the current year and the 'birthday' of the object "Person".

3-3. Eliminating incorrect attributes

Attributes are rarely fully described in a requirements document. Fortunately, they seldom affect the basic structure of the model of object-orientation. Analysts must draw upon their knowledge of the application domain and the real world to find them. Because most guidelines for identifying attributes do not help differentiate between incorrect attributes and real attributes, the following suggestions help eliminate incorrect attributes ([8],[9], [15]):

- **Objects:** If the independent existence of the attribute is more important than its value, then the attribute is an object and there needs to be a link to it. For example, consider a "Person" object, an instance of the class person, in an application of Staff Administration. The 'address' or 'city' in which the Person lives is an attribute. In this application, if analysts do not manipulate the 'address' without knowing to which person the address belongs, then it is an attribute. However, if analysts manipulate the 'address' as an entity itself, like Military Service application, then the 'address' or 'city' should be an object with a link between it and Person.
- **Qualifiers:** If the value of an attribute depends on a particular context such as Sport or Police, then we must consider it as a qualifier. For example, the badgeNumber of Player/Police is not really his/her attribute. It really qualifies as a link "plays/works" between the object Player/Police and the object Club/Police-Center.
- **Names:** It is noted that a name is an attribute when it does not depend on the context. For example, a 'person' name is an attribute of "Person". Note that an attribute, as in a person's name, does not have to be unique. However, names are usually qualifiers and not attributes. They usually either define a role in an association or define a subclass or superclass abstraction. For example, 'parent' and 'teacher' are not attributes of the object "Person". Both are probably roles for associations. Another example is a male person and a female person. There are two ways to capture this: we must (a) consider 'gender' as an attribute of the object "Person" or (b) make two subclasses.
- **Identifiers:** We must not prepare a list of the unique identifiers that object-oriented languages need for making an unambiguous reference to an object. This is implicitly assumed to be part of the model; however, the application domain identifiers are listed. For example, in most accounting applications, an 'account code' is an attribute of an object "Account", whereas a transaction identifier is probably not an attribute of the "Account".
- **Link attributes:** If an attribute of an object depends on the presence of a link, then it is an attribute of the link and not of the objects in the link. We must consider the link as an associative object and make the proposed attribute as one of its attributes. For example, assume that Ali is married to Maryam. The date of their marriage is an attribute of the "is_married" association and not an attribute of Ali or Maryam.
- **Discordant:** We must omit minor attributes that do not affect the methods. For example, in the application of student registration in university, the number of brothers/sister of a student must be removed from the list of attributes of the student.

3-4. Verify attributes

During the development of an analysis model, these are some issues concerning normalization and performance. These issues should be left to the design and implementation phase. However, the type of data such as Character, Integer, String, or Boolean should be specified. Its range, constraints, and invariants should also be captured. We recommend capturing constraints and invariants using declarative semantics as [25]. Because verifying attributes is difficult, the studies in the References ([10], [16]) offered several rules to which an attribute must adhere:

- **Rule-1**: An attribute must capture a characteristic that is consistent with the semantic domain in which an object resides. For instance, consider the object "Programmer" in an application of software management. The property of the programmer is the number of years of experience in writing computer programs. However, 'age' is probably not an attribute of "Programmer"; it is probably an attribute of "Person", which is a different object from "Programmer". We know that all programmers are also people, then we can create a "Human Programmer" object by having it inherit the programmer's attributes (e.g., 'years of writing computer programs') from the "Programmer" object and the human attributes (e.g., 'age') from the "Person" object. Thus, it must be noted that a "Human Programmer" is not a composite of two objects.
- **Rule-2**: An instance/object has exactly one value (within its range) for each attribute at any given time. For example, in the application of medical administration, eye color is chosen as an attribute of the Person object with a range of colors such as black, brown, blue, and green. If it is discovered that a person, Mohammad, which should be an instance of Person, has one green eye and one brown eye, then both green and brown cannot be assigned as the eye color of Mohammad.
- **Rule-3**: An attribute must not contain an internal structure. For example, if "name" is considered as an attribute of "Person", then we are not interested in manipulating the given 'name' and the 'family name' independently in the problem domain.
- **Rule-4**: An attribute must be a characteristic of the entire entity and not a characteristic of its composite parts. For example, if "computer" is specified as an object that is composed of a "terminal", "keyboard", "mouse", and "CPU", the size of the screen is an attribute of "monitor" and not "computer".
- **Rule-5**: When an object is an abstraction of a concept that interacts with other objects, the attribute of the object must be associated with the concept and not the other objects. For example, in the application of corn transportation, assume that we want to transfer corn from a "Primary Depot" to a "Second Depot" and define an "Intermediate Truck" for transferring. Then, if the capacity of the "Intermediate Truck" is indicated with the attribute 'number of tons', it must represent the number of transferred tons. It may not be used to represent the number of tons in the "Primary Deport" or in the "Second Depot".
- **Rule-6**: When an object/class has a relationship with another object/class, especially an object/class of the same kind, the attribute must capture the characteristics of the object/class, not the relationship or the other object(s)/class(es) in the relationship. For example, if salary is added as an attribute and spousal relationship to "Person", the spouse's pay cannot be used as the value for the salary attribute of a nonworking spouse, and the date of their marriage is not an attribute of either spouse.

4. Experimental Results

In order to make specific guidelines for developing an analysis model according to which attributes of classes/objects must be identified and verified, this paper used a Control Command Police System (CCPS). For this system, there is a brief description with a mini-requirement in [17]. This system is extended in [14]; then, it is used in our study due to its reusability and fertility in both application and system software. This system must respond as quickly as possible to report incidents. Its objectives are to make sure that all incidents are logged and routed to the most appropriate police vehicle. The most important factors concerning which vehicle to

choose or assign to an incident include the following:

- **Type of incident**: Some critical events need an immediate response. It is suggested that some specific categories of response actions are assigned to a definite type of incident.
- **Location of available vehicles**: Generally, the best strategy is to send the closest vehicle to the accident to address the problem. We must keep in mind that it is not possible to know the exact position of the vehicles and may need to send a message to the car to determine its current location.
- **Type of available vehicles**: Some incidents need vehicles and some special incidents such as traffic accidents may need ambulance and vehicles with specific equipment.
- **Location of incident**: In some specific areas, sending only one vehicle for a response action is enough. In other areas, maybe a police vehicle to respond to the same type of accident is enough.
- **Other emergency services such as firefighter and ambulance**: The system must spontaneously alert the needs to these services.
- **Reporting details**: The system should record details of each incident and make them available for any information required.

The Class Diagram of this system is depicted in Fig. 3. In this diagram, there are many classes. The main classes, here, include 'Incident', 'Police Staff', 'Police Vehicle', 'Police Officer', 'Director', 'Route Manager', 'Incident Waiting List', 'Response', and 'GPS Receiver'.

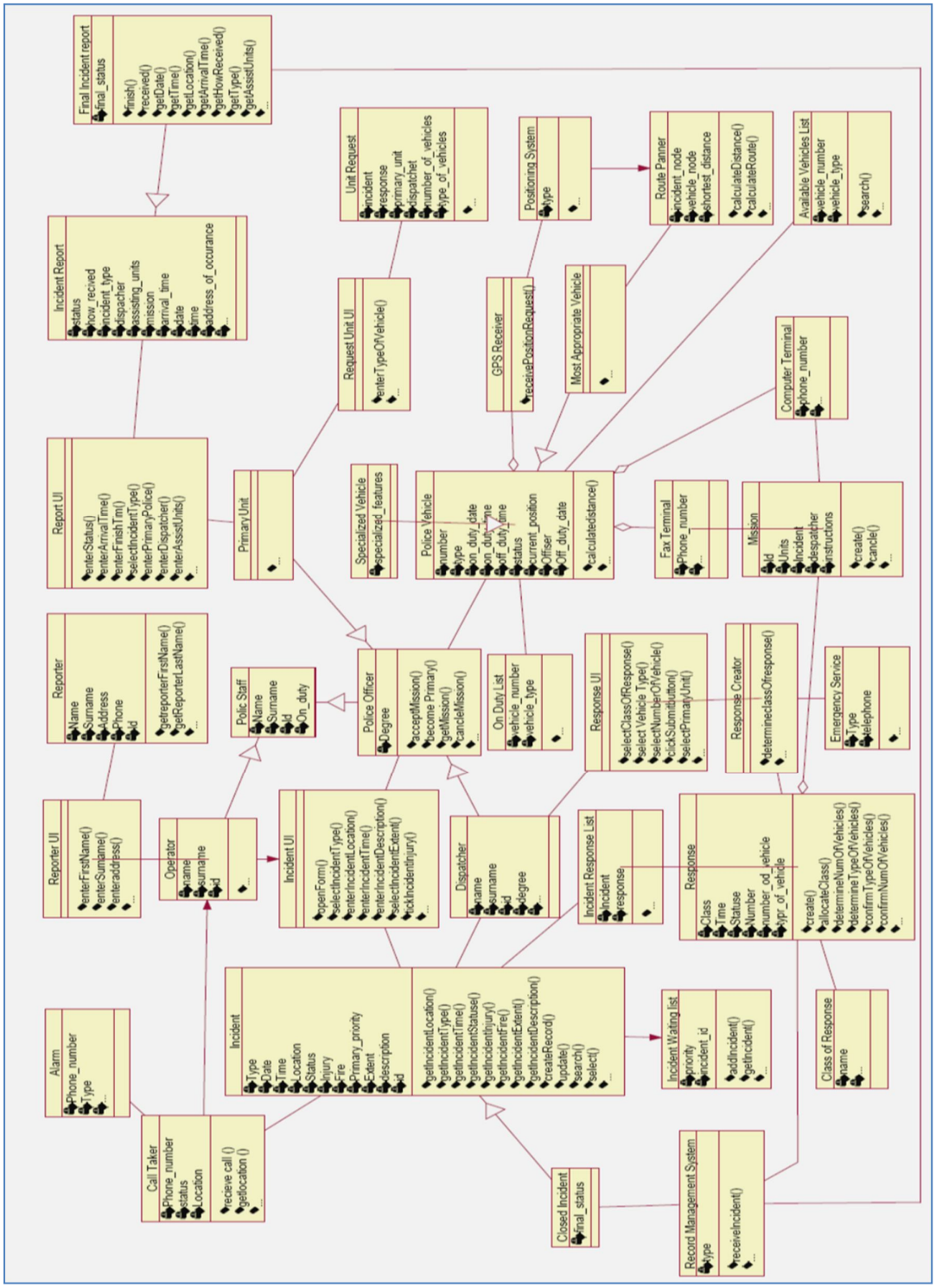

Fig. 3. The class diagram of the control command police system

We applied the steps defined in Section 3 with a view that attributes are properties of individual objects. For example, an "Incident" object, as described in [14], has a 'Type', a 'Location', and a 'Description' property (see Figure 3). These are entered by the "Call Taker" object when she reports an incident and are subsequently tracked by the system. Based on the experiences in our study, some guidelines are presented as follows:

- **Guidline-1:** We should verify that the attribute adheres to all of the rules suggested in Section 3-4 and that a group the attributes is in the same semantic domain to provide a good cohesion in the model. A class should

be coherent and simple. It must represent a concept that operates in a single semantic domain.

- **GuidLine-2**: We should eliminate attributes that are calculable or derivable from the basic attributes. These attributes are usually related to address normalization, object identification or performance of overall supplication, which could be eliminated during the analysis step. We must note that some attributes are related to using the resources efficiently inside the application such as "PoliceVehicle" in the CCPS. The attributes of these resources must be identified during the analysis step.
- **GuidLine-3**: When identifying properties of classes/objects, only the attributes relevant to the system should be considered. For example, each "PoliceStaff" has a social security number that is not relevant to the CCPS. Instead, "PoliceStaff" is identified by 'Name', 'Surname', and 'Id" which are represented as badge number property.
- **GuidLine-4**: When an attribute of an object seems completely unrelated to all other attributes, it may indicate that the object may need to be split into two or more objects. For example, the attribute 'status' in the CCPS must be considered for the "Incident" and "PoliceVehicle" objects.
- **GuidLine-5**: Properties that are represented by objects are not attributes. For example, in the CCPS, every "Incident" has a reporter that is represented by an association to the "CallTaker" class.
- **GuidLine-6**: Analysts should identify as many associations as possible before identifying attributes to avoid confusing attributes and objects. Attributes that related to States-Information and Referential-Information, as described in Section 3-2, are implementation issues.
- **GuidLine-7**: Each attribute must have (a) Name, (b) Type, and (C) Description.
- **GuidLine-8**: When we are identifying an attribute of the object with a specific **Name**, we must not confuse it with other names used in the domain. For example, in the CCPS, the "IncidentReport" has several attributes, a couple of which are called 'Incident_Type' and 'Mission'. The 'Incident_Type' describes the kind of report being filed (e.g., initial report, request for a resource, and final report). The mission describes the type of emergency service (e.g., fire, traffic, etc.) that must be taken care of. To avoid confusion, these attributes should not both be called type.
- **GuidLine-9**: Each attribute must have a **Type** that describes the legal values it can take. For example, in the CCPS, the 'Description' attribute of an "Incident" is a string. The Type attribute of "Incident" is an enumeration that can take one of three values: fire, traffic, etc. Attribute types are based on predefined basic types in UML.
- **GuidLine-10**: Each attribute must have a **Description.** It must not be confused with the other attributes of objects. For example, in the CCPS, the 'Description' attribute of an "Incident" is a String, which must not be confused with the description of each attribute.

5. Summary and Conclusion

This paper reviewed the steps of making an analysis model in object-oriented software development. Additionally, it proposed a method to identify attributes of objects and verify them. The method is also concerned with classifying and eliminating the incorrect attributes of objects. Then, the method was evaluated through an application, Control Command Police System. Several guidelines for finding attributes of objects based on some practical experience obtained from the evaluation were provided. Sometimes, attributes were discovered or added late when a prototype of a system was developed and evaluated by the users. Unless the added attributes were associated with additional functionality in the application, the added attributes did not entail major changes in the object and application structure. The most important point is that a class must be coherent and simple so that it can represent a concept that operates in a single semantic domain. The developers do not need to spend excessive resources in identifying and detailing attributes that represent fewer important aspects of the system. These attributes can be added later after the analysis model, and the user interfaces are validated.

References

[1] Bavota G, Lucia A De, Marcus A, Oliveto R, "Automating extract class refactoring: an improved method and its evaluation", Empir Software Engineering, Vol. 19, (2014), PP. 1616-1664.

[2] Beck K. and Cunningham W., "A laboratory for teaching object oriented thinking", OOPSLA '89 Conference proceedings on Object-oriented programming systems, languages and applications, ACM SIGPLAN Notices, (1989).

[3] Booch, G., J. Rumbaugh, and I. Jacobson. The Unified Software Development Process, Addison- Wesley, (1998).

[4] Bruegge B. and Dutoit A. H., "Object-Oriented Software Engineering: Using UML, Patterns, and Java", Pearson Prentice Hall, (2010).

[5] Coad P. and Yourdon E., "Object-Oriented Analysis", Yourdon Press, (1991).

[6] Deursen A. V, Kuipers T., "Identifying Objects Using Cluster and Concept Analysis", Proc. of 21st International Conference on Software Engineering, Los Angeles, CA, ACM Press, New York, (1999), PP. 246-255.

[7] Fokaefs M., Tsantalis N., Strouliaa E., Chatzigeorgioub A., "Identification and Application of Extract Class Refactoring in Object-Oriented Systems ", Journal of Systems and Software, Vol. 85, (2012), PP. 2241-2260.

[8] Gurp J.V and Bosch J., "Design, Implementation and Evolution of Object-Oriented Frameworks: Concepts and Guidelines", Software-Practice and Experience, Vol. 31, (2001), PP. 277-300.

[9] Langer M., "Analysis and Design of Information Systems", 3rd Edition, Springer-Verlag London Limited, (2008).

[10] Lee R.C and Tepfenhart W.M., "UML and C++: A Practical Guide to Object-Oriented Development", 2nd Edition, Pearson Prentice Hall, (2005).

[11] Martin, J. and J. Odell, *Object-Oriented Analysis and Design,* Prentice-Hall, (1992).

[12] PfleegerS.h., Atlee J.M., "Software Engineering: Theory and Practice", 4th Edition, Pearson, (2010).

[13] Pressman R. S., "Software Engineering: A Practitioner's Approach", 8th Edition, McGraw-Hill, (2014).

[14] Rashidi H., "Software Engineering-A programming approach", 2nd Edition, Allameh Tabataba'i University Press (in Persian), Iran, (2014).

[15] Rumbaugh, J. "Getting Started: Using Use Cases To Capture Requirements", Object-Oriented Programming, Vol. 7, No. 5, (1994), PP. 8-12.

[16] Schlaer, S., and S. Melior, "Object-Oriented Systems Analysis: Modeling the World in Data", Yourdon Press, (1988).

[17] Sommerville Y., "Software Engineering", 10th Edition, Pearson Education, (2018).

[18] Subhash K.S, Bhojane V., Mahajan P., "NLP based Object-Oriented Analysis and Design from Requirement Specification", International Journal of Computer Applications, Vol. 47, No. 21, (2012).

[19] Yourdon, E. N. and Constatine L. L.,"Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design", Prentice-Hall, Englewood Cliffs, New Jersey, (1979).

[20] Rosenberg D., and Stephens M., "Use Case Driven Object Modeling with UML: Theory and Practice", Apress, (2007).

[21] Larman, C., "Applying UML and Patterns – An Introduction to Object-Oriented Analysis and Design and Iterative Development", 3rd Edition, Prentice-Hall, (2005).

[22] Rashidi H., "A Systematic Approach to

Financial Planning in Firms and Its Implementation in an Enterprise", Quarterly Journal of Fiscal and Economic Policies, Vol. 2, No. 8, (2014), PP. 73-92.

[23] Rashidi H., "Objects Identification in Object-Oriented Software Development - A Taxonomy and Survey on Techniques", Journal of Electrical and Computer Engineering Innovations, Vol. 3, No. 2, (2015), pp. 27-43.

[24] Rashidi H., Fundamental Concepts in Computers and Information Technology, Allameh Tabataba'i Press, (2017).

[25] Rashidi H., Declarative Semantics in Object-Oriented Software Development - A Taxonomy and Survey, Journal of Electrical and Computer Engineering Innovations, Vol. 4, No. 1, (2016), PP 57-68.

[26] Rashidi H., Discrete simulation software: a survey on taxonomies, Journal of Simulation, Vol. 11, No. 2, (0217), PP 174–184.

[27] Rashidi H., Tsang E., Vehicle Scheduling in Port Automation, Advanced Algorithms for Minimum Cost Flow Problem (2nd Edition), Taylor and Forensics, (2016).

[28] Asadi M., Rashidi H., A Model for Object-Oriented Software Maintainability Measurement, International Journal of Intelligent Systems and Applications(IJISA),Vol. 1, (2016), PP. 60-66.

[29] Rashidi H., Tsang E. P. K., A Complete and an Incomplete Algorithm for Automated Guided Vehicle Scheduling in Container Terminals, Computer and Mathematics with applications, Vol. 61, No. 3, (2011), PP. 630-641.

[30] Rashidi Z., Rashidi Z., Rashidi H., Use Case Modeling in Software Development: A Survey and Taxonomy, Int. J. Advanced Networking and Applications Vol. 8, No. 5, (2017), PP. 3188-3200.

[31] Rashidi Z., "Properties of Relationships among objects in Object-Oriented Software Design," International Journal of Programming Languages and Applications, Vol. 5, No. 4, (2015), PP. 1-13.

Follow This Article at The Following Site:

Rashidi H, Parand F A. On Attributes of Objects in Object-Oriented Software Analysis. IJIEPR. 2019; 30 (3) :341-352
URL: http://ijiepr.iust.ac.ir/article-1-873-en.html