A taxonomic class modeling methodology for object-oriented analysis

Il-Yeol Song[1], Kurt Yano[1], Juan Trujillo[2] and Sergio Luján Mora[2]
[1]College of Information Science and Technology, Drexel University, USA
[2]Department of Computer Science, University of Alicante, Spain
(song,msis24)@drexel.edu, (jtryjillo, slujan)@dlsi.ua.es

Abstract

An object-oriented analysis requires the identification of domain classes. Discovering a set of domain classes in a problem domain is intellectually challenging and time-consuming for novice analyzers. In this paper, we present a taxonomic class modeling (TCM) methodology that can be used for object-oriented analysis in business applications. In our approach, we synthesize several different class modeling techniques under one framework. Our framework integrates the noun analysis method, class categories, English sentence structures, and other heuristic rules for modeling. Our methodology allows us to discover classes represented by nouns in the requirement specifications, to find classes whose concepts were represented by verb phrases, and to discover hidden classes that were not explicitly stated in the problem statement. We illustrate our approach using a detailed case study. Our experience shows that our method is effective in identifying classes for many object-oriented applications.

1 INTRODUCTION

An object-oriented system decomposes its structure into classes. In object-oriented systems, the notion of the class is carried over from analysis to design, implementation, and testing. Thus, finding a set of domain classes is the most important skill in developing an object-oriented system. However, finding classes is a discovery process [Booch93]. Discovering a set of domain classes in a problem domain is intellectually challenging and time-consuming for novice analyzers. We need systematic methods and guidelines to discover classes.

A class is an abstraction of meaningful real-world objects. A class is a description of objects that share the same attributes, exhibit the same behaviors, and are constrained by the same rules [Starr01]. Classes are organized into a class diagram.

A class diagram can be developed at different levels of abstraction. Classes can be *domain (or analysis) classes, design classes,* or *implementation classes.* Domain classes represent important business activities at the analysis level such as Customer or Account. Domain classes are enduring classes regardless of the functionality required today [SP99]. In this paper, we focus on identifying domain classes that capture fundamental business activities at the analysis level.

In order to discover classes for a problem domain, we have to examine various sources and documentation, and apply various techniques to those specifications. We frequently begin to identify classes from the problem statement or use case descriptions. Rosenberg [Rose99] states that the best sources of classes are the high-level problem statement, lower-level requirements, and expert knowledge of the problem space.

Blaha and Premerlani [BP98] recommend that we always begin analysis with a written problem statement. Thus, in this paper, we assume the modeler has a specification in the form of a problem statement or a use case description in a written form. The statement, written in English, usually defines goals, scope, important functional requirements, and some non-functional requirements of the domain. The problem statement, however, does not give us a complete list of classes necessary for an object-oriented analysis. Nevertheless, beginning with the problem statement is the easiest method for modeling classes for a draft of a class model. Identifying classes from a written source, however, has at least three major limitations as follows [PW97, Rich99, Maci01]:

- Natural language is ambiguous. Thus, rigorous and precise analysis is very difficult, and we need techniques and guidelines for modeling.
- The same semantics could be represented in different ways. Thus, a way of handling this style variation is necessary.
- Concepts that were not explicitly expressed in a written source are often very difficult to model. Thus, we need expert domain knowledge to identify the hidden classes.

The methodology we present in this paper will address all three limitations stated above.

There are several approaches for identifying classes. Our survey shows that noun analysis is the most popular approach [Abbo83, Chen83, WWW90, RBPE91, HKK93, Booc94, HL96, SP99, Rich99, Rose99, Maci01]. Other methods used are the use of class categories as tips [Rose88, Booc94, RJB99, Starr01, Larm01], the use of use case descriptions [Jaco93, Rich99, DE00], and CRC (Class-Responsibilities-Collaborators) cards [WWW90, Wilk94]. Literature and experiences show that no single approach works best. Ideally, several approaches can be used together for a domain.

These approaches have been used either separately or together without any specific guidelines under a single framework. In this paper, we present an integrated class modeling methodology that can be used for object-oriented analysis in business applications. Our framework integrates the noun analysis method, class categories, English sentence structure, and other modeling heuristics. Our methodology allows us to identify classes represented by nouns in the requirement specifications, to find classes whose concepts were represented by verb phrases, and to discover hidden classes that were not explicitly stated in the problem statement. We illustrate our approach using a case study and summarize the results from seven other case studies. Our students have found that our TCM methodology is practical and could be easily and effectively applied to their project domains.

The rest of this paper is organized as follows: Section 2 presents an overview of our TCM methodology. Section 3 presents the detail of the taxonomic class modeling methodology. Section 4 presents a case study based on our methodology. Section 5 concludes our paper.

2 AN OVERVIEW OF TAXONOMIC CLASS MODELING METHODOLOGY AND CLASS CATEGORIES

2.1 An Overview of TCM Methodology

The primary purpose of TCM method is to create an integrated methodology that integrates many existing modeling techniques. TCM incorporates the noun analysis, class categories, English sentence structure rules, checklists, and other heuristic rules for modeling.

Domain classes in TCM consist of three types of classes: Noun classes, transformed classes, and discovered classes. *Noun classes* are those that are identified from noun phrases of requirement specification. *Transformed classes* are those that are identified from verb phrases of requirement specification and transformed into classes by heuristics. *Discovered classes* are those that have not been explicitly stated in the requirement specification but discovered by applying domain knowledge to class categories.

2.2 Class Categories

A widely-used method for class modeling is to use categories of classes [Ross88, SM88, CY91, CAB94, RJB99, Star01, Larm01]. Modelers apply domain expertise to those categories to create classes. Many authors have compiled those lists. See Table 1 for a comparison of class categories used by different authors. In the rightmost column of Table 1, we show the class categories we have adopted in our TCM methodology. We note the following in the use of class categories: (1)They are not mutually exclusive. (2)They are dependent on domains. (3) We use class categories as a tip for identifying classes, not as an absolute list. We believe our chosen 14 class categories subsume most of categories used by other authors. We explain our class categories with examples in Section 3.2.2 in detail.

Table 1. Class Categories used by other authors and our TCM method

	[Ross 88]	[Rich99]; [SM8	[Star01]	[Bahr99]	[CD91]	[Larm01]	**TCM**
Roles of People	X	X	X	X	X	X	X
Places (Locations)	X			X	X	X	X
Physical Things	X	X	X		X	X	X
Organizations	X			X	X	X	X
Events (Incident) / Transactions	X	X	X	X	X	X	X
Transaction Line Item						X	X
Concepts (Discovered Class; Intan things)	X		X	X		X	X
Specifications		X	X			X	X
Interactions		X	X				X
Rules / Policies					X	X	X
Invented Class			X				
Simulated Class			X				
Structure					X		
Other (External) Systems					X	X	
Device					X		
Containers of other things						X	X
Things in a container						X	X
Financial instruments and services						X	X
Look up (References)							X

3 THE TAXONOMIC CLASS MODELING METHODOLOGY

Section 3.1 explains the workflow of our methodology. Section 3.2 present the rules used in the methodology.

3.1 The Workflows of Our Methodology

The actual step-by step activities of our methodology are outlined in Figure 1 in the form of an activity diagram in the UML. In Figure 1, the three swimlanes have their own goals and perform the following activities:

- **The middle swimlane**: *The goal of these swimlane activities is to identify classes from the concepts that were explicitly stated as noun phrases in the problem statement.* We call the classes found using this method ***noun-classes.***

- **The rightmost swimlane**: *The goal of this swimlane is to identify classes that were stated as a verb phrase in the problem statement*. Thus, this swimlane deals with style variation of the written problem statement. We call these association-converted-classes ***transformed class*****es.**

- **The leftmost swimlane**: *The goal of this swimlane is to discover hidden classes that were not explicitly stated in the problem domain but are necessary for the domain modeling*. Note that this swimlane does not directly use any part of the problem statement. We discover those hidden classes by applying domain knowledge to class categories. We call these classes ***discovered-classes.***

Activities of Middle Swimlane of Figure 2

- Begin with the problem statement
- **Step N1**: Pick up noun phrases
  - We will classify them into *problem-description nouns* (PDN) and *problem-solving nouns* (PSN). PSNs are those nouns that become classes, attributes, or values.

- **Step N2**: Test *Class Elimination Rules*
  - If a noun phrase satisfies one of the Class Elimination Rules shown in Section 3.2.1, then it is not a class. Eliminate it from the candidate of a class.

- **Step N3**: Apply *Class Category Rules*
  - If a noun phrase represents a category from the *Class Categories* shown in Section 3.2.2, then the noun represents a class. The selected class is called a *NOUN-CLASS.* If the noun does not belong to an existing category, the modeler should carefully analyze and decide whether or not to keep it as a class.

- **Step N4**: Apply English Sentence Structure Rule and other heuristics. Classify non-class nouns into attributes or values. If a noun represents a class, an attribute, or a value of an attribute, then the noun is a PSN. Otherwise, it is a PDN.

Activities of Rightmost Swimlane of Figure 2

- **Step V1**: Pick up verb phrases (noun-verb-noun) and prepositional phrases (noun-preposition-noun) as a candidate for an association.
  - We will classify them into *problem-description verbs* (PDV) and *problem-solving verbs* (PSV). A PSV is a verb whose concept could be represented as an association in the class diagram. A PDV is a verb that was used in describing the context of the problem and not modeled as an association in the class diagram.

- **Step V2**: Apply Noun-Class Verb Rule
  - If one of two nouns surrounding the verb or the preposition is not a class, then the verb is a PDV. Eliminate the verb phrase.

- **Step V3**: Apply Verb Elimination Rules
  - If the verb is in the list of the Verb Elimination Rules, then the verb is a PDV. Eliminate the verb phrase.

- **Step V4**: Apply Need-to-Know Rule
  - If the verb phrase represents an association that does not have to be remembered between two classes, the verb phrase represents a comprehension association. The verb is a PDV. Eliminate it.

- **Step V5**: Apply Identification Rule
  - If the concept represented by the verb phrase needs to have a unique identifier, then the verb phrase needs to be transformed into a class. We call such a class a TRANSFORMED-CLASS.
  - Otherwise, we adopt the verb phrase as an association between the two classes.

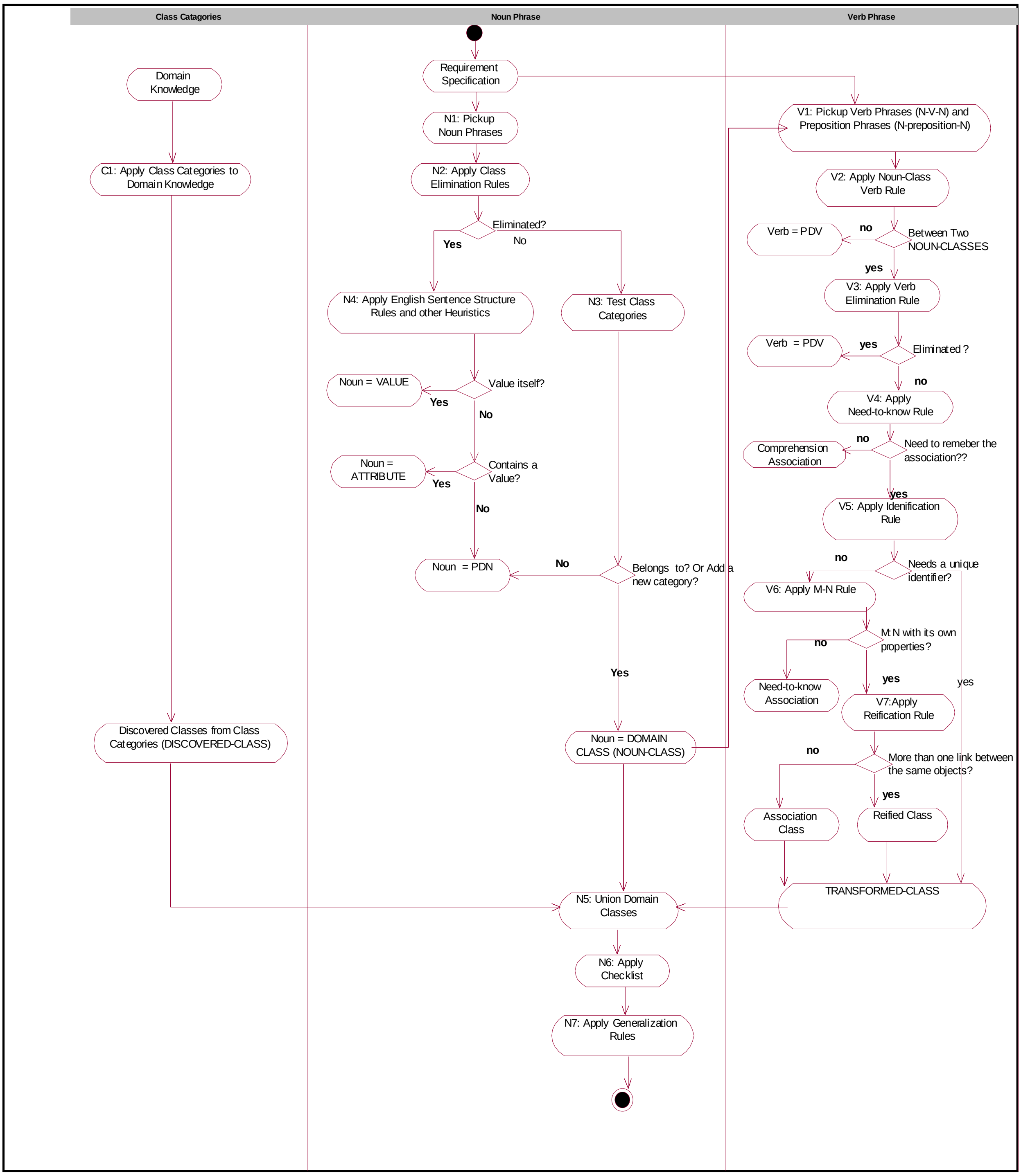

Figure 2. Activities of the taxonomic class modeling methodology in an activity diagram. {Domain classes} = {DISCOVERED-CLASS} ∪ {NOUN-CLASS} ∪{TRANSFORMED CLASS}

(PDN= Problem-Description Noun; PSN= Problem-Solving Noun;
PDV= Problem-Description Verb; PSV= Problem-Solving Verb)

- **Step V6**: Apply M:N Rule
  - If an association has many-to-many multiplicity between two classes and the association has its own properties or constraints, model it as an association class. We call such a class a TRANSFORMED-CLASS.

- **Step V7**: Apply Reification Rule
  - If the association class could have more than one link between the same object instance, then model the association as a class known as a reified class. We also call such a class a TRANSFORMED-CLASS.

Activities of Leftmost Swimlane of Figure 2

- **Step C1**: Apply domain knowledge to Class Categories
  - For each class category discussed in Section 3.2.2, check whether all the classes representing the class category have been already captured. Otherwise create a new class based on the class category. We call the newly added class a DISCOVERED-CLASS.

A set of domain classes identified from our methodology is a union of the classes identified from the three swimlanes (Step N5). That is: {Domain classes} = {DISCOVERED-CLASS} ∪ {NOUN-CLASS} ∪{TRANSFORMED CLASS}

The domain classes are compared against the checklist for a final sanity checking in Step N6.

3.2 Rules Used in the Taxonomic Class Modeling Methodology

In this section, we present various rules used in our methodology.

3.2.1 Applying the Class Elimination Rule (Step N2)

The Class Elimination Rules are used in Step N2 for each noun phrase selected from the problem statement. Rumbaugh et al. [RBP91] popularized seven Class Elimination Rules (CER). These rules and variations were subsequently used by many other authors [Derr85, Rich99, BP98, SP99]. In our methodology, we have adopted CER1-CER5, and CER7 from Rumbaugh et al. [RBP91], CER6 from Stevens and Pooley [SP99], and CER9 from Blaha and Premerlani [BP98]. We have added CER8 and heuristic rules R1-R3 based on our own experience [SF95].

- **CER1: Redundant classes**. Two nouns represent the same abstraction. We keep the more descriptive noun. For example, we use *customer*, instead of *user* in ATM domain.
- **CER2: Irrelevant classes**. The nouns have nothing to do with the problem to be solved. The noun is beyond the scope of the problem being modeled. For example, in a video rental domain, the *occupations of the customers* are irrelevant when we focus on rental transactions.
- **CER3: Vague classes**. The nouns have ill-defined or too broad scope. For example, *business activities* are vague in most domains.
- **CER4: Operations**. The nouns represent operations. For example, ROI (Return-on-Investment) is an operation [BP98].
- **CER5: Implementation constructs**. The nouns represent an implementation-related class such as *set*, *string*, or *algorithm*. These implementation classes can be added at the design or implementation stages, but not at the conceptual level.
- **CER6: Meta-language**. The noun is used to describe and explain requirements and the system at a very high level. Examples are *systems*, *information*, or *reporting requirements*.
- **CER7: Attributes**. The nouns represent a text or a number. For example, *name*, *age*, and *phone number* represent attributes that carry a value. There are often delicate cases where it seems uncertain whether a noun should be modeled as an attribute or a class. In those cases, we use the following rules in determining whether a noun represents an attribute or a class as follows:

  /* R1: The Rule of One-Property */
  IF a noun has only one property to remember **THEN** it is an attribute of another class **ELSE** it is a class.
  /* R2: The Rule of Dependence */
  IF the identification of an object (noun) relies on another concept object (noun) **THEN** it is an attribute.
  /* R3: The Rule of Independence */
  IF the noun represents an object which is important in its own right **THEN** it is a class.

- **CER8:Values**. The nouns represent a value itself. For example, in "an account will be put on *hold state* if the balance is unpaid for more than 100 days," the noun phrase "hold state" represents a value of another attribute, possibly, account-status.
- **CER9: Derived classes**. The concepts can be derived from other domain classes. The decision to include a derived class in the analysis model should be deferred until the design stage. However, we add a derived class in the data dictionary. Derived classes do not add new information, but they could be useful in real-world and in design

We note that these class elimination rules may not be mutually exclusive. They are used to admonish the modeler to include only meaningful classes at the analysis level.

/* R4: The Class Elimination Rule */
IF the noun candidate belongs to one of the nine CER rules **THEN** it is not a class.

Any noun phrases that pass these nine rules are candidates for classes.

3.2.2 Applying the Class Category Rule (Step N3)

In the previous section, the Class Elimination Rules were used to reject bad classes. In this section, we apply class categories to select good domain classes. Various class categories were discussed and presented in Section 2.2, and were summarized in Table 1. Our class categories were inspired by Larman [Larm01], but were modified based on our own teaching experience in business applications. They are as follows:

- **CC1: Roles of People.** They represent humans who carry out some important function. Examples are *Student, Employee*, and *Customer*.
- **CC2: Places.** They represent locations where important business activities are held. Examples are *Office, Warehouse*, and *Store*.
- **CC3: Physical Things.** They represent tangible objects that are important in business activities. Examples are *Machine, Product, Device,* and *Book*.
- **CC4: Organizations.** They represent important business units. Examples are *Company, Team*, and *Department*.
- **CC5: Events (Transactions).** They represent important activities that need to record some data with the time the event occurred. Examples are *Order, Promotion*, and *Payment*.
- **CC6: Transaction Line Items.** They represent an element of a transaction. Examples are *Order-Line-Item, Purchase-Line-Item*, and *Rental-Line-Item*.
- **CC7: Concepts (Discovered Class; Intangible Things).** They represent intangible ideas used to keep track of business activities. Examples are *Project, Account*, and *Complaint*
- **CC8: Specification.** They represent a description of other items that need to be distinguished from one another. Examples are *Video-Title*[1] or *Flight-Plan*. For example, a movie called Harry Potter is a title, but a store may have many tape instances, where each tape has a different barcode.
- **CC9: Interaction.** They represent an association between two classes, where the association has meaningful attributes. An example of this class is *Reservation* between Passenger and Flight classes.
- **CC10: Rules/Policies.** They represent important business rules. Examples are *Rental-Policy* and *ShippingMethod*. The *Rule* here does not mean if-then-else logic. Instead, it represents a business rule that can be broken down into several attributes. For example, a rental policy may state rental charges and rental durations.
- **CC11: Containers of other things.** They represent classes that will contain other classes. Examples are *Store, Shelf, Catalog,* and *Bin*.
- **CC12: Things in a container.** They represent classes that will be contained in another class. Examples are *Item, Passenger,* and *Video-Title* in a catalog.
- **CC13: Financial Instruments and Services.** They represent class that are used to support financial activities Examples are *Stock, Bond,* and *Mortgage.*
- **CC14: Lookup/References.** They represent a single class that is used for referring to a list of predefined items. Examples are *Airport codes* and *Accounting codes*.

We note that our class categories are neither mutually exclusive nor closed for all domains[2]. We use these categories as a thinking tip to identify classes.

/* R5: The Class Category Rule*/
IF the candidate noun which passed the Class Elimination Rules belongs to one of the fourteen class categories
THEN it is a domain class and we call it a NOUN-CLASS
ELSE use domain knowledge to decide whether to keep the class.

For those nouns that do not belong to a category, we caution modelers to carefully analyze the domain and decide whether or not to keep the class. We call the classes passed from the Class Elimination Rule and the Class Category Rule *Noun-classes*.

---

[1] Note here Video Title is not just one attribute that stores a title of a tape. Instead, it is a class that keeps track of title, actors, release year, running time, etc. In a video store, one title may have many video tapes.
[2] For example, we have not tested our class categories on domains such as CAD/CAM or GIS.

3.2.3 Identifying Attributes and Values (Step N4)

In our middle swimlane, we also identify attributes and important attribute values that were mentioned in the Problem statement. In order to identify attributes of a class, we use two techniques. One technique is to use CER7. Another technique is to use Rules 3, 6, 7, and 8 of Chen's English Sentence Structure rules [Chen83]. We do not reproduce those rules due to the lack of space. We use CER8 to identify important values of attributes. These artifacts are recorded in the data dictionary.

3.2.4 Verb Phrases and Noun-Preposition-Noun Phrase (Step V1)

Rumbaugh et al. [RBJ91] use verb phrases to identify associations. Blaha and Premerlani [BP98] use both verb phrases and preposition phrases, in the form *of noun-preposition-noun*, to identify associations. We call both of them simply verb-phrases for convenience.

3.2.5 Applying the Noun-Class Verb Rule (Step V2)

For each verb phrase, we apply the problem-solving verb rule as follows:

/* R6: The Noun-Class Verb Rule */
IF one of two nouns surrounding the verb or the preposition is not a NOUN-CLASS
THEN the verb is a problem-description verb. Eliminate it.

The verb phrases that satisfy R6 represent associations that do not have to be kept track of.

3.2.6 Applying the Verb Elimination Rule (Step V3)

Rumbaugh et al. [RBP91, BP98] present six verb elimination rules. We have named one of them as the Noun-Class Verb rule in the previous section since it is very important in our methodology. We adopted four Verb Elimination Rules from [RBP91, BP98] as follows:

- **VER1: Irrelevant Associations**. Eliminate the verbs that represent associations beyond the scope of the problem domain.
- **VER2: Implementation Associations**. Eliminate the verbs that deal with implementation constructs.
- **VER3: Actions**. Eliminate the verbs that represent transient actions, as in *ATM prints receipts.* They can be represented in interaction or activity diagrams, but not in class diagrams.
- **VER4: Derived Association**. Eliminate the verbs that clearly represent derived associations. As in the case of derived classes, we document derived associations in the data dictionary since they could be important during design.

/* R7: The Verb Elimination Rule */
IF the verb candidate belongs to one of the four verb elimination rules
THEN the verb is a problem-description verb. Eliminate it.

3.2.7 Applying the Need-to-Know Rule (Step V4)

We keep only need-to-know associations as follows:

/* R8: The Need-to-Know Association Rule */
IF the verb represents a persistent relationship that needs to be remembered for a certain duration of time
THEN the verb is represented as a need-to-know association (Problem-Solving-Verb)
ELSE the verb represents a comprehension association and is removed (Problem-Description-Verb).

3.2.8 Applying the Identification Rule (Step V5)

Because of style variations, a concept representing a class is often represented by a verb phrase, instead of a noun phrase. In this case, the noun analysis method does not identify this kind of class. Thus, for each verb selected so far, we need to apply the Verb Identification Rule to see whether we need to transform a verb into a class as follows:

/* R9: The Identification Rule */
IF the concept represented by a verb (or a noun) requires a unique identifier **THEN** model it as a class

We call the class converted from a verb, together with association class and reified class discussed in the next section, a TRANSFORMED-CLASS. We skip the rules for the M:N rules and reification rules for the lack of space.

3.2.10 Applying the Class Categories to Domain Knowledge (Step C1)

Since a problem statement is a short description of a domain by its nature, there may be omissions of functional requirements. Therefore, there may be hidden classes caused by these omissions. In this section, we discuss a way of mitigating the problem. We apply class categories discussed in Section 3.2.3 to discover hidden classes as follows:

For each class category, use domain knowledge to discover any class belonging to the class category. We call the classes identified via this method DISCOVERED-CLASS.

3.2.11 Union of Domain Classes (Step N5)

We have identified three types of classes – noun-classes, transformed-classes, and discovered-classes. The union of these three types of classes forms our domain classes and will be represented in our class diagram. In Step N6, various check lists are applied. They include: need-to-know; multiple attributes; always-applicable attributes; always-applicable operations; not merely derived results; a single-personality definition, and a single-sentence definition.

4 A CASE STUDY

To illustrate our TCM methodology, we elaborate a case study on a video rental store. The problem statement with all the nouns highlighted is shown in Figure 2. Table 2 in Appendix shows our analysis by applying the Class Elimination Rules and Class Categories. The adopted classes (Noun-classes) are check-marked in the right-most column.

From the verb analysis, we have not found any interesting transformed-class candidates, which would be a verb that needs to have a unique identifier. However, applying Step C1, we found the three new classes (Discovered-Classes) from the list of our fourteen categories as follows:

Category 1 People: Staff
Category 2 Location: Shelf
Category 10 Rules/Policies: LoanPolicy.

These three classes were not explicitly mentioned in the problem statement. A simple noun analysis method could never find these classes. The final adopted class diagram, which captures all the classes, attributes and important values mentioned in the problem statement, is shown in Figure 3. Using other documentation and expert knowledge, more attributes will be added to this first-cut domain model.

This problem is about a small, local **video rental store** (**VRS**). The **problem** will be limited to rental, return, management of **inventory** (add/delete new **tapes**, change **rental prices**, etc.) and producing **reports** summarizing various **business activities**. The rental **items** of the **store** are limited to **video tapes**. **Customer ID number** (arbitrary **number**), **phone number** or the combination of **first name** and **last name** are entered to identify **customer data** and create an **order**. The **bar code ID** for each **item** is entered and **video information** from **inventory** is displayed. The **video inventory file** is decreased by one when an **item** is checked out. When all **tape IDs** are entered, the **system** computes the **total rental fee**, and **payments** are processed. A **return** is processed by reading the **bar code** of returned **tapes**. Any outstanding **video rentals** are displayed with the **amount** due on each **tape** and a **total amount due**. The **past-due amount** must be reduced to zero when new **tapes** are taken out. For new **customers**, the unique **customer ID** is generated and the **customer information** is entered into the **system**. **Videos** are stacked by their **category** such as **Drama, Comedy, Action**, etc. Any **conflict** between a **customer** and **computer data** is resolved by the **store manager**. **Rental fees** can be paid by either **cash, check** or a major **credit card**. **Reporting requirements** include viewing **customer rental history**, **video rental history**, and **titles** by **category**, **top ten rentals**, and **items** by **status**, and **overdue videos** by **customers** and **outstanding balances** by **customers**.

Figure 2. The Problem Statement of Video Rental Store with Nouns highlighted.

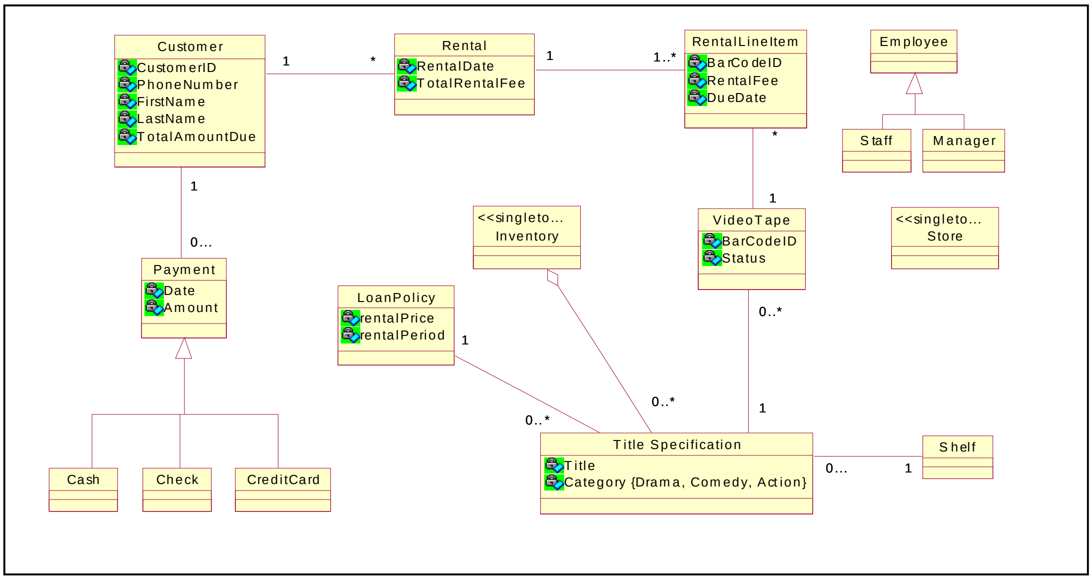

Figure 3. The class diagram built based on our methodology

5 CONCLUSION

In this paper, we presented a Taxonomic Class Modeling (TCM) methodology that can be used for object-oriented analysis in business applications. In our methodology, we synthesized several different class modeling techniques under one framework. Our framework integrates the noun analysis method, class categories, English sentence structures, and other heuristic rules for modeling.

Our methodology allows us to identify classes explicitly represented by nouns in the requirement specifications, to find classes whose concepts were represented by verb phrases, and to discover hidden classes that were not explicitly stated in the problem statement. We illustrated our methodology using a case study. Our teaching experience shows that our method is effective in identifying domain classes for many business-oriented object-oriented applications. Our students have found that our TCM methodology is practical and can be easily and effectively applied to their project domains.

Appendix A:

Table 2: The Result of Applying Class Elimination Rules and Class Categories to VRS domain

Nouns	**Class Elimination Rules Applied (Step N2)**	**Class Categories Applied (Step N3)**	**Class**
Video Rental Store	Adopt VRS; Redundant (CER1)		
VRS	NO	Place (CC2)	√
Problem	Meta languages (CER6)		
Rental	NO	Transaction (CC5)	√
Return	Reverse of Rental (CER1)		
Management	Meta language (CER5)		
Inventory	NO (Singleton)	Catalog (CC11)	√
(Video)Tapes	NO	Physical Thing (CC3)	√
Rental Prices	Attribute (CER7)		
Reports	Derived (CER9)		
Business Activities	Meta language (CER6)		
Rental Items	NO	Transaction Line Item (CC6)	√
Store	The same as VRS; Redundant (CER1)		
Customer ID Number	Attribute (CER7)		
Arbitrary Number	Vague (CER3)		
Phone Number	Attribute (CER7)		
Combination	Irrelevant (CER2)		
First Name	Attribute (CER7)		
Last Name	Attribute (CER7)		
Customer Data	Vague (CER3)		
Order	The same as Rental; Redundant (CER1		
Bar Code ID	Attribute (CER7)		
Video Information	Vague (CER3)		
Video Inventory File	Same as Inventory; Redundant (CER1)		
Tape ID	Attribute (CER7)		
System	Meta language (CER6)		
Total Rental Fee	Attribute (CER7)		
Payments	NO	Transaction (CC5)	√
Amount due	Attribute (CER7)		
Total Amount due	Attribute (CER7)		
Past-due Amount	Attribute (CER7)		
Zero	Value (CER8)		
Customer	NO	Roles of People (CC1)	√
Customer Information	Vague (CER3)		
Category	Attribute (CER7)		
Drama	Value (CER8)		
Comedy	Value (CER8)		
Action	Value (CER8)		
Conflict	Irrelevant (CER2)		
Computer Data	Vague (CER3)		
Store Manager	NO	Roles of People (CC1)	√

Rental fee	Attribute (CER7)		
Cash	NO	Physical Thing (CC3)	√
Check	NO	Physical Thing (CC3)	√
Credit Card	NO	Physical Thing (CC3)	√
Reporting Requirements	Meta language (CER6)		
Customer Rental History	Derived (CER9)		
Video Rental History	Derived (CER9)		
Titles	NO	Specification (CC8)	√
Top Ten Rentals	Derived (CER9)		
Item Status	Attribute (CER8)		
Overdue Videos	Roles (CER1)		
Outstanding Balances	Attribute (CER7)		

References

[Abbo83] Abbot, R., Program Design by Informal English Description, *CACM*, Vol. 26, No.11, 1983, pp. 882-894.
[Ambl01] Ambler, S. *Building Object Applications That Work*, 2^{nd} edition, SIGS Books, 2001
[BP98] Blaha, M. and Premerlani, W. (1998). *Object-Oriented Modeling and Design for Database Applications*, Prentice Hall.
[Bahr99] Bahrami, A. (1999). *Object-Oriented Systems Development*, Irwin McGraw-Hill.
[BRJ99] Booch, G., Rumbaugh, J., and Jacobson, I (1999). *The Unified Modeling Language: User Guide*. Addison Wesley.
[Booc94] Booch, G. (1994). *Object-Oriented Analysis and Design with Applications*, 2^{nd} Ed., Benjamin Cummings.
[Chen83] Chen, P.P. English Sentence Structure and Entity-Relationship Diagrams, *Information Sciences*, 1983, pp. 127-149.
[CNM97] Coad, P, North, D., and Mayfield M. *Object Models: Strategies, Patterns & Applications*, 2^{nd} ed., Prentice Hall, 1997.
[CY91] Coad, P. and Yourdon, E. *Object-Oriented Analysis*, 2^{nd} ed., Prentice Hall.
[Derr95] Derr, K.W., *Applying OMT,* Prentice-Hall, 1995.
[CAB94] Coleman, D., Arnold, P., Bodoff, S., Dollin, C., etc., *Object-Oriented Development: The Fusion Method,* Prentice-Hall,1994.
[DE00] Delcambre, L.M.L. and Eckland, E., A Behaviorally driven Approach to Object-Oriented Analysis and Design with Object-Oriented Data Modeling, In *Advances in Object-Oriented Data Modeling*, MIT Press, 2000, pp. 21-40.
[DP00] Dobing, B. and Parsons, J. (2000). Understanding the Role of Use Cases in UML: A Review and Research Agenda, *J. of Database Management*, Vol. 11, No. 4, Oct.-Dec. 2000, pp. 28-36.
[Fowl99] Fowler, M., *UML Distilled: Applying the Standard Object Modeling Language*, Addison Wesley, 2^{nd} ed., 1999.
[HM96] Holland, I.M. and Lieberherr. Object-Oriented Design, *ACM Computing Survey*, March 1996, pp. 273-275.
[HKK93] Honiden, S., Kotaka, N., and Kishimoto, Y. Formalizing Specification Modeling in OOA. *IEEE Software*, January 1993, pp. 54-66.
[JCJ92] Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, 1992.
[Larm01] Larman, C., *Applying UML and Patterns,* 2^{nd}, Prentice Hall, 2001.
[Maci01] Maciaszek, L. A., *Requirement Analysis and System Design: Developing Information Systems with UML*. Addison Wesley, 2001.
[Meye97] Meyer, B. (1997). *Object-Oriented Software Construction*. Upper Saddle River, NJ: Prentice Hall.
[PW97] Parsons,J. and Ward, Y. (1997). Choosing Classes in Conceptual Modeling, CACM, June 1997, pp. 63-69..
Quat98] Quantrani, T. *Visual Modeling with Rational Rose and UML*. Addison Wesley, 1998.
[RBP91] Rumbaugh, J., Blaha, M., Premerlani, W. etc. *Object-Oriented Modeling and Design*, Prentice-Hall, 1991.
[Rich99] Richter, C. (1999). *Designing Flexible Object-Oriented Systems with UML*, Macmillan Technical Publishing.
[RJB99] Rumbaugh, J., Jacobson, I., and Booch, G., (1999). *The Unified Modeling Language: Reference Manual*. Addison Wesley.
[Ross88] Ross, R.G. (1988). *Entity Modeling: Techniques and Applications*, Database Research Group, Inc.
[Rose99] Rosenberg, D. (1999). *Use Case Driven Object Modeling with UML: A Practical Approach*, Addison Wesley.
[Siau01] Siau, K. (2001). *Unified Modeling Language: Systems Analysis, Design and Development Issues*. Idea Publishing Group.
[SF95] Song, I. and Froehlich, K. "Entity-Relationship Modeling: A Practical How-to Guide," *IEEE Potentials*, Vol. 13, No. 5, Dec/Jan 1994-1995, pp. 29-34.
[SM88] Shlaer, S. and Mellor, S.J. (1988). *Object-Oriented Systems Analysis: Modeling the world in Data*, Englewood Cliffs, NJ: Yourdon Press.
[SP99] Stevens, P. and Pooley, R. (1999). *Using UML: Software Engineering with Objects and Components*, Reading, MA: Addison Wesley.
[Star01] Starr, L., *Executable UML: How to Build Class Models* , Prentice Hall, 2001.
[Wilk95] Wilkinson, N.M., *Using CRC Cards: An Informal Approach to Object-Oriented Development*, SIGS Books, 1995.
[YA96] Yourdon, E. and Argila C. (1996). *Case Studies in Object-Oriented Analysis and Design*, prentice Hall.