

Auto-generation of Class Diagram from Free-text Functional Specifications and Domain Ontology

Xiaohua Zhou, Nan Zhou
{Xiaohua.Zhou, Nan.Zhou}@drexel.edu

Abstract

In this paper, we propose and implement a system that automates the building of class diagram from free-text requirement documents. Our approach first applies natural language processing (NLP) techniques to understanding of the written requirements, and then uses domain knowledge represented by domain ontology to improve the performance of class identification. The basic notion of our approach is that, a class especially core class of the domain is always semantically connected with other classes and its attributes. Based upon this, our system applies a spider model to search classes of interest. It first identifies core classes of the domain as a starting point, with which we are usually highly confident, and further finds classes that are semantically related with the identified ones. The strength of this method is that it identifies classes and relationships at one step. Specifically, our approach aims at addressing the following challenging research questions: (1) How to use domain ontology but not limited to domain ontology? (2) How to find candidate concepts? (3) How to find concept pairs with strong semantic connection in the context? (4) How to distinguish attribute name from class name for each concept? (5) How to name each inter-class relationship, aggregation, generalization, or association? (6) Are there any missing classes or attributes? This methodology finds its way through NLP, which is proved to be effective and sufficient to provide solutions to these problems in this paper.

1. Introduction

In object-oriented systems, the notion of 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 (Song, 2004). Auto-generation of class diagram will without doubt save the time and effort of system analyst, especially for novice. However, the automation of class generation from a written source is highly challenging due to the following three reasons (Richter, 1999; Maciaszek, 2001).

- Natural language is ambiguous. Thus, rigorous and precise analysis is very difficult
- The same semantics could be represented in different ways.
- Concepts that were not explicitly expressed in a written source are often very difficult to model. Usually, expert domain knowledge is needed to identify the hidden classes.

As far as the best knowledge to us, the previous researchers have not offered any sound solutions to above-mentioned problems yet. There are two representative streams in the previous works. One regards class elicitation as a dual process: class suggestion and class rejection (Meyer, 1997). The other thinks that class identification involves two activities: discovery and invention. No matter which notion the researchers adopt, they employ general but rigid heuristics, guidelines or rules to suggest, reject and invent classes for the variety of domains. Consequently, when their approaches are implemented, the systems don't infer class diagrams based on the understanding of the written documents. Thus, the quality of the resulting class diagrams is highly restrained.

Our approach proposed in this paper tries to address the problem mentioned but not intends to resolve all. The approach first applies natural language processing (NLP) techniques to understanding of the written requirements, and then uses domain knowledge represented by domain ontology to improve the performance of class identification. The basic notion of our approach is based upon the belief from heuristics that *a class especially core class of the domain is always semantically connected with other classes and its attributes*. The novice part of this approach is the use of a *NLP-based spider model* to identify core classes of the domain as starting point, for which we are usually highly confident, and further find classes that are related with the identified ones. The essential part of the methodology is full application of NLP techniques.

Specifically, our methodology use NLP techniques to address the following challenging research questions: (1) How to use domain ontology but not limited to domain ontology? (2) How to find candidate concepts? (3) How to find concept pairs with strong semantic connection in the context? (4) How to distinguish attribute name from class name for each concept? (5) How to name each inter-class relationship, aggregation, generalization, or association? (6) Are there any missing classes or attributes?

The rest of the paper is organized as follows: section 2 is the literature review of the related works; the framework of our approach and the algorithms of each component in the framework are presented in section 3; section 4 briefly introduces the implementation of the system; the last section presents the conclusion of the paper.

2. Related Works

2.1 Approaches for Identifying Classes in Conceptual Modeling

There are several approaches for identifying classes, among which the noun analysis is the most popular one (Abbot, 1983; Chen, 1983; Rumbaugh, Blaha, & Premerlani, 1991; Booch, 1994; Richter, 1999; Rosenberg, 1999). Other methods use of class categories as tips (Booch, 1994; Rumbaugh, Jacobson, & Booch, 1999; Starr, 2001; Larman, 2001), use case descriptions (Jacobson, 1992; Richter, 1999), or CRC (Class-Responsibilities-Collaborators) cards (Wirfs-Brock, Wilkerson, & Wiener, 1990).

Taxonomic Class Modeling (TCM) methodology proposed by Song (Song et al., 2004) is the most recent and most complete work of identifying classes to our best knowledge at the time being. Starting with problem statement, it incorporates the noun analysis, class categories, English sentence structure rules, checklists, and other heuristic rules for modeling. TCM tries to identify three types of classes: noun classes, transformed classes (from verbs), and discovered classes (from domain knowledge). Workflows of TCM methodology are as follows:

1. Use noun analysis to find candidate classes
2. Use class elimination rules to eliminate spurious classes
3. Use pre-defined class categories to discover hidden classes
4. Use domain knowledge to review the list

Starting with written documentation has been the most commonly suggested way to analyze the proposed system in order to capture the classes to be used in the modeling. These written documents can be problem statement of the proposed system, use case descriptions (in UML), functional specification, etc. Rosenberg (Rosenberg, 1999) 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 (1998) recommend that we begin analysis with a written problem statement.

Though beginning with the problem statement is the easiest method for modeling classes for a system, it does not give us a complete list of classes for analysis. In contrast to problem statement, functional specification provides more detailed description of how the system will function thus it is more useful in terms of giving a more complete list of classes. Also, usually written in natural language, functional specification is one of the most common documentations that come before the design and implementation of the system and it's easy to get access to. Based upon these considerations, our method chooses *functional specification* of

the system to begin the analysis to identify classes, which is different from those pre-mentioned works.

2.2 Automation of Conceptual Modeling

Manually going through unstructured documentations to identify classes is laborious as well as intellectually challenging thus it is appealing to automate the process. Though TCM methodology is practical to use, it deals well with analyzing relatively short problem statement. But the documentation such as functional spec can go to dozens or even hundreds of pages, manual method is not very practical at all and we need to have the analysis done automatically with the help of certain tools. Automating modeling from text is considered as state of the art. Unfortunately, most of the approaches proposed still remain manual or semi-automatic.

SMART (system model acquisition from requirements text) (Dori et al., 2004) is a good attempt to construct a system model in a semi-automatic way from the system's free text documentation of the requirements. It uses a LISP-based, heuristics-directed categorization engine GRACE (Grid Search and Categorization Engine) to extract categories from the document set. It also uses a set of templates to search object-process methodology (OPM) relations and generate object-process language (OPL) sentences. But apart from this, the rest of the work is manual, including manually editing the extracted categories and the results.

Zisman et al (Zisman et al, 2003) proposed an approach to support automatic generation and maintenance of traceability relations between different types of software requirements artifacts, which are represented in the eXtensible Markup Language (XML). This approach apparently is not applicable to unstructured documentations in natural language.

An important approach to supporting the Requirements engineering (RE) process is linguistic approach, which is achieved by language manipulation, including the following steps:

1. Acquisition of domain-dependent knowledge using NL statements
2. Graphic representation of the semantic contents of the NL statements
3. Mapping of the real-world description to a conceptual schema

This paper focuses on describing our *spider model* algorithm that is in the process of implementation though our approach aims at doing the analysis using our algorithm *automatically*.

2.3 How NLP is used

Studies have tried to apply NL parsing and understanding techniques to automatic extraction of models from NL requirements (Macias & Pullman, 1993; Fantechi, Ristori, Carenini, Vanocchi, & Moreschini, 1994; Juristo, Moreno, & Lopez, 2000)

Goldin et al. did lexical analysis to find abstractions in unstructured and un-interpreted text (Goldin and Berry, 1997). Overmyer et al. proposed a methodology and prototype tool, Linguistic Assistant for Domain Analysis (LIDA), which provide linguistic assistance in the model development process (Overmyer et al, 2001). LIDA provides tools for the user to analyze texts, but it does not analyze texts itself. It doesn't do dependable deep semantic analysis or even adequate syntactic analysis.

In *Requirements Validation via Automated Natural Language Parsing* (Nanduri & Rugaber, 1995), the analysis process starts with collecting a list of guidelines on object modeling (Rumbaugh et al, 1991), expresses the guidelines in terms of the parsing rules for a publicly available natural language parser, and applies them to the parser output of high level specifications.

The main focus of our methodology is Natural Language Processing (NLP) techniques. It makes use of both syntactic (POS tagger, sentence structure, etc.) and semantic (like concept relationships in context) approaches.

2.4 Ontology

Relationship among classes is essential in object-oriented analysis of conceptual modeling. Identifying these relationships has been recognized as a difficult topic. It remains not well studied though lots of work has been put in this field (Chen, 1983; Rumbaugh et al, 1991; Jacobson et al, 1992; Booch, 1994; Kaindl, 1996).

Recently, ontology has been a hot topic in conceptual modeling (Wand et al, 1999; Shanks et al, 2003; Weber, R., 2003). In *Automatically Acquiring Requirements of Business Information Systems by Reusing Business Ontology*, an approach is proposed for automatically acquiring customers' requirements by reusing the domain knowledge that consists of a business ontology and a domain ontology. The approach is based upon the representation of conceptual graphs.

3. Algorithm

3.1 NLP-based Spider Model

In this section, we propose the algorithm for automatic generation of class diagrams. The input files include free-text functional specifications and structured domain ontology while the output is class diagram components including classes, attributes of each class, and inter-class relationships. The inter-class relationships can be classified into generalization, aggregation, and association. The association relationships can be further classified into one-to-one, one-to-many, and many-to-many.

Our approach applies a NLP-based spider model to search classes of interest, that is, we identify core classes of the domain as starting point, for which we are usually highly confident, and further find classes that are related with the identified ones. The strength of this method is that it identifies classes and relationships at one step. Before moving into the spider model, we present the following important components of the approach and how they are ordered in the paper, which will help to understand its rationale.

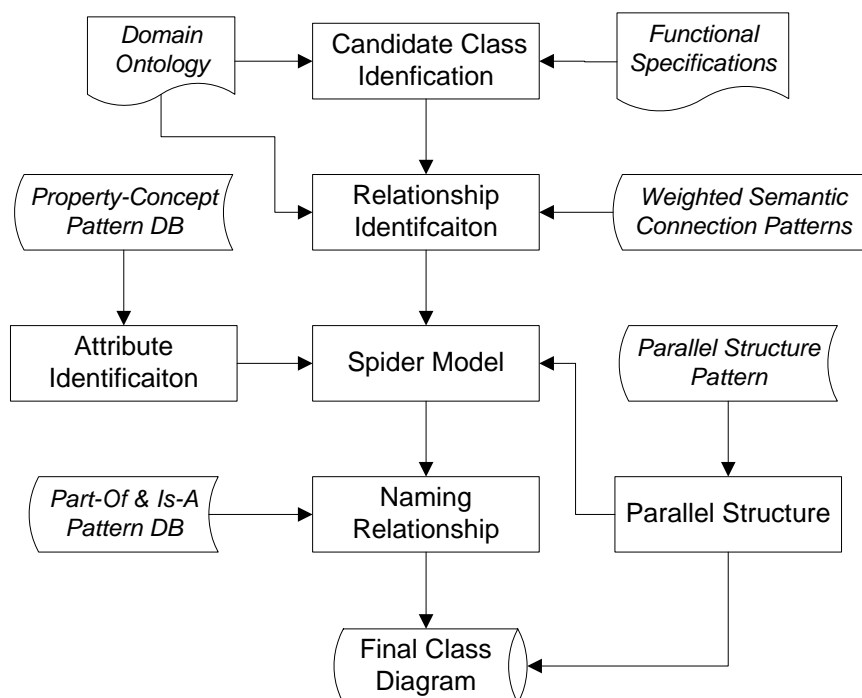


Figure 1. Framework of NLP-based Spider Model

Domain Ontology serves as domain knowledge to improve the performance of conceptual modeling. Its hierarchical structure is illustrated in section 3.2.

Candidate Class Identification module outputs **Preliminary Candidates** and **Refined Candidates**, which serve as input of spider model. The part of speech (POS) tagger and sentence parser produce preliminary candidates. Further, word sense disambiguation technique and semantic network are employed to produce refined candidates, which are semantically related to the concepts defined in the ontology. This module is introduced in section 3.3.

Relationship Identification module is the most important component of the spider model. It uses linkage distance to identify all concept pairs with strong semantic association within a sentence. We further assign different weight to each concept pair to indicate how strong the connection is according to the constituents the concepts serve as in the sentence. The method is presented in section 3.4.

Attribute Identification module distinguishes attributes from classes. After two concepts are found to be strong associated to each other, we need to determine whether the concepts stand for a class or an attribute of a class. See details of the method in section 3.5.

Naming Relationship module applies linguistic patterns to find aggregation relationship and generalization relationship. It uses numeric relationship understanding technique to distinguish one-to-one association, one-to-many association, and many-to-many association. The algorithm is described in section 3.6.

Parallel Structure uses the feature of English sentence structure to help elicit more attributes of a class, more child classes of a super class, and more aggregation part of a class. The module is presented in section 3.7.

The main algorithm of our approach is an iterative process to identify classes and attributes. In each step, it chooses one concept from the specified set with maximum relationship score and connected with identified classes. The relationship score is an indicator of how the current concept semantically connects with all other concepts of the set of candidate concepts. If the relationship score is less than the set threshold, the flow stops. Otherwise it puts the concept into the set of classes or the set of attributes against the number of properties the concept owns. The algorithm is presented below.

```

c = MaxW(C) /* choose one class from candidate set as starting point */
N = {c} /* initialize the set of identified classes */
C = C - {c}
While(true){
    S = {c | R(c, m) > 0, c ∈ C, m ∈ N}
    c = MaxW(S) /* choose next concept */
    If Weight(c) < Threshold Then break
    C = C - {c} /* remove from the candidate set */
    If PN(c) ≥ 2 Then
        N = N ∪ {c} /* add to the set of classes */
    Else
        A = A ∪ {c} /* add to the set of attributes */
    End If
}

```

3.2 Domain Ontology

The ontology aims at capturing the knowledge artifacts within a business domain where business transactions occur. It consists of structured concepts that are semantically related to each other. Our proposed ontology is built on the previous work of Valaradi et al (2001). In our

ontology, the semantic relations are either vertical or horizontal. Vertical relations include broader (B), part-of (PA), or instance-of (I). Horizontal relations are similarity (S) or relatedness (R). Each concept C is represented as follows:

$$C = \{N, D, CT, B, PA, HAS-PART, I, HAS-INSTANCE, S, R\}$$

Where n is the name of the concept;

D is the description of the concept;

CT is the category of the concept (i.e. one of the primary categories described shortly);

B is the broader concept;

PA is part-of;

I is instance-of;

S is similar to;

R is related to.

Here we classify the basic concepts that need to be modeled within the business domain into the following categories:

- (1) Roles – roles of people. They represent actors who perform important functions or activities.
- (2) Organizations – they represent important business units, like company, functional division, departments, etc.
- (3) Processes – they represent important activities that need to record some data with the time the event occurs. They could be transactions like payment or interaction between two classes like reservation of a flight.
- (4) Objects – they represent physical (tangible) things or intangible objects. Examples are product, project, account, etc.

The following example concept, customer, illustrates the representation of the ontology. (For more complete information, refer to the appendix)

{customer, actors in the system who obtain products or services from the company in most cases by paying, roles, actor, null, (customer ID, first name, last name, phone, address), null, customer of a certain product, (potential customer, user, subscriber), (account, payment, product)}

3.3 Candidate Classes Identification

Most previous works think that sources of candidate class name are major noun phrases and minor verbs. By using part of speech (POS) tagger, we obtain a list of verbs and nouns. However, nouns are not equal to noun phrases in many cases. We further use sentence parser to get noun phrases of interest. Here we only consider nouns with pre-noun modifiers like NN+NN and JJ+NN. In the parser of Link Grammar, a linkage name beginning with capital letter 'A' indicates this linkage connects pre-noun modifier to noun. In figure 1, for instance, there is a link called "AN" between "rent" and "process", so "rent process" is a noun phrase.

After POS tagging and sentence parsing, we get a set of noun phrases and verbs, namely **Preliminary Candidates**. Most preliminary candidates are not good classes, and are even irrelevant to the domain at all. The domain ontology offers the chance to further refine the classes for the domain model. Obviously, we can't use exact string match to refine the candidate class names because ontology provides just general knowledge of the domain rather than exact conceptual model of a specific system. Instead, we search concepts out of the result of tagger and parser, which are related to the concepts defined in the ontology.

WordNet provides the semantic network of different senses of a word (Miller et al., 1990). The synonyms, hypernyms, hyponyms, holonyms, meronyms, derivations, coordinate terms and so on will appear in the nodes of the network of the target word. If a noun phrase or a verb appears in the nodes of the semantic network of a concept of the ontology, it is considered related to the domain ontology. The algorithm to refine the candidate class names is presented as follows.

Step 1: Disambiguate the sense of the concepts in ontology.

We use a simple unsupervised method (Agiree and Rigau, 1996; Agiree 1998) to disambiguate the sense of each concept in ontology.

Step 2: Build the set of related concepts of the target concept.

Since we disambiguate the sense of the target concept in the preceding step, we can easily build the semantic network using WordNet. Apart from the WordNet semantic network, the concepts defined in the ontology including N, B, PA, HAS-PART, I, HAS-INSTANCE, S, and R, will be added to the set of related concepts.

Step 3: Determine the final refined candidates.

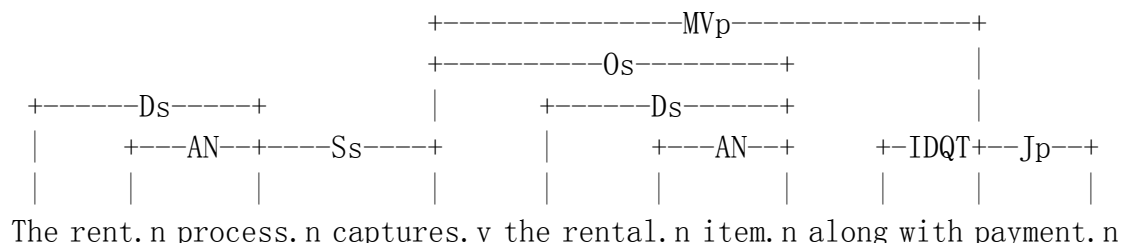
If the preliminary candidate is one of the elements of the result set in the second step, it is taken as a final **Refined Candidate**. Besides, any preliminary verb that can be derived from any noun refined candidate is also a final **Refined Candidate**.

3.4 Relationship Identification

Relationship identification refers to discover any semantic concept pairs within a sentence. The relationship of concept pair can be roughly classified into three categories: (1) relationship of two classes, (2) attribute of a class, and (3) value of an attribute. The relationship of two classes can be further classified into generalization, aggregation and association. The association relationship can be either one-to-one, one-to-many, or many-to-many. However, in this step we don't have to specify the name of the relationship, but to determine whether two concepts have relationship and how strong the relationship is if any.

Because it is difficult to directly test whether two concepts are semantically connected with each other in the context, we use linkage distance to measure the semantic connection of two concepts.

Linkage is a term defined in the famous parser, Link Grammar (Sleator & Temperley, 1993). It refers to the direct connection of two words within a sentence. In the following example,



word “process” and “captures” have a linkage called Ss; word “captures” and “item” have a linkage called Os. So word “process” can reach word “item” by crossing two linkages. Intuitively, the less linkage number between two concepts, the closer they are semantically associated in the context. In order to avoid concepts pairs without any semantic connection at all, we only keep the concept pairs with less than or equal to 3 linkages in relationship analysis.

Obviously, the confidence on the relationship of concept pair varies at the positions the concept pair appears in the sentence. For example, two concepts appeared in subject phrase and object phrase respectively definitely have high chance of being semantically connected with each other. So we will assign different weights to concept pairs against their positions in the sentence. The position of a concept in a sentence can be one of the following values: SP (subject phrase), OP (object phrase), PP (prepositional phrase), CP (complementary phrase) and Verb (the derivation of verb can be a concept, e.g. pay is the derivation of payment). If the concept pair is also defined in the ontology, the corresponding weight (confidence) will be increased.

Each pair of concepts may occur multiple times in the document. The sum of weight of each occurrence is served as the final weight of the concept pair.

3.5 Attribute Identification

When a concept is found to have strong semantic association with an existing class, it is still needed to tell whether the concept is an attribute of the associated class or another independent class. Usually, if there are more than one property associated with the concept, it is a class otherwise an attribute. For example, if only the amount of the price is necessary to remember, the price serves as an attribute, but if it is necessary to remember the amount, the discount, and the effective date range, it might well be a class.

Property and concept is a special subordinate-master relationship in terms of semantics. There are strongly indicative linguistics patterns for such relationship. In the proceeding section, concept pairs are already identified. We use the following 7-tuple linguistic pattern to determine if the concept pair is property-concept relationship.

{first, second, first-role, first-prep, second-role, second-prep, verb}

Where

first: the first concept is property or concept

second: the second concept is property or concept

first-role: the role the first concept serves in the sentence

first-prep: the preposition before the first concept if any

second-role: the role the second concept serves in the sentence

second-prep: the preposition before the second concept if any

verb: the verb used when the two concepts are subject and object respectively

The following 4 examples illustrated the linguistic patterns. Since prepositions are limited and very few verbs indicate master-subordinate relationship, the patterns are limited.

Example pattern 1: {concept, property, pre-noun modifier, null, modified noun, null, null }

Any overdue fee is added up to *customer's outstanding balance*.

Example pattern 2: {property, concept, noun, null, post pp, for, null }

The *bar code ID* for each *item* is entered and video information from inventory is displayed.

Example pattern 3: {property, concept, object, null, pp, in, null }

System determines the rental price and due date and displays *title*, *due date*, and *price* in the *transaction*.

Example pattern 4: {property, concept, object, null, pp, for, null }

A store employee creates an *account* for a new *customer* by inputting customer information.

3.6 Naming Relationship

In the standard of Unified Modeling Language (UML), inter-class relationship can be classified into aggregation, generalization and association. Association relationship can be either one-to-one, one-to-many or many-to-many.

Aggregation is a part-whole relationship while generalization is a general-special relationship. Both two types of relationships have strong linguistic patterns associated with them. Many previous researches on this topic are well done so that we simply follow their methods (Berlan & Charniak, 1999; Hearst, 1992). However, we adopt a conservative strategy for the identification of aggregation and generalization relationship in the paper. In addition to the criterion of linguistic patterns described in previous works, they must meet two additional criterions:

- (1) The parts of the composite class and the child classes of the super class must be appeared in the parallel structure. Refer to section 3.7 for details.
- (2) At least one of the elements in the parallel structure has part-of or is-a relationship with the super class by looking up WordNet.

Regarding the association relationship, a naive algorithm is applied to tell three types of sub-relationships. The numeric words or phrases modifying the concept will be checked. For examples:

- (1) The customer can create more than one account. The customer and account have a one-to-many relationship since “more than one” modifies “account”.
- (2) Each customer can keep only one registered credit card in the profile. Here customer and credit card are one-to-one relationship because “only one” modifies “credit card”.

If the concept A and concept B are one-to-many relationship and vice versa, the two concepts have a many-to-many relationship.

3.7 Parallel Structure

Our algorithm fully uses the domain knowledge represented by domain ontology to conceptually model the system. However, for the purpose of generalization, the ontology usually contains the general knowledge only. Ontology often covers core or important concepts in the domain, and important attributes of the concepts. Thus it doesn't exhaust all concepts and attributes for a specific system. Fortunately, parallel structure within an English sentence helps us find missed concepts or attributes. In detail, parallel structure can find (1) rest parts of a composite, (2) rest children of a super concept, and (3) rest attributes of a class.

For examples:

- (1) Rental fees can be paid by *either cash, check or a major credit card*.

If the system already identifies credit card as one of the child classes of payment (we know paid is the derivation of payment by looking up WordNet), we are then confident that cash and check are the other two child classes of payment.

- (2) In use case Maintain Customer, a store employee creates an account for a new customer by inputting customer information, i.e. *last name, first name, address, zip code, phone number, rental items, outstanding balance, etc.*

The system thinks that address, zip code, phone number, outstanding balance are also good candidate attributes of the class customer because last name and first name are the attributes of the class customer. Here we exclude rental items as attribute because it is already identified as an associated class of customer.

The use of parallel structure follows 2 steps:

- (1) Use linguistic patterns to identify parallel structure within a sentence. The frequently used parallel structure patterns include:
 - A. words: and, or, etc.
 - B. phrases: both... and..., ... as well as..., either... or..., not only... but also...
- (2) If one of the elements within the parallel structure has relationship with already identified concepts appeared in the other constituents of the sentence, the rest elements is highly possible to have the same relationship with the concept.

4. Implementation

The implementation of the system involves not only complex algorithm described in section 3, but also many external programs and resources. We choose Eric Brill's rule-based transformation tagger for part-of-speech tagging (Brill, 1994), and pick up Link Grammar as sentence parser because linkage information is very important to our approach. Meanwhile, WordNet serves as lexicon resources in our system. Though Brill's tagger, Link Grammar Parser and WordNet are all written in native code, we find third party Java Native Interfaces (JNI) for these programs. Therefore, we can still integrate them with our main application written in Java well. Since all external programs support Windows, Linux and UNIX, and Java is platform independent, our system suppose to be able to run in different platforms though we develop and test it on Windows platform only. The architecture of the system is presented in the following figure.

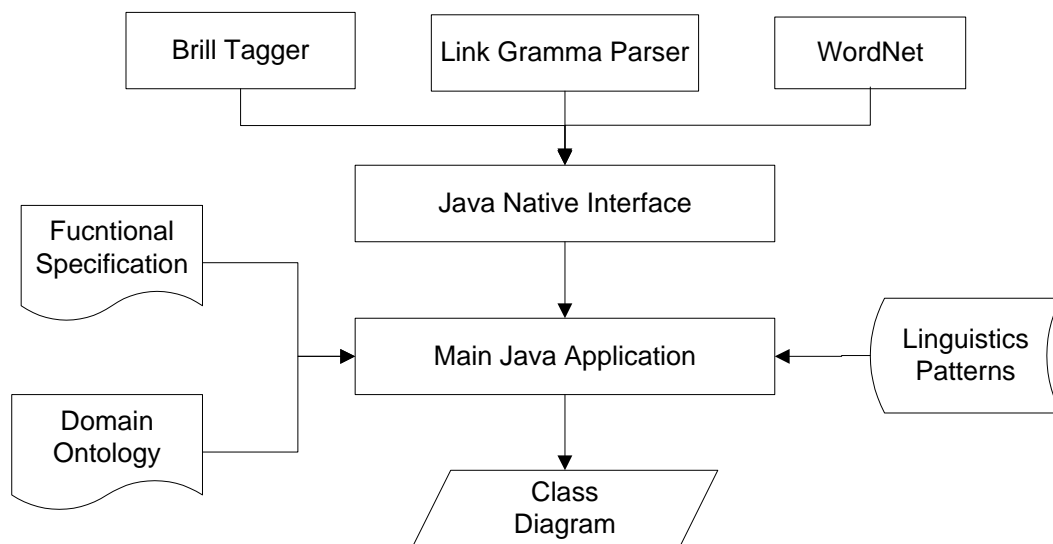


Figure 2. System Architecture

The system can accept any domain ontologies. But the domain ontology should be structured in the form as follows.

$$C = \{N, D, CT, B, PA, HAS-PART, I, HAS-INSTANCE, S, R\}$$

Theoretically the system can accept any free-text document describing functional specifications. The version implemented here would have best performance to deal with problem statement and use case description written in natural language.

Our approach also needs external linguistic patterns including weighted semantic connection pattern, property-concept pattern, parallel structure pattern, part-of pattern, and is-a pattern. Currently we build all linguistic patterns manually. They may be automatically or semi-automatically learned from training corpora in future. Fortunately, these linguistic patterns are domain independent, that is, they are built once, but worked for all.

5. Conclusion

In this paper, we not only provide the framework for auto-generation of class diagrams from free-text functional specification documents, but also figure out the specific algorithms for each component of the framework. In comparison with previous works on class generation methods or automating algorithms, it has several unique features as follows.

First of all, our method fully uses natural language processing (NLP) technique to identify classes, attributes and inter-class relationships. Most previous works just use NLP technique (usually part-of-speech tagger and parser) to find candidate class names, and after that, they often use general but rigid heuristics, guidelines or rules to generate class. In contrast, NLP technique is applied through our approach.

- (1) Part-of-speech (POS) tagger and sentence parser are used to produce preliminary class candidates while word sense disambiguation (WSD) and semantic network are employed to refine preliminary candidates.
- (2) Linkage distance produced by Link Grammar Parser is used to find concept pairs within strong semantic association within a sentence. Different weights are assigned to each concept pair according to the linguistic patterns the concept pair belongs to.
- (3) Linguistics patterns are used to evidence class-attribute relationship, aggregation relationship, and generalization relationship, and distinguish three types of association relationships.
- (4) Parallel structure serves as a clue to find more attributes of a class, more child classes of a super class, and more parts of a composite class.

Secondly, as far as the best knowledge to us, our approach well integrates domain ontology with class generation for the first time. The domain ontology contains important or core domain knowledge. Thus, it should be the right way to greatly improve the quality of resulting class diagrams. The ontology feeds the system important classes and their attributes, but resulting classes and attributes are not limited to those listed in ontology.

The full application of domain ontology and NLP technique helps to understand the meaning of functional specifications and in return might improve the quality of the resulting class diagrams. However, our approach correspondingly depends on more external resources than

previous approaches do. It needs not only domain ontology as input, but also many linguistic patterns for natural language processing. Currently we manually build the linguistic pattern databases. As one of the future works, we plan to learn linguistic patterns of interest from training corpora automatically since the patterns are domain independent.

The other urgent work is the implementation of the system. At the time being, we finish the Java Native Interfaces with Brill Tagger, Link Grammar Parser and WordNet, and complete the coding of a couple of components of the framework. Upon the completion of the system, we will do case studies to compare the performance of our approach with that of other approaches such as TCM method and that of human experts.

References

Abbot, R. (1985). *Program Design by Informal English Description*, Communication of ACM, Vol. 26(11), 882-894.

Agirre, E. and G. Rigau. *Word Sense Disambiguation Using Conceptual Density*. In Proceedings of COLING'96. Copenhagen, Demark, 1996.

Agirre, E. *Formalization of Concept-Relatedness Using Ontologies: Conceptual Density*. Ph.D. thesis, LSI saila, University of the Basque Country, 1998.

Berlan, M. & Charniak E., (1999). *Finding Parts in Very Large Corpora*. In Proceedings of the the 37th Annual Meeting of the Association for Computational Linguistics (ACL-99).

Booch, G. (1994). *Object-Oriented Analysis and Design with Applications*, 2nd Ed., Benjamin Cummings.

Booch, G., Rumbaugh, J., & Jacobson, I (1999). *The Unified Modeling Language: User Guide*. Addison Wesley.

Brill E., *Some Advances in Transformation-Based Part of Speech Tagging*. Proceedings of the twelfth national conference on Artificial intelligence, Pages: 722 – 727, 1994.

Chen, P.P. (1983). *English Sentence Structure and Entity-Relationship Diagrams*, Information Sciences, 29, 127-149.

Dori, D., et al, (2004). SMART: System Model Acquisition from Requirements Text. BPM 2004, LNCS 3080, pp. 179-194.

Macias, B. and Pullman, S.G., (1993). *Natural Language Processing for Requirements Specification, Safety-Critical Systems*, Chapman and Hall: London, 57-59.

- Fantechi, A., Gnesi, S., Ristori, G., Carenini, M., Vanocchi, M., and Moreschini, P., (1994). *Assisting Requirement Formalization by Means of Natural Language Translation*, Formal Methods in System Design, 4(3), 243-263.
- Goldin, L. and Berry, D.M., (1997). *A prototype Natural Language Text Abstraction Finder for Use in Requirements Elicitation*, Automated Software Engineering Journal 4, (4) 375-412.
- Heart, M.A., (1992). *Automatic Acquisition of Hyponyms from Large Text Corpora*. In Proceedings of the 14th International Conference on Computational Linguistics.
- Jacobson, I., Christerson, M., Jonsson, P., & Overgaard, G. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley.
- Juristo, N., Moreno, A.M., Lopez, M., (2000). *How to Use Linguistic Instruments for Object-Oriented Analysis*, IEEE Software, 17(3), 80-89.
- Kaindl, H., (1996). *How to Identify Binary Relations for Domain Models*, IEEE Proceedings of ICSE-18.
- Larman, C., (2001). *Applying UML and Patterns (2nd)*. Prentice Hall.
- Maciaszek, L. A., (2001). *Requirement Analysis and System Design: Developing Information System with UML*. Addison Wesley.
- Meyer, B. (1997). *Object-Oriented Software Construction*. Prentice Hall.
- Miller G., et al. (1990). *WordNet: an On-line Lexical Database*. International Journal of Lexicography, 3, 235-245.
- Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., *Object-oriented Modeling and Design*, Prentice Hall, 1991.
- Rumbaugh, J., Jacobson, I., & Booch, G. (1999). *The Unified Modeling Language: Reference Manual*. Addison Wesley.
- Richter, C. (1999). *Designing Flexible Object-Oriented Systems with UML*, Macmillan Technical Publishing.
- Rosenberg, D. (1999). *Use Case Driven Object Modeling with UML: A Practical Approach*, Addison Wesley.
- Shanks, G., Tansley, E., & Weber, R., (2003). *Using Ontology to Validate Conceptual Models*, Communications of the ACM, Vol 46, No. 10, October 2003.

Sleator, D. & Temperley D., (1993). *Parsing English with a Link Grammar*. Third International Workshop on Parsing Technologies.

Song, Il-Yeol, et al, (2004). *A Taxonomic Class Modeling Methodology for Object-Oriented Analysis*, to be published on Information Modeling Methods and Methodologies.

Starr, L., (2001). *Executable UML: How to Build Class Models*, Prentice Hall.

Velardi, P., Fabriani, P., & Missikoff, M., (2001). *Using Text Processing Techniques to Automatically Enrich a Domain Ontology*, FOIS '01, October 17-19, 2001, Ogunquie, Maine, USA, ACM.

Wirfs-Brock, R., Wilkerson, B., & Wiener, L. (1990). *Designing Object-Oriented Software*. Prentice Hall.

Wand, Y., Storey, V.C., & Weber, R., (1999). *An Ontological Analysis of the Relationship Construct in Conceptual Modeling*, ACM Transactions on Database Systems, Vol. 24, No. 4, 494-528, December 1999.

Weber, R., (2003). *Conceptual Modeling and Ontology: Possibilities and Pitfalls*, Journal of Database Management, 14(3), 1-20, July-Sept 2003.

Zisman, A., Spanoudakis, G., Perez-Minana, E., & Krause, P., (2003). *Tracing Software Requirements Artefacts*, Proceedings of the 2003 International Conference on Software Engineering Research and Practice (SERP '03).

Appendixes

VRS Functional Specification

Functional Specifications for a Video Rental Store (VRS)

Overview of VRS system

The 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.

The actors in the system include customer, staff, and manager. Customers don't access the system directly. They interact with staff and staff interacts with the system to perform the transactions, which are defined in the form of use cases. The use cases performed by staff include Log in, Maintain Customer, Validate Customer, Process Rents, Process Returns, Maintain Inventory, Generate Reports, Pay Fee, Check Overdue, etc. Manager is also a staff so the manager has the right to perform all the activities that a staff can do. Apart from the regular business activities, manager is in charge of the system by starting up and shutting down the system. Manager also can override any management activities. The description of each use case is as follows:

Use case 1: Process Rents

As a primary use case, the purpose of Process Rents is to capture rental items along with payment. A store employee (staff) checks out items for a customer by calculating due dates and correct charges. The use case also checks any overdue items. The store employee accepts payments for the items and any late payments. A rental slip is issued and stored by the store employee. As preconditions, the staff is logged on and the barcodes of items are known. A

customer arrives at the point of sell terminal (POST) with tapes to rent. If the person is a new customer, the staff creates an account for the customer. If the person is an existing customer, the staff verifies customer status by scanning the ID card. System finds customer information and displays it. System also checks overdue items and displays associated information like item information, due date, and fine. The staff records new rental items by scanning the bar codes. This is repeated for all the items to be checked out. Rental object and rental line item objects are created. System determines the rental price and due date and displays title, due date, and price in the transaction. On completion of item entry, the staff indicates to the POST that the item entry is complete. System records the date, calculates tax, and displays the total rental fee. Any overdue fee is added up to customer's outstanding balance. Customer pays the amount due either by cash, check, debit card, or credit card. When paid by debit card or credit card, system contacts card authorization system, which validates the card and process the payment. If pay fee is successfully done, inventory is reduced for the rental items. Receipt is printed. All transaction data is saved onto the database. The staff gives items and receipt to customer and stores the rental order list.

In other successful scenarios, if scanning ID card fails, the staff enters ID number manually. If the ID number fails, the staff searches the system using customer's last name and phone number. In unsuccessful scenarios, if customer challenges the overdue fee, the staff may waive the overdue fee, reduce it by 50% or call management.

Use case 2: Maintain Customer

In use case Maintain Customer, a store employee creates an account for a new customer by inputting customer information, i.e. last name, first name, address, zip code, phone number, rental items, outstanding balance, etc. When customer asks for personal information update, the staff logs on the system and changes the corresponding information. When processing rents, the rental records of the customer get updated.

Use case 3: Process Returns

In use case Process Returns, customer brings the rental tapes to the counter. The staff scans the rental items one by one. The system takes in the scanned bar code and finds the corresponding customer and associated rental records. If scanning the bar code fails, the staff enters the code manually. If the code fails, the title of the item is entered to search through the system. The system checks overdue. If the item is overdue, the overdue fee is calculated and applied to the outstanding balance. The system proceeds to process payment. A receipt is printed for the customer. The system updates the inventory by increasing the returned item and updates the rental records by resetting the rental items and payment.

Use case 4: Maintain Inventory

Inventory has four types of media, which are CD, DVD, Video, or Game. The items with the same title could have different media types. And each title of a certain media type usually has multiple copies of items. Each item has one unique ID for the system to keep track of it. When new purchased items arrive, the staff creates new entry for items with new titles by inputting the title name, number of copies, available number, purchase cost, date obtained, etc.

If the title already exists in the system, the number of copies and available number of that title get updated by incrementing the number of items.

After a rental is processed, the number of available items under the title is reduced by one. After a return is processed, the number of available items under the title is increased by one.

Business Ontology

1. {customer, actors in the system who obtain products or services from the company in most cases by paying, roles, actor, null, (customer ID, first name, last name, phone, address), null, customer of a certain product, (potential customer, user, subscriber), (account, payment, product)}
2. {product, products or services produced by the company with the intention to sell to the customers, objects, null, null, (product components, product ID, price), null, (video tapes, software, book, consulting service), (item, merchandise), (payment, account, inventory)}
3. {account, customer account maintained by the company that keeps the customer information including personal information and transaction records, objects, customer information, customer maintenance, (customer, outstanding balance, transaction, date), null, user account, (customer record, transaction record), (payment, customer, transaction)}
4. {payment, payment made by customers while purchasing products, processes, transaction, (amount due, amount paid, date, transaction), null, (cash, check, debit card, credit card), null, (price, balance, transaction, customer)}
5. {employee, individual employed by the company, roles, (project group, division), group, (first name, last name, ID number, address, phone number, title, division), staff, (manager, technician, developer), (personae, staff), (human resource, project group, payroll, performance record, information systems)}
6. {order, a transaction made as an agreement to purchase the products, processes, transaction, null, (amount due, amount paid, date, purchased item), transaction, (mail order, online order, phone order), (purchase, reservation), (payment, customer, product)}
7. {credit card, a payment method used to process the purchase of products, objects, payment, order, (credit card number, CC type, expiration date, card holder name, amount paid), payment, (master card, visa card), (debit card), (payment, customer, order)}