GENERATING JAVA CLASS SKELETON USING A NATURAL LANGUAGE INTERFACE

Ender ÖZCAN, Şadi Evren ŞEKER, Zeynep İlknur KARADENİZ

{eozcan, seseker, ikaradeniz}@cse.yeditepe.edu.tr
Yeditepe University, Department of Computer Engineering
Artificial Intelligence Laboratory (AR+I)
26 Ağustos Yerleşkesi Kayışdağı/İstanbul
Turkey

Abstract. An intelligent natural language interface based on Turkish Language is designed for creating Java class skeleton, listing the class and its members. This interface is developed as a part of a project named as TUJA, a tool for producing Java programs using Turkish sentences. Turkish sentences are converted into instances of schemata, representing classes and their members. Concept hierarchies are utilized for building the classes and their hierarchical representation for Java class skeleton generation. In this paper, the details of the design and the implementation are described and a sample run is provided.

1 Introduction

Programming languages are machine processible, precise and *mostly* unambiguous with predefined syntax and semantics. Still, a novice programmer spends a lot of effort in learning syntactic rules and at the same time developing general programming skills. Even an experienced programmer may face the same problems, if the programming language is a new one. On the other hand, natural languages are more declarative, flexible, powerful and richer, being useful even for occasional users. Also, the programmer may not know the language used in the resources, such as books, to learn a new programming language.

There are visual tools for creating object oriented designs, furthermore, generating Java/C++ skeletal programs, such as Rational Rose (an IBM product). Turkish to Java (TUJA) is a natural language processing (NLP) application, designed with two modes of operation, where each mode is to be implemented as a phase. First phase involves in building an interface for creating a skeletal Java program, including all classes, their attributes (data) and prototypes of member methods of each class. Second phase involves in enlarging the functionality of the same interface to convert each skeletal class into full Java programs by allowing users to express them in Turkish sentences. In this paper, details of the first phase of TUJA projects are described. TUJA accepts Turkish sentences,

describing a class, a member method or a member attribute of a class, using a conversational front end. Then the input is fed into an augmented transition network (ATN) 1 for parsing and semantic analysis. At the end of this process knowledge database is updated using the current command. Knowledge is represented using schemata. At any instant, the user can ask TUJA to produce the Java skeletal code, saving it into a file. Architecture of TUJA is illustrated in Fig. 1.

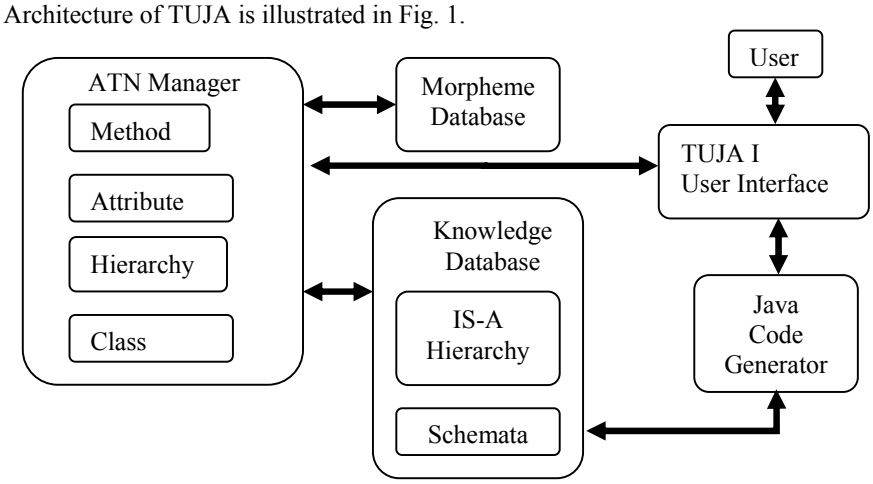


Fig. 1. Framework of TUJA.

2 Natural Language Processing using Turkish

NLP consists of 5 layers: morphology, syntax, semantics, pragmatics and phonetics (2, 10). Due to our scope and purposes we have limited our work in morphology and especially in syntax and semantics layers.

Turkish is one of the most widely spoken languages in the world, distributed over a large geographical region in Europe and Asia, as pointed in 6 based on United Nations sources. Note that there are many Turkish dialects, such as the Azeri, the Türkmen, the Tartar, the Uzbek, the Baskurti, the Nogay, the Kyrgyz, the Kazakh, the Yakuti, the Cuvas. Turkish is similar to Mongolian, Manchu-Tungus, Korean belonging to the same family of languages: the Altaic branch of the Ural-Altaic family. There are 29 letters based on the Latin alphabet in Turkish, including 8 vowels. There is a vowel harmony in Turkish words. Words do not have gender. In Turkish sentences adjectives precede nouns. It is unfortunate that there are a few number of natural language applications (1, [3],[5], 7-[9]) based on Turkish language due to its agglutinative nature.

The same suffix can be attached to different words in different ways. Sometimes, a vowel or a consonant towards the end of a word may deform. For this reason, morphological analysis in Turkish is not straightforward as shown in Table 1.

Table 1. Some deformation examples in Turkish words due to suffixes.

Word	**(Stem) + Suffixes**
görünürlerde (in sight)	(gör) + ün + ür + ler +de (görmek - to see)
ağaca (towards the tree)	(ağaç) + a (tree)
ağlıyor (he/she/it is crying)	(ağla) + yor (ağlamak – to cry)

There are seven morphological categories in Turkish: nouns, private nouns, compound nouns, adjectives, verbs, adverbs and conjunctions. In Turkish, another difficulty rises due to the syntax. Sentences with different syntaxes using the same words are allowed in Turkish (*free world-order*), yielding a group of sentences with the same meaning as illustrated in Table 2. The common property of all these three sentences is a feature of Turkish language, that is, the verb appears at the end of the sentences.

Table 2. Turkish sentences with different syntax having the same meaning.

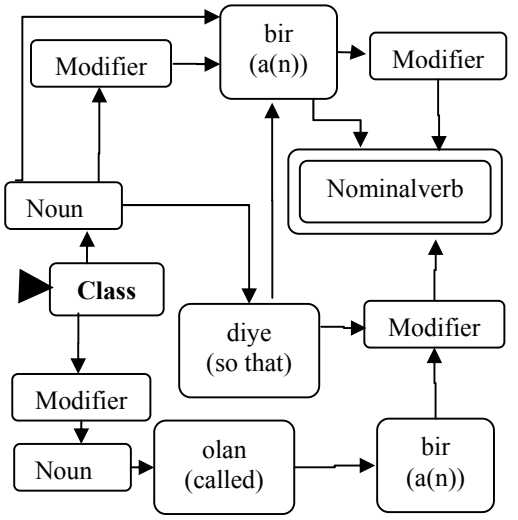
Sentence (I gave the book to the child)
Çocuğa kitabı ben verdim
Çocuğa ben kitabı verdim
Ben çocuğa kitabı verdim

3 Morphology, Syntax and Semantics

It is assumed that Object Oriented Programming terminology is known. Morphology of TUJA is inherited from a previous project, TUSA [6] based on PROLOG. The sentences are categorized into four different groups: (a) Class Declaration Sentences, (b) Attribute Declaration Sentences, (c) Method Declaration Sentences, (d) Relation Declaration Sentences. All possible syntax types are supported to create an abstract model representing the classes. An augmented transition network (ATN) is developed for TUJA interface. HASA relationship is used for composing classes and ISA relationship is used for building the class hierarchy. TUJA assumes that *in general*, a noun in a sentence refers to a class, interface, or an attribute (primitive or an object), and a verb refers to a method.

3.1 Class Declaration Sentences

This group of sentences is used to create a new class or name an existing class as shown in Fig. 2b. Note that declaration of abstract classes; Java interfaces are also supported.



(a)

- **nokta diye bir kavram vardır. (there exists a concept called point)**
- **şekil bir kuraldır. (shape is a rule)**

- liste soyut bir kavramdır. (list is an abstract concept)
- liste diye soyut bir kavram vardır. (there exists an abstract concept called list)
- bağlı liste soyut bir kavramdır. (link list is an abstract concept)

(b)

Fig. 2 (a) ATN, **(b)** some sample sentences for class declaration sentences.

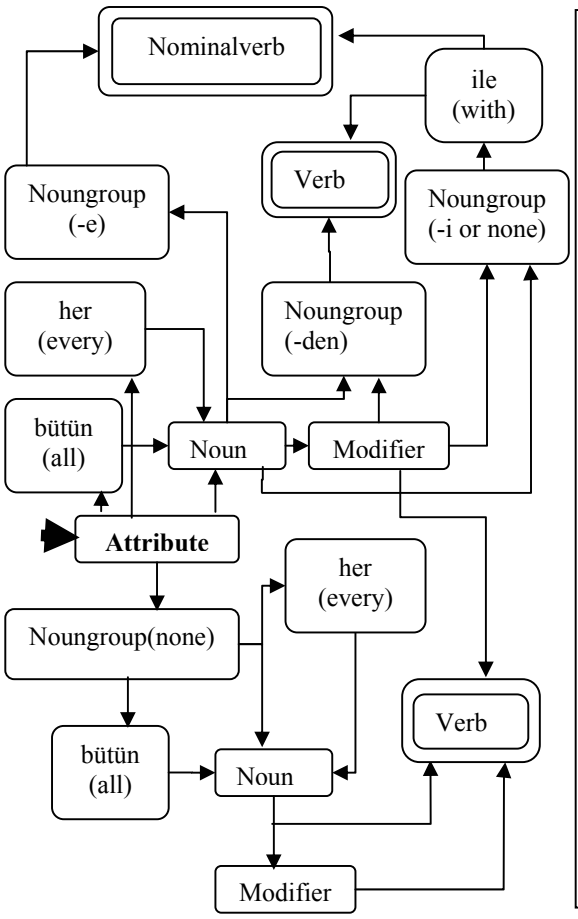
Part of the ATN for TUJA detects class declarations as illustrated in Fig. 2a. *Nominalverb* component alone and combined with the *Modifier* component in the ATN determines whether a class is abstract or not.

3.2 Attribute Declaration Sentences

This group of sentences is used to define the attributes of an existing class or to define a new class with specified attributes as shown Fig. 3. HAS relation represents the inclusion relationship, determining the elements included by an object. In other words, HAS relation is used to define the members of a class. Part of the ATN for TUJA detects attribute declarations as shown in Fig. 3.

3. 3 Method Declaration Sentences

This group of sentences is used to define the methods of predefined classes or to define a new class with specified member methods as shown in Fig. 4b. Part of the ATN for TUJA determines method declarations as shown in Fig. 4a.



(b)

- **nokta x adlı korunan tamsayıya sahiptir (point has a protected integer named x)**
- **nokta y adlı korunan tamsayıya sahiptir (point has a protected integer named y)**
- **çember çap adlı korunan çifte sahiptir (circle has a protected double named radius)**

- liste kuralı yeni uzunluk adlı tamsayıya sahiptir (list rule has an integer named new length)
- sıra soyut kavramı eski uzunluk adlı tamsayıya sahiptir (the abstract concept named queue has an integer named old length)
- her otomobil renge modele markaya sahiptir (all automobiles has color, model and brand)
- bütün otomobil kavramları rengi markası modeli ile tanımlanabilir (all automobiles can be defined by color, brand and models)

Fig. 3 (a) ATN, **(b)** sample sentences for attribute (HASA relationship) declaration sentences.

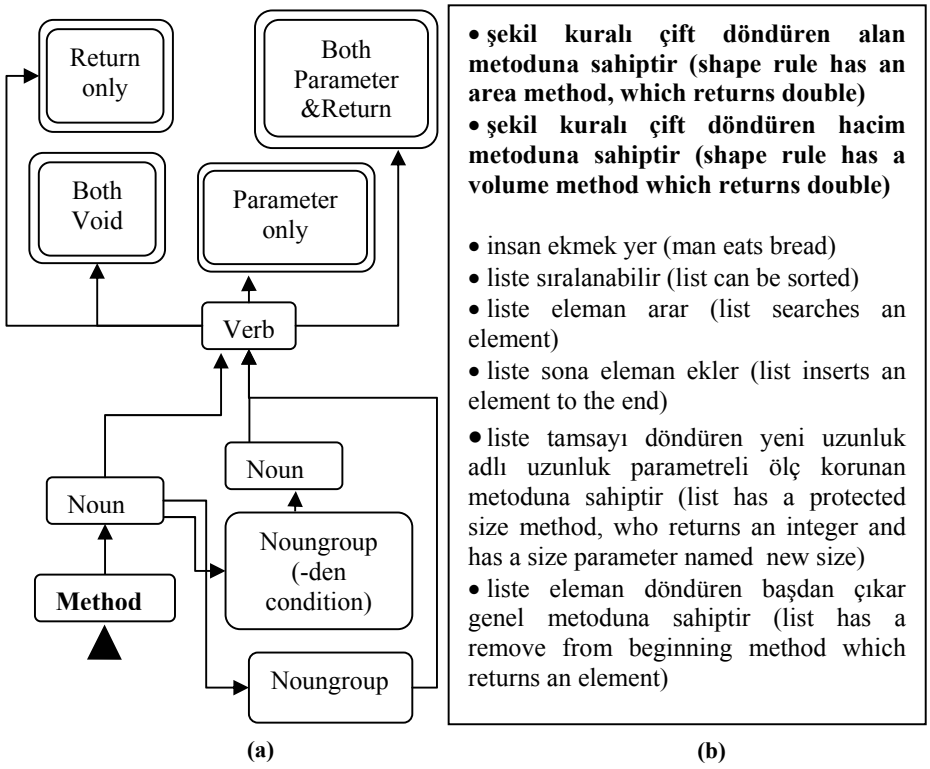


Fig. 4. (a) ATN, **(b)** sample sentences for method declaration sentences.

3.4 Hierarchy Declaration Sentences

Our knowledge about the world can be organized hierarchically using a naming convention for each class including a set of objects with common properties. For example, *cows* and *horses* represent two different set of objects, and *mammals* contains both of them as a super class. Note that cows and horses carry the properties of mammals. Similarly objects defined by a formal object oriented programming language can be organized hierarchically, forming a class hierarchy, supporting inheritance.
ISA hierarchy is used to represent inclusion relationship between classes. A class can be defined to be a subset of two or more super classes (multiple inheritance). Since the goal is generating JAVA class skeletal codes and JAVA does not support multiple inheritance, such sentences are converted into JAVA class templates assuming that at least one of

them is a class and the rest are interfaces. Part of the ATN for TUJA detects relation declarations as shown in Fig. 5a.

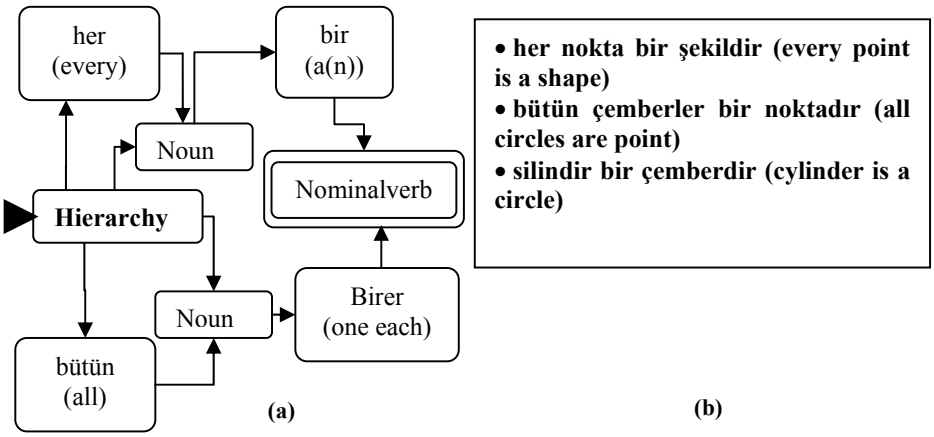


Fig. 5. (a) ATN, **(b)** sample sentences for hierarchy (ISA relationship) declaration sentences.

4 Knowledge Database

Knowledge database keeps all valuable data resulting from the object oriented design, retrieved from sentences, necessary to generate the skeleton of each class. After parsing and understanding a sentence, TUJA converts the input into an appropriate instance of a schema or modifies an existing schema in the knowledge database. PROLOG provides the advantage of keeping the instances in a relational database form. There are three basic schemata, each generated by the corresponding ATN: (a) Method Schema, (b) Attribute Schema, (c) Class/Interface Schema. Furthermore, ISA hierarchy is embedded into the class schema.

Details of each schema are shown in Table 3. ISA hierarchy is supported using, *InheritedClass* slot of the class schema, a pointer to the parent class. *ClassSpecifier* slot keeps the access specifier of the related class (e.g. public, private, protected).

Table 3. Schemata details supported by TUJA.

class/interface(ClassName, InheritedClass, ListofAttributes, ListofMethods, ListofImplementedIntefaces, ClassSpecifier)
attribute(AttributeName, AttributeType, AttributeSpecifier)
method(MethodName, ListofParameters, ListofReturnValues,MethodSpecifier)

4.1 Method Schema

Besides certain sentences, such as, “get X as input and return Y value”, there are uncertainties for the verb in some sentences like “Routers send messages”. In this specific example, it is obvious that the verb “send” gets “message” as a parameter. On the other hand, there might be an input, such as, “Routers generate error message”, where the verb of the sentence “generate” should return “error message” as an object. In order to retrieve the prototype of a method fully, verbs in Turkish are categorized as follows (Table 4):

- *Consuming verbs* identify methods requiring a list of parameters with no return value
- *Producing verbs* identify methods requiring no parameters and returning a value
- *Unaffecting verbs* identify methods requiring no parameters with no return value
- *Modifying verbs* identify methods requiring a list of parameters and returning a value

Table 4. Verb categories supported by TUJA to determine method parameters and return values.

Verb Type	**Example**	**Parameters**	**Return**
Consuming	“İnsanlar ekmek yer” (*Human-beings eat bread*)	Ekmek (bread)	Void
Producing	“At tay doğurur” (*A horse gives birth to a foal*)	Void	Tay (foal)
Unaffecting	“At hızlanır” (*The horse speeds up)*	Void	Void
Modifying	“İnsan undan sudan ekmek pişirir” (*Man cooks bread from flour and water)*	Un, su (Flour, water)	Ekmek (bread)

There is a special case for modifying verbs: “İnsan kuş avlar” (*Man hunts for bird*). The question, “what is the result of the hunt?”, can be answered as “bird”, just as the question, “what does man hunt for?”. Both return value and parameter of the method are the same. ATN parses Turkish sentences, and if a sentence defines a method, nouns are identified as a list of parameters or a list of return values of the related method.

In Java, a method may have only one return value, but our sentence might contain more than one return values. For example, “insan undan sudan ekmek börek pişirir” (man cooks bread and pastry from flavor and water) consist of two return values; bread and pastry. In such cases, TUJA produces two functions, one with return type of bread, second with return type of pastry with different function names.

5 Code Generation

User can trigger the code generation at any time by using the “kodla”(code) command at the prompt. TUJA code generator is a Prolog based predicate to java code translator.

Schemata, kept in the memory all the time, are processed and the code is generated. This way of approach provides flexibility to handle any changes in the declarations.

TUJA uses its defaults in the case of obscurities in access modifiers, getter/setter methods and interface rules. TUJA generates all member methods as public and all member attributes as private, generating public getter and setter methods for each attribute during the code generation.

TUJA inherits all the member methods of an interface, generating each prototype of the method as a member inside the class implementing it.

Following output is produced after feeding the input sentences in Figures 2b, 3b, 4b, 5b, formatted as bold. This example is the class skeleton of the Java codes provided as a case study in 4:

```
class Point implements Shape{
 protected int x;
 public int getX(){return x;}
 public void setX(int x){this.x=x;}
 protected int y;
 public int getY(){return y;}
 public void setY(int y){this.y=y;}
 public double volume(){}
 public double area(){}
}
class Circle extends Point{
 protected double radius;
 public double getRadius(){return radius;}
 public void setRadius(double radius){this.radius=radius;}
 public double area(){}
}
class Cylinder extends Circle{
 protected double heigth;
 public double getHeigth(){return heigth;}
 public void setHeigth(double heigth){this.heigth=heigth;}
 public double volume(){}
 public double area(){}
}
interface Shape{
 public double volume(){}
 public double area(){}
}
```

6 Conclusion

TUJA is a project for creating skeletal Java codes using our natural language, Turkish. This project will be extended further allowing users to enter Turkish sentences, producing the body of each member method. Yet, there are some issues to be attacked before this

next phase. For example, nested class declarations are not allowed. By the time being, TUJA does not handle the cases, if an interface extends another. Supporting punctuations and anaphora resolution will make TUJA to accept more *natural* sentences.

At the best of our knowledge, this is the first time verbs are categorized for retrieving possible parameters and return values from the sentences. TUJA is useful for experienced programmers. Novice programmers should be familiar with at least object oriented concepts. Since the code generator and knowledge database is separate, TUJA can be easily modified to support other object oriented or object based programming languages. Adapting the system to use a different natural language requires more effort, necessitating modification of ATN Manager and morpheme database components of TUJA.

7 Acknowledgement

Special thanks to Prof. Dr. A. C. Cem SAY from Boğaziçi University for providing the morphological analyzer.

References

1. A. Cetinoglu, Prolog Based Natural Language Processing Infrastructure for Turkish, M.Sc. Thesis, Bogazici University, 2001
2. M.A. Covington, *Natural Language Programming for Prolog Programmers*, (Englewood Cliffs, NJ:Prentice-Hall, 1994)
3. O. N. Darcan, An intelligent database interface for Turkish, M.Sc. Thesis, Bogazici University, Istanbul, 1991
4. H. M.Deitel, P. J. Deitel, Java: How To Program, 1998, 389-392
5. S. Demir, Improved treatment of word meaning in a Turkish conversational agent, M.Sc. Thesis, Bogazici University, Istanbul, 2003.
6. M.U.Karakas, E. Inan, *Current Status in Turkish Code Table Problem*, Bilisim, Bogazici University, Istanbul, 1996
7. K. Köymen, Prototype DMBS with a Turkish Query Language, University of Petroleum and Minerals, Dhahran, Saudi Arabia, 1976
8. S.E. Seker, Türkçe Doğal Dil Arayüzlü Bir Kişisel Takvim Programının, Tasarım ve Kodlamasi, *TAINN 2003*, Canakkale, Turkey.
9. S.E. Seker, A Personal Assistant with A Natural Language Interface in Turkish, M.Sc. Thesis, Yeditepe University, Istanbul, 2003.
10. J. Weizenbaum, ELIZA: A Computer Program for the Study of Natural Language Communication between Man and Machine, ACM Press, NY, USA, 1983, 23-28
11. W. A. Woods, Transition Network Grammars for Natural Language Analysis, Harvard University, Cambridge, Massachusetts, 1970