1-1-2012

Java API-Aware Code Generation Engine: A Prototype

Chandra Sekhar Vijyapurpu
Utah State University

JAVA API-AWARE CODE GENERATION ENGINE: A PROTOTYPE

by

Chandra Sekhar Vijyapurpu

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Stephen W. Clyde Major Professor	Nicholas Flann Committee Member
Curtis Dyreson Committee Member	Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2012

ABSTRACT

Java API-Aware Code Generation Engine: A Prototype

by

Chandra Sekhar Vijyapurpu, Master of Science

Utah State University, 2011

Major Professor: Dr. Stephen W. Clyde
Department: Computer Science

Software reuse enhances a programmer's productivity and reduces programming errors. Improving software reuse through libraries and frameworks is a vast problem area. This thesis offers an approach to solve two sub-problems within the problem area – to identify the right library components, and to offer code snippets that use the components correctly. The *Java API-aware Code Generation Engine*, or JAGE for short, is a prototype system that demonstrates the feasibility of generating semantically valid code snippets consisting of method calls to classes in the J2SDK library.

Developers often search for sample code snippets that describe how to use the library. This thesis describes the design and implementation of JAGE, which allows software developers to use an English sentence to generate helpful code snippets in Java. This thesis also discusses the related concepts in natural-language processing including ontology, Wordnet, and object-orientation in the area of automatic code snippet generation.

(71 pages)

PUBLIC ABSTRACT

One of Dr. Stephen Clyde's research interests at the Department of Computer Science, Utah State University, is object-oriented software design. Recent advances in the area of natural-language processing and the need for a better specification of aspect-pointcuts spurned our interest in describing object oriented elements.

Chandra Sekhar Vijyapurpu's thesis proposes a way to describe a subset of Java SDK's library elements and generate valid code snippets from a subset of English language sentences. This thesis also describes the design of a prototype based on proposal and its results.

ACKNOWLEDGMENTS

I am deeply indebted to my major advisor, Professor Stephen Clyde, who kept motivating me to finish my thesis, for his invaluable advice, guidance, and constant patience. I would also like to thank the members of my committee, Dr. Nicholas Flann and Dr. Curtis Dyreson. Their suggestions, support, and hard work through my tight schedule are appreciated.

Many thanks to Myra Cook; her great work and help were indispensable for completing the thesis.

I would also like to thank my colleagues and friends. Thanks for guiding me to become familiar with the project, as well as resolving many problems along the way.

Finally, I appreciate the continuous support and encouragement of my beloved family through the duration of my academic pursuits.

Chandra Sekhar Vijyapurpu

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Software reuse reduces software development effort and time, and hence saves money [1]. Software libraries and frameworks, which are forms of reuse, aid programmers by abstracting low-level details of implementation and providing more user-friendly handles to do the same task. Object-oriented languages have been gaining a wide user-base in the software industry since the introduction of Java in 1995 [2]. Java remains one of the mainstream object-oriented programming languages, along with C# and C++. The J2SDK library provided by the developers of Java makes writing useful and complex applications possible [3].

There are a large number of libraries and frameworks that have been written by different developers in open-source[1], as well as proprietary libraries for use with Java. For example, J2SDK has a very large number (running into thousands) of classes and methods [4]. Most developers are unaware of what possible open-source or commercial libraries and frameworks are available that might help them with their programming task. Coping with volume of library components and frameworks is the first hurdle to effectively reusing them.

Search engines solve a part of the volume problem by performing free text searches on documents containing keywords. The burden of skimming through all the irrelevant documents and choosing the right usage scenario might solve the problem; however, the burden of such an endeavor makes this solution less than ideal. Other

[1] http://sourceforge.net/search/?&fq[]=trove%3A198&fq[]=trove%3A15 is one example of a place where software developers can browse through and contribute to actual source code of a variety of Java software applications

approaches suggest querying repositories of usage scenarios to match the developer's intention with the code snippet in the repository using heuristics. While these approaches and some domain-specific tools address the problem to some extent, the general problem of recommending usable code snippets with a simple user interface remains hard.

Another problem with software re-use is that developers must adhere to the API specification of the library when using it, but compilers do not enforce the usage protocol of a library. For example, a programmer cannot call any instance methods on an object reference without assigning it to a valid object instance. Compilers cannot check for such mistakes. Illegal sequences of method calls for an API lead to program failures during execution time.

A static protocol checker could validate correct use of a library at compile time, if the library includes formal definitions for its usages. The disadvantage of this approach is that it is tedious to create such formal definitions. Expressiveness of the formalisms for defining protocols is still limited. As a consequence, the ability for formal checkers to find all possible problems is restricted. Not surprisingly, formal protocol definitions and static protocol checkers are uncommon even for the most common libraries.

A more common approach is for library developers to document the protocols informally in simple text documents or in the code directly and then use tools, like *Javadoc* [5], to create such text documents. Such informal documentation might say, "It is mandatory to open a *java.io.Socket* before reading from it." Even informal protocol

documentation is difficult to create and maintain. As a consequence, protocol documentation tends to be incomplete and less helpful than programmers would desire.

Another approach is to describe these protocols through sample usage scenarios. However, this puts a large burden on the library developer to write sample scenarios for each possible use-case his library can support. Most often these scenarios are documented through sample programs. While some libraries come with a thorough set of sample programs, most libraries do not.

While improving reuse through libraries and frameworks is a vast problem area, this thesis focuses on two small sub-problems that, if solved, would significantly increase programmer productivity and reduce program errors. These two sub-problems are a) helping programmers find appropriate components from a collection of libraries and b) assisting programmer in using those components correctly.

This thesis introduces a two-step based approach to these problems in the context of object-oriented libraries, like Java's SDK. Specifically, it introduces a tool, called JAGE (Java API-Aware Code Generation Engine), which constructs correct code snippets from natural-language statements provided by the programmer. First, it parses natural-language statements into their constituent phrases and then uses information-retrieval techniques to search for appropriate library components. It next uses the knowledge of semi-formal protocol descriptions to generate viable code.

The key contributions of JAGE are the techniques it employs in its semantic matching and code-snippet generation process. JAGE's semantic matching first builds on the notion that objects in the object-oriented world are *things*; their attributes link

them to other *things*. It builds on the notion that an object's methods are ways others can perform actions on that object. In natural-language, objects and their attributes are expressed by nouns or noun phrases, whereas actions are expressed by verbs or verb phrases. JAGE's semantic-matching extracts noun phrases from a natural-language sentence and uses them to find relevant components in the object-oriented library, as well determine what instances (objects) need to be created from these components. It also extracts verb phrases from a sentence to determine what actions need to be performed on this component in code that it will generate.

JAGE's code generation process relies on class *typestates*, which are an idea proposed by Strom, et al., for enhancing program reliability [6]. Specifically, typestates determine the permitted sequences of operations on a class depending on some context. Collectively, the set of permitted sequences is referred to as *protocol*. JAGE uses the typestate specification of a class to determine what method calls need to be inserted to generate a viable code snippet. JAGE will generate code snippets containing the actual sequence of method calls that forms the essence of the solution along with the *pre* and *post* method calls necessary to adhere to the protocol.

Chapter 2 provides the reader with more background on typestates and other technologies used by JAGE. The full details of the natural-language and code generation are discussed in Chapter 3. Chapter 4 describes the implementation and results of a prototype. This effort led to insights about JAGE's possibilities and limitations, which are described in Chapter 5, along with ideas for future work. Chapter 6 discusses related work.

CHAPTER 2

BACKGROUND

This chapter describes the key concepts used in JAGE, namely, natural-language processing, WordNet, ontology, signature-matching, and typestates.

2.1 Natural-language Processing

Natural-language processing (NLP) is the field of computing that enables human-computer interaction through natural-language [7]. NLP involves various techniques that allow computers to understand natural-language as spoken or written by humans, or to generate natural-language as heard or read by humans.

NLP facilitates communication between humans and machines, which at the basic level only understand machine instructions and binary data. Compilers can translate a sentence in a context-free programming language to machine instructions, and existing development environments can help programmers write sentences in such languages. However, natural-languages are more complex; they are more complex than context-free languages [8]. The meaning of a phrase can depend on the previous sentence or even the background of the involved human languages. Natural-language parsers attempt to bridge the gap between context-sensitive and context-free languages.

While a sentence in context-free language has one parse tree representation, a sentence in natural-language can have many possible parse-tree representations. A statistical parser chooses the most likely parse-tree representation based on word-use probabilities in a large training dataset [9]. As Chapter 3 explains in detail, JAGE delegates the burden of natural-language parsing to an open-source statistical parser

developed by the Stanford NLP research group [10].

Like most software systems that accept natural-language input, JAGE needs to place bounds on the input domain. Specifically, it needs to constrain the acceptable sentence structures. To do this, it adapts a technique, called *case frames* [11], that have been previously used for translating natural-language into machine-understandable data-structures and for information retrieval [12]. A case frame is an ordered list of *slots*, wherein each slot represents some grammatical construct, e.g., subject, verb phrase, direct object, indirect object [13].

As a notation, we represent a case frame as a string containing parts-of-speech separated by "-". Each part-of-speech represents an empty slot in which only the instances of the specific parts-of-speech will fit when the case frame is applied to a sentence. For example, when the case frame represented by *PRONOUN-VERB* is applied to the phrase, "I ran", then the PRONOUN slot has the value I and the VERB slot has the value ran. This case frame accepts "I ran", and "You sleep". However, it will reject "Joe ran" because the first word in this sentence is a noun and not a pronoun. In addition, "I ran fast" will be rejected, since this sentence has an additional adverb following the verb. The case frame PRONOUN-VERB-ADVERB, on the other hand, would accept this sentence.

As a whole, a case frame is a template that represents a specific sentence structure. Table 2.1 lists five common sentence structures in the English language along with examples of acceptable sentences that adhere to the case frame and unacceptable examples for that case frame. Since each case frame accepts only those sentences whose

slots can be exactly filled, a set of case frames can describe a domain of acceptable natural-language input for a system.

Table 2.1 Common Case Frames in English.

Sentence structure (case frame)	Acceptable	Unacceptable
PRONOUN VERB NOUN	I am John. I ran home.	I am happy.
PRONOUN VERB ADJECTIVE	I feel happy. I am surprised.	I run fast.
PRONOUN VERB ADVERB	I run fast. I sleep heavily.	I feel happy.
PRONOUN VERB PRONOUN VERB	Where do you live? How do I look?	Where is your home?
PRONOUN VERB VERB NOUN	What is your name? Where is your home?	Where do you live?

2.2 WordNet

WordNet [14] is a large database of words in the English language that organizes words into groups, or *synsets*, based on their meaning or semantic relevance. WordNet also has information about relationships, such as antonyms, between such synsets. An English word can be a part of several synsets. For example, pick as a noun is synonymous with selection, while as a verb it is synonymous with blame. WordNet has a file-backed lexicon of these relationships and offers several open-source APIs to enable systems to interact with it programmatically. JAGE uses JWNL [15].

2.3 Ontology

An ontology is an alphabet to frame the facts for a domain [16] and could provide JAGE with the primitives to express concepts and relationships between objects in a domain. W3C contains specifications for ontology description languages, like RDF

and OWL [17, 18]. An ontology acts like the glue that binds library components to machine-understandable concepts and, thus, enables a system to reason about the purpose of library components relative to programmer needs.

OWL2 Query Language [19,20] and SPARQL [21] are two standard languages used to express queries on knowledge databases or ontologies. These languages lend to a separation of concerns from the software design perspective and help JAGE manage the complexity of parsing and using the ontology. These languages also have frameworks like Jena, which enable programmatic access to the answering engine and for parsing the results in XML [22].

2.4 Signature Matching

Signature matching is a search technique used to retrieve a particular software library component (module, method, or procedure) based on the type of information in its method signature [23]. This technique requires the programming language to be strongly-typed like Java. Signature matching returns sets of exact and close matches of methods from the defined universe of a software library given a query string that describes the types of arguments and return value. For example, if the user requests input type as *java.io.File* and output type as *byte*, signature matching defines techniques to find the *read* method.

2.5 Typestates

Typestates are a semantic refinement of the concept of type. While a type defines the possible operations on itself, typestates define subsets of operations that are semantically valid on a type when it is in a particular state. Each row of Table 2.2

specifies a typestate transition for the *java.io.Socket* Class. The first two columns represent initial typestates, the third represent actions, and the fourth column represents final typestates, which are the results of the actions taking place.

Table 2.2 Typestates of java.io.Socket class.

Typestate			
Type	**Initial State**	**Method**	**Final Typestate**
java.io.Socket	Start	Socket()	Raw
java.io.Socket	Start	Socket(String host, int port)	Connected
java.io.Socket	Raw	connect(String host, int port)	Connected
java.io.Socket	Raw	close()	End
java.io.Socket	Connected	close()	End
java.io.Socket	Connected	getInputStream()	Connected

As per Table 2.2, a call to *getInputStream()* on the Socket object when it is in the *Raw* state is illegal. If a program has to issue a call to *getInputStream()*, it should make sure that the Socket object is in the Connected state. If a program issues the method-call, *getInputStream()* to an object reference when it is in the Raw state, the Java virtual environment will throw a sub-class of *java.lang.RuntimeException*. While typestates define the subset of sequences of operations that are valid on a type, sequences that do not conform to the typestates are invalid.

CHAPTER 3

DESIGN OF JAGE

This chapter focuses on the two objectives of JAGE: a) helping programmers find appropriate components from a collection of libraries, and b) assisting programmer in using those components correctly. To explore the feasibility and the detailed issues involved in solving these two problems, there is value in creating a prototype system that solves the problem in a limited context. This chapter describes a prototype code-generator that operates with the following restrictions: a) constrained grammar defined by set of acceptable case frames, b) constrained vocabulary, and c) limited library components. See Section 3.1 for further explanation and justification of these constraints. Sections 3.2 and 3.3 describe JAGE's architecture and workflow for the matching and generation processes.

3.1 Constraints

3.1.1 Grammar Constraints for the Prototype Using Case Frames

Research on NLP systems indicates that neither the system designer nor the users can predict all possible words or input sentences to a question-answering system on a database of facts [24]. This principle applies to JAGE, since the ontology for a library of components represents a database of facts. To keep JAGE, and particularly the initial prototype within the range of solvable problems and focused on its objectives, we place certain constraints on the domain of acceptable input sentences.

A convenient way to constrain the domain is to limit the number of acceptable

sentence structures, which JAGE does by pre-defining the possible case frames. Furthermore, JAGE restricts the sentences to be questions that start with the phrase "How do I", so the case frames only have to model the completion of the question. JAGE's initial prototype uses two pre-defined case frames, namely, a) VERB-ARTICLE-NOUN, and b) VERB-ARTICLE-NOUN-PREPOSITION-ARTICLE-NOUN. The first case frame models common questions about the public methods available for the classes in libraries. Below are some typical examples:

- Rename a file
- Open a socket
- Read a file

Even though each of these questions appear simply and relate directly to a single method call on a single object, their valid use within the context of a protocol may involve other objects and methods calls. For example, reading a file requires the file to be opened before the read and closed sometime after the read.

The second case frame includes prepositions, which allows user to ask questions that might involve more than one object. Below are some typical examples:

- Read a line from a file
- Write a file to a Socket

Although using just two case frames might seem limited, the domain of valid sentences is large and interesting, covering a wide range of the practical questions. Since the first case frame deals with methods on single object and the second covers questions involving two objects, the prototype can handle all method-call compositions

involving two classes. Even with these constraints, the prototype is interesting since it needs to find which methods to compose to obtain the required essence of the solution. The prototype then needs to generate the code required to put the objects in the right typestate to invoke the method-call forming the essence of the solution, which corresponds to one of the code structures – a) *y = x.m();* and b) *x.m(y);*

However, the prototype cannot handle method-call compositions involving more than two types of parameters. For example, the prototype cannot generate a code structure of the form, *x.m(y, z)*. A full-blown JAGE would need to implement case frames involving at least three more parts-of-speech, namely, prepositions, conjunctions and disjunctions. With additional case frames, JAGE would be able to generate method calls with any number of parameter or conditional control structures.

3.1.2 Vocabulary Constraints for the Prototype Using WordNet

If JAGE were to store all possible user words used in a particular part-of-speech in all possible user sentences, storage and search would be very expensive. Additionally, the user need not type in the exact word stored in JAGE, he can use any synonym of the stored word and JAGE can consult WordNet to match the two words. Hence, using WordNet in JAGE helped reduce the storage requirement and improve its user experience. While WordNet supports many inter-word relations, such as synonyms, antonyms, and hyponyms, JAGE is interested in the lexicon and the synonym relation. The synonym relation allows JAGE to search for library components that match the user intention but not its exact description.

Unfortunately, WordNet's lexicon is oriented towards common speech and not

programming jargon. For example, '*fetch*' and '*read'* are synonyms in WordNet, while '*get'* and '*read'* – two common synonyms in programming jargon – are not. To overcome this limitation, the prototype has added this instance explicitly to the lexicon of WordNet. Section 4.2 elaborates on this example and Appendix A has examples about how this relation is added to the lexicon. Even though the prototype accommodates some programming jargon, a full-blown JAGE would need a more substantial customization to the standard WordNet lexicon and synonym relations.

By using WordNet, JAGE's vocabulary is only limited to the extent that current the lexicon and synonyms capture English words and programmer jargon, which we do not view as a serious constraint relative to the purpose of the prototype.

3.1.3 Constraints on Library

According to DeLine and Fähnrich, usage protocols in object-oriented libraries are of two major types: *state-machine protocols* and *resource protocols* [25]. State-machine protocols describe which subset of operations is permitted by contract in the API documentation when the instance is in a particular state. Consider the *java.io.File* and *java.io.BufferedReader* classes. The state-machine protocol for these classes ensures that the object reference is prepared for a method to be called. For example, an object reference of type *java.io.File* will not be prepared to invoke the method *renameTo()* unless that object reference is tied to an instance in memory.

The prototype has the state-machine protocol information in a hash table for each type. The hash table contains the type of an object as the key, and the value is another hash table. The inner hash table contains the start state as the key for each entry

and an object containing the method name and target state as the value for the entry. This storage schema enables the prototype to search for available transitions on a state. However, each object type also has a start state and an end state associated with it. Each valid code snippet involving a method call on an object contains all the method invocations necessary to transition the state of the object from start state to end state while involving the actual method call as the essence of the code snippet.

Resource protocols describe object creation and destruction protocols. An example of a creation pattern in J2SDK is found in *java.util.Calendar* class. Since the *java.util.Calendar* class is abstract, instantiation is possible only through a static method, *getInstance()*, called directly on the *java.util.Calendar* class or by calling its subclass, e.g., *java.util.GregorianCalendar*, constructor.

The prototype cannot generate code adhering to resource protocols. Resource protocol information can be obtained by using API's of *java.lang.reflect* package. This package provides information about, which method is abstract, which constructor is private, and so on. To enable a full-blown JAGE to generate code adhering to resource protocols of a library, JAGE needs information about access modifiers, like *private*, and about keywords, like *static* and *abstract*. Resource protocols are tied to the Java language rather than to the library, which means that scale is not an issue for addition of these protocols.

3.1.4 Summary of Constraints

Within the bounds of constraints as described above, the JAGE prototype addresses interesting use cases that shed light on potential value and possible limitations

of a full natural-language-based code generator.

3.2 Workflow

The input to JAGE is an English sentence, and the output is a code snippet. The workflow is a two-step sequential process as shown in Figure 3.1. The first step in JAGE is semantic matching. The input to the semantic matching step is a user-specified sentence in English (See Section 3.3). The output of the semantic matching step is a set of classes and their methods that potentially match with what the user desired, which forms the input to the second step of JAGE – code-snippet generation (See Section 3.4). The output of the second step is a code snippet that provides code for the use-case. In other words, the code snippet helps the user understand how to use certain classes in the standard J2SDK library.

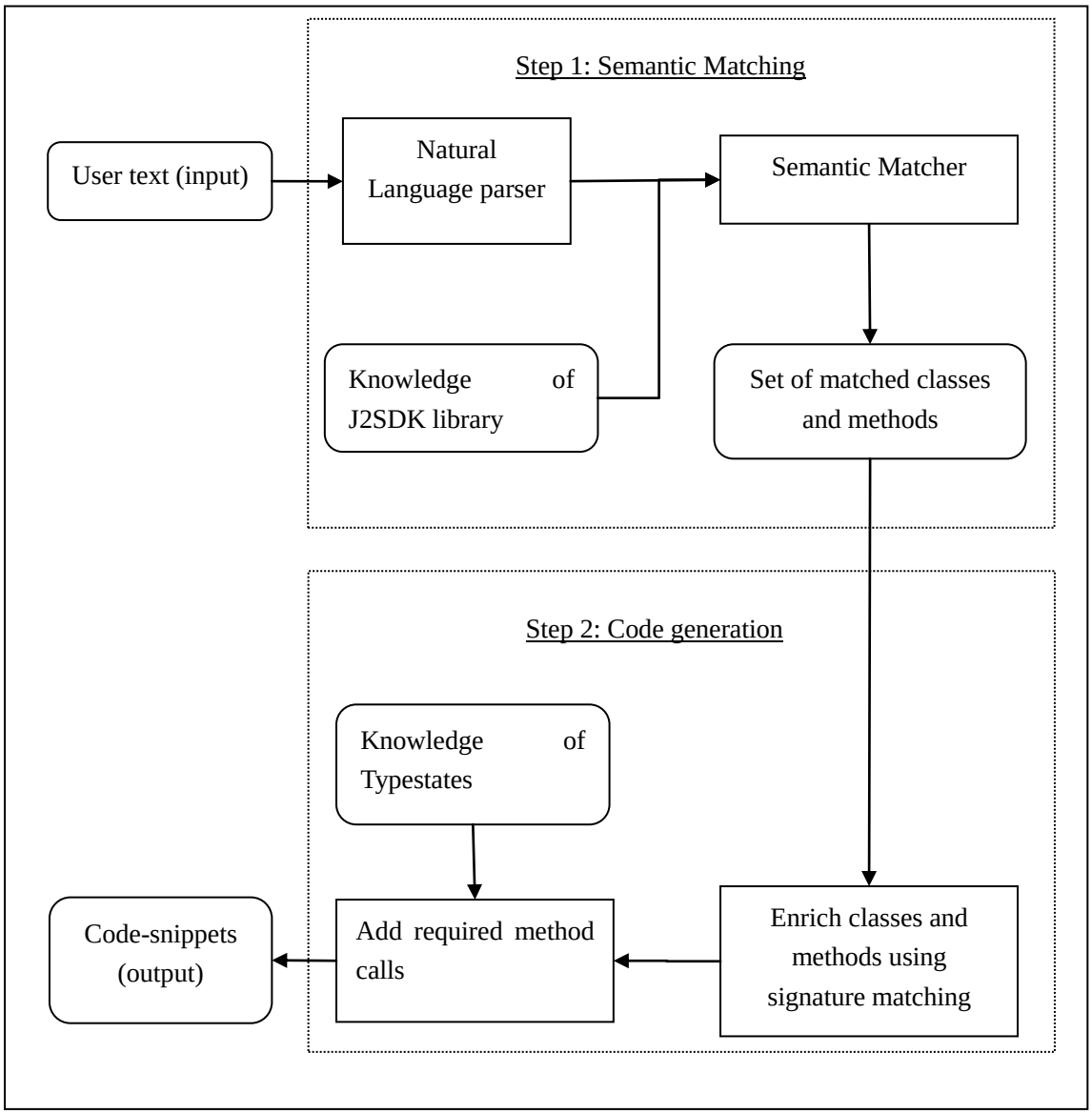

Figure 3.1 Overall workflow in JAGE.

3.3 Semantic Matching

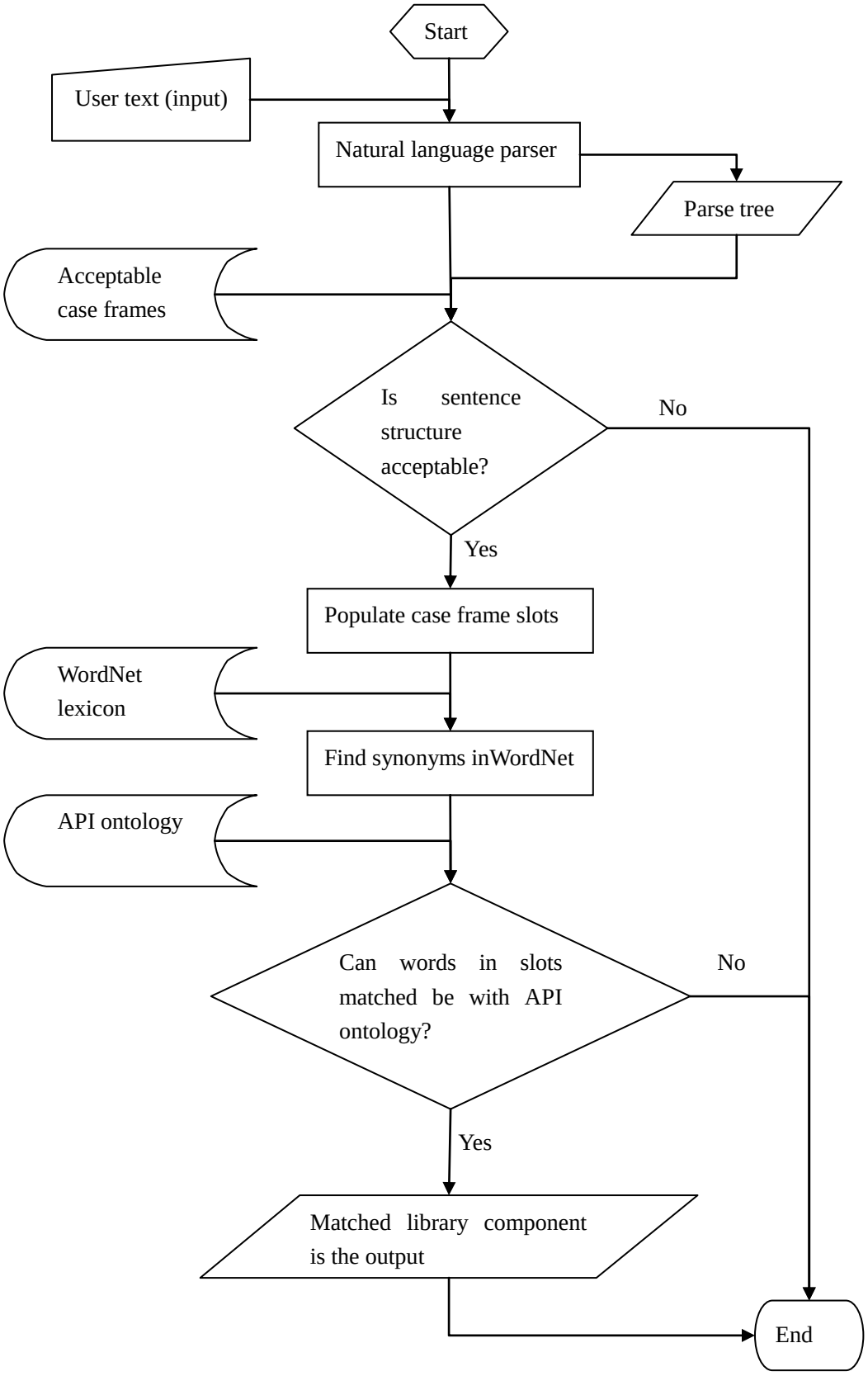

Figure 3.2 Semantic matching flowchart.

In semantic matching, shown in Figure 3.2, JAGE tries to retrieve a set of classes and methods to enable the second step of JAGE, code generation. To accomplish this step, the prototype of JAGE uses case frames, knowledge of an OWL ontology for the restricted library components, knowledge from WordNet, and a parse-tree representation of the user's sentence.

JAGE's user-interface is a textbox through which the user enters the English sentence that best describes his/her use-case. The textbox has the phrase, "How do I" as a prefix. The natural-language parser picks up the input English sentence and outputs a parse tree representation of the sentence. The semantic matcher has access to the knowledge of the J2SDK ontology, the WordNet lexicon through its API, and natural-language sentence structures hard-coded within it. The semantic matcher applies an algorithm, described in Section 3.2.4, to retrieve the right set of classes and methods that best satisfy the user's use-case as described in his input text.

3.3.1 Parsing Input Sentence

A free-text search is more like a regular-expression matching process [26]. However, a natural-language sentence is not just a simple bag of words. The underlying structure of the sentence also can impact the meaning of the words. For example, "read from a Socket to a File" and "read from a File to a Socket" have very different meanings. A free-text search based on both these sentence, however, would use the same bag of words, e.g., "read", "File", and "Socket". A natural-language parser can provide the addition information in the form a parse tree, where nodes can represent the subject, action, direct object, and other parts of speech. This is the reason behind using

a natural-language interface for the user input.

The user interface of JAGE is primitive – a simple text box that has the prefix, “How do I ….” The user completes the sentence by adding words into the text box. The reason behind this design decision is to avoid the complexity behind parsing the dependant clause of the input query. If the user is not provided with a prefix, JAGE needs to parse the sentence in its entirety to understand its meaning. For example, the developer may type in one of the following, “How to rename a file” or “What should I do to rename a file?” While both these queries are semantically identical, they are syntactically different. However, the dependant clause in both “How to” or “What should I do,” is immaterial to the actual search criteria that JAGE needs to perform. In addition to reducing the complexity of parsing, the prefix gives the developer a starting point to enter his query, making the user interface more intuitive, and hence more user-friendly.

Since the user input is in English, a machine cannot understand it directly. Natural-language parsing is the first step of semantic matching in JAGE. This sentence is run through the Stanford NLP probabilistic parser to get the parse tree representation of the input sentence. For example, the sentence, “How do I read a line from a file” gives the following output as a parse tree (see Figure 3.3).

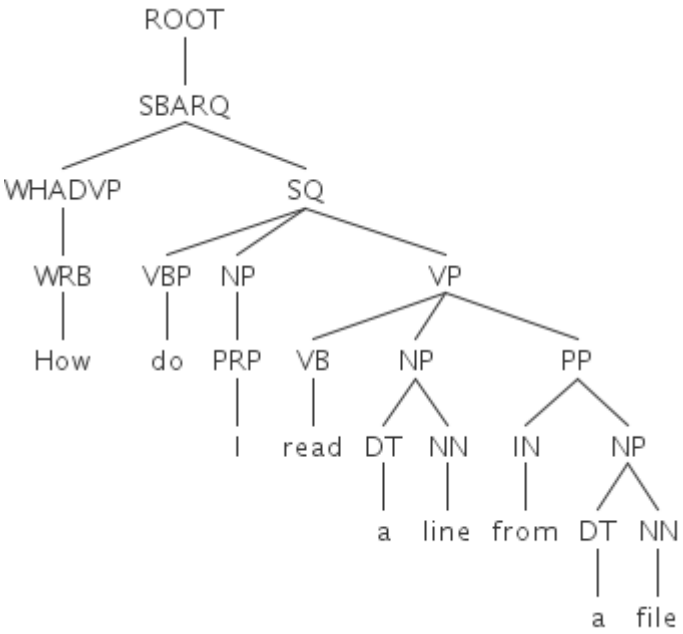

Figure 3.3 Parse tree for the sentence, “How do I read a line from a file”.

3.3.2 Populating the Slots of a Case Frame

JAGE uses the structure of the user sentence to determine whether it can extract the required data to proceed with semantic matching or not. The parse tree tags the words to parts-of-speech and also provides the phrases that constitute the input sentence. JAGE uses this information to match with phrase structures or case frames it is interested in.

The first part of rejecting or accepting an input sentence is straightforward – accept a sentence, even if one of the allowed case frames match. The second part of finding the right case frame is largely dependent on the output or accuracy of the natural-language parser. Parsing a natural-language sentence is a complex problem and is still an area of active research [27]. Delegating the process of transformation an English sentence into a parse tree to the open-source parsing technique allows the prototype to focus on other issues like semantic matching and code generation. Chapter

5 deals with possible future work in this aspect.

As explained earlier, the JAGE prototype supports only two case frames, and therefore the following two syntactic structures:

- VERB-ARTICLE-NOUN. An example is “rename a file.” This is the simplest query structure against the prototype. The ARTICLE is not considered in matching.
- VERB-ARTICLE-NOUN-PREPOSITION-ARTICLE-NOUN. An example sentence could be “read a line from a file.” Semantic matching in JAGE is currently at the parts-of-speech level. The first noun is the direct object of the verb and aids in semantic matching to find all methods indicated by the verb that are related to the class equivalent to the noun of the direct object by composition. The noun extracted from the prepositional phrase is the indirect object and helps in finding the class that needs some transformations, thus necessitating the code snippet generation step of JAGE.

These parts-of-speech values are used in SPARQL query templates and executed against OWL ontology to give JAGE the required classes and methods. However, this process is not straight-forward due to a related problem of the vocabulary of English, which the prototype tries to overcome by using WordNet.

3.3.3 Using Ontology

The prototype needs knowledge about what a class represents and what methods correspond to which actions. The library developer provides this information in XML

files adhering to specifications of OWL ontology [28]. The prototype uses an open-source inference engine inside Protégé [29], to reason about the available classes, their properties, and behaviors.

JAGE encodes knowledge of object-oriented concepts like inheritance in OWL so the machine can decide at runtime that *java.util.HashMap* implements the *java.util.Map* interface. The primitives in OWL, like *isA* and *hasA*, map well to the object-oriented concepts of inheritance and abstraction. For example, JAGE should be able to tell the machine what a method in a class does. Ontology acts like the language that both the API developer and the machine understand. The J2SDK library consists of various classes and packages that allow its user to write applications. JAGE uses knowledge from ontologies in its semantic matching step as well as in its code generation step.

Each J2SDK element (e.g. a class, a field, or a method) is associated with a phrase describing itself. A class has a noun-phrase while a method has a verb-phrase associated with it. Classes are arranged in a tree in the ontology according the inheritance hierarchy. Hence, if a parent class has a method, the child class also has the method. For the initial version, JAGE neglects the access modifiers of private, public, etc. Please refer to Appendix B for details of OWL representation of J2SDK elements.

SPARQL is used for querying ontology, as SQL is used for querying databases. JAGE uses the sentence structure to extract the verb phrase and noun phrase for constructing the SPARQL query. The substituted SPARQL query is then executed by Jena on the OWL ontology of J2SDK library. To extract the verb phrase, the substituted

SPARQL query for the sentence “Read a line from a file” looks like this.

```
SELECT ?class ?subject ?object WHERE {
?subject
<http://www.owl-ontologies.com/2009/8/16/Ontology1250442606.owl#verb_phrase> ?object.
subject
<http://www.owl-ontologies.com/2009/8/16/Ontology1250442606.owl#enclosing_class> ?class.

FILTER regex(?object, "^read a line$") }
```

Executing this query on OWL ontology gives the following output:

```
DefaultOWLIndividual(#BufferedReader of [(#Java_Class)])
DefaultOWLIndividual(#readLine of [#Method)]) read a line.
```

3.3.4 Semantic Matching Algorithm

Below are the five steps that comprise the matching algorithm.

Step 1: The user enters text in the input text box and clicks the find matches button.

Step 2: JAGE tries to match this sentence with sentence structures and returns an output of SPARQL query for querying the J2SDK ontology with substitutions for verb phrases and noun phrases.

Step 3: The user executes the SPARQL query against the J2SDK ontology in Protégé OWL.

Step 4: If a match found in Step 3, go to Step 7. Else go to Step 5.

Step 5: JAGE will try the next combination of synonyms suggested by computer science ontology and WordNet by replacing the verbs and nouns in the SPARQL query to query the J2SDK Ontology – go to Step 3.

Step 6: If all combinations of synonyms are exhausted, terminate with result “No matches found.”

Step 7: Use the output classes and methods in the next step of JAGE – code snippet generator.

The JAGE prototype returns the first match of the matching process. The semantic matching is hence partially dependent on the reasoning engine used by Protégé. The SPARQL query returns individuals from the OWL ontology, which is similar to the concept of objects of classes. The library elements returned by the query are the source type that has the method of interest and the target type that is the class of interest.

3.4 Code Snippet Generation

Code-snippet generation is a two-step process. The first step is to enrich the classes obtained from the semantic matching step. The second step is generating actual code.

3.4.1 Type-Based Composition

J2SDK offers library functions that enable listing of method signatures in a Java class. The package *java.lang.reflect* contains API to list signatures of methods and the inheritance information of classes. JAGE performs a variant of signature matching using J2SDK's reflection API with one level of breadth-first search. Imagine a multi-partite graph wherein at each stage, the methods of a class are the nodes. Each edge of this graph that joins vertices between two stages denotes a signature match. An edge between a method and a class is possible if the argument of the method is a type or super-type of the class. The goal is to find a path from a node in the first stage to the node in the last stage. For simplicity, JAGE uses one additional stage in between the

first and the last.

As an example, assume the input type to be *java.io.FileReader* and output type to be *java.io.File*. The class *java.io.FileReader* has a constructor that takes an argument of *java.io.File* type and returns a *java.io.FileReader*. In this case, JAGE needs to find a method of source type (*java.io.FileReader*) that accepts a target type (*java.io.File*).

For another example, assume the input type is *java.io.BufferedReader* and output type is *java.io.File*. There is no method in the class *java.io.BufferedReader* that takes in an argument of type *java.io.File*. However, *java.io.BufferedReader* accepts a *java.io.Reader* in one of its constructors of which *java.io.FileReader* is a child class. So, JAGE lists all distinct types of arguments that all the methods of source type require that return the source type. Then, JAGE searches for all the methods of these argument types that take in a target object type and return the argument object type.

JAGE can now synthesize a series of method calls, one on the source object type in conjunction with another method call on the argument type of the method in the source object type. This enables JAGE to obtain a reference to a *java.io.BufferedReader* from a *java.io.File* object through a *java.io.FileReader* object. This concept is akin to that of Jungloids [29] wherein a series of method calls or object type-casting results in a desired target object from a source object. However, the JAGE prototype does not implement type-casting and differs considerably in its use of signature matching to generate code snippets from Jungloid mining. For the following example, consider the source class of method *readLine()* to be *java.io.BufferedReader* and target class to be *java.io.File*. The output looks like the following code snippet.

```
File _file_1;
FileReader _fileReader_1 = new FileReader(_file_1);
BufferedReader _bufferedReader_1 = new
BufferedReader(_fileReader_1);

_bufferedReader_1.readLine();
```

3.4.2 Typestate-Based Composition

The second step of code generation ensures that the generated code adheres to the usage protocol of the classes as defined in their API. JAGE relies on the concept of typestates to get this information. While typestates have been used for static-checking [30], JAGE uses them to generate code that follows the usage protocol.

Consider the code snippet shown above. While the file object is declared, it is never constructed nor is it ever closed. Running this code as it is as a Java program will lead to a *java.io.IOException*. While it is easy to point that the null object in the trivial example mentioned as the object is never constructed, there could be more complex scenarios like binding a socket instance to an address. To capture the protocols of usage, JAGE uses typestate information.

Consider the table of typestates for *java.io.BufferedReader* (similar for *java.io.FileReader*) class shown in Table 3.1.

Table 3.1. Typestates for *java.io.BufferedReader* Class.

Initial State	**Method**	**Final State**
Start	BufferedReader(Reader)	Connected
Connected	readLine()	Connected
Connected	Close()	End
Start	Close()	End

To be able to call a *readLine()* method on a *java.io.BufferedReader* object, that object must be in connected state. This step in code snippet generation ensures that an object begins in start state and ends in end state. The first path possible using the typestate information as a graph helps generate code that adheres to the protocols of usage of a class. After this step, the code snippet looks like the following:

```
File _file_1 = new File(java.lang.String);
FileReader _fileReader_1 = new FileReader(_file_1);
BufferedReader _bufferedReader_1 = new
BufferedReader_fileReader_1);
_bufferedReader_1.readLine();
_bufferedReader_1.close();
_fileReader_1.close();
```

CHAPTER 4

SAMPLE WORKFLOWS

4.1 A Detailed Example

This section provides a detailed example of JAGE interpreting a sentence and generating relevant code snippets. Consider the user sentence to be “How do I make a directory.” Since the semantic matching step neglects the static “How do I” clause, this sentence structure matches the first case frame, VERB-ARTICLE-NOUN, specified in Section 3.2.2. Figure 4.1 shows the parse-tree representation of the clause, “make a directory”.

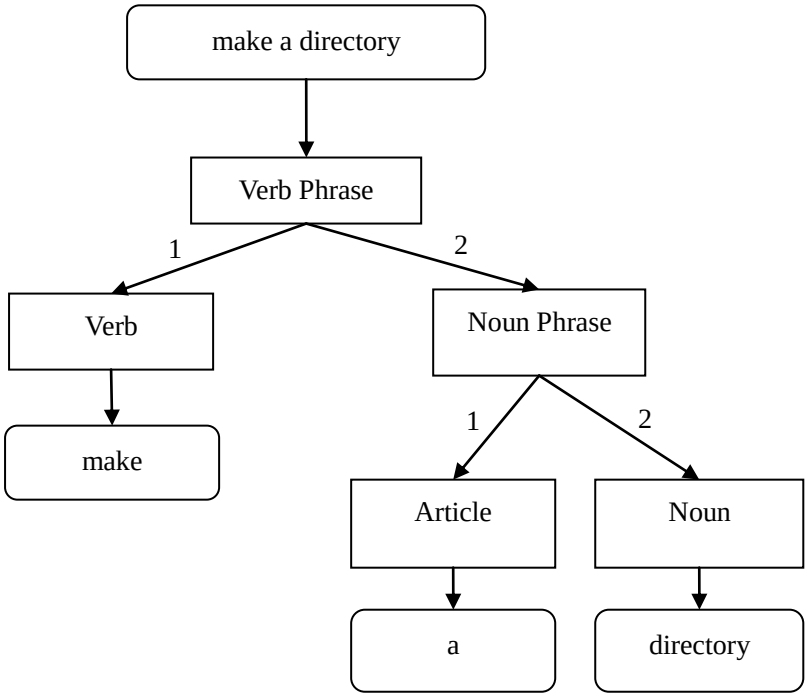

Figure 4.1. Parse sub-tree for the clause, “make a directory.”

The case-frame technique extracts the verb make and the noun directory for querying against the OWL ontology. The SPARQL query is constructed by substituting for these values for the parts-of-speech slots in the following query:

```
SELECT ?class ?subject ?full ?object WHERE { ?subject
<http://www.owl-ontologies.com/2009/8/16/Ontology1250442606.owl#
enclosing_class> ?class. ?subject
<http://www.owl-ontologies.com/2009/8/16/Ontology1250442606.owl#
verb_phrase> ?object. ?class
<http://www.owl-ontologies.com/2009/8/16/Ontology1250442606.owl#
qualified_name> ?full. FILTER regex(?object, "make") }
```

When this SPARQL query is executed against the OWL ontology of J2SDK, the output is a null set. JAGE then tries to get all synonyms of the verb make to fill the slot of the verb and executes against the ontology using WordNet. WordNet suggests create as a synonym to make, thus enabling JAGE to find a match for the verb in the ontology in the method, *mkdir()*. The noun File matches the class, *java.io.File*. The case frame dictates that the verb belonged to the direct object in the structure. Hence, JAGE returns a match of method, *mkdir()* in the class, *java.io.File*.

JAGE then uses the output of method *mkdir()* and class *java.io.File* to generate a code snippet. In this case, the first step of code snippet generation is simple, as no type-safe transformations are required to invoke the method on the class. So, the type-based composition returns the method *mkdir()* and class *java.io.File* to the next step of typestate-based code snippet generation.

Typestate specification of *java.io.File* mentions that the object of type *java.io.File* should be in the created state to invoke the method *mkdir()*. Table 4.1 gives some insight into typestate specification of *java.io.File* class.

Table 4.1. Typestates for *java.io.File* Class.

Initial State	**Method**	**Final State**
Start	File(String)	Created
Created	mkdir()	Created

The typestate specification in Table 4.1 dictates JAGE to generate a call to the contructor of *java.io.File*, *File(String pathname)* to take the object into created state. After invoking the constructor, JAGE generates a call to the *mkdir()* method on the *java.io.File* object. The final code snippet looks like this:

```
File _file_1 = new File(java.lang.String);

_file_1.mkdir();
```

4.2 Other Examples

This section provides results of the prototype on some additional examples of user input to shed further light on the overall effect of the design choices that went into the implementation. In all the examples below, the user input is shown in bold and the essence of the solution to the user query is highlighted in gray.

1. Case Frame: VERB-ARTICLE-NOUN. Section 4.1 had the simple case.
    a. To perform the action, read, a second type is required – *java.io.FileInputStream*

       Text input:
       How do I **read a file**

       Results of semantic matching:
       ```
       Found match in
       Source: class java.io.FileInputStream
       Target: class java.io.File
       ```

       Generated code snippet:
       ```
       File _file_1 = new File(java.lang.String);
       FileInputStream _fileInputStream_1 = new
       FileInputStream(_file_1);
       _fileInputStream_1.read();
       _fileInputStream_1.close();
       ```

2. Case Frame: VERB-ARTICLE-NOUN-PREPOSITION-ARTICLE-NOUN
   a. Happy case: Only two types are needed to answer the user query

      Text input:
      How do I **read a line from a file**

      Output of semantic matching:
      ```
      Found match in
      Source: class java.io.BufferedReader
      Target: class java.io.File
      ```

      Output of code snippet generation:
      ```
      File _file_1 = new File(java.lang.String);
      FileReader _fileReader_1 = new FileReader(_file_1);
      BufferedReader _bufferedReader_1 = new
      BufferedReader_fileReader_1);
      _bufferedReader_1.readLine();
      _bufferedReader_1.close();
      _fileReader_1.close();
      ```

   b. Preposition issue: The results of the semantic matching do not make any sense relative to the input sentence. However, the prototype thought that the user asked for "how do I read **from** a file". This is because the prototype does not consider the preposition in its semantic matching. (Please refer to section, 5.2.1.)

      Text input:
      How do I **read a line to a file**

      Results of semantic matching:
      ```
      Found match in
      Source: class java.io.BufferedReader
      Target: class java.io.File
      ```

      Generated code snippet generation:
      ```
      File _file_1 = new File(java.lang.String);
      FileReader _fileReader_1 = new FileReader(_file_1);
      BufferedReader _bufferedReader_1 = new
      BufferedReader_fileReader_1);
      _bufferedReader_1.readLine();
      _bufferedReader_1.close();
      _fileReader_1.close();
      ```

3. Undefined Case Frame: The prototype rejects a sentence when the parse tree from the parser does not match any pre-defined case frames.

   Text input:
   ```
   How do I not read a line from a file
   ```

   Output of semantic matching:
   ```
   User sentence not understood: How do I not read a line from a file
   ```

4. Affect of extending WordNet

a. Simple case: When the verb matches exactly the method description

Text input:
```
How do I write a line to a file
```

Output of semantic matching:
```
Found match in
Source: class java.io.PrintWriter
Target: class java.io.File
```

Output of code snippet generation:
```
File _file_1 = new File(java.lang.String);
PrintWriter _printwriter_1 = new PrintWriter_file_1);
_printwriter_1.println(java.lang.String);
_printwriter_1.close();
```

b. Before WordNet was extended: WordNet does not think put and write are synonyms. Please see section 5.2.1

Text input:
```
How do I put a line into a file
```

Output of semantic matching:
```
No matches found in OWL
```

The reason this sentence was rejected by the prototype is that WordNet did not think put and read are synonyms. Please see section 5.2.1 which describes an idea that can alleviate this problem.

c. After WordNet was extended: The synset for write was modified by adding put. Please see Appendix A

Text input:
```
How do I put a line into a file
```

Output of semantic matching:
```
Found match in
Source: class java.io.PrintWriter
Target: class java.io.File
```

Output of code snippet generation:
```
File _file_1 = new File(java.lang.String);
PrintWriter _printwriter_1 = new PrintWriter_file_1);
_printwriter_1.println(java.lang.String);
_printwriter_1.close();
```

CHAPTER 5

DISCUSSION OF THE PROTOTYPE

Previous chapters described the JAGE prototype and its capabilities. This chapter discusses the limitations of the prototype and possible ways of overcoming those limitations in a full-blown JAGE. Major topics beyond the scope of a full-blown JAGE are deferred to the Future Work section of Chapter 7.

5.1 Probabilistic parsing

The prototype is largely dependent on the accuracy of the Stanford's probabilistic parser. The prototype rejects an English sentence whose structure does not match a pre-defined case frame. For a natural-language sentence, several parse-tree representations are possible. Probabilistic parser selects one parse tree based on the most likely use of the words, which may not match the user's intention. In the prototype, there is no way for the system to know what the user's intention really was.

One approach to overcome this limitation is to have the probabilistic parser return the top *n* choices, and thereby, increase the odds that one of them was what the user intended. However, this approach generates ambiguity for the code generator since there can be multiple case frames that match the user input and hence, multiple slot values. This approach also increases the search time within the ontology *n* times in addition to increasing the code generation complexity *n* times.

Another approach to overcome the multiple-parse-tree-representation problem can be to extend the previous approach with user supervision. Multiple parse-tree representations can be shown via a UI to the user, and he can choose the closest

representation he intended. However, this approach may not always work, for example, when the user's interpretation and the case-frame developer's interpretation of the parse tree do not match.

5.2 Sub-Classing Problem in Semantic Matching

As outlined in Section 3.1.3, generating code adhering to the resource protocol is a limitation of the prototype, which the full-blown JAGE should overcome. To achieve this, JAGE should have knowledge of Java's access modifiers and the concepts of abstract classes, and static methods.

To overcome the limitation of abstract classes and methods, one approach is to choose the first concrete implementing class of an interface or abstract class. While this approach would work for some cases, it is not optimal as the user's intent may be different. For example, the user wants a list to behave as a stack while the phrase "get me a list" might return an instance of a *java.util.LinkedList* whose behavior is that of a queue. The issue here is that *java.util.List* [31] has as children, *java.util.LinkedList* and *java.util.Stack*, both of which while being valid implementations of the List interface, have contrasting behaviors with respect to the order of element insertion and retrieval. Hence, a first-fit strategy will not always help answer the use-case.

Another approach is a guided search, wherein the system can ask the user for more information based on distinguishing attributes of the implementing classes on the same level of inheritances from the interface or abstract classes until the system hits a concrete class without ambiguity. To take this approach, the ontology of the class description should include another attribute called a distinguishing attribute. This

attribute could be a list of name-value pairs as there can be multiple distinguishing attributes in a class hierarchy.

For example, if the user's intent is to get a data structure for holding elements that are sorted upon insertion but does not allow duplicates, the questions can be similar to any of the following:

- "does the holder needs one key to index another?" Answer to this question will help the system to select one among *java.util.List*, *java.util.Set*, and *java.util.Map*.
- "does the holder allow duplicates?" Answer to this question will help the system to select one among *java.util.List* and *java.util.Set*?
- "does the Set have its elements in a sorted order?" Answer to this question will help the system to select one of the concrete classes of *java.util.SortedSet*, which can be a *java.util.TreeSet*.

These questions can be derived from the name-value pairs of distinguishing attributes in the ontology of J2SDK. The user interface can be as simple as a multiple choice screen with radio buttons, each of them having the name of the attribute and radio button's text as the value.

One approach to overcoming the limitation of access modifiers is to restrict the code generation module to generate code involving only public and default methods. *java.lang.reflect* package offers API to determine the access modifier of each library component as well as information about static methods and classes.

5.3 Multiple Paths in Typestate-Transition Graphs

The code-snippet generation process deals with the issue of selecting one code snippet from among several possibilities, based on typestate transitions. The typestates and allowed transition discussed in Chapter 3 can be modeled as a direct graph where the nodes are typestates and the links are transitions.

Consider a hypothetical class whose typestates are represented in Table 5.1 and graphically in Figure 5.1. Any valid code snippet involving method calls on an object of this class should transition the object from start state to end state. For example, consider the essence of the solution to a user's query involves invoking the method, m1. If the typestate specification dictates that any valid code snippet should take the object to state end, the code generation module can take the object from state S1 to state end by invoking – a) method *m2()*, or b) method *m3()*.

Table 5.1. Typestates for a Hypothetical Class.

Initial typestate	Method name	Final typestate
start	m1()	S1
S1	m2()	end
S1	m3()	end

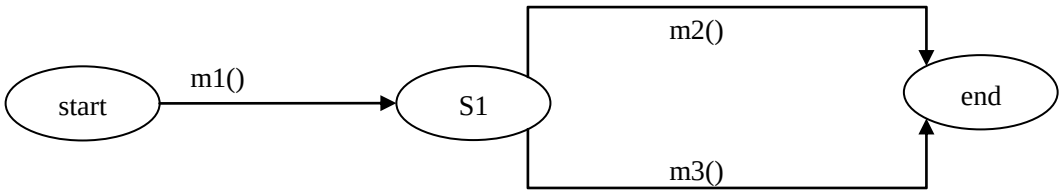

Figure 5.1. Graphical representation of the typestates.

The prototype uses a depth-first strategy for selecting a path but does not backtrack when the path it pursues does not lead to the end state. An approach to solve this issue is by modeling this problem as a graph traversal. A variant of the shortest-path algorithm to find the route from a node to end node can offer a code snippet. Such a code snippet represents the code of least complexity, when the measurement unit of complexity is lines of code.

Another approach to solving this limitation would be to display all possible paths and have the user select a desired one. However, this approach is only effective if the system can provide the users a basis for evaluating the different choice. Statistical analysis of code bases offers another possible approach to solve this issue. If large open-source code bases are indexed based on library elements, pattern-matching techniques can offer most preferred code paths.

5.4 Improving Semantic Matching

The preposition in the prototype's second case frame, VERB-ARTICLE-NOUN-PREPOSITION-ARTICLE-NOUN, raises some interesting scenarios. For example, "read from a Socket to a File" and "read to a Socket from a File" have the same meaning. The order of words and the prepositions indicate the source and

destination of the data flow. While the current implementation drops the prepositions and looks solely at the position of the words in a case frame, a more intelligent semantic matching algorithm should use the information provided by the prepositions in the input sentence. Typed dependencies provide this information [32], along with new case frames involving prepositions and other parts-of-speech should overcome this limitation of the prototype.

While JAGE uses simple word-sense matching in semantic matching, research in the area of semantic distance can help JAGE improve the semantic matching process. For example, JAGE does not use adjectives and adverbs in matching. However, adjectives can play an important role in choosing between classes in a tree hierarchy when a parent class matches the requirement and so do all its child classes. Adjectives can serve as the distinguishing attributes in such a scenario. Consider the example of *java.io.Socket*. If the user is interested in getting a secure socket, the adjective secure can allow a matcher to choose *javax.net.ssl.SSLSocket* over *java.net.Socket*.

Another limitation of the prototype is its inability to resolve pronouns in a sentence. Any sentence that contains pronouns relies on tying the pronoun to an object that is defined or referred to in an earlier phrase. Consider the example of “Read a line from a file and write it to the console.” The pronoun “it” needs to be resolved to a noun, and only then JAGE will be able to generate the code snippet accordingly. Since some control-structure generating systems, outlined in Section 6.3 have demonstrated the feasibility of resolving pronouns in a domain restricted by case frames, a full-blown JAGE can reuse those techniques.

5.5 Customizing WordNet for Computer Science Domain

The current prototype is limited by the completeness and accuracy of WordNet. For example, in computer science jargon, the word write and the word put are synonyms when used as verbs. However, the standard lexicon of WordNet does not contain a synonym relationship between these two words. To make WordNet more suitable for use in a computer science domain, a full-blown JAGE will need a customized lexicon. Appendix A contains details of how we added a synonym relationship between two words to the standard lexicon of WordNet.

CHAPTER 6

RELATED WORK

Similar work in suggesting code snippets that answer a user's requirement can be broadly classified into retrieval techniques, code-snippet generation techniques, and program-structure generation techniques. Each of these approaches has its merits and drawbacks.

6.1 Retrieval Techniques

The common method of this class of techniques is that they need a large repository of valid usage scenarios of an API to recommend examples to the user. They differ in types of knowledge stored in the repository and in the types of information extraction techniques used to interact with the user. The general drawback of retrieval techniques is that they need a repository of valid usage scenarios. While the large repository may include all possible usage scenarios, it remains only an assumption. These approaches also leave this question unanswered – how about usage scenarios for a new API of a new library? It is too cumbersome for a framework developer to document all possible usage scenarios of a library for the API he develops. Some of the existing code-snippet retrieval techniques are

- Approximate structural context matching [33] which takes a partial code snippet as input
- XSnippet [34] and PARSEWeb [35] which require knowledge of the library elements to query
- Sematics-based code search [36] which has multiple query options ranging

from keywords to method signatures

- SPARS-J [37] that maintains a large database of API elements to enable keyword search

SNIFF: a search engine for Java based on free-form queries, the most recent of these techniques, annotates usage scenarios with natural-language text [38]. The system then searches through indexes when a user query is entered. The system also clusters common usage scenarios and returns a set of code snippets that might follow the protocol of API usage. This approach, as other retrieval techniques, still requires a repository of usage scenarios. In addition, this system does not try to match the API element with a natural-language sentence, but it does a bag of words akin to keyword search.

6.2 Code-Snippet Generation Techniques

There are code snippet generation tools that use the type information to match with required library elements. The upside of these techniques is that there is no need for a large repository of usage scenarios to mine. The downside is that most of these techniques require partial knowledge of the API to specify what the user needs.

Signature matching and Jungloid mining techniques require the user query to specify the source and target types in the library. These techniques then suggest missing links of method call chains or class casting to extract the object of the target type from the source type. This places the burden on the user to know or learn part of the API to be able to query this tool.

JAGE does not use signature matching to retrieve the initial set of classes and

methods that answer a user's query. Instead, JAGE relies on parts-of-speech mapping to retrieve the initial set of classes. JAGE then uses signature matching only if there are missing gaps in the conversion of the source type to the target type. Hence, JAGE overcomes the disadvantage of learning the API about the source and target types to use signature matching.

6.3 Program-Structure Generation Techniques

The idea of using natural-language to generate code has been the subject of active research at least since 1979 [39]. There is a class of techniques that help a programmer to generate the body of a method from natural-language sentences. Metafor [40] is such a system that generates structure of classes and methods from a user story. NaturalJava [41] and Pegasus [42] take instructions from user's text in natural-language to generate code.

These systems can understand iteration, array operations, and variable name resolution from anaphoric relations in text. An instruction to assign to variable, *i*, the value of 1 would generate an assignment statement of, *i=1*, in a Java-like programming language. While these program-generation techniques make good use of natural-language processing techniques, they do not interface with software libraries. They can recommend control-flow structures like loops, but cannot instantiate or call J2SDK elements.

6.4 Web-Service Composition

The area of automatic web-service composition has witnessed much research on using ontologies like OWL to describe software components [43]. JAGE differs from

these methods by using ontology to describe J2SDK elements due to the difference in the domain of knowledge. The attributes and ontology hierarchy used to describe J2SDK elements are different from those of web services.

CHAPTER 7

SUMMARY AND FUTURE WORK

This thesis introduced the concept of JAGE, which is the generation of meaningful code-snippets from natural-language input, and demonstrated its feasibility through a functioning prototype. The two core objectives of JAGE were to make searching for a library component easier through natural-language, and to assist the programmer in using these components by suggesting valid code snippets.

The prototype using a constrained grammar, constrained vocabulary and limited library components could answer some simple user queries by providing them with what components might fit their use case. The prototype in its second step of code generation could generate one code snippet per query which illustrated the right way to use the API of these library components. Chapter 5 and Appendix C contain sample results of the prototype.

While the previous chapters discussed the limitations of the prototype and some approaches to overcome those in a full-blown JAGE, there are still sufficiently hard problems beyond the scope of a full-blown JAGE, some of which are outlined below.

JAGE relies on the assumption that the framework developer can provide the system with a machine-readable ontology of facts in the library. However, this exercise becomes quite cumbersome over large software libraries involving hundreds of classes and a number of methods in each class. One future work can be using machine learning techniques to automatically extract the relevant facts of domain as an ontology. Terminology extraction is one such technique [44]. Such a method would reduce the

burden on framework developers and JAGE can automatically learn what other libraries contain.

JAGE produces single-threaded code. Most Java applications, however, require multiple threads. Generating multi-threaded code presents new challenges of data synchronization and controlling the life-cycle of threads, which is a possible area of more research.

Fragility of point-cut specification in aspect-oriented languages like AspectJ has been the target of considerable research recently [45]. Point-cuts in AspectJ, for example, rely on syntactic matching of method and class names to weave advice. Several techniques have been proposed to improve the point-cut specification by using point-cut specification not on syntactic method signatures, but instead using a higher level of abstraction like UML diagrams and XML descriptors to describe methods, hence improving the precision and recall of point-cut specification [46, 47]. Further investigation is needed to determine if a point-cut specification strategy using natural-language elements can be developed using parts-of-speech description of library elements (methods and classes).

The JAGE prototype cannot understand or generate control flow constructs. While a full-blown JAGE might learn from other program structure generation techniques on how to generate control structures like loops and conditionals, doing so would require sufficient research to use the Java's exception handling primitives correctly.

REFERENCES

[1] Sommerville, I. Software Engineering. Addison Wesley 1992.

[2] Jackson, J., Google exec worries over 'rudderless' Java. IT World Canada. http://www.itworldcanada.com/news/google-exec-worries-over-rudderless-java/140426-pg2. Accessed, Aug, 2011

[3] Oracle. Java SE Documentation. http://download.oracle.com/javase/6/docs/. Accessed on 2nd Feb, 2011.

[4] Mandelin, D., Xu, L., Bodik, R., and Kimelman, D. Jungloid mining: helping to navigate the API jungle. In Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation,48-61.

[5] Oracle. Java SE Documentation. http://download.oracle.com/javase/6/docs/. Accessed on 2nd Feb, 2011.

[6] Strom, R.E., Yemini, S. Typestate: A programming language concept for enhancing software reliability. IEEE TSE 12 (1986) 157-171

[7] Jurafsky, D. and Martin, J.H. Speech and Language Processing. Prentice Hall, 2000.

[8] Shieber, S. Evidence against the context-freeness of natural language. Linguistics and Philosophy 8 (1985), 333–343.

[9] Manning, C. and Schutze, H. Foundations of Statistical Natural Language Processing. MIT Press, 1999.

[10] Klein, D. and Manning, C. 2003. Accurate unlexicalized parsing. In Proceedings of the 41st Meeting of the Association for Computational Linguistics, 2003, 423-430.

[11] Fillmore, C. The case for case. In Texas Symposium On Linguistic Universals, 1967.

[12] Nomiyama, H. 1992. Machine translation by case generalization. In Proceedings of the 14th Conference on Computational Linguistics, vol. 2. Association for Computational Linguistics, 714-720.

[13] Trandabat, D. Natural Language Processing Using Semantic Frames. Doctoral Dissertation, Universitatea Alexandru Ioan Cuza, 2010.

[14] Princeton University.About WordNet. Princeton University. http://wordnet.princeton.edu. Accessed, Feb, 2011.

[15] Bwalenz, J. JWNL (Java WordNet Library).

http://sourceforge.net/projects/jwordnet/. Accessed, Feb, 2011.

[16] Gruber, T.R. A translation approach to portable ontology specifications. Knowledge Acquisition 5, 2 (1993), 199-220.

[17] W3C. Resource Description Framework (RDF). http://www.w3.org/RDF/. Accessed, Feb, 2011.

[18] W3C. OWL Web Ontology Lanugage Overview. http://www.w3.org/TR/owl-features/. Accessed, Feb, 2011.

[19] Fikes, R., Hayes, P., and Horrocks, I. OWL-QL - A Language for Deductive Query Answering on the Semantic Web. Technical Report 03-14, Knowledge Systems Laboratory, Stanford University, 2003.

[20] W3C. Owl 2 Web Ontology Language Profiles. http://www.w3.org/TR/owl2-profiles/#OWL_2_QL. Accessed, Feb, 2011.

[21] W3C. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/. Accessed, Feb, 2011.

[22] Jena. Jena – A Semantic Web Framework for Java. http://jena.sourceforge.net/. Accessed, Feb, 2011.

[23] Moormann Zaremski, A. and Wing, J.M. Signature matching: a tool for using software libraries. ACM Trans. on Software Engineering and Methodology 4, 2 (1995), 146-170.

[24] Tennant, H., 1979, Experience with the evaluation of natural language question answerers. In Proceedings 6th Internation Joint Conference on Artificial Intelligence, 1979, 874-876.

[25] R. DeLine and M. Fähndrich. Typestates for objects. In European Conference on Object-Oriented Programming. Springer-Verlag, 2004.

[26] Fidel, R. Searchers' selection of search keys: II. Controlled vocabulary or free-text searching. Journal of the American Society for Information Science 42, 7 (1991), 501-514

[27] Dan Klein, Christopher D. Manning, Accurate Unlexicalized Parsing, In Proceedings of the 41ST Annual Meeting of the Association for Computational Linguistics, 2003, 423--430.

[28] W3C. OWL Implemenations. http://www.w3.org/2007/OWL/wiki/Implementations. Accessed, Feb, 2011.

[29] Noy, N. F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R., Musen, M. A., Creating Semantic Web Contents with Protege-2000. IEEE Intelligent Systems, 16,

2, 2001, 60-71. http://protegewiki.stanford.edu/wiki/Main_Page. Accessed, Aug 2011

[30] Bierhoff, K. 2008. Checking API protocol compliance in java. In Companion To the 23rd ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications, 2008, 915-916.

[31] Oracle. Interface List<E>. http://download.oracle.com/ javase/1.5.0/docs/api/java/util/List.html. Accessed on 2 Feb, 2011.

[32] Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The Stanford typed dependencies representation. In COLING, Workshop on Cross-framework and Cross-domain Parser Evaluation, 2008, 1-8.

[33] Holmes, R., Walker, R.J., and Murphy, G.C. Approximate structural context matching: An approach to recommend relevant examples. IEEE Trans. Software Engineering 32, 12 (2006), 952-970.

[34] Sahavechaphan, N. and Claypool, K. 2006. XSnippet: Mining for sample code. In Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications, 2006, 413-430.

[35] Thummalapenta, S. and Xie, T. Parseweb: a programmer assistant for reusing open source code on the web. In Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering, 2007, 204-213.

[36] Reiss, S.P. 2009. Semantics-based code search. In Proceedings of the 31st IEEE International Conference on Software Engineering, 2009, 243-253.

[37] Yokomori, R., Umemori, F., Nishi, H., Yamamoto, T., Matsushita, M., Kusumoto, S., and Inoue, K. Java software component retrieval system SPARS-J. Trans.of the Institution of Electronics, Information and Communication Engineers J87-D-1, 12 (2004), 1060-1068.

[38] Chatterjee, S., Juvekar, S., and Sen, K. SNIFF: A search engine for Java using free-form queries. Fundamental Approaches to Software Engineering, Lecture Notes in Computer Science, (2009), 385-400.

[39] Ballard, B. W. and Biermann, A. W. 1979. Programming in natural language: “NLC” as a prototype. In Proceedings of the 1979 Annual Conference A. L. Martin and J. L. Elshoff, Eds. ACM 79. ACM, New York, NY, 228-237.

[40] Liu, H. and Lieberman, H. 2005. Metafor: Visualizing stories as code. In Proceedings of the 10th International Conference on Intelligent User Interfaces, 2005, 305-307.

[41] Price, D., Rilofff, E., Zachary, J., and Harvey, B. 2000. NaturalJava: a natural language interface for programming in Java. In Proceedings of the 5th

International Conference on Intelligent User Interfaces,2000, 207-211.

[42] Knöll, R. and Mezini, M. 2006. Pegasus: First steps toward a naturalistic programming language. In Companion to the 21st ACM Symposium on Object-Oriented Programming Systems, Languages, and Applications, 2006, 542-559.

[43] Paolucci, M., and Kawamura, K., and Payne, T.R., and Sycara, K. Semantic matching of web services capabilities. In Proceedings of 1st International Semantic Web Conference, 2002, http://eprints.ecs.soton.ac.uk/7606/1/ ISWC2002-Matchmaker.pdf.

[44] Stylos, J., Faulring, A., Z., and Myers, B.A. Improving API documentation using API usage information. In Proceedings of the 2009 IEEE Symposium on Visual Languages and Human-Centric Computing, 2009, 119-126.

[45] R. Filman and D. Friedman, Aspect-oriented programming is quantification and obliviousness. In Workshop on Advanced Separation of Concerns, 2000, 21-35.

[46] Walter Cazzola, Sonia Pini, and Ancona Massimo, Design-based pointcuts robustness against software evolution. In 3rd ECOOP Workshop on Reflection, AOP and Meta-Data for Software Evolution, 2006, 35-45.

[47] Yang, Z. and Zhao, T. 2007. Improve pointcut definitions with program views. In Proceedings of the 5th Workshop on Engineering Properties of Languages and Aspect Technologie, 2007, Article 9.

APPENDICES

Appendix A:

Adding a Word to the Synset of the Standard WordNet Lexicon

This appendix shows the steps to extend the standard WordNet lexicon with one custom relationship between two words in a synset. As an example, we add the word put to the synset of the word write when both the words are verbs.

WordNet is organized as a tree of pointers with relationships. To add a word to a synset is equivalent to adding a pointer to an existing pointer. extJWNL, http://sourceforge.net/projects/extjwnl/, is an open source library that offers API to edit WordNet dictionaries. In the following example, *write%2:36:00::* and *put%2:31:13*:: are pointers to nodes of information. The command to add a word to a synset with the extJWNL is, *./ewn write%2:36:00:: -addptr put%2:31:13:: @*

The *ewn* tool accepts a script containing a set of exceptions one per line, which the full-blown JAGE can execute before starting up. The following set of commands in bold show the work log of the exercise to modify the standard WordNet lexicon

```
$ export WNHOME=/WordNet/WordNet-3.0/dict/
$ ./ewn write -g -k -l -synsv

Synonyms of verb write
Sense 1
write [write%2:36:00::], compose [compose%2:36:01::], pen
[pen%2:36:00::], indite [indite%2:36:00::] -- (produce a literary
work; "She composed a poem"; "He wrote four novels")
      create verbally [create_verbally%2:36:00::] -- (create with
or from words)
            make [make%2:36:00::], create [create%2:36:00::] --
(make or cause to be or to become; "make a mess in one's office";
"create a furor")

$ ./ewn put -g -k -l -synsv

Synonyms of verb put

Sense 6
place [place%2:31:13::], put [put%2:31:13::], set [set%2:31:13::]
-- (estimate; "We put the time of arrival at 8 P.M.")
      estimate [estimate%2:31:00::], gauge [gauge%2:31:00::],
approximate [approximate%2:31:00::], guess [guess%2:31:01::],
judge [judge%2:31:01::] -- (judge tentatively or form an estimate
of (quantities or time); "I estimate this chicken to weigh three
pounds")
```

```
            calculate [calculate%2:31:00::], cipher
[cipher%2:31:00::], cypher [cypher%2:31:00::], compute
[compute%2:31:00::], work out [work_out%2:31:06::], reckon
[reckon%2:31:01::], figure [figure%2:31:00::] -- (make a
mathematical calculation or computation)
                    reason [reason%2:31:00::] -- (think logically;
"The children must learn to reason")
                          think [think%2:31:00::], cogitate
[cogitate%2:31:00::], cerebrate [cerebrate%2:31:00::] -- (use or
exercise the mind or one's power of reason in order to make inferences,
decisions, or arrive at a solution or judgments; "I've been thinking
all day and getting nowhere")
```

```
$ ./ewn write%2:36:00:: -addptr  put%2:31:13:: @
```

```
$ ./ewn write -g -k -l -synsv
```

```
Synonyms of verb write

Sense 1
write [write%2:36:00::], compose [compose%2:36:01::], pen
[pen%2:36:00::], indite [indite%2:36:00::] -- (produce a literary
work; "She composed a poem"; "He wrote four novels")
       create verbally [create_verbally%2:36:00::] -- (create with
or from words)
            make [make%2:36:00::], create [create%2:36:00::] --
(make or cause to be or to become; "make a mess in one's office";
"create a furor")
       place [place%2:31:13::], put [put%2:31:13::], set
[set%2:31:13::] -- (estimate; "We put the time of arrival at 8 P.M.")
            estimate [estimate%2:31:00::], gauge
[gauge%2:31:00::], approximate [approximate%2:31:00::], guess
[guess%2:31:01::], judge [judge%2:31:01::] -- (judge tentatively or
form an estimate of (quantities or time); "I estimate this chicken
to weigh three pounds")
                    calculate [calculate%2:31:00::], cipher
[cipher%2:31:00::], cypher [cypher%2:31:00::], compute
[compute%2:31:00::], work out [work_out%2:31:06::], reckon
[reckon%2:31:01::], figure [figure%2:31:00::] -- (make a
mathematical calculation or computation)
                          reason [reason%2:31:00::] -- (think
logically; "The children must learn to reason")
                                think [think%2:31:00::],
```

cogitate [cogitate%2:31:00::], cerebrate [cerebrate%2:31:00::] -- (use or exercise the mind or one's power of reason in order to make inferences, decisions, or arrive at a solution or judgments; "I've been thinking all day and getting nowhere")

Appendix B:

Ontology of J2SDK Classes in OWL


```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
    xmlns:assert="http://www.owl-ontologies.com/assert.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"

xmlns="http://www.owl-ontologies.com/2009/8/16/Ontology1250442606.owl#
"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

xml:base="http://www.owl-ontologies.com/2009/8/16/Ontology1250442606.o
wl">
  <owl:Ontology rdf:about="">
    <rdfs:comment
rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Author: Chandra
Vijyapurpu</rdfs:comment>
  </owl:Ontology>
  <owl:Class rdf:ID="Typestate"/>
  <owl:Class rdf:ID="Field"/>
  <owl:Class rdf:ID="Method"/>
  <owl:Class rdf:ID="Java_Class"/>
  <owl:Class rdf:ID="Primitive"/>
  <owl:ObjectProperty rdf:ID="start_state">
    <rdfs:range rdf:resource="#Typestate"/>
    <rdfs:domain rdf:resource="#Java_Class"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="has_state">
    <owl:inverseOf>
      <owl:ObjectProperty rdf:ID="of_class"/>
    </owl:inverseOf>
    <rdfs:range rdf:resource="#Typestate"/>
    <rdfs:domain rdf:resource="#Java_Class"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="has_field">
    <owl:inverseOf>
      <owl:FunctionalProperty rdf:ID="declaring_class"/>
    </owl:inverseOf>
    <rdfs:domain rdf:resource="#Java_Class"/>
    <rdfs:range rdf:resource="#Field"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="end_state">
    <rdfs:domain rdf:resource="#Java_Class"/>
    <rdfs:range rdf:resource="#Typestate"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="super_class">
    <owl:inverseOf>
```

```
      <owl:TransitiveProperty rdf:ID="child_class"/>
    </owl:inverseOf>
    <rdfs:domain rdf:resource="#Java_Class"/>
    <rdfs:range rdf:resource="#Java_Class"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#of_class">
    <owl:inverseOf rdf:resource="#has_state"/>
    <rdfs:domain rdf:resource="#Typestate"/>
    <rdfs:range rdf:resource="#Java_Class"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="noun_phrase">
    <rdfs:domain rdf:resource="#Java_Class"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="argumentList">
    <rdfs:domain rdf:resource="#Method"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="state_name">
    <rdfs:domain rdf:resource="#Typestate"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="verb_phrase">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Method"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="field_noun_phrase">
    <rdfs:domain rdf:resource="#Field"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="returnType">
    <rdfs:domain rdf:resource="#Method"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  </owl:DatatypeProperty>
  <owl:TransitiveProperty rdf:about="#child_class">
    <rdfs:range rdf:resource="#Java_Class"/>
    <owl:inverseOf rdf:resource="#super_class"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:domain rdf:resource="#Java_Class"/>
  </owl:TransitiveProperty>
  <owl:FunctionalProperty rdf:ID="field_is_static">
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#Field"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="class_name">
    <rdfs:domain rdf:resource="#Java_Class"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="is_constructor">
    <rdf:type
```

```
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#Method"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="method_name">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Method"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="is_abstract">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#Java_Class"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="field_name">
    <rdfs:domain rdf:resource="#Field"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="qualified_name">
    <rdfs:domain rdf:resource="#Java_Class"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="enclosing_class">
    <rdfs:range rdf:resource="#Java_Class"/>
    <owl:inverseOf>
      <owl:InverseFunctionalProperty rdf:ID="has_method"/>
    </owl:inverseOf>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:domain rdf:resource="#Method"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:about="#declaring_class">
    <rdfs:domain rdf:resource="#Field"/>
    <rdfs:range rdf:resource="#Java_Class"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <owl:inverseOf rdf:resource="#has_field"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="is_static">
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#Method"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
  </owl:FunctionalProperty>
  <owl:InverseFunctionalProperty rdf:about="#has_method">
    <rdfs:range rdf:resource="#Method"/>
    <owl:inverseOf rdf:resource="#enclosing_class"/>
```

```
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:domain rdf:resource="#Java_Class"/>
  </owl:InverseFunctionalProperty>
  <Primitive rdf:ID="long"/>
  <Primitive rdf:ID="int"/>
  <Primitive rdf:ID="char"/>
  <Java_Class rdf:ID="Socket">
    <super_class>
      <Java_Class rdf:ID="Object">
        <child_class>
          <Java_Class rdf:ID="FileInputStream">
            <has_method>
              <Method rdf:ID="fileInputStream">
                <method_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >FileInputStream</method_name>
                <enclosing_class rdf:resource="#FileInputStream"/>
                <is_static
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                >false</is_static>
                <argumentList
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >File</argumentList>
                <is_constructor
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                >true</is_constructor>
              </Method>
            </has_method>
            <has_method>
              <rdf:Description
rdf:about="http://www.owl-ontologies.com/2009/8/16/Ontology1250442606.
owlread">
                <method_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >read</method_name>
                <enclosing_class rdf:resource="#FileInputStream"/>
                <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >read</verb_phrase>
                <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >read bytes</verb_phrase>
                <is_constructor
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                >false</is_constructor>
                <is_static
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                >false</is_static>
              </rdf:Description>
            </has_method>
            <qualified_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >java.io.FileInputStream</qualified_name>
            <class_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >FileInputStream</class_name>
            <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

```
          >read byte stream</noun_phrase>
          <super_class rdf:resource="#Object"/>
          <is_abstract
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
          >false</is_abstract>
        </Java_Class>
      </child_class>
      <child_class>
        <Java_Class rdf:ID="Reader">
          <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >read from character stream</noun_phrase>
          <is_abstract
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
          >true</is_abstract>
          <qualified_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >java.io.Reader</qualified_name>
          <child_class>
            <Java_Class rdf:ID="FileReader">
              <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >read character stream</noun_phrase>
              <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >read character input stream</noun_phrase>
              <class_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >FileReader</class_name>
              <qualified_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >java.io.FileReader</qualified_name>
              <super_class rdf:resource="#Object"/>
              <super_class rdf:resource="#Reader"/>
              <super_class>
                <Java_Class rdf:ID="InputStreamReader">
                  <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                  >bridge from byte to character streams</noun_phrase>
                  <is_abstract
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                  >false</is_abstract>
                  <has_method>
                    <Method rdf:ID="Method_10">
                      <method_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                      >InputStreamReader</method_name>
                      <returnType
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                      >InputStreamReader</returnType>
                      <enclosing_class
rdf:resource="#InputStreamReader"/>
                      <is_static
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                      >false</is_static>
                      <argumentList
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                      >InputStream</argumentList>
                      <is_constructor
```

```
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                    >true</is_constructor>
                  </Method>
                </has_method>
                <super_class rdf:resource="#Reader"/>
                <super_class rdf:resource="#Object"/>
                <qualified_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >java.io.InputStreamReader</qualified_name>
                <class_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >InputStreamReader</class_name>
                <child_class rdf:resource="#FileReader"/>
              </Java_Class>
            </super_class>
            <is_abstract
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
            >false</is_abstract>
          </Java_Class>
        </child_class>
        <class_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >Reader</class_name>
        <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >character stream</noun_phrase>
        <child_class>
          <Java_Class rdf:ID="BufferedReader">
            <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >read line</noun_phrase>
            <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >read string</noun_phrase>
            <super_class rdf:resource="#Reader"/>
            <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >read character stream</noun_phrase>
            <class_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >BufferedReader</class_name>
            <is_abstract
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
            >false</is_abstract>
            <super_class rdf:resource="#Object"/>
            <qualified_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >java.io.BufferedReader</qualified_name>
            <has_method>
              <Method rdf:ID="readLine">
                <method_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >readLine</method_name>
                <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >read line</verb_phrase>
                <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >read a line of text</verb_phrase>
```

```
                <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >read a line</verb_phrase>
                <is_static
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                >false</is_static>
                <is_constructor
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                >false</is_constructor>
                <enclosing_class rdf:resource="#BufferedReader"/>
              </Method>
            </has_method>
          </Java_Class>
        </child_class>
        <child_class rdf:resource="#InputStreamReader"/>
        <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >read character stream</noun_phrase>
        <super_class rdf:resource="#Object"/>
      </Java_Class>
    </child_class>
    <child_class rdf:resource="#FileReader"/>
    <child_class rdf:resource="#Socket"/>
    <qualified_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >java.lang.Object</qualified_name>
    <child_class>
      <Java_Class rdf:ID="System">
        <has_field>
          <Field rdf:ID="out">
            <field_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >out</field_name>
            <field_is_static
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
            >true</field_is_static>
            <declaring_class rdf:resource="#System"/>
            <field_noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >monitor</field_noun_phrase>
            <field_noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >screen</field_noun_phrase>
            <field_noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >console</field_noun_phrase>
          </Field>
        </has_field>
        <has_field>
          <Field rdf:ID="in">
            <declaring_class rdf:resource="#System"/>
            <field_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >in</field_name>
            <field_noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >user input</field_noun_phrase>
            <field_noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

```
              >console</field_noun_phrase>
              <field_is_static
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
              >true</field_is_static>
            </Field>
          </has_field>
          <is_abstract
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
          >false</is_abstract>
          <class_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >System</class_name>
          <qualified_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >java.lang.System</qualified_name>
          <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >system constants</noun_phrase>
          <super_class rdf:resource="#Object"/>
        </Java_Class>
      </child_class>
      <child_class rdf:resource="#BufferedReader"/>
      <child_class>
        <Java_Class rdf:ID="Date">
          <super_class rdf:resource="#Object"/>
          <class_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >Date</class_name>
          <is_abstract
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
          >false</is_abstract>
          <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >Date</noun_phrase>
          <has_method>
            <Method rdf:about="#Date(constructor)">
              <enclosing_class rdf:resource="#Date"/>
              <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >time now</verb_phrase>
              <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >today's date</verb_phrase>
              <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >current date</verb_phrase>
              <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >current system date</verb_phrase>
              <method_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >Date</method_name>
            </Method>
          </has_method>
          <qualified_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >java.util.Date</qualified_name>
        </Java_Class>
      </child_class>
```

```
      <child_class>
        <Java_Class rdf:ID="File">
          <has_method>
            <Method rdf:ID="delete">
              <enclosing_class rdf:resource="#File"/>
              <is_constructor
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
              >false</is_constructor>
              <method_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >delete</method_name>
              <returnType
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >boolean</returnType>
              <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >delete</verb_phrase>
              <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >remove</verb_phrase>
              <is_static
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
              >false</is_static>
            </Method>
          </has_method>
          <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >file in the fiile system</noun_phrase>
          <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >directory</noun_phrase>
          <has_method>
            <Method rdf:ID="isDirectory">
              <is_static
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
              >false</is_static>
              <method_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >isDirectory</method_name>
              <returnType
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >boolean</returnType>
              <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >tests whether the file denoted by this pathname is a
directory</verb_phrase>
              <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >is directory</verb_phrase>
              <is_constructor
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
              >false</is_constructor>
              <enclosing_class rdf:resource="#File"/>
            </Method>
          </has_method>
          <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >file</noun_phrase>
          <qualified_name
```

```
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >java.io.File</qualified_name>
          <has_method>
            <Method rdf:ID="file">
              <is_static
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
              >false</is_static>
              <method_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >File</method_name>
              <enclosing_class rdf:resource="#File"/>
              <argumentList
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >path</argumentList>
              <is_constructor
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
              >true</is_constructor>
              <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >new file</verb_phrase>
              <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >create</verb_phrase>
            </Method>
          </has_method>
          <is_abstract
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
          >false</is_abstract>
          <has_method>
            <Method rdf:ID="listFiles">
              <is_constructor
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
              >false</is_constructor>
              <method_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >listFiles</method_name>
              <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >list files in this directory</verb_phrase>
              <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >list files</verb_phrase>
              <returnType
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >File[]</returnType>
              <is_static
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
              >false</is_static>
              <enclosing_class rdf:resource="#File"/>
            </Method>
          </has_method>
          <class_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >File</class_name>
          <has_method>
            <Method rdf:ID="exists">
              <method_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >exists</method_name>
```

```
              <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >exists</verb_phrase>
              <is_constructor
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
              >false</is_constructor>
              <returnType
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >boolean</returnType>
              <is_static
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
              >false</is_static>
              <enclosing_class rdf:resource="#File"/>
            </Method>
          </has_method>
          <super_class rdf:resource="#Object"/>
          <has_method>
            <Method rdf:ID="renameTo">
              <method_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >renameTo</method_name>
              <is_constructor
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
              >false</is_constructor>
              <argumentList
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >File</argumentList>
              <returnType
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >boolean</returnType>
              <enclosing_class rdf:resource="#File"/>
              <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >rename</verb_phrase>
              <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >move</verb_phrase>
              <is_static
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
              >false</is_static>
            </Method>
          </has_method>
          <has_method>
            <Method rdf:ID="getAbsolutePath">
              <returnType
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >String</returnType>
              <method_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >getAbsolutePath</method_name>
              <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >get absolute path on the file system</verb_phrase>
              <is_static
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
              >false</is_static>
              <enclosing_class rdf:resource="#File"/>
              <is_constructor
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
```

```
            >false</is_constructor>
          </Method>
        </has_method>
      </Java_Class>
    </child_class>
    <child_class>
      <Java_Class rdf:ID="String">
        <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >string</noun_phrase>
        <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >stream of characters</noun_phrase>
        <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >character sequence</noun_phrase>
        <qualified_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >java.lang.String</qualified_name>
        <super_class rdf:resource="#Object"/>
        <is_abstract
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
        >false</is_abstract>
        <class_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >String</class_name>
      </Java_Class>
    </child_class>
    <child_class rdf:resource="#InputStreamReader"/>
    <class_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Object</class_name>
    <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >any java instance</noun_phrase>
    <is_abstract
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
    >false</is_abstract>
    <child_class>
      <Java_Class rdf:ID="Writer">
        <child_class>
          <Java_Class rdf:ID="PrintWriter">
            <has_method>
              <Method rdf:ID="println">
                <method_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >println</method_name>
                <enclosing_class rdf:resource="#PrintWriter"/>
                <is_constructor
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                >false</is_constructor>
                <returnType
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >void</returnType>
                <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >write line</verb_phrase>
                <is_static
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
```

```
                >false</is_static>
                <argumentList
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >String</argumentList>
              </Method>
            </has_method>
            <is_abstract
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
            >false</is_abstract>
            <super_class rdf:resource="#Writer"/>
            <class_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >PrintWriter</class_name>
            <noun_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >write to character stream</noun_phrase>
            <qualified_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >java.io.PrintWriter</qualified_name>
          </Java_Class>
        </child_class>
        <super_class rdf:resource="#Object"/>
        <qualified_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >java.io.Writer</qualified_name>
        <class_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >Writer</class_name>
      </Java_Class>
    </child_class>
  </Java_Class>
</super_class>
<is_abstract
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
>false</is_abstract>
<qualified_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>java.net.Socket</qualified_name>
<has_method>
  <Method rdf:ID="Method_11">
    <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >create a stream socket</verb_phrase>
    <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >create a socket</verb_phrase>
    <method_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Socket</method_name>
    <enclosing_class rdf:resource="#Socket"/>
    <is_constructor
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
    >true</is_constructor>
  </Method>
</has_method>
<has_method>
  <Method rdf:ID="Method_3">
    <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

```
      >create a stream socket</verb_phrase>
      <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >create a socket</verb_phrase>
      <method_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Socket</method_name>
      <argumentList
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >port</argumentList>
      <argumentList
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >host</argumentList>
      <enclosing_class rdf:resource="#Socket"/>
      <is_constructor
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
      >true</is_constructor>
    </Method>
  </has_method>
  <has_method>
    <Method rdf:ID="getInputStream">
      <verb_phrase
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >get input stream</verb_phrase>
      <method_name
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >getInputStream</method_name>
      <enclosing_class rdf:resource="#Socket"/>
      <is_static
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
      >false</is_static>
      <returnType
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >InputStream</returnType>
      <is_constructor
rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
      >false</is_constructor>
    </Method>
  </has_method>
  <class_name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Socket</class_name>
 </Java_Class>
</rdf:RDF>
```

<!-- Created with Protege (with OWL Plugin 3.4.1, Build 536)
http://protege.stanford.edu -->

Appendix C:

Sample Results

The classes used in this experiment are the same classes that are described using OWL in Appendix A.

- java.lang.Object
- java.io.File
- java.net.Socket
- java.io.Writer
- java.io.PrintWriter
- java.io.BufferedReader
- java.io.FileInputStream
- java.io.Reader

The following examples show the user text input, as well as the output of semantic matching of JAGE, and the generated code snippet.

1. Text input:
   ```
   How do I read a file
   ```
   Output of Step 1:
   ```
   Found match in Source: class java.io.FileInputStream Target: class
   java.io.File
   ```
   Output of Step 2:
   ```
   File _file_1 = new File(java.lang.String);
   FileInputStream _fileInputStream_1 = new FileInputStream(_file_1);
   _fileInputStream _1.readLine();
   _fileInputStream _1.close();
   ```

2. Text input:
   ```
   How do I read a line from a file
   ```
   Output of Step 1:
   ```
   Found match in Source: class java.io.BufferedReader Target: class
   java.io.File
   ```
   Output of Step 2:
   ```
   File _file_1 = new File(java.lang.String);
   FileReader _fileReader_1 = new FileReader(_file_1);
   BufferedReader _bufferedReader_1 = new
   BufferedReader_fileReader_1);
   _bufferedReader_1.readLine();
   _bufferedReader_1.close();
   _fileReader_1.close();
   ```

3. Text input:
   ```
   How do I rename a file
   ```
   Output of Step 1:
   ```
   Found match in Source: class java.io.File Target: null
   ```
   Output of Step 2:

```
File _file_1 = new File(java.lang.String);
_file_1.renameTo(java.io.File);
```

4. Text input:
   ```
   How do I create a file
   ```
   Output of Step 1:
   ```
   Found match in Source: class java.io.File Target: null
   ```
   Output of Step 2:
   ```
   File _file_1 = new File(java.lang.String);
   ```

5. Text input:
   ```
   How do I read a line to a file
   ```
   Output of Step 1:
   ```
   Found match in Source: class java.io.BufferedReader Target: class
   java.io.File
   ```
   Output of Step 2:
   ```
   File _file_1 = new File(java.lang.String);
   FileReader _fileReader_1 = new FileReader(_file_1);
   BufferedReader _bufferedReader_1 = new
   BufferedReader_fileReader_1);
   _bufferedReader_1.readLine();
   _bufferedReader_1.close();
   _fileReader_1.close();
   ```

6. Text input:
   ```
   How do I not read a line from a file
   ```
   Output of Step 1:
   ```
   User sentence not understood: How do I not read a line from a file
   ```

7. Text input:
   ```
   How do I write a line to a file
   ```
   Output of Step 1:
   ```
   Found match in Source: class java.io.PrintWriter Target: class
   java.io.File
   ```
   Output of Step 2:
   ```
   File _file_1 = new File(java.lang.String);
   PrintWriter _printwriter_1 = new PrintWriter_file_1);
   _printwriter_1.println(java.lang.String);
   _printwriter_1.close();
   ```

8. Text input:
   ```
   How do I put a line into a file
   ```
   Output of Step 1:
   ```
   No matches found in OWL
   ```

9. Text input:
   ```
   How do I give a file another name
   ```
   Output of Step 1:
   ```
   User sentence not understood: How do I give a file another name
   ```

10. Text input:
    ```
    How do I move a file
    ```
    Output of Step 1:
    ```
    Found match in Source: class java.io.File Target: null
    ```

Output of Step 2:
```
File _file_1 = new File(java.lang.String);
_file_1.renameTo(java.io.File);
```

11. Text input:
```
How do I delete a file
```
Output of Step 1:
```
Found match in Source: class java.io.File Target: null
```
Output of Step 2:
```
File _file_1 = new File(java.lang.String);
_file_1.delete();
```

12. Text input:
```
How do I open a file
```
Output of Step 1:
No matches found in OWL

13. Text input:
```
How do I see a file
```
Output of Step 1:
```
No matches found in OWL
```

14. Text input:
```
How do I delete a directory
```
Output of Step 1:
```
Found match in Source: class java.io.File Target: null
```
Output of Step 2:
```
File _file_1 = new File(java.lang.String);
_file_1.delete();
```

15. Text input:
```
How do I close a file
```
Output of Step 1:
```
No matches found in OWL
```

16. Text input:
```
How do I list files in a directory
```
Output of Step 1:
```
User sentence not understood: How do I list files in a directory
```

17. Text input:
```
How do I check if a file is readonly
```
Output of Step 1:
```
User sentence not understood: How do I check if a file is readonly
```

18. Text input:
```
How do I make a directory
```
Output of Step 1:
```
Found match in Source: class java.io.File Target: null
```
Output of Step 2:
```
File _file_1 = new File(java.lang.String);
```
This example illustrates how WordNet improved matching process.

19. Text input:
    `How do I check if the file is a directory`
    Output of Step 1:
    `User sentence not understood: How do I list files in a directory`