Journal of Theoretical and Applied Information Technology
15th October 2011. Vol. 32 No.1

ISSN: **1992-8645** www.jatit.org E-ISSN: **1817-3195**

OBJECT-ORIENTED PROGRAMMING SEMANTICS REPRESENTATION UTILIZING AGENTS

TEH NORANIS MOHD ARIS
Senior Lecturer, Department of Computer Science, Faculty of Computer Science and Information Technology, 43400 UPM Serdang, Selangor, Malaysia
E-mail: nuranis@fsktm.upm.edu.my

ABSTRACT

Comprehending Object-Oriented Programming (OOP) is not an easy task especially by novice students. The problem occurs during the transition from learning fundamental programming language concept to OOP concept. It is very important to handle this problem from the beginning before novices learn more advanced OOP concepts like encapsulation, inheritance, and polymorphism. Learning programming from source code examples is a common behavior among novices. Novices tend to refer to source codes examples and adapt the source codes to the problem given in their assignments. To cater the problems faced by these novices, a novel agent-based model have been designed to assist them in comprehending OOP concepts through source codes examples. The instructor needs to provide two related source codes that are similar but in different domain. Generally, these source codes go through the preprocessing, comparison, extraction, generate program semantics and classification processes. A formal algorithm that can be applied to any two related Java-based source codes examples is invented to generate the semantics of these source codes. The algorithm requires source codes comparison based on keyword similarity to extract the words that exist in the two related source codes. Three agents namely SemanticAgentGUI, semanticAgent and noviceAgent are designed in the proposed model. The running system shows an OOP semantic knowledge representation by intelligent agents.

Keywords: *OOP Semantics, Source Codes Comparison, Keyword Similarity, Extraction, Classification*

1. INTRODUCTION

OOP is taught as the first programming course in most universities in Malaysia. Usually, students who enrol this subject are in the novice category and it is a difficult subject for them to comprehend. The main issue is the difficulty for them to understand OOP concepts and apply it to a specific problem. Many researches have focused on the teaching methods to enhance the comprehension of novices in OOP using eight-queen puzzle [1], Abstract Data Type (ADT) set [2], pedagogical features (metaphor, learning module, structure editor, friendly compiler, program animation and visualization, open environment, problem solving techniques and tutorial) [3], integration of several languages (HyperText Markup Language (HTML), JavaScript, and Java) [4], game projects [5], instant graphical feedback [6] and design language principles (no conceptual redundancy, clean concepts, readability, software engineering support) [7].

In our university, students are taught the concepts of OOP in the first Computer Programming course that use the popular language, Java. We teach the fundamentals of Java consisting of data types, control statements, methods, arrays and basics of Object-Oriented (OO). Based on teaching experience, these students encounter problems when shifting from arrays to the basics of OO. This transition involves totally a different thinking from fundamental programming concept to OOP. These students are unable to relate the real world objects concepts to the actual problem given to them. It would be very useful if a source code example of an object can be compared with a source code example of an operation and then mapped to the semantics or meaning of the source codes. The mapping process that provides the semantics of the source codes contribute a clear comprehension through explanation of the relation between the two source codes that are compared.

Intelligent agents has been applied in various fields such as interactive tutoring [8] medical diagnosis system [9], image analysis [10], robotics [11], refinery controller [12], and simulation [13]. Intelligent agents are reactive system that react autonomously and determines how to achieve a particular task. They are located in some environment and are able to sense their environment through sensors, and have a selection of possible actions to perform through effectors or actuators so that they can modify their environment. For example, the sensors in a tutoring system is the input from the keyboard and the actuators is the display exercises, suggestions and corrections actions by the agents. Agents features four properties namely autonomy, pro-activeness, reactivity and social ability [14].

Author proposes a new model based on intelligent agents to represent the OOP semantic knowledge. The agent model is based on the Belief, Desire and Intention (BDI) architecture to perform a comparison between two related OO Java source codes and produce a mapping explaining the semantic of these source codes. The Jason AgentSpeak interpreter which adopts the BDI architecture will be used to construct the agents. Therefore, a model of agents featuring four properties mentioned (autonomy, pro-activeness, reactivity and social ability) perform the task of teaching novices to understand the OO concepts.

The paper is organized as follows: Section 2 highlights the proposed method to teach OO and the proposed agent formalization is given in Section 3. Section 4 explains works that are related to our research. Section 5 shows the experimental results of running sample programs using the proposed system. Section 6 gives a brief discussion on the proposed system. Finally, in Section 7 author concludes the paper and suggests future works.

2. PROPOSED TECHNIQUE TO UNDERSTAND OO CONCEPTS

As mentioned in Section 1, novice students face difficulties during the transition from array to OO fundamentals. They find it difficult to relate the OO concepts with the actual problem given to them. When given an OO lab assignment, students will often refer to an example given in the lectures and modify the example to suit it with the problem. A detailed scenario is given as follows.

An easy way to help these novices is to provide an OO source code that is easy to comprehend for example an object is something that we can see, a car with attributes engine, wheel and steering with behaviors reverse, forward and stop; or perhaps, a circle with attribute radius with behavior find the area of the circle. Other objects that can be used as simple examples are table, chair, rectangle, computer, whiteboard, etc. and from these objects it is easy to identify the attribute(s) and the behavior(s) of them.

Lab assignments are given to students to test their ability to apply OO concepts that has been taught in the lectures. Questions are based on operations that we perform everyday, for example accounts transaction, flight information, course registration, bill payment, etc. In Java, these operations are also objects but students find it hard to relate them with simple objects mentioned earlier.

To cater this problem, two OO source codes that are similar but in, a different perspective are provided. The difference is one source code represents a simple example with its own attribute(s) and behavior(s); and another source code represents an operation with its own attribute(s) and behavior(s). These two source codes will be compared based on keyword similarity and the semantics of these source codes will be generated by extracting the words from the source codes. The detailed process will be explained in the next section.

3. THE AGENT FORMALIZATION

Currently, no other research has utilized the intelligent agent technique to understand the OOP semantics.

Author’s research focuses on modeling agents for comprehending OOP semantics using the Jason AgentSpeak interpreter. Jason is an interpreter for an extended version of AgentSpeak based on the BDI architecture. Jason is developed by Jomi F. Hübner and Rafael H. Bordini and implements the operational semantics of AgentSpeak, provides a platform for the development of multi-agent systems, with many user-customizable features [15]. Besides its user-customizable features, Jason is chosen because it is an open source software and easy to access.

The BDI software model is a software model developed for programming intelligent agents [16]. Agents have the beliefs, desires, intentions and events features, and use these concepts to solve a particular problem in agent programming. Beliefs represent the informational state of the agent, and can also include inference rules that allow forward chaining to lead to new beliefs. Desires represent the motivational state of the agent, objectives, situations or goal that the agent would like to accomplish. Intentions represent the deliberative state of the agent or desires, the agent chosen action, which means the agent has begun executing a plan. Plans are sequences of actions that an agent can do to achieve one or more of its intentions. Events are triggers for reactive activity by the agent, and may update beliefs, trigger plans or modify goals.

In our research, author proposed a model for OOP semantics using the reasoning cycle concepts of the Jason agents. The Jason reasoning cycle algorithm is represented in Figure 1. The variables *B*, *D* and *I* is the agent's current belief, desire and intentions. In the outer loop on lines (3) – (4), the agent observes its environment to get the next percept. On lines (3.1) – (3.5), the variables *B*, *D* and *I* is processed and plan, π is generated to achieve intentions based on a set of actions, A_c. The inner loop on lines (3.6) – (3.7), captures the execution of a plan to achieve the agent's intentions. If no problem exists, the agents execute its action from its plan until all the plan has been executed represented in lines (3.6.1) – (3.6.2). However, after executing an action from the plan, on lines (3.6.3) – (3.6.5), the agent pauses to observe its environment, and update its belief again. Lines (3.6.6) – (3.6.7) are executed if the agent needs to reconsider its intentions that lead to a change of intentions. Finally, if the plan is no longer a sound one, the agent replans – lines (3.6.8) – (3.6.9).

```
1. initialize initial beliefs, $B \leftarrow B_0$
2. initialize initial intentions, $I \leftarrow I_0$
3. while true do
   3.1 get next percept, $\rho$ thru sensors
   3.2 update belief, $B \leftarrow brf(b, \rho)$
   3.3 agent determine desires or options, $D \leftarrow options(B,I)$
   3.4 agent choose options, selecting some to become
       intentions, $I \leftarrow filter(B,D,I)$
   3.5 generate a plan to achieve intentions based on set of
       actions, $\pi \leftarrow plan(B,I,A_c)$
   3.6 while not (plan is empty, $empty(\pi)$) do
       3.6.1 process first plan element, $\alpha \leftarrow \pi$
       3.6.2 execute plan element, $execute(\alpha)$
       3.6.3 pause to preserve environment, $\pi \leftarrow$ tail of $\pi$
       3.6.4 observe environment to get next percept, $\rho$
       3.6.5 update belief, $B \leftarrow brf(b,\rho)$
       3.6.6 if $reconsider(I,B)$ then
             3.6.6.1 $D \leftarrow options(B,I)$
             3.6.6.2 $I \leftarrow filter(B,D,I)$
       3.6.7 end-if
       3.6.8 if not $sound(\pi,I,B)$ then
             3.6.8.1 $\pi \leftarrow plan(B,I,A_c)$
       3.6.9 end-if
   3.7 end-while
4. end-while
```

Figure 1. Jason Reasoning Cycle Algorithm

3.1 THE AGENT MODEL

Three agents are designed in author's proposed system. They are SemanticAgentGUI, semanticAgent and noviceAgent.

The proposed agent model is shown in Figure 2. The model is designed using Prometheus notation [17]. The model applies the Jason reasoning cycle algorithm explained in the previous section. The agents are labeled and shown as 'stickman' in a rectangle as shown in Figure 2. The program starts with a desire/goal that is initialized in the SemanticAgentGUI program. The user controls the SemanticAgentGUI to comprehend the source codes. This is represented as user control in Figure 2 and is a type of incident in Prometheus notation. The SemanticAgentGUI retrieves two related source codes from two separate databases. One database stores the simple example source code, another database stores the operation example source code, mentioned in Section 2. The createLiteral method from the Jason ASSyntax library is assigned to the goal that passes three parameters, namely gui_semantic, and the two source codes. The SemanticAgentGUI submit the three parameters to the semanticAgent. The type of triggering event of the goal is achievement-goal and is represented as follows:

+!gui_semantic(code1, code2)

where code1 represents simple example source codes and code2 represents operation example source codes.

The semanticAgent receives the programs with a desire/goal !semantic_novice(code1, code2) and the intention/plan is represented as:

+!semantic_novice(code1, code2)

gui_semantic and semantic_novice are two types of actions exhibit by the proposed agents as in Figure 2.

Next, the source codes are sent to the noviceAgent using the following message:

.send(noviceAgent,tell,give_programs(code1, code2))

The noviceAgent receives the programs, prints them and sends a message to inform the semanticAgent as follows:

.send(S,tell,message(M))

where S represents the semanticAgent and M represents the message.

The semanticAgent receives the message and prints it. The control is then passed to the SemanticAgentGUI to process the source codes and generate the semantics of them. The SemanticAgentGUI interacts with the semanticAgent to give the semantic or meaning of the two compared source codes. This is represented as the 'program semantic explanation and classification' agent action in Figure 2. The 'envelope' notation shown in Figure 2 represents the message that is sent from one agent to another agent. The SemanticAgentGUI keep on running by retrieving source codes in the databases and the loop continues until the user stops/kill the running agent and lastly, the process ends.

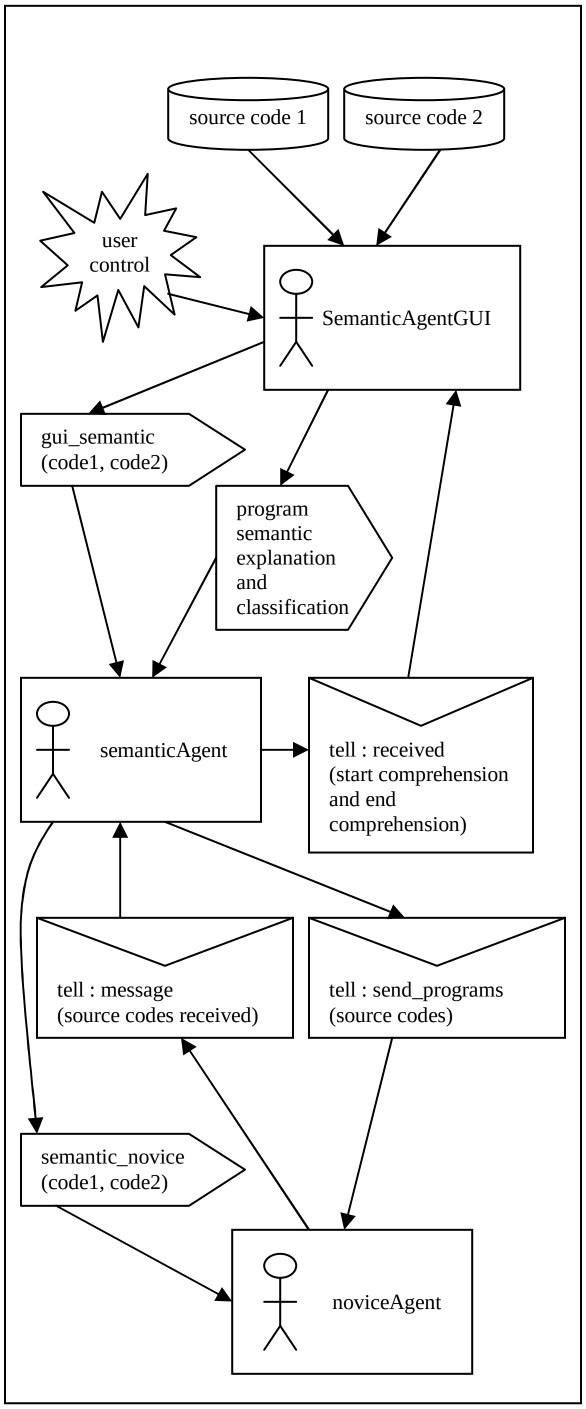

source code 1
source code 2
user control
SemanticAgentGUI
gui_semantic (code1, code2)
program semantic explanation and classification
semanticAgent
tell : received (start comprehension and end comprehension)
tell : message (source codes received)
tell : send_programs (source codes)
semantic_novice (code1, code2)
noviceAgent

Figure 2. Proposed Agent Model

Table 1 shows the constructor/method, belief, desire and intention of the SemanticAgentGUI, semanticAgent and noviceAgent.

TABLE 1. Agent Properties

Statements	Semantic-AgentGUI	semantic-Agent	novice-Agent
Constructor/ Method	Semantic AgentGUI(), act(), stopAg(), processing()	-	-
Belief/ Rule	-	message(M)	give_ programs (code1, code2)
Desire/ Goal	createLiteral() in Semantic AgentGUI constructor	-	-
Intention/ Plan	-	gui_ semantic (code1, code2), semantic_ novice (code1, code2)	-

Figure 3 shows the class diagram for the agents that have been modeled. The SemanticAgentGUI extends the AgArch class which is in the jason.architecture package. The SemanticAgentGUI communicates with the semanticAgent in a bi-direction way, and also the same for semanticAgent and noviceAgent. The SemanticAgentGUI constructor displays the interface containing a button that is controlled by the user. The act method executes an action and, when finished, adds it back in the feedback actions. The stopAg method is a call-back method called by the infrastructure tier when the agent is about to be killed. The processing method will be explained in the next section.

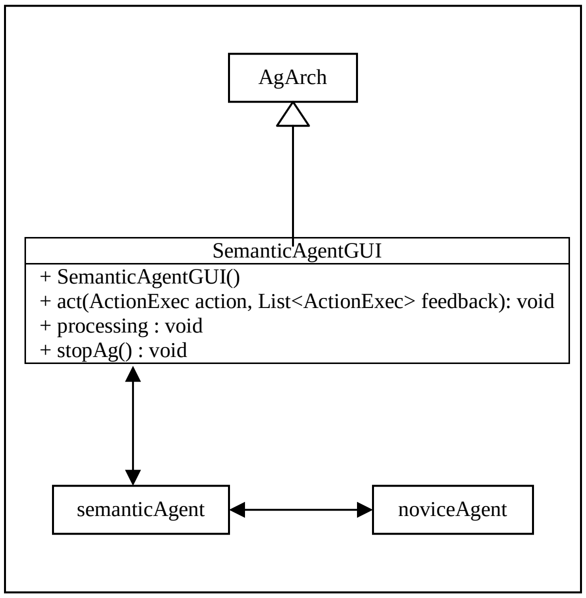


FIGURE 3. Proposed agent class diagram

3.2 AGENT PROCESS AND ALGORITHM

In author's work, semantic knowledge is represented by the agent model. The agent process is shown in Figure 4. code1 represents the simple example source code and code2 represents the operation example source code. code1 and code2 are the input that undergo preprocessing where they will be partitioned into smaller units or granularity using the tokenization technique. In preprocessing, the spaces in the source code will be omitted. The output of the preprocessing will be the preprocessed code1 and code2 that will be stored in an array list data structure. Next, is the comparison process where keywords based on the Java programming language concepts will be compared with the source codes to find the similarities between it. The output of the comparison process is the keyword similarity that exists in the source codes. Then, the following word in the source codes which is different will be extracted from the source codes. The word will be used as an input to generate the meaning or semantics of both source codes. The meaning of the source code will be given based on the extracted word. Lastly, the classification process will classify and count the number of java files, classes, objects, constructors, etc. and produce the result.

Journal of Theoretical and Applied Information Technology
15th October 2011. Vol. 32 No.1
© 2005 - 2011 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

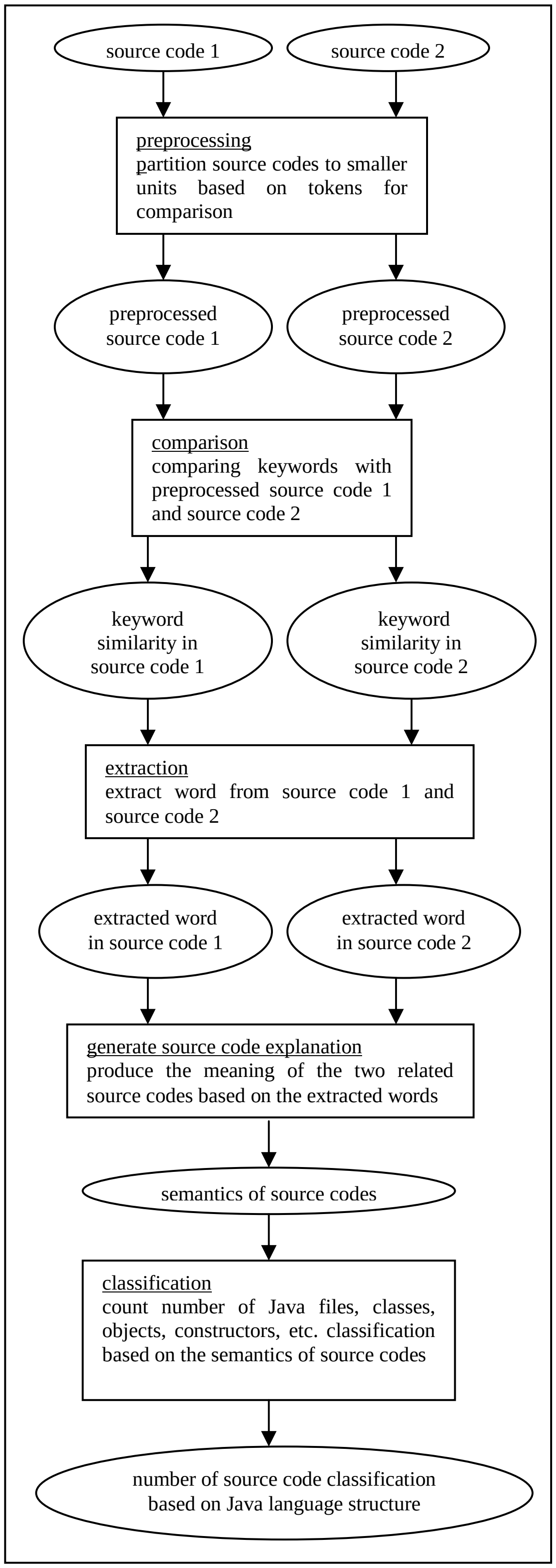

source code 1
source code 2
preprocessing
partition source codes to smaller units based on tokens for comparison
preprocessed source code 1
preprocessed source code 2
comparison
comparing keywords with preprocessed source code 1 and source code 2
keyword similarity in source code 1
keyword similarity in source code 2
extraction
extract word from source code 1 and source code 2
extracted word in source code 1
extracted word in source code 2
generate source code explanation
produce the meaning of the two related source codes based on the extracted words
semantics of source codes
classification
count number of Java files, classes, objects, constructors, etc. classification based on the semantics of source codes
number of source code classification based on Java language structure

FIGURE 4. Proposed agent process

Besides the model, the contribution of this research work is the algorithm that is integrated in the SemanticAgentGUI.java program. Figure 5 shows the algorithm to process the source codes.

```
algorithm processing()
1. retrieve code1 from database
2. retrieve code2 from database
3. read code1 token by token and store in an array list, x
4. read code2 token by token and store in an array list, y.
5. initialize 0 to variables a, b, c, d, e, f, g to calculate the
   classifications of  java file, classes, objects, printouts,
   default constructors, other constructors and returned
   values from a method.
6. initialize 0 to i
7. loop (i < length of array list x && y)
      7.1    if (x[i] = = y[i])
             7.1.1  if (x[i] && y[i] = = "keyword1")
                    7.1.1.1 extract word from x[i] and y[i]
                            and print out the program semantic
                    7.1.1.2 a+ +
             7.1.2  if (x[i] && y[i] = = "keyword2")
                    7.1.2.1 extract word from x[i] and y[i]
                            and print out the program semantic
                    7.1.2.2 b+ +
             7.1.3  if (x[i] && y[i] = = "keyword3")
                    7.1.3.1 extract word from x[i] and y[i]
                            and print out the program semantic
                    7.1.3.2 c+ +
             7.1.4  if (x[i] && y[i] = = "keyword4")
                    7.1.3.1 extract word from x[i] and y[i]
                            and print out the program semantic
                    7.1.3.2 d+ +
             7.1.4  if (x[i] && y[i] = = "keyword5")
                    7.1.4.1 extract word from x[i] and y[i]
                            and print out the program semantic
                    7.1.4.2 e+ +
             7.1.5  if (x[i] && y[i] = = "keyword6")
                    7.1.3.1 extract word from x[i] and y[i]
                            and print out the program semantic
                    7.1.3.2 f+ +
             7.1.6  if (x[i] && y[i] = = "keyword7")
                    7.1.6.1 extract word from x[i] and y[i]
                            and print out the program semantic
                    7.1.6.2 g+ +
      7.2 end-if
      7.3 i+ +
8. end-loop
9. print out the variables a, b, c, d, e, f, g for classification
```

FIGURE 5. Proposed semantic algorithm

4. RELATED RESEARCH

The reasons of source code comparison research implementations are to detect the differences in Source Code Management (SCM), to detect plagiarism in students' assignments and to detect clone in software development.

SCM, a tool in software development projects provides the ability to store and retrieve past versions of source files. Comparison tools help SCM in highlighting the differences. A metaphor for comparison based on a single-pane interface where common text is displayed only once with differences combined into a single text to improve readability and usability in terms of difference classification (additions, deletion, and modifications) was invented [18]. Other comparison tools are built around a two-pane interface, with files displayed side by side. These types of interfaces are inefficient in the use of screen space and ineffective because duplication makes text more difficult to read, result difficulties to the user in performing comparison tasks. These works are mainly focused on improving the interface and classification. Author's research work does not focus on improving the interface but classification is used to classify the words in the source codes into java classes, objects, constructors, etc.

Detecting plagiarism manually is very time consuming. A plagiarism detection system integrated with Online Course Management System (OCMS) was designed [19] to overcome this problem. An agent serves as a daemon to analyze the program codes in terms of textual analysis for strings, structural analysis for method collections and variable analysis for code line collections. This work is different from author's work where agents are used to analyze two program source codes by comparing them to find the keyword similarity, extract words from the source codes and generate the semantics of the codes.

Code clones are software systems that contain sections of code that are similar. Cloning may be useful in many ways [20], [21] but can also be harmful in software maintenance and evolution [22]. Authors provide a qualitative comparison and evaluation in clone detection techniques and tools, and organize the information into a conceptual framework [23]. In their work, they identified four different types of clones that are similar. In this paper, author compare source codes, get the keyword similarity that exist in the source codes and extract words that can help novice to understand the OOP semantics. To extract words author use the tokenization approach. The tokenization approach has been used in CCFinder [24] and Dup [25] clone detection tools. However, these tools do not generate the semantics of program statements.

An approach to teaching algorithms called Structured Hypermedia Algorithm Explanation (SHALEX) system have been developed [26]. SHALEX uses hypermedia and represents algorithms as an abstract tree structure. An intelligent agent is integrated in SHALEX to monitor student progress, to provide the students with hints where necessary and to record the results of student interaction that shows the level of comprehension the students has achieved. In author's system, the intelligent agents represent the semantics of the OOP knowledge.

A conceptual framework [27] has been proposed using a knowledge representation language named Telos [28], developed at the University of Toronto. The architecture consists of three layers: agent layer, server layer and repository layer. The agent layer contains all the application agents. All communication among application agents is through the knowledge server. The repository stores all the common knowledge and information to run the system [29]. Knowledge representation in authors' work is represented by an algorithm that is integrated in the Jason AgentSpeak language which represents the semantics of OOP. In the author's proposed architecture, the user interact with the agents through a Graphical User Interface (GUI) and the agent present source codes samples and generate the semantics of the source codes that explains the relation and meaning of the source codes.

A unified formalism has been proposed based on the BDI architecture to model computational rational agents to understand Natural Language [30]. Agents parse sentences and uses the proposed formalism to represent them. Then, the agents perform actions based on the problem domain using the information provided in the sentences. Besides that, the agent is also able to carry on other requests. Although author model the agents using the BDI architecture, author's research work is different, where the agents are programmed to understand program source code by giving the semantics of the codes.

In author’s approach source code comparison are used to comprehend OO concepts by providing the semantics of two OO source codes, which is different from other authors approach mentioned. To date, no other research works have utilized intelligent agents explaining the semantics of source codes as a knowledge representation perspective. Therefore, the design of the architecture is very much different from other existing research.

5. EXPERIMENTAL RESULTS

Figure 6 and Figure 7 show the source codes, which are the input to the preprocessing process. Basically, the codes are similar but they are in different domains. In the CircleObject source code, the object circle is easy to view because simply a circle has the attribute radius and we can find its radius and calculate its area. In the AccountObject source code, balance is the attribute and we can find the withdrawal value and calculate the balance. However, it is not an easy task for the novice to write the AccountObject source code. Therefore, by providing the CircleObject source code, comparing it with the AccountObject source code and extract the different words that exist in the source code based on similar keyword, will present a clear comprehension by giving the meaning or semantics of them.

```
public class CircleObject {
public static void main(String[] args) {
Circle circle = new Circle ( 2.5 );
double RadiusValue = circle . getRadius() ;
double AreaValue   = circle . getArea() ;
System.out.println ( " CircleRadius     = " + RadiusValue );
System.out.println ( " CircleArea       = " + AreaValue );
}
}
class Circle {
double radius  ;
 Circle () {
 radius = 1 ;
 }
 Circle (double newRadius ) {
 radius = newRadius;
}
double getRadius() {
 return radius  ;
}
double getArea() {
 return radius * radius * Math.PI;
}
}
```

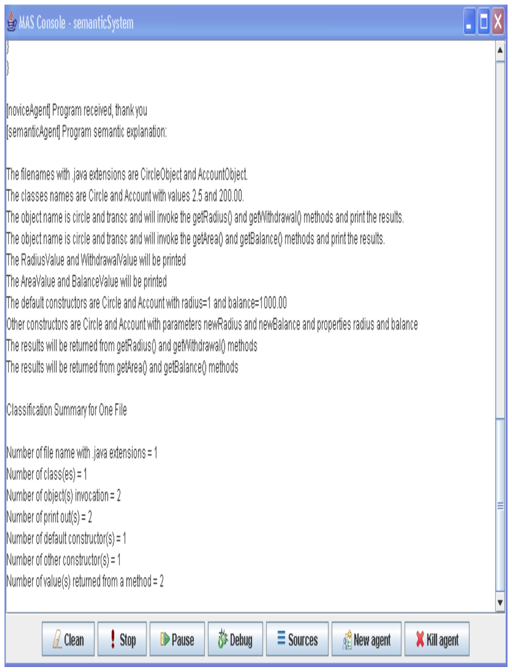
FIGURE 6. CircleObject source code

```
public class AccountObject {
public static void main(String[] args) {
Account transc = new Account ( 200.00 );
double WithdrawalValue = transc . getWithdrawal() ;
double BalanceValue = transc . getBalance() ;
System.out.println ( " WithdrawalAmount = " +
WithdrawalValue );
System.out.println ( " BalanceAmount    = " + BalanceValue
);
}
}
class Account {
double balance ;
 Account () {
 balance = 1000.00 ;
 }
 Account (double newBalance ) {
 balance = newBalance;
}
double getWithdrawal() {
 return balance ;
}
double getBalance() {
 return 1000.00 - balance;
}
}
```

FIGURE 7. AccountObject source code

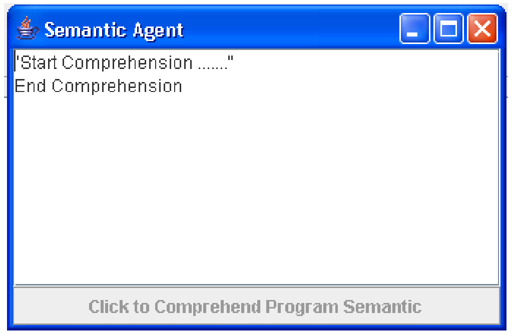
Figure 8 shows the output of the system. An example of words that are extracted from the source codes are CircleObject and AccountObject. The keyword similarity compared is the word ‘class’. Comparing the two source codes provide the difference between them in terms of Java constructs, namely file names, classes, object name, print out, constructors and methods. The classification result show the number of java constructs that exist in one source code.

The user ‘clicks’ the button ‘Click to Comprehend Program Semantic’ to start comprehending the source codes as shown in Figure 9. The figure shows the GUI that is controlled by the user continuously until the agent is terminated/killed and the process ends.



```
}
}


[noviceAgent] Program received, thank you
[semanticAgent] Program semantic explanation:


The filenames with .java extensions are CircleObject and AccountObject.
The classes names are Circle and Account with values 2.5 and 200.00.
The object name is circle and transc and will invoke the getRadius() and getWithdrawal() methods and print the results.
The object name is circle and transc and will invoke the getArea() and getBalance() methods and print the results.
The RadiusValue and WithdrawalValue will be printed
The AreaValue and BalanceValue will be printed
The default constructors are Circle and Account with radius=1 and balance=1000.00
Other constructors are Circle and Account with parameters newRadius and newBalance and properties radius and balance
The results will be returned from getRadius() and getWithdrawal() methods
The results will be returned from getArea() and getBalance() methods


Classification Summary for One File


Number of file name with .java extensions = 1
Number of class(es) = 1
Number of object(s) invocation = 2
Number of print out(s) = 2
Number of default constructor(s) = 1
Number of other constructor(s) = 1
Number of value(s) returned from a method = 2
```

FIGURE 8. Program output



```
"Start Comprehension ......."
End Comprehension
```

FIGURE 9. GUI controlled by the user

6. DISCUSSION

The proposed OOP semantic system illustrates the four agent features: autonomy, pro-activeness, reactivity and social ability. The agent is able to operate independently, to achieve the goals to submit programs that we delegate to it and make independent decision to generate the OO semantics and classify programs under its own control. The agent also exhibit pro-activeness feature in the sense of its goal-directed behavior and its success to achieve the goals. In addition, the agent poses reactive characteristic equivalent to the changes in the user's action that control the GUI system. Lastly, the agent social ability attribute show the communication between agents in sending source codes and generating the OOP semantics. Combining all the agent features, students comprehend the OOP semantics based on the role of the agents. Therefore, knowledge representation is portrayed by the agents.

For the time being, the proposed system works well to generate the semantics of source codes statements based on certain Java keywords for example, class, new, double and return, etc. Author plan to extend and improve the algorithm to cater more semantics of source code statements based on a wide range of Java keywords. In addition, various way of writing program source codes to produce the semantics of them will also be considered.

The classification summary can also be improved by giving the relation meanings of words that appear more than once for example in Figure 6, words like Circle, circle, RadiusValue, AreaValue, getRadius, getArea, etc. In Figure 6, the use of new keyword to create the circle object reference variable is related to circle.getRadius() and circle.getArea() but the meanings of the three statements are different and can be classified as create object and accessing the class methods.

The constraint of the research work is that it needs a pair of source codes associated with each other to execute using the proposed system. Therefore, the instructors need to think and provide examples of these source codes to be stored in the database before the system can produce the results. It may take some time to provide source codes examples.

Program source codes can be represented in various ways for instance, through software interface visualizations and abstract tree structure. Although current state-of-the-art used agents to detect source codes similarities but they do not provide the semantics of the source codes. The significance of the proposed model is that it explains the semantics of two related source codes that can aid novices to comprehend in depth the concepts of OOP. Here, program source codes are represented by the role of intelligent agents that hold the semantic knowledge of the program source codes.

7. CONCLUSION AND FUTURE WORKS

An agent-based model featuring three intelligent agents namely, SemanticAgentGUI, semanticAgent and noviceAgent have been designed to assist novice students to comprehend the OOP concepts. The whole process involves preprocessing, comparison, extraction, generate semantics and classification techniques. In this paper, semantic knowledge is represented by intelligent agents. This system is programming language dependent which is based on Java programming source codes.

Author plan to extend and improve the proposed algorithm to cover a large number of keyword similarity exist in Java constructs. The database will also be added to include more source codes examples to be compared. In addition, the algorithm should also adapt with source codes statements in different writing ways but same meaning. For example, an object and method can be accessed by assigning them to a variable and then print the result. Therefore, this involves two statements as in our example in Figure 6 and 7. The two statements are also equivalent to one statement where the object and method can be written in one print statement. Future work also includes integrating the proposed agent model in an Integrated Development Environment (IDE).

REFERENCES:

[1] L. Longshu, and X. Yi, “The Teaching Research on a Case of Object-Oriented Programming”, *The 5th International Conference on Computer Science & Education*, Hefei, China, August 24-27, 2010, pp. 619-621.

[2] R. Noa, “A Pedagogical Approach to Discussing Fundamental Object-Oriented Programming Principles”, *ACM Inroads*, Vol. 1, No. 2, 2010, pp. 43-52.

[3] X. Stelios, “An Interactive Learning Environment for Teaching the Imperative and Object-Oriented Programming Techniques in Various Learning Context”, *Communications in Computer and Information Science*, 111 CCIS (PART 1), 2010, pp. 512-520.

[4] H.M. Qusay, D. Wlodek, and S. David, “Making Computer Programming Fun and Accessible”, *Computer*, Vol. 37, No. 2, 2004, pp. 108+106-107.

[5] L. Chuck, and R. John, “Learning O-O Concepts in CS1 Using Game Projects”, *ITiCSE '04: Proceedings of the 9th Annual SIGCSE Conference on Innovation and technology in Computer Science Education*, Leeds, United Kingdom, June 28-30, 2004, pp. 237.

[6] K. Michael, “GreenFoot – A Highly Graphical IDE for Learning of Object-Oriented Programming”, *ITiCSE '08 Proceedings of the 13th annual conference on Innovation and technology in computer science education*, Madrid, Spain, June 30-July 2, 2008, pp. 327.

[7] K. Machael, and R. John, “Blue – A Language for Teaching Object-Oriented Programming”, *SIGCSE '96: Proceedings of the Twenty-seventh SIGCSE Technical Symposium on Computer Science Education*, Philadelphia, USA, 1996, pp. 190-194.

[8] S. Yaskawa, and A. Sakata, “The Application of Intelligent Agent Technology to Simulation”, Mathematical and Computer Modelling 37, *Mathematical and Computer Modelling*, Vol. 37, No. 9-10, 2003 pp. 1083-1092.

[9] B.L. Iantovics, “Agent-based Medical Diagnosis Systems”, *Computing and Informatics*, Vol. 27, No.2, 2008, pp. 593-625.

[10] D.A. Bell, A. Beck, P. Miller, Q.X Wu, and A. Herrera, “Video Mining –Learning Patterns of Behaviour via an Intelligent Image Analysis System”, *Proceedings of the 7th International Conference on Intelligent Systems Design and Applications*, ISDA 2007, art. no. 4389651, 2007, pp. 460-464.

[11] H. Li, F. Karray, and O. Basir, “A Framework for Coordinated Control of Multi-agent Systems”, *Studies in Computational Intelligence*, Vol. 310, 2010, pp. 43-67.

[12] J. Aguilar, and W. Zayas, “A Multiagents System to Create Control Agents”, *Applied Artificial Intelligence*, Vol. 24, No. 8, 2010, pp. 785-806.

[13] J. Tvarožek, J., and M. Bielikovå, “Feasibility of a Socially Intelligent Tutor”, *Lecture Notes in Computer Science*, 2010. pp. 423-425.

[14] M. Wooldridge, and M.R. Jennings, “Intelligents: Theory and Practice”, *The Knowledge Engineering Review*, Vol. 10, No. 2, 1995, pp. 115-152.

[15] http://jason.sourceforge.net/Jason/Jason.html

[16]http://en.wikipedia.org/wiki/Belief-Desire-Intention_software_model

[17] L., Padgham, and M., Winikoff, 2004. “Developing Intelligent Agent Systems: A Practical Guide”, Wiley, Chichester, 2004.

[18] M. Lanna, and D. Amyot, “Spotting the difference, *Software Practice and Experience*, Vol. 41, No. 6, 2010, pp. 607-626.

[19] J.K. Yih, and F.H. Chu, “Code Analyzer for an Online Course Management System”, *The Journal of Systems and Software*, Vol. 83, No. 12, 2010, pp. 2478-2486.

[20] L. Aversano, L. Cerulo, and M.P. Di, “How Clones are Maintained: An Empirical Study”, *in Proceedings of the 11th European Conference on Software maintenance and Reengineering, CSMR 2007*, art. no. 4145027, 2007, pp. 81-90.

[21] K. Cory, and W.G. Michael, “Cloning Considered Harmful: Patterns of Cloning in Software”, *Empirical Software Engineering*, Vol. 13, No. 6, 2008, pp. 645-692.

[22] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, “Do code clones matter?” *in Proceedings of the 31st International Conference on Software Engineering, ICSE’09*, Vancouver, Canada, May 16-24, 2009, pp. 485-495.

[23] K. R. , Chanchal, R. C., James, and K. Rainer, “Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach”, *Science of Computer Programming*, Vol. 74, 2009, pp. 470-495.

[24] T. Kamiya, S. Kusumoto, and K. Inoue, “CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code”, *IEEE Transactions on Software Engineering*, Vol. 28, No. 7, 2002, pp. 654-670.

[25] B. Baker, “A Program for Identifying Duplicated Code”, *in Proceedings of Computing Science and Statistics: 24th Symposium on the Interface*, vol. 24, 1992, pp. 49-57.

[26] M. S., Elhadi, and H., Richard, “An Agent for Web-based Structured Hypermedia Algorithm Explanation System”, *Journal of Universal Computer Science*, Vol. 15, No. 10, 2009, pp. 2078-2108.

[27] W., HuaiQing, “LearnOOP: An Active Agent-Based Educational System”, *Expert Systems with Applications*, Vol. 12, No. 2, 1997, pp. 153-162.

[28] M. John, B. Alex, J. Matthias, and K. Manolis, “Telos: A Language for Representing Knowledge About Information Systems” *ACM Transactions on Information Systems*, Vol. 8, No. 4, 1990, pp. 325-362.

[29] H., Wang, “Repositories for Co-operative Information Systems”, *Information and Software Technology*, Vol. 38, No.5, 1996, pp. 333-341.

[30] K., Deepak, H., Susan, and S.A., Syed, “Towards a Unified AI Formalism”, *Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences*, Vol. 3, 1994, pp. 92-101.