An Innovative Instructional Method for Teaching Object-Oriented Modelling

Khalid Al-Tahat
Department of Information Technology and Computing, Arab Open University, Jordan

Abstract: *Object-oriented modelling is considered to be complicated to teach and learn in introductory courses in computer science and software engineering. Animated program visualisation can be significantly used to support teaching object-oriented modelling for beginners. However, there is a lack in instructional methods that support such approach. This paper bridges the gap by introducing a new instructional method for teaching object-oriented modelling using the Unified Modelling Language (UML) with the help of a visualised 3D programming environment; Alice. The proposed instructional method is the first of its kind to introduce teaching software modelling with the support of animated visualisation tool. The basic approach is to extract the UML components from a 3D Alice world, then map these components to build the UML model that represents that world. This approach shows that animated program visualisation has the potential to help in teaching other aspects of software development process besides programming including modelling.*

Keywords: *Teaching object-oriented modelling, UML, object-oriented paradigm, animated program visualisation, alice.*

Received May 8, 2012; accepted April 16, 2013; published online March 13, 2014

1. Introduction

Models are typically "scaled down" representations of real-world objects and systems. The reduction in scale may include size, detail, and aspect. In the development of information technology solutions, models are used to manage complexity and to communicate system requirements between business stakeholders, solution and system architects, developers, and operations personnel [5]. Modelling is central for learning and performing object-oriented development [12]. Starrett [20] states that "modelling can be taught and perhaps should be taught before programming; modelling is more generally applicable, methodical and feasible".

Formal presentations of object-oriented concepts and modelling are mainly theoretical; object-oriented software engineering books and resources are extremely wordy with minimum diagrams and illustrations. As a result, students find it hard to understand these concepts without being mapped to diagrams drawn with the instructor in the class.

When teaching Unified Modelling Language (UML) [11], instructors typically refer to written requirement specifications that describe the system's entities, functionalities and restrictions. Students are then asked to carefully read these requirements, understand them and then draw UML diagrams based on this understanding. For example, to build the class diagram, the requirement specifications is examined for all nouns. These nouns become the list of potential classes. Information related to these nouns is further examined to determine the attributes, behaviours and relationships of these classes. Our experience shows that beginners find it hard to extract enough information from written requirements specification to draw accurate UML diagrams. We believe that introducing the system to beginners through visual representations of its objects gives them a big picture view of the system and helps them understand it better. This would lead to students producing more accurate UML diagrams of the system model. Interactive animated program visualisation can be significantly useful for teaching software modelling using UML, as we propose in this work.

To support teaching object-oriented modelling for beginners, we suggest an instructional method that benefits from the use of a visualisation tool. Using visualisation technologies help students understand object-oriented modelling by keeping the focus on the visual representation of the objects while learning about modelling them and modelling their relationships.

The tool used in our approach is Alice (www.alice.org). Alice is an incredibly fun 3D programming environment created and distributed by Carnegie Mellon University. Alice has an object-oriented flavour and has recently gained attention as a gentle introduction to object-oriented programming [16]. The special characteristics of Alice such as 3D animation, real-life experience of dealing with objects, a direct-manipulation editor for instantiating objects, simple event handling mechanism, multi-media capabilities and "syntax-free" IDE, engage students and raises their interest in learning programming [2, 16]. Several researchers and computing scholars such as [2, 4, 7] report the benefits of using Alice as an object-first tool; students confidence had increased in

their programming ability and understanding of basic programming concepts.

Inspired by the benefits of using Alice in teaching programming, we investigated using Alice in teaching software modelling with UML. We believe that students who enjoyed creating Alice worlds by choosing their objects and designing their interactions will find it interesting as well as easy to understand and build the models behind these worlds.

The structure of the rest of paper is as follows: Section 2 introduces UML as a modelling tool. We present the most relevant related work in the field in section 3. Alice worlds are explored in section 4. The main elements of the UML static model from the object-oriented paradigm point of view and showing how to teach those using the suggested approach are covered in section 5. Section 6 includes our observation on using the proposed approach. Concluding remarks are given in section 7.

2. UML as an Object-Oriented Modelling Tool

Object-oriented modelling is important to help students visualising the real world and object-oriented concepts. Object-oriented modelling is considered to be complicated for introductory courses in computer science and software engineering; students need to learn both large number of new concepts and large number of symbols of new graphical notation. The key for understanding object-oriented modelling is to understand how to model objects and classes and model the relationships between them.

UML is a diagrammatic notation that has been accepted as a standard notation in industry and academia [3]. UML is widely used in the software industry and often taught in universities as a way to represent software requirements specification and design descriptions [9]. The widespread academic interest in the UML helped it find its way into many computer science and software engineering curricula. In these curricula, UML has been used as a tool for modelling systems and teaching the object oriented paradigm [18]. Some computing educators believe that UML is one important tool/ technology that must be taught to students besides the other things such as programming, technologies and paradigms [20].

UML helps students express their ideas about software modelling in the form of pictures. Finding new and easy approaches for teaching UML helps instructors incorporate UML into introductory computer science and software engineering modules. This offers an opportunity to achieve many recommended teaching requirements [14, 15]; involving students in modelling will raise their interest in learning, help them realise the relevance of the concepts taught and better engage them in the problems they are trying to solve.

UML provides several diagram types that can be used to view and model the software system from different views, perspectives and levels of abstraction [7]. UML has two main views of the software system; static and dynamic. The static view of the model concerns mostly structural issues such as attributes and behaviours for system's classes. Whereas, a dynamic view of the model concerns the manner in with the system components interact. For example, a class diagram shows a static view of the system, where interactions diagrams show a dynamic view of the system [18].

3. Related Work

The issues of teaching object-oriented modelling are also discussed Tamami [21] where the author of that paper shows how graph theory formalisms can be used to teach modelling. Another approach that considered teaching software modelling should be done in courses which contain project activities and use students as subjects in software engineering experiments is introduced Filho [8]. In this paper, we are the first to propose an instructional method for teaching modelling with the support of visualization tool. Visualization has been employed in computer science education since, the 1980s and has been successfully used to actively engage learners in different types of activities [1, 16, 17].

4. Exploring Alice World

This section explores Alice worlds; introducing the Alice interface components and describes a simple Alice world that was created to demonstrate the proposed approach of teaching UML with the help of Alice. In the screenshot of the Alice interface, Figure 1:

a. The world window provides a view of the virtual world that a students' program will control.
b. The object tree contains a list of the 3D objects in the virtual world.
c. The details area shows the properties, methods and functions for the object selected in the object tree.
d. The methods editor shows the code that defines a method a student is working on.
e. The events area allows students to call methods based on events in the world, such as mouse clicks or changes in the value of a variable.

Alice programmer can chose to add objects from the Alice gallery which includes more than 350 3D objects. These objects are organised into groups including people and animals. Alice programmer can access the Alice gallery by pressing the "Add Objects" button underneath the world window as shown in Figure 1-a. This causes the gallery to expand and allows the programmer to select the object they need. When selecting the object, a dialog with the object's

information appears with a button to confirm the adding of this object to Alice world.

To build an Alice program, students choose the object they want to program from the object tree as shown in Figure 1-b. This will show all the properties, methods and function of that object as command tiles. Then, programmers drag command tiles of that object from the details area as shown in Figure 1-c into the method editor as shown in Figure 1-d, they then drop it and select the parameter values from a pop-up menu that appears. Method editors can contain any number of method calls as the programmer desires. These methods perform animations on whole 3D objects (e.g., moving a penguin through space) as well as an object's subparts (e.g., penguin's head).

Alice supports collaborative programming and when students press the play button as shown in Figure 1 they are immediately able to see how their own animated program runs. Alice worlds can be seen as simulations for the real-world and students can train to model these worlds as part of their training to be future software developers.

As this work is focusing on the static model of UML, our concern will be regarding the objects of Alice world; their characteristics, their behaviours and relationships existing among and between them. Object interactions and communications in the Alice world come under the interaction model of UML, which is not our concern in this work.

The rest of this work will show how to build the static model diagrams for objects appearing in a snapshot of the Alice scenario in Figure 1-a.

Figure 1. A screenshot of the Alice interface.

5. Teaching the UML Static Model with the Help of Alice

This section covers the main object-oriented concepts in the static model. We start by explaining the object-oriented concept from the object-oriented paradigm point of view, then show how Alice supports this concept and then model it using UML notation.

5.1. Objects, Attributes, Operations and Encapsulation

In the object-oriented paradigm, objects in a software system are viewed, just as in the real world, as things around us [5]. These things have certain properties or attributes, and are capable of performing certain behaviours or operations. The collection of these attributes and behaviours identify an object and give it its identity. Objects have relationships with other objects and they may exchange messages based on these relationships.

In Alice, programmers can add objects to their world by pressing the "Add Objects" button underneath the world window as shown in Figure 1-a. Students can add an object to their virtual world by dragging the object into the 3D scene and/or by clicking on the picture of the object and pressing the "Add instance to world" button on the dialog box that appears.

Figure 2 gives a closer look at the Alice world snapshot in Figure 1-a. It shows four objects living in this world. The object tree, Figure 1-b, tells us the names of these objects. They are 'Sam', 'Wie', 'Ali' and 'Sally'.

An attribute, in the object-oriented paradigm, describes a property of an object. The value of an attribute is a piece of information that an object knows.

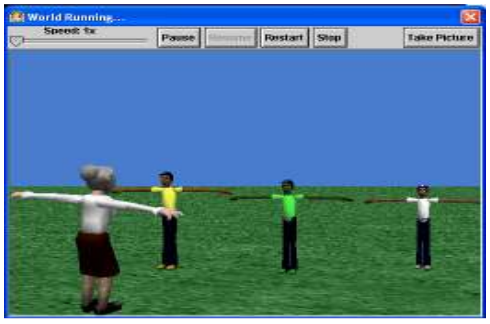

Figure 2. A closer look at Alice world window in Figure 1-a.

Clicking the properties tab in the Alice world's details area as shown in Figure 1-c shows the objects' attributes/ properties. Alice defines a number of default properties for its objects and gives Alice programmers the chance to define more properties as part of their program development. Figure 3 shows Sam's properties; besides others, 'Sam' has the properties 'course', 'age', 'gender' and 'color'.

Operations, in the object-oriented paradigm, are procedures that an object knows how to follow to perform its behaviours. Clicking the methods tab in the Alice world's details area, as shown in Figure 1-c shows the objects' behaviour/ methods.

Alice implements a number of default methods for its objects and gives programmers the chance to implement more methods as part of their program development. Figure 4 shows the methods of the Student's object 'Sam'; besides others, 'Sam' has the methods 'SetForExam', 'PresentWork', 'move' and 'say'.

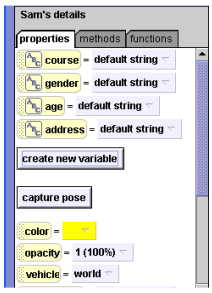

Figure 3. Sam's attributes/ properties.

Packaging the attribute of an object with its behaviours within that object makes the object seen as being an entity and that is how encapsulation is defined in object-oriented paradigm. It makes us see the object as one unit. It also includes hiding the internal structure of an object so other objects can only access the object's data or change its attributes only through its public methods. Alice supports encapsulation by introducing an object as a 3D entity associated with its attributes and behaviours. In the following section, we introduce the UML object diagram with the help of Alice objects described earlier.

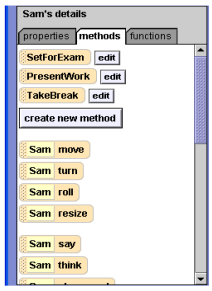

Figure 4. Sam's behaviours/ methods.

5.1.1.Object Diagram

Students start UML modelling by modelling the object diagram. In UML, an object diagram is a snapshot of the objects of an application at a point in time. Since it shows instances rather than classes, an object diagram is often called an instance diagram [19]. In UML, the object diagram is composed of three compartments, as shown in Figure 5. The first compartment shows the object name and its class name. The second compartment contains the properties of that object, called attributes in UML, and their values. The last compartment contains the methods of that objects, called behaviours in UML, which the object can perform and know how to carry out.

ObjectName: ClassName
-Attributes + Values
+Behaviours()

Figure 5. UML object diagram.

Student and teacher are examples of objects that live in the Alice world in Figure 1. Four objects appear in the snapshot as shown in Figure 2, namely, 'Sally', 'Sam', 'Ali' and 'Wie'. To draw the object Students start UML modelling by modelling the object diagram. In UML, an object diagram is a snapshot of the objects of an application at a point in time. Since, it shows instances rather than classes, an object diagram is often called an instance diagram [19]. In UML, the object diagram is composed of three compartments, as shown in Figure 5. The first compartment shows the object name and its class name. The second compartment contains the properties of that object, called attributes in UML, and their values. The last compartment contains the methods of that objects, called behaviours in UML, which the object can perform and know how to carry out.

Diagram for these objects, we will refer to the objects' details in the Alice world and extract objects' properties and methods from there. For example, to draw the object diagram for the object 'Sam', select the object from the object tree in Figure 1-b. The object name, 'Sam', goes into the first compartment of the object diagram. At this stage, Alice doesn't help us know the name of the class of the objects in the snapshot, so we will leave this part until we cover the class diagram in section 5.2. The object's details appear in the details area, Figure 1-c. The properties tab in the details area shows the properties/attributes of the object 'Sam' as explained earlier, as shown in Figure 3. The attributes of the object 'Sam' go into the second compartment of the object diagram as shown in Figure 6.

Sam:
-course = 'Computing' -gender = 'Male' -age = 20 -address ='UK' -color = 'yellow'

Figure 6. Sam's attributes.

The methods tab, in the details area, shows the methods/ behaviours of the object 'Sam', as shown in Figure 4. The behaviours of the object go into the third compartment of the object diagram. Figure 7 shows the complete object diagram for the object Sam after adding the behaviours into the third compartment.

Sam:
-course = 'Computing' -gender = 'Male' -age = 20 -address ='UK' -color = 'yellow'
+setForExam() +presentWork() +takeBreak() +move() +say() +think()

Figure 7. Sam's object diagram.

By performing the same process on the other objects, in the same way as in Sam's object, we can

complete the object diagrams for the other objects in the snap shot in Figure 2, as shown in Figure 8.

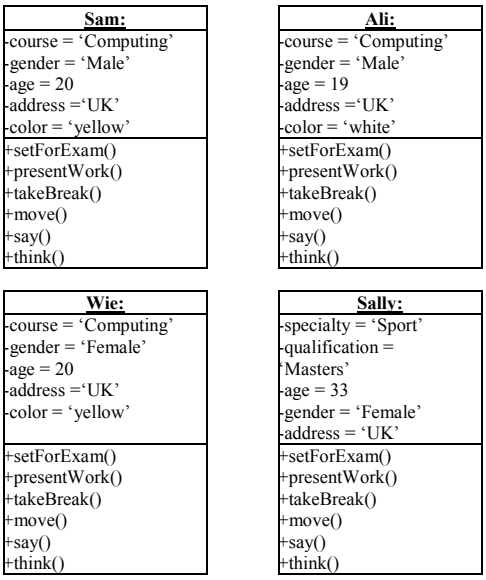

Figure 8. The object diagrams for the objects in Figure 2.

In Figure 8, the relationships between the objects are not specified yet. In section 5.3, we will be learning about the UML relationships that may exist between objects and complete the object diagram by adding the relations between the objects shown in Figure 8.

After introducing the object concept in object-oriented paradigm and object diagram in UML, the next section introduces the class concepts as an abstract data type and shows how the abstraction mechanism is used to identify abstract data types. The section will also show how to draw the class diagram in UML.

5.2. Class of Objects and Abstraction

Based on the above definitions of attribute and behaviour, we may define a class of objects on the basis of having common attributes and behaviours. These are common to all the objects in the class. Objects that have the same attributes, operations and relationships belong to the same class. A relationship between two objects results in association among their classes. The class diagram in UML as shown in Figure 9 is very similar to the object diagram but, with two main differences; the first compartment shows the class name only and the second compartment shows the attributes without assigning them any specific values. The later is because an attribute value is not specified until it is associated with a particular object.

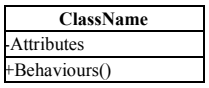

Figure 9. UML class diagram.

Alice programmers are introduced to the class concept when adding new objects to their Alice virtual world. For instance, when Alice programmers choose to add a new 'Student' object to their Alice world, they drag the object of the chosen class into the 3D scene. Figure 10 shows adding an object of type 'Student' to the Alice world. This method for instantiating, that is creating an instance of an object, Alice objects makes Alice programmers see a class as a template for creating objects, which is another common definition for the class in object-oriented paradigm.

As we are working on snapshots from a ready Alice world, we are not aware of the classes that the objects in the snapshot as shown in Figure 2 were created from. So, we need to depend on the information provided to us by Alice, at this point, to identify the classes of these objects.

Figure10. Instantiating an object from class Student in Alice.

The object diagrams produced in section 5.1.1, Figure 8, shows that the objects 'Sam', 'Ali', and 'Wie' all have common attributes and behaviours. All three objects have the attributes 'course', 'age', 'gender' and 'color' and they all can perform the behaviours 'SetForExam', 'PresentWork', 'TakeBreak', 'move', 'say' and 'think'. This means that these three objects are instances of the same class. If we look carefully at these attributes and behaviours, we will find that they are the attributes and behaviours of a student. So, we can confirm that these three objects are of the abstract data type (class) 'Student'. The class diagram for the 'Student' class, abstracted from the three objects diagrams for 'Sam', 'Ali', and 'Wie' as shown in Figure 11.

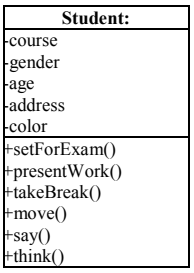

Figure 11. 'Student' class diagram.

The fourth, and last, object, in the Alice world shown in Figure 8 is 'Sally'. This object has different attributes and behaviours from the other objects. This means that this object belongs to a different class. Studying the attributes and behaviours of the object 'Sally', tells us that 'Sally' has all the attributes and behaviours of a teacher. This leads us to confirm that 'Sally' is a member of the class 'Teacher'. The class

diagram for 'Teacher' class, abstracted from the object 'Sally' as shown in Figure 12.

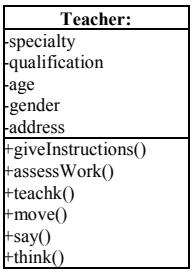

Figure 12. 'Teacher' class diagram.

To carry out the abstraction process in the classroom, students are given objects of the same class and asked to extract their common attributes and behaviours to create an abstract data type; the class. As a result the class diagram will naturally evolve from the object diagram(s) as demonstrated above. This helps us introduce the concepts of abstraction and abstract data type (class) in the object-oriented paradigm.

After identifying the class names for the objects in our Alice snapshot, we can now complete the object diagram, in Figure 8, by adding the class names in the first compartment of the object diagram(s).

So far, we have explaining the basic concepts in the object-oriented paradigm, namely, object and class, and shown how to draw the UML diagrams for them. The rest of the work will explore the different types of relations that may exist between classes in the object-oriented paradigm and how to draw them using UML notation with the help of Alice.

5.3. Association Relationship and Multiplicity

An association relationship, in the object-oriented paradigm, means that objects of one class are naturally associated with objects of another class in some way [5]. Association shows how an object of one class communicates with and gains access to an object of another class. In UML, the association is reflected by a solid line with two association ends; each end is attached to one of the objects/ classes in the association. The association can also be given a name. In the snapshot as shown in Figure 2 a single object of type 'Teacher' is associated with three objects of type 'Student' with a relationship named 'teaches', see Figure 13. In the UML, this means that 'Sally' teaches a number of students or these students are taught by the teacher 'Sally'. Besides other communication forms, this association allows 'Sally' to 'giveInstructions' to her students and, on the other hand, allows students to 'presentWork' to their teacher.

Figure 13 shows the complete object diagram for the objects in the snapshot in Figure 2; it includes class names and the relations between the objects involved. This diagram can be used to draw the association relationship between the classes of these objects.

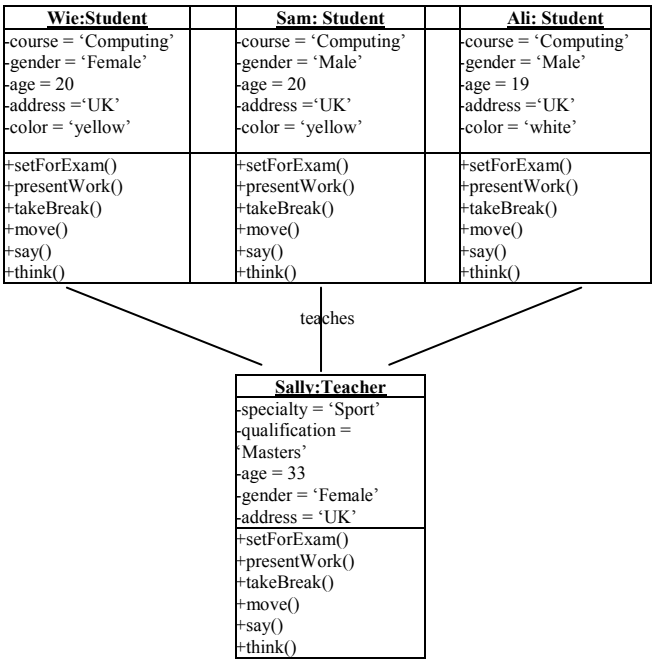

Figure 13. The object diagrams for the objects in Figure 2 after adding in the relationships between the objects.

We just need to replace the objects by their classes; identified in section 5.2. Figure 14 shows the association between classes 'Teacher' and 'Student' represented using UML notation.

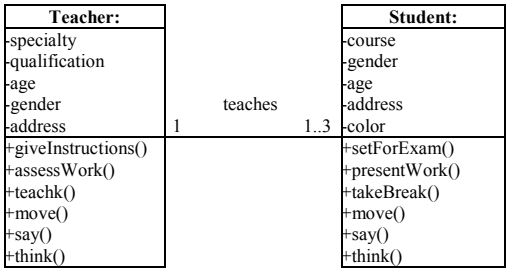

Figure 14. 'Teacher', 'Student' association.

An association end also has multiplicity, which is an indication of how many objects may participate in the given relationship. In UML, the multiplicity of a relationship among objects can be one-to-one, zero-to-many, one-to-many, or many-to-many. The numbers and symbols in UML representing multiplicity include a '0' for zero, a '1' for one and an asterisk (*) for many. In Figure 14, the association multiplicity shows that one teacher may teach one-to-3 students.

5.4. Composition Relationship

Composition is a type of class association relationship. It represents the whole-part relationship in the object-oriented paradigm. In a whole-part relationship one class is regarded as being made up of one or more other classes [6]. Composition relationship states that the part cannot exist independently of the whole. In UML, the whole-part relationship is reflected using a hollow filled in diamond and a straight line. The diamond is attached to the whole classifier, and the

arrow is attached to the part classifier. Figure 15 shows the composition relationship between a whole and its two parts.

Alice programmers are introduced to the concept of composition when adding new objects to their Alice virtual world. Before conforming the adding of new instance by clicking the button "add instance to the world", an Alice programmer is given some information about the object to be added, including of how many parts it is composed. An object of 'Student' is composed of 16 parts as shown in Figure 16, each of which is a smaller object itself.

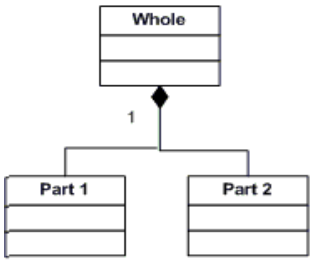

Figure 15. Composition, whole-part, relationship.

Figure 16. A Student object is composed of 16 smaller objects/parts.

The Alice object tree for an object of type 'Student', Figure 17, shows that a 'Student' object is made up of the objects 'upperBody', 'leftLeg' and 'rightLeg'. The 'leftLeg' object, which is a part of the bigger object 'Student', is made up of the objects 'lowerLeg' and 'foot'. The '+' sign next to the objects 'upperBody' and 'rightLeg' indicates that these objects are also whole objects composed of smaller part objects.

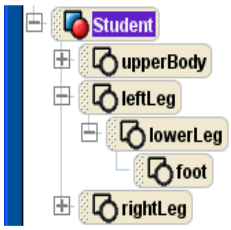

Figure 17. Alice object tree for an object of type 'Student' showing the object composition.

Using the UML composition notation introduced earlier, Figure 18 represents the equivalent UML class diagram for the Alice object tree for an object of type student seen in Figure 17.

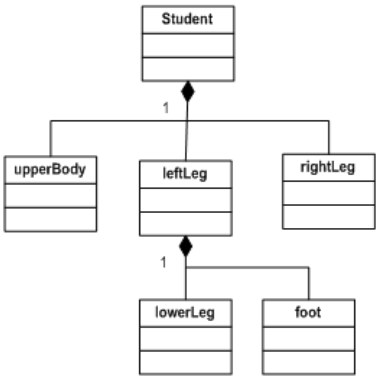

Figure 18. The equivalent class diagram for the object composition in Figure 17.

5.5. Generalisation/ Inheritance Relationship

Generalisation, in the object-oriented paradigm, is the process of extracting shared characteristics from two or more classes (called child classes or subclasses) [19], and combining them into a generalised parent or superclass. Shared characteristics can be attributes, associations or behaviours. According to Fowler [10], "generalisation at the implementation perspective is associated with inheritance in programming languages". Kimmel [13] explains the relationship between generalisation and inheritance when it comes to UML, "In UML-speak, inheritance is generalisation. This means that programmers may say inheritance when they mean generalisation, and when they say generalisation, they may mean inheritance."

Generalisation refers to an 'is-a' or 'substitutability' relationship and is reflected in a UML class diagram by a solid line connector with a hollow triangle at one end [13] as shown in Figure 19. The triangle points at the superclass, and the other end is connected to the subclass(es). Through this relationship, a subclass can make use of the attributes and operations of its superclass. In addition to the attributes and operation they inherit, subclasses can define their own attributes and operations and also redefine the meaning of the inherited operations.

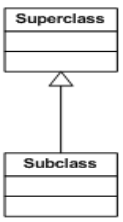

Figure 19. The generalisation/inheritance relationship.

The class diagrams for the class 'Student' as shown in Figure 11 and the class 'Teacher' as shown in Figure 12 show some common attributes and behaviours between the two classes. By performing the generalisation process on these classes; extracting shared characteristics from the two classes and combining them into a generalised class, we can create a superclass for both classes. The attributes 'gender',

'age' and 'address' are common between 'Student' and 'Teacher' classes and so are the behaviours 'move', 'say' and 'think'.

As discussed earlier, common attributes go into the attributes compartment of the new class diagram (i.e., the second compartment). Common behaviours go into the behaviours compartment in the new class diagram (i.e., the third compartment).

Now, we need to find a suitable name for our new class that fulfils the 'is-a' relationship requirements; we can describe a 'Teacher' and a 'Student' 'as-a' type of this new class. Consider class 'Person', we can say that a teacher 'is-a' person and also a student 'is-a' person. So, 'Person' is one possible name for the new superclass. As we know, the class name goes into the first compartment of the UML class diagram. Figure 20 shows the new superclass 'Person' in a generalisation relationship with the subclasses 'Teacher' and 'Student' represented using UML notation.

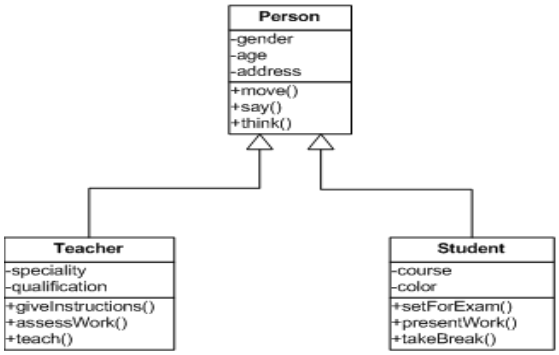

Figure 20. 'Person' is a superclass for 'Teacher' and 'Student'.

Figure 14, in section 5.3, shows another relationship that does exist between the two classes 'Teacher' and 'Student'; association relationship. Now, combining the association relationship diagram as shown in Figure 14 with the inheritance relationship diagram as shown in Figure 20 gives us the complete class diagram for the Alice world snapshot in Figure 2, as shown in Figure 21.

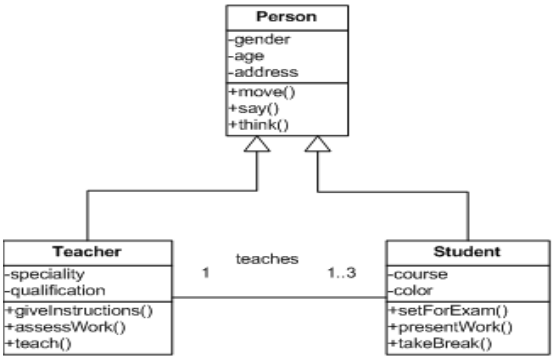

Figure 21. The complete class diagram for the snapshot in Figure 2.

In this section, we have demonstrated the proposed instructional approach, of using an animated visualisation tool (Alice) to introduce the main concepts of the object-oriented paradigm and represent them using UML notation, to extract every single detail in the complete class diagram shown on Figure 21.

6. Observations

The suggested approach was used to teach UML modelling to second year students in Foundation Degree in Computing at Wakefield College (UK) in their System Analysis and Design module. Students had no knowledge of UML or modelling except for simple flowchart modelling. Students were familiar with Alice as the tool was introduced to them during their entry-level programming module.

The students' response for using Alice, again, to introduce new computing concepts and as part of the teaching tools was very positive. The idea of using their own Alice worlds that they have built before, in learning UML modelling, was warmly welcomed. Monitoring the students' performance and the diagrams they produced, we noticed that students' understanding of object-oriented concept is deepened and their problem solving skills have also enhanced. Class relationships that students used to call 'tricky', such as inheritance and composition, don't seem to be tricky any more. We also noticed clear improvements in students' skills in understanding abstractions and abstract data types and identifying real-life objects, their attributes, their behaviours, and means of their interactions. Regarding UML modelling, students' showed good understanding of the UML static model notation. The object and class diagrams they produced are reasonably accurate. They included a well-defined attributes and behaviours for objects and classes. The notations of association, composition and inheritance are properly used. When moving to other diagrams, some students asked whether author has any suggestion of how to relate the new diagrams, particularly dynamic diagrams, to Alice so they can be easily understood and modelled. The good ones had suggested such links based on the dynamics of the scenarios in their Alice worlds. In many occasions, students expressed their appreciations for using Alice to help understanding hard programming concepts and modelling notations.

7. Conclusions

In this paper, a new approach for teach object-oriented modelling with the help of a 3D visualisation environment Alice has been discussed. The importance of this approach lies in its applicability and effectiveness in teaching object-oriented modelling for beginners, where it has been successfully tested with students and approved to be easy to learn and consequently apply. The proposed approach benefits from the visual presentation of the object and the object's details provided by the Alice environment to aid understanding objects and model them. It starts with students taking a snapshot of a scenario in an Alice world, then drawing the object

diagram for all the objects in that snapshot. Using the object diagram(s), students easily derive conceptual class diagram(s) by applying the concept of abstraction. The relationships that may exist between and among these classes were then identified. Combining all these findings together, students can produce the final class diagram for the objects in the chosen snapshot. It was found that by using this approach, students do not just learn UML modelling and notation, but they also learn object-oriented concepts and problem solving techniques.

References

[1] Abudul-Kader H., "E-learning System in Virtual Environment," *the International Arab Journal of Information Technology*, vol. 8, no. 1, pp. 23-29, 2009.

[2] Al-Tahat K., "Using Alice for Teaching Object-Oriented Programming for Novice Programmers," *in Proceedings of Electronic Education: Towards an Immersive Learning Environment*, Kuala Lumpur, Malaysia, pp. 189-195, 2011.

[3] Al-Tahat K., Idris S., Sembok T., and Yousof M., "Using Hot-Spot-Driven Approach in the Development of a Framework for Multimedia Presentation on the Web," *in Proceedings of Information Systems Analysis and Specification*, pp. 873-876, 2001.

[4] Bishop-Clark C., Courte J., and Howard E., "Programming in Pairs with Alice to Improve Confidence, Enjoyment, and Achievement," *Journal of Educational Computing Research*, vol. 34, no. 2, pp. 213-228, 2006.

[5] Burner M., Microsoft Corporation, Modelling for Software Development and Operations, available at: http://msdn.microsoft.com/en-us/library/ ms954611.aspx, last visited 2011.

[6] Burton P. and Bruhn R., "Using UML to Facilitate the Teaching of Object-Oriented Systems Analysis and Design," *Journal of Computing Sciences in Colleges*, vol. 19 no. 3, 278-290, 2004.

[7] Cooper S., Dann W., and Pausch R., "Teaching Objects-First in Introductory Computer Science," *in Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, Reno, Nevada, USA, pp. 191-195, 2003.

[8] Filho W., "Process Issues in Course Projects," *in Proceedings of the 27th International Conference on Software Engineering*, Saint Louis, USA, pp. 629-630, 2005.

[9] Flint S., Gardner H., and Boughton C., "Executable/Translatable UML in Computing Education," *in Proceedings of the 6th Australasian Conference on Computing Education*, Darlinghurst, Australia, vol. 30, pp. 69-75, 2004.

[10] Fowler M. and Scott K., *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley Longman, Boston, USA, 1997.

[11] Grady B., Rumbaugh J., and Jacobson I., *The Unified Modeling Language User Guide*, Addison-Wesley, Boston, USA, 1999.

[12] Hansen K. and Ratzer A., "Tool Support for Collaborative Teaching and Learning of Object-Oriented Modeling," *in Proceedings of the 7th Annual Conference on Innovation and Technology in Computer Science Education*, Aarhus, Denmark, pp. 146-150, 2002.

[13] Kimmel P., *UML demystified*, McGraw Hill, 2005.

[14] Lesgold A., "The Nature and Methods of Learning by Doing," *American Psychologist*, vol. 56, no. 11, pp. 964-973, 2001.

[15] Mrozek Z., Mrozek B., and Adjei O., "Teaching Object Oriented Software Engineering with UML," *in Proceedings of 13th EAEEIE Annual Conference on Innovations in Education for Electrical and Information Engineering*, New York, USA, pp. 8-108, 2002.

[16] Mullins P., Whitfield D., and Conlon M., "Using Alice 2.0 as a First Language," *Journal of Computing Sciences in Colleges*, vol. 24 no. 3, pp. 136-143, 2009.

[17] Naps T., Rößling G., Almstrum V., Dann W., Fleischer R., Hundhausen C., Korhonen A., Malmi L., McNally M., Rodger S., and Velázquez-Iturbide J., "Exploring the Role of Visualization and Engagement in Computer Science Education," *ACM SIGCSE Bulliten*, vol. 35, no. 2, pp. 131-152, 2003.

[18] Rivera-Lopez R., Rivera-Lopez E., and Rodriguez-Leon A., "Another Approach for the Teaching of the Foundations of Programming Using UML and Java," *in Proceedings of the 3rd World Scientific and Engineering Academy and Society International Conference on Computer Engineering and Application*, Wisconsin, USA, pp. 279-283, 2009.

[19] SourceMaking., available at: http://sourcemaking.com/uml, last visited 2011.

[20] Starrett C., "Teaching UML Modeling before Programming at the High School Level," *in Proceedings of the 7th IEEE International Conference on Advanced Learning Technologies*, Niigata, Japan, pp. 713-714, 2007.

[21] Tamai T., "How to Teach Software Modeling," *in Proceedings of the 27th International Conference on Software Engineering*, Saint Louis, USA, pp. 609-610, 2005.

Khalid Al-Tahat holds a BSc in computer science from Yarmouk University in Jordan, an MSc in AI from Malaysian University for Science and Technology and a PhD in Software Engineering from the National University of Malaysia. Currently, he is the coordinator of the information technology program at the Arab Open University-Jordan Branch. He worked at educational institutes in UK, Malaysia and Jordan. His research interest lies in the fields of modern software engineering, AI and teaching computing. He is an author of about 20 papers published in international journals, conference proceedings and invited book chapters.