Enhancing the WinCASE Tool

Alhadi Ali Klaib
Department of Computer Science, Faculty of Information Technology, Elmergib University, Libya
alhadi.klaib@elmergib.edu.ly

Abstract— **Computer Aided Software Engineering (CASE) tools are very important for software engineering professionals. These tools offer great support to software developers. WinCASE tool is one of these tools. it has been improved over a period of time. Moreover, dataflow Algebra is a methodology that used to describe a formal specification of a system. The WinCASE and Dataflow Algebra were both invented in the Department of Computer Science at Sheffield University. Unified Modeling Language is a standard language for developing software systems. The Class diagram is one of the UML diagrams which demonstrates the system objects. This paper aims to enhance the WinCASE framework by integrating the Class diagram in UML notation within this framework. Therefore, this paper studied the previous work and the background of the WinCASE framework, class diagram, and Dataflow Algebra. As a result, the requirements for integrating the Class Diagram within the WinCASE framework were identified. The current WinCASE structure was analysed. Furthermore, the research method for this improvement was designed and developed**.

Keywords **— CASE tools, Software Engineering, Software Tools, UML.**

I. INTRODUCTION

Computer Aided Software Engineering (CASE) tools are very beneficial as they provide reliability for software engineering. Thus, CASE tools have been developed rapidly. They also offer easy and flexible approaches for software developers to construct systems. Historically, these tools appeared in the 1980s. However, they were developed rapidly. Furthermore, these tools offer many advantages, such as saving time, effort and money, as well as offering high reliability [1-4]. WinCASE is one of the CASE tools. Initially, it was invented in the Computer Science Department at the University of Sheffield. Essentially, it was intended to be a configurable CASE tool for experimental purposes [2, 5]. It has been further developed several times over a while. First of all, it was adapted for use of Parallel Communicating Sequential Code (PCSC) methodology. Subsequently, DataFlow Algebra (DFA) methodology was incorporated within WinCASE. Lastly, Sequence Diagrams in UML notation were implemented within WinCASE[3, 6].

DFA is a methodology used to describe the formal specifications of a system. This methodology was incorporated into WinCASE. Subsequently,the specifications of DFA can be created automatically with this tool. In addition, UML is a standard language of graphical notations.It is used for building and documenting the artefacts of huge systems, particularly software systems[3, 4]. Class diagrams in UML notation demonstrate the types of system objects and different types of static relationships that exist between these objects. Furthermore, class diagrams illustrate the properties of classes, and the restrictions that apply to the approach used to link objects [6-15]. Thus, the main objective of this paper is to develop the current DataFlow Algebra (DFA) methodology within the WinCASE framework by integrating the class diagram (CDs) in UML notation within this framework [1, 3, 4, 16-18]. Therefore, a review of the WinCASE framework will be carried out in order to ensure the feasibility of extending and adapting the existing system and repository to include the class diagram. In more details, the WinCASE tool needs to be enhanced to allow the new diagram to work effectively.To summaries, the remainder of this paper is organized as follows; the second section discusses the background and previous work of this research. Subsequently, the third section illustrates the identification of the requirements of this research. Section four discusses the analysis of the current WinCASE. Section five demonstrates the proposed approach. Section six discusses the conclusion of this research.

II. BACKGROUND AND PREVIOUS WORK

This section begins with a further detailed introduction to the CASE tools. WinCASE tool developments and history. Subsequently, DFA will be covered. Finally, this section explains the class diagram in the UML notation and its properties.

A. CASE Tools

Since the early days of coding software, there has been an understanding of the need for automated tools to assist software developers. Essentially, the focus was on the programming support tools such as compilers, macro processor and translators. Over time, as computers became more powerful, and the software that ran on them became more sophisticated, various support tools started to develop considerably. The software engineering stages, such as design, implementation, delivery and maintenance, are complicated and expensive. Therefore, the development and management of these stages need to be automated. As a result, the CASE tools field was suggested, and subsequently, a large number of CASE products have been produced commercially providing a wide range of functionality [1, 9, 19-21].

B. WinCASE Tool

Initially, the WinCASE project was started in the early 1990s by Dr. Manson in the Department of Computer Science at the University of Sheffield. Essentially, the WinCASE tool was aimed to be a fully configurable CASE tool for experimental purposes. As its name may suggest, it was adapted very much towards providing a system that can be run under a particular proprietary operating system. Basically, the WinCASE system provides the function of constructing diagrams from objects and abiding rules. The methodology in the WinCASE is a set of diagrams that are visual components. Properties of methodology objects are defined as separate components. This feature gives the WinCASE a great deal of flexibility since these properties can be easily customised by a methodology engineer [3, 4]. The implementation stage of the DFA methodology within WinCASE framework was undertaken by **Wyles.** The repository of the WinCASE was expanded in order to integrate the DataFlow Algebra. Thus, an editor for data flow diagrams was also provided. Aida Manan implemented a state chart diagram within the WinCASE framework in 2005. Consequently, Di Liu studied the integration of WinCASE tool into Eclipse. Joseph Czucha implemented the development of diagram editors within the WinCASE framework [2-4, 9].

C. Dataflow Algebra

DFA is a methodology that is used to describe a formal specification of a system. The first idea of DFA was to construct a formal specification using a Data Flow Diagram (DFD). In other words, DFA is used to illustrate a system in terms of the data flows within it. The development of DFA was planned to add to the process of integration methods. These integration methods are formal and diagrammatic [2, 3, 22, 23].

D. UML Notation

UML is a standard language for specifying, visualizing, constructing, and documenting the artefacts of software systems, as well as for business modeling and other non-software systems. UML is commonly used and accepted for describing and constructing software systems. Basically, UML was developed from object-oriented graphical modeling languages which became widespread in the late 1980s and early 1990s. Since the complexity of software systems increased dramatically, visualizing and modeling these systems became an important requirement, and as a result, the UML is a broadly accepted solution for this requirement. Moreover, UML is a significant approach to developing object-oriented software. Mainly, the UML adopted graphical notations to explain the design of software projects [7, 8, 19].

E. Class Diagram in UML Notation

The class diagram is one of the UML classes and illustrates the system objects, as well as the static relationships that may exist between them. Furthermore, the class diagram illustrates the properties and operations of class objects, as well as the restrictions that apply to the connection [3, 7, 8]. Figure1 shows a detailed class diagram. The objects of the class diagram are the following:

1) Class Diagram

The boxes within the diagram represent classes. Each class is divided into three sections. The upper section is for the class name. The middle section is for the class attributes. The lower section is for the class operation [8, 19].

2) Properties

Essentially, each property is a single concept, which represents a class feature. The properties are represented in two different approaches, namely attributes and associations[7, 8].

3) Attributes

Attributes describe a property in a text form in the class box [7, 8, 19]

4) Associations

simply, the association is the other approach to notating a property. About the same information that appears in the attributes can be shown in the association. Fundamentally, the association is a firm line that connects two classes that start from the source class and go to the target one. Association can be either unidirectional or bidirectional [7, 8, 19]. Figure 2, illustrates these two kinds of associations.

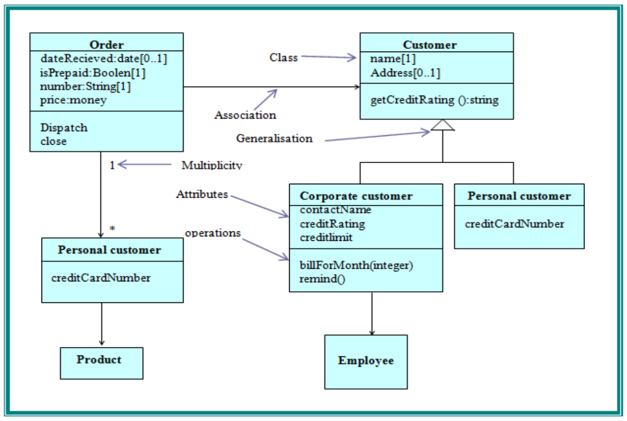


Fig. 1. A class diagram

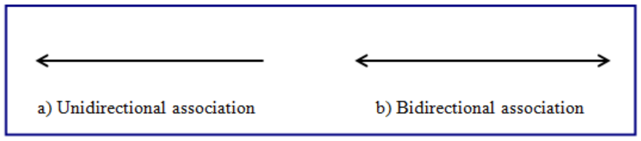


Fig. 2. Two kinds of associations

The association name appears around the middle of its association. The multiplicities appear at the beginning and the end of the association. More detail about the multiplicities is provided below. The direction of association is an arrow that fits somewhere near the association name [2, 19]. Figure 3 shows the association name.

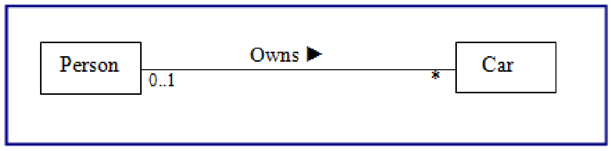


Fig. 3. Properties of an association

5) *Multiplicity*

The multiplicity of a property is the number of objects that might fill the property. There are some common multiplicities as follows:

- (1): which means (exactly one). For example, an order must have only one customer.
- (*): which means (zero or more). For instance, a customer does not have to place an order, however, this customer can order an unlimited number of orders.
- (0...1): which means zero or one. For example, a corporate customer might or might not have a single representative.
- (m..n): which means numerically specified. Thus, the number of multiplicities can be customised by the user [7, 8]. Figure 4 shows these common multiplicities.

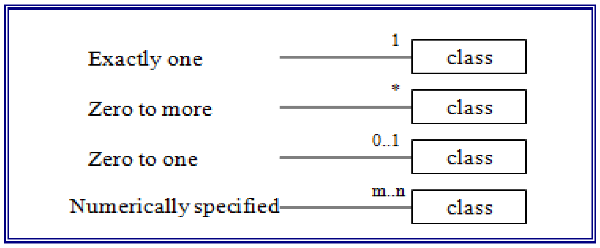


Fig. 4. Some common kinds of multiplicity

6) *Operations*

they are the actions that a class can undertake. The full UML syntax for operations is the following:

Visibility name (parameter-list) : return-type {property string}

- The **name** is a string.
- The **parameter-list** is the list of parameters for the operation.
- The **return-string** points out properties values that apply to the given operation[7].

To conclude, this section reviewed the relevant background to this paper.

III. Identification of the Requirements

This section is concerned with the identification of the requirements for integrating the class diagram in UML notation within the WinCASE framework. Moreover, the methodology and design of the class diagram are also discussed and illustrated in this section. Basically, the class box and the associations are the fundamental components of the class diagram. Their properties are also significant to make these components expressive and complete. The most important properties that need to be considered are as follows:

1) *The Class Object Properties*

The class name is the most important property and should to be given the first priority. The second priority is the attributes of the class object. Several of these attributes should be available for each class object. The lower priority is the operations.

2) *The Association Object Properties*

The association name and multiplicities are significant properties. Thus, these two properties should be given priority. The direction arrow is an important feature to show the flow of data. Finally, headarrows are also important properties. Therefore, the last two properties should be given the second priority. Consequently, by developing these components and their properties, the user should be able to draw large diagrams.

A. *Functionality*

A diagram editor within the WinCASE framework for drawing class diagram objects and also make them movable around the window need to be developed. In addition, the function of editing objects properties needs to be available. Manipulation of objects and diagrams also needs to be available, such as naming, renaming and saving diagrams. The look and feel of the class diagram editor need to keep pace with the existing ones. Thus, the class diagram editor should provide similar forms and icons to the previous editors. It should also be possible to adapt WinCASE to these features since some of them are already available in the current diagrams.

B. *Connectivity*

The class box to class box connection is the only connection that has to be carried out by associations. The connection detail also needs to be provided.

C. *Methodology Objects*

the methodology objects that need to be implemented are the following:

1) *Class Box*

The class object in the class diagram is represented by a box rectangle. This box is divided into three sections. The compulsory property is the class name, whereas the other properties are optional. Therefore, the class box can include the object name only. However, it can include the object name as well as a number of attributes and operations.

2) *Association*

The association object represents a connection between class objects. This association will be represented by a line. The properties of an association need to be developed as well. These properties are the association name, multiplicities, direction arrow of the data flow, and arrowheads. These properties also need to be implemented and each of them needs to be fit in the applicable location.

D. The Basic Class Diagram

The class object in the class diagram will be represented by a rectangle as mentioned above. This rectangle will be divided into three interior sections. The name of the class is placed into the higher section. The attributes of the class occupy the middle section, whereas the operations occupy the lower section. Figure 5 shows a simple example of a class diagram.

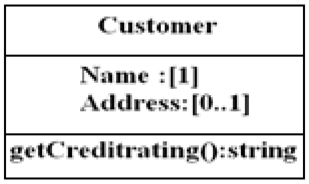


Fig. 5. A class diagram

E. Associations

The association is another class diagram object and it is an important requirement of this research. An association has several properties which are association name, multiplicities, arrow heads, and a direction arrow. A detailed explanation is given below about the appearance of these properties.

1) *The Class Object Properties*

The multiplicity is the number of objects that might fill the property. There are some common multiplicities Such as 1 (exactly one), 0...1 (zero or one), * (many) and so on and so forth. Figure 6 illustrates these common multiplicities.

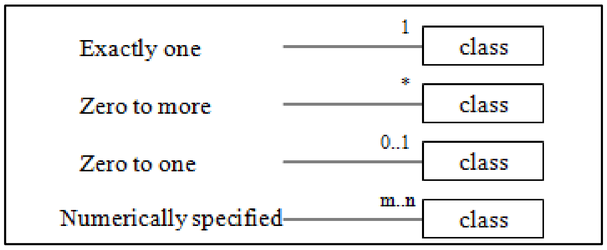


Fig. 6. Some common kinds of multiplicities

2) *The Association Name*

The association usually has a name. This name provides understanding and avoids ambiguity. The name fits around the middle of its association. Therefore, the association name will be implemented in this research. Figure 7 shows a simple example of an association.

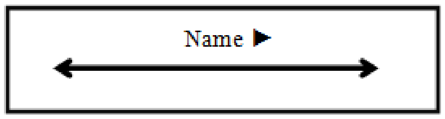


Fig. 7. An association

3) *Bidirectional Association*

This is a common type of association. A bidirectional association is a couple of head arrows that are connected together as inverses as shown in figure 7.

4) *Direction of Association*

This property shows the direction of the data flow between the classes.

F. Properties of Class Object

The properties of a class object are namely the class name, attributes and operations. Further details as follows:

1) *Class Name*

The class name fits in the higher section of the class box. Figure 8 shows an example of attributes.

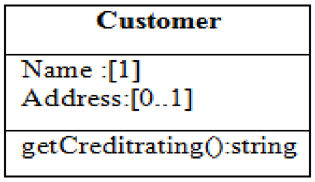


Fig. 8. An example of attributes

2) Attributes of Classes

A class attribute is a text that fits into the class box itself. There is no limited number of attributes of a class. However, they are usually a small number. Thus, it would be reasonable to set them up to five attributes for a class.

3) Operations of Classes

Operations of a class are called methods. These operations are the actions that a class performs. These operations are located in the bottom section of the class. Since there is no limited number of class operations, it would be sensible to set them up to five operations in a class.

G. Object Repository

The objects that need to be stored in the repository are shown in figure 9. The connection points are recognised by WinCASE when the user moves the cursor around.

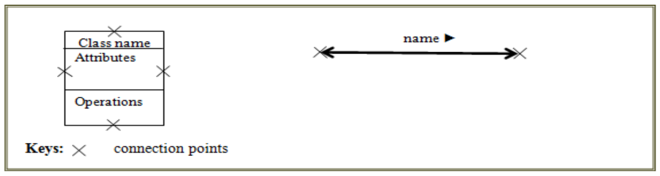


Fig. 9. Object representation of the repository

To conclude, the basic requirements of a class diagram have been identified and established. These requirements can set up the basic objects of the class diagram. The following section discusses the design of the editor and the diagrams.

IV. Analysis of current wincase Structure

Oxspring, Wyles and Denton stated that the WinCASE system was designed in a quite sophisticated way in order to allow high flexibility. Therefore, further symbols can be added to this system with minimum additions of representations. Moreover, WinCASE has a well-organised directory structure with the methodologies and associated elements in particular locations. This design provides WinCASE with the advantage that adding and using existing elements is easy to work [2-4]. Figure 10 illustrates the current WinCASE structure. More detail as follows:

A. *The Key Functions of WinCASE*

The current functions within the WinCASE tool are the following:

1) *File Manipulations*

WinCASE can load and open a project file. This file can be saved and closed once it is modified. Diagrams can also be saved in this file as well.

2) *Diagram Objects Manipulations*

There are several functions within diagrams themselves in the DFA methodology which are the following:

- Creating a new object of a diagram.
- Provision of a Diagram Name: a diagram is given a name to be saved and restored by this name.
- Renaming an Object: having given a name for a file, it can be renamed again.
- Moving an Object: it is a good advantage to move objects of diagrams around the specified window. This gives flexibility to the WinCASE framework as diagrams can be changed and expanded very easily.
- Deletion of Objects: this function allows objects to be deleted. It is also useful for making changes in diagrams.

B. *WinCASE Files Structure*

As can be seen in figure 10, the file structure has three main packages which are the following:

1) *CoreSystem Package*

As its name suggests, this is the main part of the WinCASE system. Basically, it controls the whole application and supplies its functionality. Structurally, it includes a number of classes and two packages in itself. These packages include the main classes for defining all symbols and properties of objects that are used in this application [2, 3].

2) *Interfaces Package*

Java provides the inheritance technique, which allows a great deal of generalisation among similar classes. For instance, as all methodology objects in WinCASE are similar, this lets them extend the *MethodologyObject* class. Thus, all these objects will implement the same interface which is the *IMethodologyObject* class. In other words, the WinCASE system utilises the interfaces to control methodology objects that have the same base functionality [2, 3].

3) *Methodologies Package*

This package defines the details of any implemented methodology. Currently, there is only one methodology, which is DFA. This methodology has a package called *DFA* which contains two sub-packages, namely *Diagrams* and *MethodologyObjects.* There are also a number of defining classes for each diagram in the *DFA* package. The *Diagrams* package has two sub packages, *DFA* and *SD* packages. Every one of them holds a class that defines its diagram detail. The *MethodologyObjects* package contains a number of sub-packages for all objects that are defined by these diagrams [2, 3].

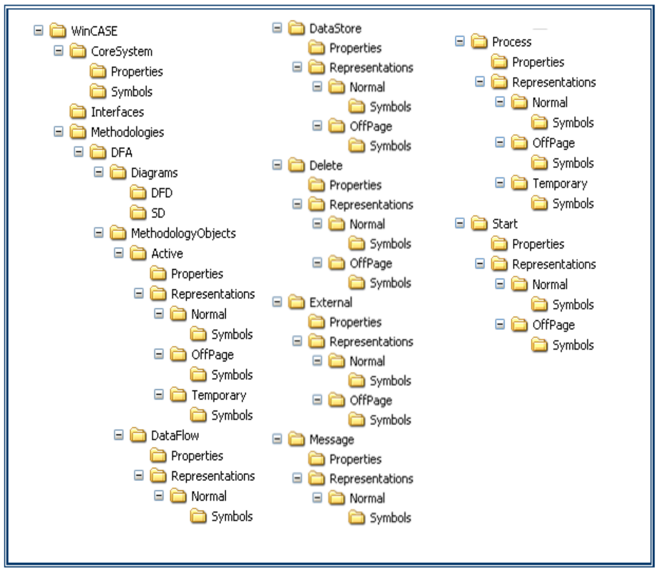
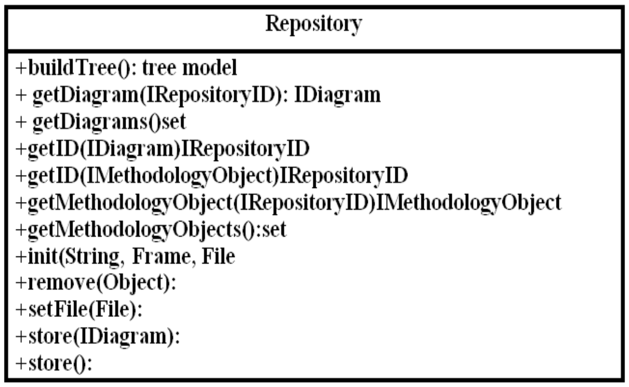


Fig. 10. The WinCASE files structure

C. *The Repository of WinCASE*

This section provides an explanation of how the repository of WinCASE stores information as this function is vital within the WinCASE system. Essentially, there is a unique identifier attached to every object in all implementations of a repository. Therefore, the repository can guarantee that only one copy of each object is saved. Other objects can store a reference to this object, from which the original object can be found. This is the main idea of dealing with objects such as saving and retrieving them. It is also the way of distinguishing objects from each other. The repository provides the *IRepositoryID* interface which is used to allow those references to existing. This interface class can be considered as a tag attached to any saved object and the content of this object depends on the implementation. This allows information to be saved. Therefore, the *IRepositoryID* class interface is used by the repository implementation to fill in the details. Notice that this interface does not specify any methods as none of them will be needed in some cases [3]. Since every representation must be linked to a methodology object, this representation will have to find out that object's *id* tag. The object has to be determined from the id when a user edits the properties of an object. This can be carried out by the "*getMethodologyObject*" method. There are other methods that enable the tagging and resolving of the diagrams. The Remove method removes any object or diagram that is not needed any more in the repository. Similarly, the Store method is responsible for ensuring that the stored version of all data which currently loaded is up-to-date. The methods *getDiagram* and *getMethodologyObject* are both responsible for retrieving a list of all diagrams and objects in the repository [3]. The class diagram with all these methods is shown in figure 11.



```
Repository
+buildTree(): tree model
+ getDiagram(IRepositoryID): IDiagram
+ getDiagrams()set
+getID(IDiagram)IRepositoryID
+getID(IMethodologyObject)IRepositoryID
+getMethodologyObject(IRepositoryID)IMethodologyObject
+getMethodologyObjects():set
+init(String, Frame, File
+remove(Object):
+setFile(File):
+store(IDiagram):
+store():
```

Fig. 11. Repository Design

D. *Diagrams*

There are two types of diagrams that have been implemented in the DFA methodology within WinCASE, which are the Dataflow Diagram and the Sequence Diagram. These two kinds of diagrams are defined by the *DFAMethodology* class. This class is responsible for defining all kinds of DFA diagrams. Notice that diagrams are defined in a project file just at the creation time of this file. That is to say, it is not possible to draw or add any kind of diagram in a project if this diagram was not defined in this project. Essentially, each diagram has its own functionality which is defined by a base diagram class, which is known as *Diagram* and exists in the *CoreSystem* package. Every methodology diagram extends this class to allow for its own functionality and properties. This basic diagram class supplies all the fundamental elements of a diagram, such as representation and object connection data structure, the function to add objects to diagrams, a paint function, and finally the basics of popup menus. The methodology diagram classes, which are *DFADiagram* and *SDDiagram*, extend the *Diagram* class to give more specific functions and details. They also establish their objects and keep them ready to use in the diagram, as well as establishing connection information. As far as consistency is concerned, this connection information is very significant since it plays the role of a consistency checker. Thus, any undefined connection is not allowed. In fact, there is no particular code for consistency checks within WinCASE. Consistency checks completely depend on the code of the *Diagram* class. The main consistency check is the allowed connections into the *Diagram* class which can be created [3].

E. *Methodology Objects*

There are a number of methodology objects in the current WinCASE. Most of them belong to the DFA diagram since they were defined in the *DFDDiagram* class. Another methodology object belongs to the SD diagram as it was defined in the *SDDiagram* class. The rest of these objects were not defined at all [2]. The methodology objects that were defined in the DFD diagram are the following:

1) *Process Object*

Which is represented by the following :

DFA.MethodologyObjects.Process

2) *External object*

Which is represented by the following:

DFA.MethodologyObjects.External

3) *Datastore object*

Which is represented by the following:

DFA.MethodologyObjects.Datastore

4) *Dataflow object*

Which is represented by the following:

DFA.MethodologyObjects.Dataflow

Each methodology has two properties, which are the object name and annotation. The only methodology object that is defined in the SD diagram is the ***Start object***: this is represented by the following:

DFA.MethodologyObjects.Start

This explanation is related to the structure of diagram packages. Basically, each methodology object extends the *MethodologyObject* class which exists in the *CoreSystem* package, and also implements its *interface* which is *IMethodologyObject* in the *Interfaces* package. The elements of an object are saved in the package of this object itself. This package is located on the following site: *DFA.MethodologyObjects*. Figure 10 shows the whole structure of the WinCASE framework. Within each object package there are two sub-packages and three other classes.

First of all, to start with the classes, they principally define their object. Consider an arbitrary object called Entity. Thus, the first class would be called *EntityMethodologyObject*. This class is the main one, and it defines the object properties and any other characteristics it has. The other classes would be the *EntityCustomiser* and *EntityMethodologyObjectBeanInfo*. They allow the object to be changed. The *EntityCustomiser* class sets and gets their object properties.

Secondly, as far as the *Properties* and *Representations* packages as concerned, they simply contain additional detail about their object. The *Properties* package includes classes that set and get properties values into the repository, These values can be changed by the user during the diagram creation and editing. The *Representations* package is more sophisticated since it has sub packages within itself. It contains classes for defining each possible representation.

In general, object properties and representation symbols all extend base classes that are defined in the following packages:

- *CoreSystem.Properties* and
- *CoreSystem.Symbols*

F. Interfaces

The *IMethodology* class in the *CoreSystem* package is a vital class, since it takes the responsibility for managing the functionality of the *IMethodologyobject* and *IDiagram* classes in this methodology. Essentially, *IMethodologyObject* holds a number of properties and representations and manages what is allowed. Some other classes such as *IDiagram*, *IRepresentation* play similar roles to the *IMethodologyObject*. To summarise, all these classes control the main mechanism of the WinCASE tool[3].

The WinCASE data model uses a number of linked interfaces, and as interfaces do not hold attributes, so that provides this tool with flexibility as mentioned above. Figure 12 illustrates the data model within WinCASE. It is obvious that the *IMethodology* class is the main class as it controls the types of diagrams and adds any new class. This class also controls the drawing of a new diagram, as well as naming the diagram. The class *IDiagram* is also a significant class, since it manages the representations and connections within a diagram. This class also manages the permitted connections. The *IRepresentation* class runs a number of important functions. It manages the representations for methodology objects and also representations of connections among these objects. The *IMethodologyObject* manages the functions associated with an object, such as editing properties and sets up the representation of that object. The *IProperty* class performs similar functions as it manages the value of properties and other functions. Lastly, the class *ISymbol* is responsible for managing the features of an object, such as the arrows and names. The rest of the WinCASE structure is constructed around this data model. In the main window where diagrams are created, they could be loaded within child windows, which hold a palette of representations that might be created in this diagram[3].

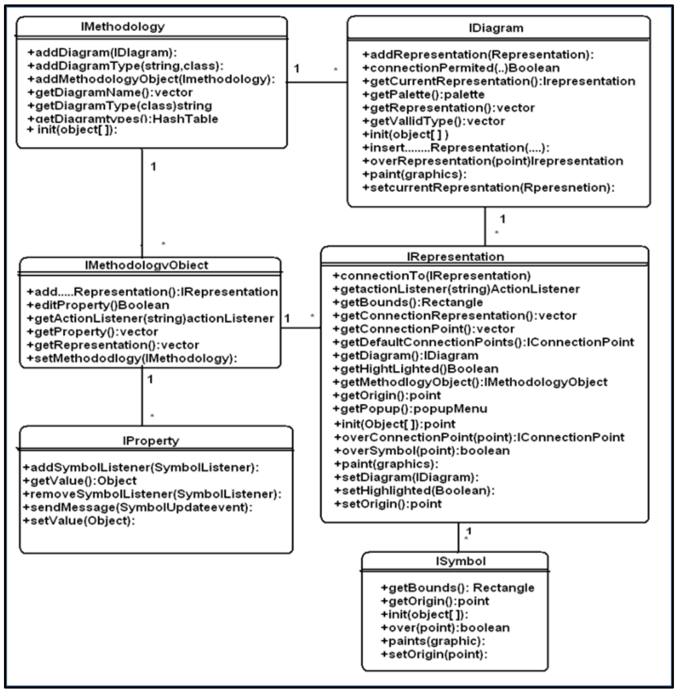

IMethodology
+addDiagram(IDIagram):
+addDiagramType(string,class):
+addMethodologyObject(Imethodology):
+getDiagramName():vector
+getDiagramType(class)string
+getDiagramtypes():HashTable
+ init(object[ ]):
IDiagram
+addRepresentation(Representation):
+connectionPermited(..)Boolean
+getCurrentRepresentation():Irepresentation
+getPalette():palette
+getRepresentation():vector
+getVallidType():vector
+init(object[ ])
+insert........Representation(....):
+overRepresentation(point)Irepresentation
+paint(graphics):
+setcurrentRepresntation(Rperesnetion):
IMethodologyObject
+add.....Representation():IRepresentation
+editProperty()Boolean
+getActionListener(string)actionListener
+getProperty():vector
+getRepresentation():vector
+setMethododlogy(IMethodology):
IRepresentation
+connectionTo(IRepresentation)
+getactionListener(string)ActionListener
+getBounds():Rectangle
+getConnectionRepresentation():vector
+getConnectionPoint():vector
+getDefaultConnectionPoints():IConnectionPoint
+getDiagram():IDiagram
+getHightLighted()Boolean
+getMethodlogyObject():IMethodologyObject
+getOrigin():point
+getPopup():popupMenu
+init(Object[ ]):point
+overConnectionPoint(point):IConnectionPoint
+overSymbol(point):boolean
+paint(graphics):
+setDiagram(IDiagram):
+setHighlighted(Boolean):
+setOrigin():point
IProperty
+addSymbolListener(SymbolListener):
+getValue():Object
+removeSymbolListener(SymbolListener):
+sendMessage(SymbolUpdateevent):
+setValue(Object):
ISymbol
+getBounds(): Rectangle
+getOrigin():point
+init(object[ ]):
+over(point):boolean
+paints(graphic):
+setOrigin(point):
1
*

Fig. 12. The Data Model within WinCASE

V. Proposed Approach

This section explains the analysis of needed developments of the current WinCASE tool structure. Subsequently, the second part of this section discusses the implementation of the class diagram within the WinCASE tool.

A. Analysis of Required Developments

In General, the similarity of the basic elements between this research and previous projects gives great advantages to this research. The key advantage is consistency, as mentioned before that WinCASE has been designed in a sophisticated and consistent way. Thus, the current WinCASE structure needs to be studied carefully in order to make this project keep pace with the same consistency. The other advantage of consistency is saving time and effort for the development of this research. An understanding of this code and the WinCASE structure is important in order to build an effective project structure. Figure 13 illustrates the main components of a class diagram that need to be implemented. Essentially, the class diagram is divided into two basic components, which are a class box and an association. The class box includes three sub components, which are the class name, attributes and operations. The association contains four components within itself, namely an association name, a multiplicity, arrow heads and a direction arrow.

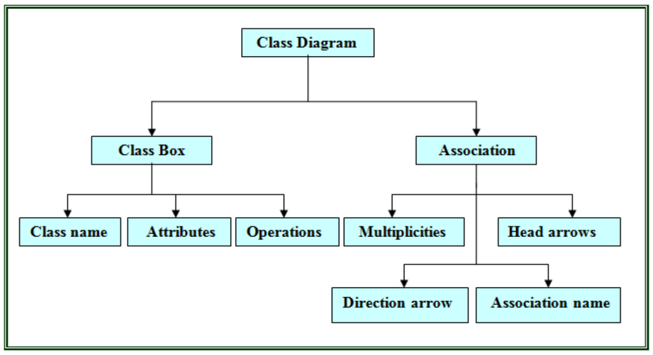


Fig. 13. The components of the class diagram structure

B. System Design

The class diagram system within WinCASE has to be designed in a similar way to the previous diagrams in order to keep pace and integrate smoothly with the current WinCASE framework. Figure 14 represents the integration of the class diagram with the WinCASE framework. Notice that creating and opening a project file are already existing functions. The project file contains all the kinds of diagrams that are defined in this methodology. The main functions that will be implemented on the class diagram are renaming, saving and closing the diagram, which are the same as the functions in the previous diagrams. Once a class diagram is either created or opened, this diagram is ready to be drawn in the specified window. The main functions for a class diagram in this window are as follows:

- Creating an Object.
- Object Modification: it means the properties of this object can be modified.
- Object Navigation: this means the object can be moved around the specified window.
- Object Deletion.

Each of these functions would be available on both class box and association objects. Figure 15 shows the main functions that need to be implemented for class diagram objects.

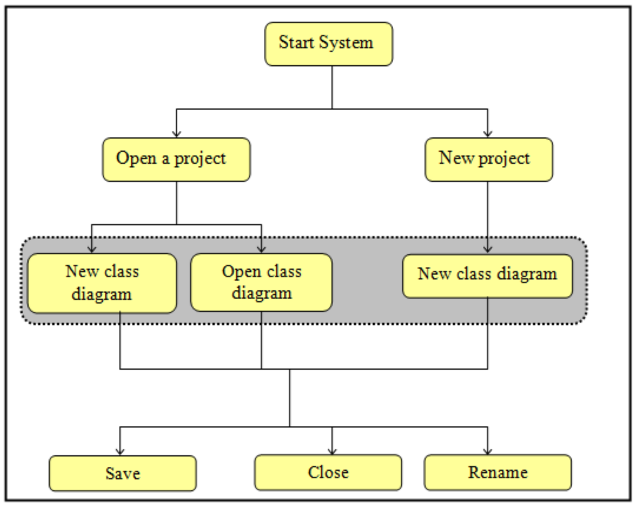


Fig. 14. The integration of the class diagram system within the WinCASE framework

C. File Structure Design

As far as the current WinCASE files structure is concerned, a package called *CD* needs to be created in the *Methodology.DFA.Diagrams* package. The *CD* package represents the class diagram and it contains a class called *CLSDiagram*. The function of this class is to initialise the class diagram by invoking other classes and methods. This class is the core one as it sets all the objects and connections of a class diagram. Conveniently, the packages of methodology objects need to be setup in the *MethodologyObjects* package. They would be named *Entity* and *Association* packages. Simply, the *Entity* package represents the class box objects, and the *Association* package represents the association objects. Each of these packages has two basic sub-packages, which are *Properties* and *Representations*. Regarding the *Properties* package; as its name suggests, it includes the main properties classes for its own methodology object. The *Representation* package contains sub-package(s) as well. With respect to the *Association* package, there is only one *Representations* sub-package called *Normal*. At the same time, in the *Entity* package there are two *Representations* sub-packages which are *Normal* and *Temporary*. Each of them has a number of classes and a package named *Symbols*. These classes represent their objects. The *Symbols* package also has classes that define their object symbols, such as the association name, the arrow heads and the direction arrow for association objects.

The structure would appear quite complicated; however it is slightly similar to the current structure techniques. Figure 16 illustrates the prospective structure for these packages along with the current packages. Notice that the package of the class box object is named *Entity* in order to avoid any confusion, as "class" is a reserved word in java.

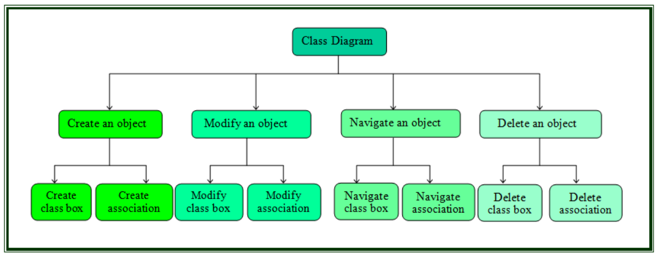


Fig. 15. The main functions are implemented for class diagram objects

D. Methodology Objects

In this research, there are two methodology objects were added to the previous ones. These objects are *Association* and *Entity*. In fact, these objects are slightly similar to some of the existing objects as follows:

- The *Association* is similar to the Dataflow object in the DFA methodology in the basic concepts since they both play the role of connecting objects to each other and carrying information between them. However, a considerable amount of work was carried out to implement the properties of this object.

- The Class object bases are similar to the *Process* object bases. Both objects have a basic rectangle shape, irrespective of the dimensions and internal partitions. They also both have an object name. Nevertheless, quite sophisticated work was performed. This work involves dividing the rectangle into three parts and adding a number of attributes and operations to this rectangle. Subsequently, fitting each of these parts in the specified section also was carried out. Figure 16 illustrates the file structure of the Class Diagram within the WinCASE file structure.

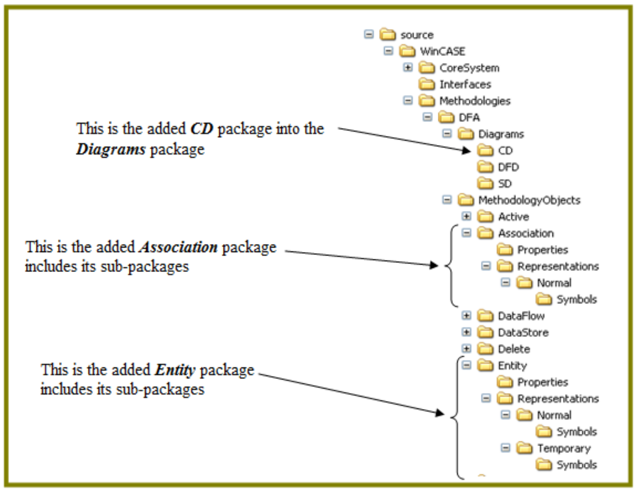


Fig. 16. The file structure of the class diagram within the WinCASE file structure

The following detail explains the implemented methodology objects of the CD in the WinCASE file structure:

1) Entity Objects

The Entity object represents the class box. This class requires name, attributes, and operations. The design of class properties is as follows:

- *Class name*: it is a string of a text which represents the name of its class.
- *Attributes*: there will be up to five attributes in a class object, and they are text strings.
- *Operations*: there also will be up to five attributes of text string within a class.

2) Association Objects

It is simply represented by a line. The properties of associations have to fit into the specified locations. Basically, the head arrows fit at the edges of this line. The association name is located around the middle of its association. The arrow direction fits near the name. All these are known as the properties of an association.

To summarise, this section discusses the analysis of the current WinCASE structure and the functionality of the WinCASE system. The Methodology package was studied in order to define the implementation of CD within it. Subsequently, the repository of the WinCASE also investigated and how the information is stored and retrieved, so it can be updated for the new integration of CD. The diagrams of the current WinCASE were studied as well. The interfaces of the WinCASE were considered in order to explore the technology of the WinCASE. As a result, the design of CD and all its elements were carried out.

VI. CONCLUSION

The aim of this paper is to investigate and develop the WinCASE framework by integrating the class diagram in UML notation within this framework. Therefore, the background and previous work of this research were reviewed and discussed deeply. In more detail, the WinCASE tool and DFA were reviewed. The UML Class Diagram was also discussed. It's found out that the integration of CD within the WinCASE is possible by making use of the current functionality and methodology and then develop and integrate the new Class diagram. Subsequently, the methodology of CD was setup. The file structure of the WinCASE was enhanced by adding the CD file structure. Furthermore, the class diagram components were implemented and integrated into the WinCASE tool. Thus, large class diagrams can be drawn by this editor. To conclude, the contribution of this paper as follows; the requirements and the methodology for the development of WinCASE tool and integrating the class diagram were identified and setup. The needed objects were also identified. Subsequently, the design of the new CD methodology and interfaces were carried out. Consequently, the Class Diagram was implemented and tested for evaluation purpose. As a result, Currently, the WinCASE tool was provided with an editor for drawing a class diagram within the DFA methodology.

REFERENCES

[1] Fuggetta, A., A classification of CASE technology. Computer, 1993. 26(12): p. 25-38.
[2] Denton, M., Implementing Sequence Diagrams within the WinCASE Framework. BSc, University of Sheffield, 2003.
[3] Klaib, A., Data Models and the Dataflow Algebra within WinCASE, in MSc Dissertation 2004/2005, Sheffield University: Sheffield University.
[4] Oxspring, R. and G. Manson, Implementing a PCSC Tool within the WinCASE Framework. 3rd Year Dissertation, Department of Computer Science, University of Sheffield, 2000.
[5] Cowling, T., Extending the Eclipse Version of WinCASE. UNIVERSITY OF SHEFFIELD.
[6] Cowling, A., Basic System and Subsystem Structures in the Dataflow Algebra. 2008, Department of Computer Science Research Report CS-08-12, University of Sheffield.
[7] Fowler, M., UML distilled: a brief guide to the standard object modeling language. 2004: Addison-Wesley Professional.

[8] Podeswa, H., UML for the IT Business Analyst. 2009: Course Technology Press.

[9] Sommerville, I., Software engineering 9th Edition. ISBN-10, 2011. 137035152: p. 18.

[10] Cowling, A., Properties of The Synchronous Merge Operation in the Dataflow Algebra. 2009, Department of Computer Science Research Report CS-09-07, University of Sheffield.

[11] Cowling, A., A Revised Denotational Semantics for the Dataflow Algebra. 2006, Department of Computer Science Research Report CS-06-11, University of Sheffield.

[12] Cowling, A., A simplified abstract syntax for the dataflow algebra. 2002, Department of Computer Science Research Report CS-02-09,

University of Sheffield.

Cowling, A., Normal Forms in the Dataflow Algebra. 2007, Department of Computer Science Research Report CS-07-11, University of Sheffield.

[14] Cowling, A. and M. Nike, Dataflow Algebra Specifications of Pipeline Structures. Sheffield University CS-97-17, 1997.

[15] Cowling, A.J., Dataflow algebras as formal specifications of data flows. 1995: University of Sheffield, Department of Computer Science.

[16] Cowling, A., Equality and Inequality in the Dataflow Algebra. 2008, Department of Computer Science Research Report CS-08-03, University of Sheffield.

[17] Cowling, A., Fundamental Compositionality Properties of Systems in the Dataflow Algebra. 2010, Department of Computer Science Research Report CS-10-03, University of Sheffield.

[18] Cowling, A.J. and M. Nike, Using dataflow algebra to analyse the alternating bit protocol, in Software Engineering for Parallel and Distributed Systems. 1996, Springer. p. 195-207.

[19] Starr, L. and S.J. Foreword By-Mellor, Executable UML: how to build class models. 2001: Prentice Hall PTR.

[20] Klaib, A. and L. Joan, Investigation into indexing XML data techniques. 2014.

[21] Thomson, C.D., Linking Dataflow Algebra with the CaDiZ Tool. The dissertation can be referred to as 3rd Year Dissertation, Department of Computer Science, University of Sheffield, 2001.

[22] Cowling, A., An Operational Semantics for the Dataflow Algebra. 2004, Department of Computer Science Research Report CS-04-16, University of Sheffield.

[23] Cowling, A., Operations for Composing Subsystems in the Dataflow Algebra. 2008, Department of Computer Science Research Report CS-08-13, University of Sheffield.