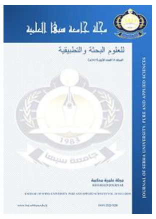
SEBHA UNIVERSITY JOURNAL OF PURE & APPLIED SCIENCES VOL.21 NO. 1 2022
DOI: 10.51984/JOPAS.V21I1.1413

مجلة جامعة سبها للعلوم البحتة والتطبيقية
Sebha University Journal of Pure & Applied Sciences

Journal homepage: www.sebhau.edu.ly/journal/index.php/jopas



Developing an Editor for Drawing Class Diagram within WinCASE Framework

*Alhadi Klaib[a], Ragab Ihnissi[b], Abdelrahman Arbi[c]

[a]Department of Software Engineering, Faculty of Information Technology, Elmergib University, Libya
[b]Department of Computer Engineering, Faculty of Engineering-Regdalayn, Sabratha University, Libya
[c]Department of Electrical and Computer Engineering, Faculty of Engineering, Elmergib University, Libya

Keywords:
Software Engineering
CASE tools
UML
Class Diagram
WinCASE

ABSTRACT

Computer Aided Software Engineering (CASE) tools are significant for the software engineering field. They provide great support to software developers. WinCASE tool is one of these tools. Furthermore, dataflow Algebra is a methodology that used to describe a formal specification of a system. Unified Modeling Language is a general purpose modlling language in the field of software engineering. Class diagram is one of the UML diagrams which illustrates the system objects. This paper aims to upgrade the WinCASE tool by implementing the Class diagram within the framework of this tool. Therefore, this paper studied the previous work and the background of the WinCASE. Subsequently, the class diagram was implemented into WinCASE framework; and then tested, and evaluated successfuly as well. Consequently, the WinCASE has been provided with an editor for drawing the class diagram.

تطوير محرر لرسم مخطط الكلاس وإدماجه في أداة WinCASE

*الهادي علي كليب[1] و رجب حنيش[2] و عبدالرحمن العربي[3]

[1]قسم هندسة البرمجيات، كلية تقنية المعلومات– جامعة المرقب، ليبيا
[2]قسم هندسة الحاسوب، كلية الهندسة - رقدالين، جامعة صبراته، ليبيا
[3] قسم الكهربائية و هندسة الحاسوب، كلية الهندسة – جامعة المرقب، ليبيا

الكلمات المفتاحية:
هندسة البرمجيات
أدوات كيسCASE
لغة النمذجة الموحدة
مخطط الكلاس
. أداة WinCASE

الملخص

تعتبر الأدوات المساعدة فيه مجال هندسة البرمجيات (CASE) مهمة في مجال هندسة البرمجيات ،حيث أنها توفر دعما كبيرا لمطوري البرمجيات. الأداة WinCASE هي إحدى هذه الأدوات. بالإضافة لذلك فإن جبر تدفق البيانات (DFA) هو نظام منهجية تستخدم لوصف نظم الحاسوب. لغة النمذجة الموحدة (UML) هي عبارة عن لغة نمذجة للأغراض العامة ووصف نماذج في مجال هندسة البرمجيات. مخطط الكلاس (Class Diagram) هو أحد مخططات لغة النمذجة الموحدة التي توضح كائنات النظام. تهدف هذه الورقة إلى تطوير أداة WinCASE من خلال تنفيذ مخطط الكلاس وإدماجه في هذه الأداة. لذلك ، تم تنفيذ مخطط الكلاس وإدماجه في هذه الأداة ؛ ثم تم اختباره وتقييمه بنجاح أيضًا. وبالتالي ، تم تزويد WinCASE بمحرر لرسم مخطط الفصل الدراسي.

Introduction:

Computer Aided Software Engineering (CASE) Tools are considerably useful since they provide support for software engineering. Thus, CASE tools have been developed quickly. They also offer easy and flexible approaches for software developers to build systems. Historically, these tools appeared in the 1980s. However, they were developed rapidly, particularly in the first decade. To summarize, these tools provide 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 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].
Essentially, DFA is a methodology used to describe a formal specification of a system. This methodology was incorporated into WinCASE. As a result, the specifications of DFA can be created

*Corresponding author:
E-mail addresses: alhadi.klaib@elmergib.edu.ly ,(R. Ihnissi) ragab000_abd@yahoo.com ,(A. Arbi) asarbi@elmergib.edu.ly

Article History : Received 11 November 2021 - Received in revised form 25 December 2021 - Accepted 01 January 2022

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]. A class diagrams in UML notation illustrate 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]. Therefore, the main objective of this paper is to investigate the possibility of developing the current DataFlow Algebra (DFA) methodology within WinCASE framework by integrating the class diagram (CDs) in UML notation within this framework [1, 3, 4, 7, 16-24]. Thus, a review for WinCASE framework needs to be carried out in order to ensure the feasibility of extending and adapting the existing system and repository to include the class diagram. In other words, WinCASE tool needs to be enhanced to allow the new diagram to work effectively.

To summarise, the remainder of this paper is organised as follows; the the second section describes the mplementation of class diagram within the WinCASE. Section three demonstrates the testing and evaluation. Section four discusses the conclusion of this research.

Comparison with Related Work

The WinCASE was invented in the 1990s by Dr. Manson in the Department of Computer Science at the University of Sheffield. The WinCASE tool was intended 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 WinCASE a great deal of flexibility since these properties can be easily customised by a methodology engineer [3, 4]. Wyles implemented the DFA methodology within WinCASE framework. 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. Di Liu studied the integration of WinCASE tool into Eclipse. Joseph Czucha implemented the development of diagram editors within the WinCASE framework.

Implementation

The basic development process was iterative, and it was found out that this is a suitable approach to implementing the class diagram with the smallest possible work stages. The approach was essentially applied to the object packages. The advantage of this approach is that it allows every small step in the work to be tested. It also minimises the potential errors in the project and makes error detection an easy task, and as a result the project goes smoothly.

1 Initialization of Class Diagram

First of all, the class diagram has been defined for the DFA methodology by adding a fragment of code into the *DFAMethodology* class. Consequently, any project file that is created will contain automatically this diagram type. The code fragment for defining the class diagram within the DFA is illustrated below.

```
// here where new kind of diagram can be defined
Class cdClass = Class.forName(suffix + "CD.CLSDiagram");
addDiagramType("UML class", cdClass);
```

As can be seen, this code indicates the location of the class that defines this type of diagram. This class is the one called *CLSDiagram*. It also provides the system with the type name of the diagram. The next step passes this information to the *Methodology* class, which creates this diagram and adds it to the repository. Thus, the system will confirm this diagram and update itself where needed. Notice that this fragment of code is vital to add a new diagram to the DFA methodology as it sets up the basics of this diagram. In order to make the class diagram available in the WinCASE system, a new project file is created. Consequently, the Diagram menu is updated automatically, so the *New UML class* item appeared in the *Diagrams* menu. Figure 1 shows this menu.

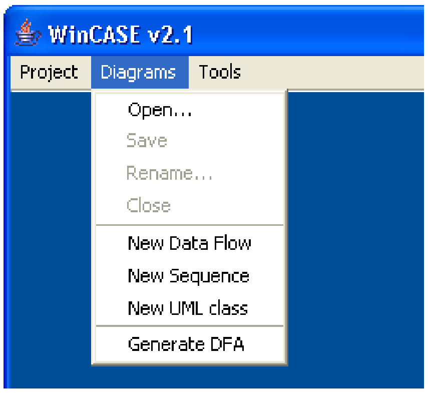


Fig. 1: The diagrams menu including the Class Diagram

The *UML Class* item also appears automatically in the *Choose Diagram* window, which pops up by the *Open* item in the *Diagrams* menu to open an existing diagram. Figure 2 shows this window. To summarise, by just defining the new type of diagram into *DFAMethodology* class, the WinCASE system is updated automatically where needed for every new project file.

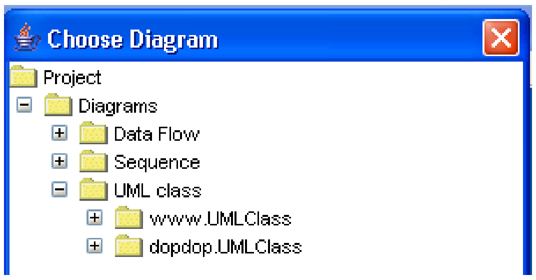


Fig. 2: The choose diagram window includes the UML class

2 Open Project file

Having chosen a *New UML class* from the *Diagrams* menu, a small window appears to provide a name for this diagram. Figure 3 shows this window. In fact, this window is a generic one and it is invoked by the method *NewDiagramAction* in the *MainFrame* class, so it works with all kinds of diagrams within the WinCASE system.

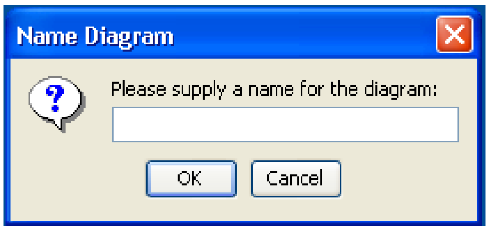


Fig. 3: Entering a new diagram name

As soon as a new diagram is created or an existing one opened, the sub window for drawing a class diagram pops up. It has the diagram name and its extension to show the diagram type, as can be seen in figure 4. This window also includes icons that invoke the creation of class and association objects, as well as the navigation function.

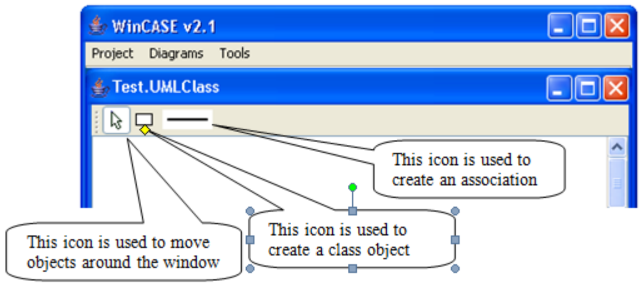

WinCASE v2.1
Project Diagrams Tools
Test.UMLClass
This icon is used to create an association
This icon is used to move objects around the window
This icon is used to create a class object

Fig. 4: Class Diagram editor

3 CLSDiagram Class:

The *CLSDiagram* class is the main one and plays a vital role in this research. This class was located in the *CD* package in order to match the WinCASE structure. Essentially, this class was designed in a similar way to the *DFDDiagram* class. This gives significant advantages to this project and reduces the potential errors and also provides consistency of the diagrams in the DFA methodology and the repository.

In practice, the *CLSDiagram* class sets up the methodology objects of the class diagram, namely the Object and the Association. This class also sets up the connectivity among objects. Notice that the only connection that needs to be set up in this project is Object *to* Object. The *getNameExtension* method in this class returns the extension of this type of diagram, which in this case is *UMLClass*. Therefore, the repository can distinguish between the types of diagrams. Figure 5 shows the *CLSDiagram* class.

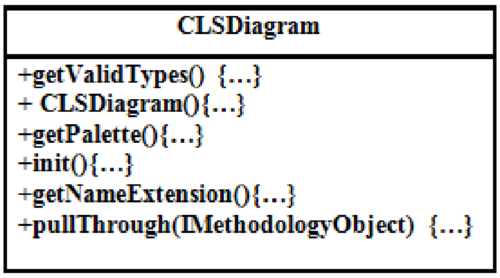


Fig. 5: CLSDiagram class

4 Class Object

In terms of the WinCASE structure, the class object is represented by the *Entity* package. This package is the main storage for all classes related with this object.

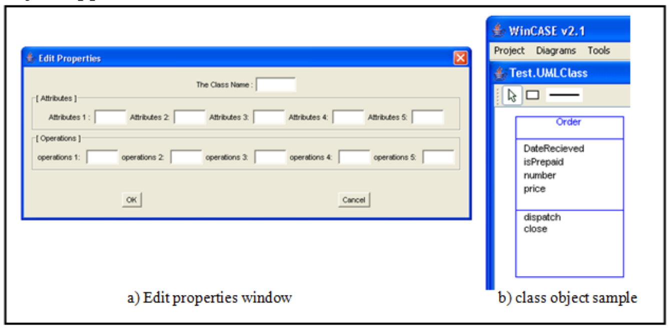
The main classes in this package are the *EntityMethodologyObject*, *EntityCustomiser* and *EntityMethodologyObjectBeanInfo*. These classes define and customise the class object. To start with the *EntityCustomiser* class is an important class since it sets all the properties of this object, which are the class name, five attributes and five operations.

The class *EntityMethodologyObjectBeanInfo* works along with the *EntityCustomiser* to customise this object. The *EntityMethodologyObject* is the main class as it defines all the properties and representations of this object by the constructor method *EntityMethodologyObject()*.

The *NormalRepresentation* class is also an important class as it is responsible for representing all the properties and symbols of the class object, such as the class box and the attributes. It also sets up the connection points for connecting this object with associations. There are four points which have been set around the object. This class exists in the *Representations.Normal* package.

With regard to the locations, the class properties occupy the *Properties* package within the *Entity* package. The symbols of an object occupy the *Symbols* package in the *Representation.Normal* package.

Essentially, the class symbols set the appearance and characteristics of all components of this object. For instance, the *DefaultSymbol* class sets the dimensions and other characteristics of a class box such as the its colour. The other symbol classes play similar roles such as setting the locations and the characteristics for the class name, attributes, and operations. Having put these classes and functions into practice, and opened or created a class diagram, the specified window appears in the main window. Consequently, a class object can be drawn by clicking the class icon and then clicking on the required location in this window. Therefore, the Edit Properties window appears in the screen and gets ready to enter the object properties. Figure 6 -a shows this window. Notice that all the properties of class object can be entered through this window. Having entered these properties and clicked the *Ok* button, the class box appears in the specified location. Figure 6- b shows an example of an object appearance.



Fig, 6: Shows The Edit properties window, and a class object sample

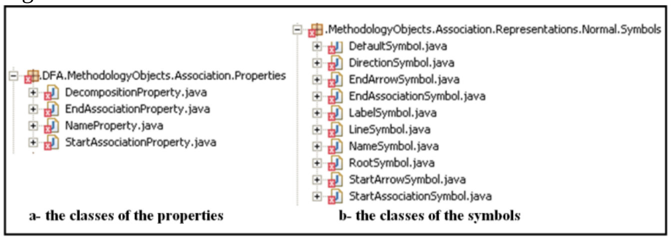
5 Association object

A package called *Association* is created to hold all the details about this object. Essentially, this package includes the *Properties* and *Representations* packages, as well as the *AssociationCustomiser*, *AssociationMethodologyObject* and *AssociationMethodologyObjectBeanInfo* classes. These classes are similar to those existing in the *Entity* package that has been described above.

The *Properties* package has the following classes:

- *NameProperty* class: this class defines the name of the association object.
- *StartMultiplicityProperty*: this class defines the first part of the multiplicity.
- *EndMultiplicityProperty*: this class defines the last part of the multiplicity.
- *DecompositionProperty*: this class defines the decomposition of a class object.

Figure 7-a illustrates the structure of these classes.



DFA.MethodologyObjects.Association.Properties
DecompositionProperty.java
EndAssociationProperty.java
NameProperty.java
StartAssociationProperty.java

.MethodologyObjects.Association.Representations.Normal.Symbols
DefaultSymbol.java
DirectionSymbol.java
EndArrowSymbol.java
EndAssociationSymbol.java
LabelSymbol.java
LineSymbol.java
NameSymbol.java
RootSymbol.java
StartArrowSymbol.java
StartAssociationSymbol.java

a- the classes of the properties
b- the classes of the symbols

Fig. 7: The structure of the Classes of the properties and the symbols of an Association object

The *Representation* package holds a package called *Normal*. The function of this package is the normal representation. Furthermore, the *Normal* package contains the *NormalRepresentation* class, which sets up the connections of this association with class objects. It also sets up all the symbols of this object such as the multiplicities and the direction arrow. The *Normal* package also includes the *Symbols* package. This package contains classes that define all the symbols of this object as follows:

- *LineSymbol* class: this class basically sets up the drawing of an association line.
- *RootSymbol* class: this class sets up the root of the association.

- *DirectionSymbol* class: this class initialises the direction arrow, and also sets up its location.
- *StartArrowSymbol* class: it initialises an arrow head and draws it at the beginning of an association line.
- *EndArrowSymbol* class: this class initialises an arrow head and draws it at the end of an association line.
- *StartMultiplicitySymbol* class: this class defines the first multiplicity part and fits it in the specified place.
- *EndMultiplicitySymbol* class: this class defines the end multiplicity part and puts it at the end of an association line.
- *NameSymbol* class: this class defines the association name.

Figure 7-b shows the structure of these classes.
Having put all these classes in practice, and created class boxes, the next step is to connect these boxes. This can be carried out by clicking the association icon, and drawing the association line from the source to the target classes. As soon as the association line is drawn, the window of edit properties appears on the screen, and then all the properties can be entered. Figure 8 shows this window.

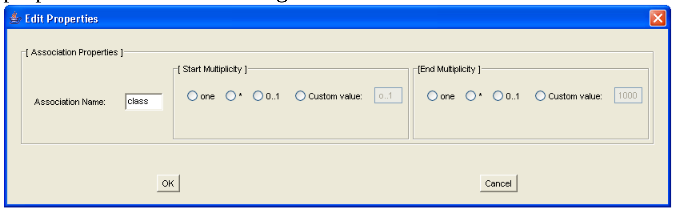


Fig. 8: The Edit properties window for associations

The *AssociationCustomiser* class controls this window and customises it. Notice that this window gives good flexibility to the user since it was designed very well. In other words, the multiplicities have been put in priorities. The high priority multiplicities have been implemented in radio buttons, so that decreases the chance of errors. The lower priority multiplicities can be entered by choosing the *custom value* radio button and customising the multiplicity. Having entered the association name and the required multiplicities and finally pressed the *Ok* button, the association appears in the specified location and connects the classes. Figure 9 shows an example of a class diagram.

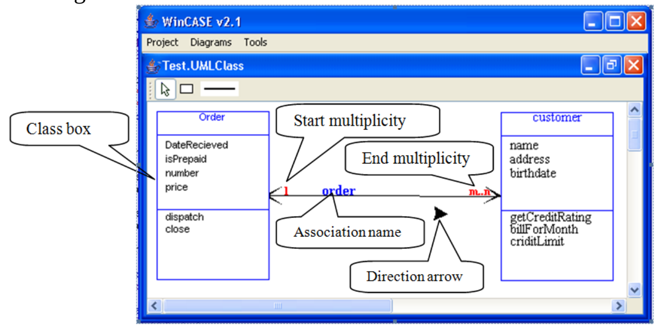


Fig. 9: An example of a class diagram in WinCASE

6 Change Object Properties

The change functions of the objects are available in the class diagram editor. In fact, these functions are defined by the *Diagram* class which is the base class for setting diagrams. They are generic functions, so they are available for all diagrams in the DFA methodology. The change task can be done by *right clicking* with the mouse on the object. This will display the object *popup menu*. This menu contains some items that depending on the type of the object. The *popup menu* of the class object contains four items which allow the user to change or delete this object. Figure 10 shows this *popup menu*. The editing properties task can be carried out by selecting the "*Properties"* item. Clicking this item will display the *Edit Properties* dialogue window, and then all the properties of this object will be available to be changed.

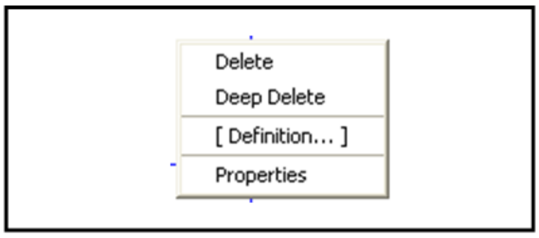


Fig. 10: The popup menu for changing a class object properties

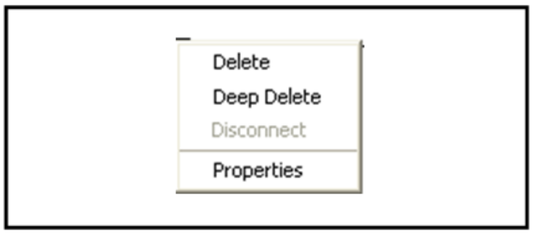


Fig 11 The popup menu for changing an association object

The *popup menu* of the association object contains four items. Figure 11 shows this *popup menu*. The highest item is the "*Delete*" and it performs the deletion of an association with all its properties. The next two items down need to be ignored since they are not related with the class diagram. The lowest item down is the *"Properties"*. Selecting this item will display the *Edit Properties* dialogue window, so all the properties of this object will be available to be changed.

To conclude, having implemented all these functions, the requirements of this research have been met. Furthermore, the class diagram editor has built including the main functions.

Testing and Evaluation

Testing the functions of drawing diagrams is the basic testing task for this research, since the fundamental aim of this research is drawing a diagram and there is no numerical or data outputs.

1 Testing and Development

One of the development process parts was testing. As soon as any class or even sometimes a fragment of code was finished, it was tested immediately. This was to ensure that it functioned properly. This kind of testing was found very convenient in the nature of this research. This technique facilitates in errors detection early.

2 Testing Tasks

a) General Testing

First of all, a general testing was undertaken which tested the main functions that are related with the main menu in the main frame of WinCASE. This testing task also ensured that the class diagram was integrated successfully within the WinCASE system.

b) Testing the Diagram Editor: this stage covers the drawing class diagram functions. The *Objects* and *Associations* were tested accordingly. Tests were carried out on the *Class* as well.

3 Usability Testing: the usability testing was carried out to discover how easy and flexible the task of drawing a class diagram. This can be carried out by giving the system to a number of users and letting them to draw diagrams. Subsequently, ask these users some questions in order to discover any limitation and complexity in the task and to catch any suggestion that may improve this research. The target number of the users are forty one. These users should be familiar with the UML diagram notations. These users were asked number of questions in order to evaluate this class diagram editor. Table number 1 demonstrates these questions.

TABLE 1: Demonstrates the Questions of Usability Testing

No	Questions
1	Is the software easy to use?
2	Does the software run quickly?
3	Is user interface is user friendly?
4	Are the menus easy to use?
5	Are the items clear?
6	Is the navigation in the menu smooth?

In general, the answers of these questions were almost positive and the users were comfortable with the editor functionality. There is just one limitation that was found during the testing stage. The limitation is related to the popup menus of changing the class and association objects. These two menus are generic since they are generated by the base class diagram. These menus contain some items that not needed in the class diagram and therefore they should be disabled.

Conclusion

The aim of this paper is to develop an editor for drawing class diagram within the WinCASE framework.. Thus, the class diagram was implemented and integrated into the WinCASE tool. Consequently, the editor was tested and evaluated. As a result, the editor of the WinCASE has an editor for drawing a class diagram. To conclude, the contribution of this research was an improvement of the WinCASE by integration the class diagram which very important diagram for software engineers that will use this tool.

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.
[13]-Cowling, A., *Normal Forms in the Dataflow Algebra*, 2007, Department of Computer Science Research Report CS-07-11, University of Sheffield.
[14]-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.
[24]-Klaib, A.A., *Integrating Class Diagram within The WinCASE Tool.* framework, 2021. **2**(4): p. 9.