INTERNATIONAL JOURNAL OF RESEARCHES IN BIOSCIENCES, AGRICULTURE AND TECHNOLOGY
© VISHWASHANTI MULTIPURPOSE SOCIETY (Global Peace Multipurpose Society) R. No. MH-659/13(N)
www.vmsindia.org

# EXTRACTING OBJECT MODEL INFORMATION FROM DATA FLOW DIAGRAM

**Ajay D. Shinde**
Departmant of Computer Studies Chhatrapati Shahu Institute of Business Education and Research, Kolhapur.

**Abstract**
The model is diagrammatic representation of artifacts to be implemented in software. Data flow diagram is an important tool in procedure oriented software engineering. It is used to represent working of system in terms of its users, functions, data stores and data flows, But this is an old technique whereas object oriented software engineering is new emerging area. It is possible to establish the relationship between procedures oriented and object oriented software engineering. In this paper an approach is presented to extract information from data flow diagram that helps to find out object oriented artifacts. The approach presented in this papers helps to extract information required for building use case model, Class model and other dynamic models. It will be a great help for developers switching from procedure oriented to object oriented domain.

**Keywords**: Data Flow Diagram, Software Engineering, Use case model, Class model, Object Oriented Software engineering

**The need**

Data Flow Diagram (DFD) is graphical diagrams for specifying and visualizing the model of a system to be developed. DFD is useful in defining the requirements of user in a graphical view. Once the requirements from user are collected in analysis phase DFD is drawn to represent working of the system in terms of users and user roles, Data flows, functions and data stores [5]. Since only one diagram is not sufficient for showing working of the system in detail, DFD is broken down in to level, where each level gives more detailed working description for one of the functions from earlier level. In order to have logical linking between levels of DFD the consistency check mechanism becomes essential between a higher level and lower level DFD [1]. As the software development and design field is maturing day by day the systems are becoming more complex, it becomes essential to have new approaches for specification and design. In additions to this there is a need to raise the level of abstraction so that designer focuses more on the concepts and different views of the system and less on the implementation details. The approach should have necessary support for automation, consistency checking and verification; this in turn will reduce costs and time required for development of the new systems [6]. In order to tackle these problems, one should use models to describe systems, starting from the high-level specification to the implementation. In last few decades object orientation has become more popular and gaining more and more support from the software community. The reasons for this may be (i) It is becoming standardized and (ii) provides support for representing system model using various diagrams from different development and implementation perspectives [13]. But still many developers use DFD as requirement analysis tool, in addition to this many old systems are still in use and DFD is part of their documentation. Now it is very necessary to find out concepts represented in DFD and relate it to object oriented models So that object model information can extracted from it. The primary aim of this paper is to study interrelation between structured methodologies like DFD and object oriented methods and design an approach for extracting object model information from DFD which is a structured method. The main contributions of this paper are (i) to establish relationship between DFD and object oriented approach (ii) to identify set of rules for object model information extraction from DFD.

**Review of Earlier Works**

Till day many researchers and academicians have already studied the relationship between of DFDs and object-oriented concepts. A survey of this relationship is given in [8]. In this paper authors discuss some of the approaches that are relevant for the work presented in this paper. In [4] Object-Process Methodology (OPM) is proposed that uses objects and processes. An Object-Process Diagram (OPD) shows both the behavior and structure of the system that include both processes and objects. Objects may be persistent or transient entities and processes perform some or the other operation on them. In addition, OPD has state diagram to describe the objects. The OPM has timing constrains, conditions, exceptions, control flow structures and events for modeling, specifying, and designing reactive and real-time systems [11]. In another proposal, two types of functional models are suggested: Object Functional Models (OFM) and Service Refinement Functional Models (SRFM). The DFD notation is modified and the roles of the functional models are redefined, in

order to use DFDs while retaining the spirit of object-orientation. OFM is used to model the services provided by individual objects, while SFRMs models aggregate objects to show the service composition of individual objects. Both models use objects, processes and data-flows as modeling objects they do not consider data store as necessary. The interactions with a data store are modeled as communications with the corresponding object.

In [10], L. B. Becker and others have proposed E-DFD (an extended version of the traditional DFD) where the functionalities are described and then associated with respective use case. SysObj, is tool designed that uses these inputs and then automatically generates an object model that describes architecture of the system. This method is also related to an integrated environment for developing distributed systems with the object-oriented paradigm [3]. In OMT, DFDs are also used to describe the functional model of a system [9]. Since in OMT the system is also specified by two other models (the object and the dynamic models), DFDs specify the meaning of the operations in the object model and the actions in the dynamic model. Although there is some attempt at integration, this correspondence is left completely vague and can not be analyzed in any useful way.

In reverse generated DFDs source code is interpreted and information about the objects is gathered to form the system [7]. This is hybrid approach, because it is semiautomatic, it requires the assistance of a human expert with domain knowledge. In [2] Alabiso proposes a method for transformation of DFDs into objects. In his work he proposes the following activities: (1) Find Data, Data Processes, Data Stores and External Entities and relate them to object-oriented concepts; (2) Interpret the DFD leveling to find out object decomposition; (3) Interpret Control Processes in view of the object-oriented model; (4) Finding object decomposition definition from data decomposition. Another interesting proposal given by P. Shoval and J. Kabeli is the Functional and Object-Oriented Methodology (FOOM) [12], which is specifically tailored for information systems. The basic idea behind FOOM is to use the functional approach during analysis phase that enables developer to define user's requirements and use object-oriented approach during design phase so as to specify the structure and behavior of the system. In FOOM, OO-DFD is used for user requirement specifications where data stores are replaced by classes and data terms are replaced by object oriented schema, or an ERD which can be transformed into an object-oriented schema. In the design phase, the information from analysis is used for detailed object-oriented and a behavior schemas. These schemas are the used as input to the implementation phase, which is ultimately transformed in to object-oriented programming language.

**Extracting information from DFD**

**Data Flow Diagram**

Data Flow Diagram (DFD) is basically used for representing model of business processes. For modeling business processes many notations are used. For our solution of transformation we have picked out DFD for the following reasons. Firstly it is easy to understand for all stakeholders. Secondly easy to create and the third one is their wide usage. In general, DFD can be defined as one of the older techniques for business process modeling. DFD is a modeling technique used for software engineering that graphically displays data flows from external entities into the IS and vice versa. DFD displays data that pass from one process to another. Data flows diagrams are used as graphical notation for process expression of data processing. In the practice, software designer often first draws the context level of DFD that represents interaction among the system and the surrounding elements. This level of DFD is decomposed into lower levels that model parts of the system in a greater detail.

A data flow diagram uses very limited number of symbols; the DFD can be represented as a set of symbol sets. In the proposed framework the DFD is represented as a graph using atomic relational grammar. The DFD may be defined as

DFD = {{SS}, {PS}, {DS}, {TS}, {RS}} where,

DS represents set of data flows

PS represents set of processes that may be either atomic or aggregate

TS represent set of data stores

SS represent set of source consumers

RS represents the set of relationships

Consider Fig.1, it is represented in framework using five sets. The source consumer set is

SS = {student, examiner, exam section}

The process set

PS = {register student, assign exam no print hall ticket, conduct examination, evaluation}

The data store set

TS = {student, attendance, answer sheet, marks}

The data flow set

DS = {appl. Form, hall ticket, attend, student data, student marks, subject knowledge, written answer sheet, record, rules}

RS = the members of relation set RS are derived from set DS. Where dataflow set members are

connected at both the ends to members of SS, PS and TS.

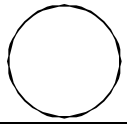
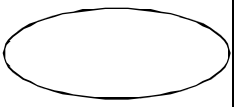
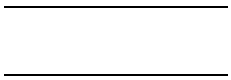
Further when a process from level-1 DFD is expanded in to level-2 DFD we can obtain a new set of symbol sets, from which we can again obtain 5 different sets {{SS}, {PS}, {DS}, {TS}, {RS} }. This recursive definition can be applied to N number of level of DFD.

**Extracting Use Case Diagram information from DFD :**

UML use case diagram capture functionality that will be covered by the future IS that means they describe the exact functions that will be performed by IS. The implemented IS will not contain anything else but what is described in the use cases. Each use case describes one of the ways of usage of the system from the user's point of view thus it describes its needed functionality.

If we compare use case diagram with DFD both of them represent functionality and users for proposed system. We can extract this information from DFD by using mapping set given in Table 1. From this information it is possible to build sets of actors and use cases that are used for constructing use case model for object oriented system. The data flows between users and processes give us connection between use cases and actors in use case model. As data stores are not part of use case diagram it is to be excluded from the design information extracted. In addition to this if a process from DFD is expanded in to next level, we can use this information to find out extends and uses relationship between use cases. The process expanded and processes shown in expanded DFD are to be represented as use cases with either uses or extend relationship.

**Table 1:** Set of Mapping symbols from DFD to use case diagram

| DFD | | Use Case Diagram | |
|---|---|---|---|
| Symbol | Meaning | Symbol | Meaning |
|  | Source and consumer of data |  | User of the system |
|  | Processes in the system |  | Use case in the system |
|  | Data flow and its direction |  | Link between user & use case |
|  | Storage of the data | No corresponding symbol available in use case model. The data store becomes class in class model. | |

By using mapping symbol table the information from DFD is mapped to Use case model of UML. This information is then used for generating use case diagram. From this information we can build following sets

UCD = {{AS}, {US}, {LS}}

Where UCD is set of sets

AS = set of actors generated from set of source and consumers from DFD

US = set of use cases generated from set of processes from DFD

LS = set of links generated from set of data flows from DFD

The above sets are used for translation of DFD to use case diagram.

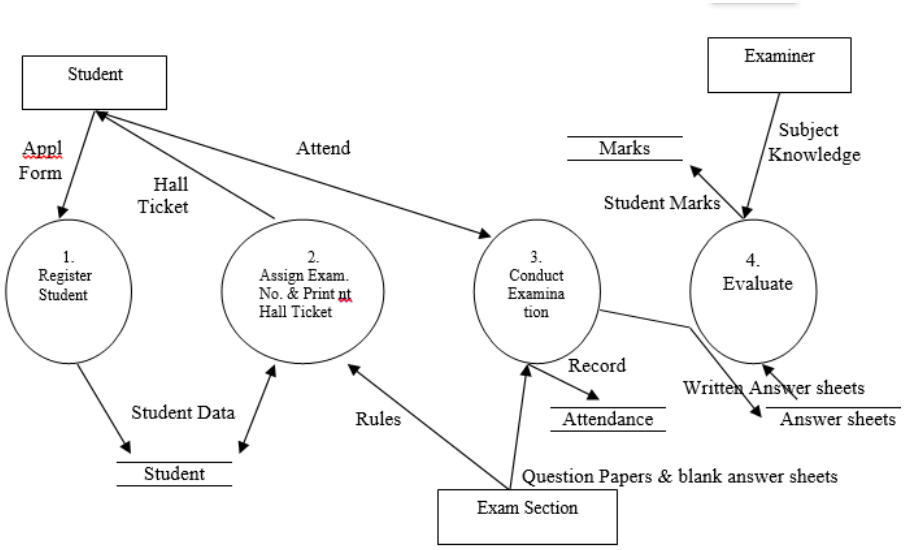



**Figure 1.** Level-1 DFD for Examination system.

**Extracting other information from DFD :**

The functionality in object oriented system is represented with the help of use cases. In order to implement this functionality in the object oriented system we need to design classes. The Class Diagram shows classes and relationship between them. It is extremely difficult or highly impossible to implement use case with the help of only one class, we may require collaboration between various classes. These classes you use in the application may be classified in to one of the following category -i) Entity ii) Control iii) Interface/Boundary classes. The DFD shown in fig. 1 has data stores and roles played by users. These are probable classes in object oriented systems. The data stores are entity classes where information is to be stored permanently in the databases. Roles played by the users are also probable classes in the proposed system and are to be represented as entity classes. In addition to this each process shown in data flow diagram requires interface this has to be implemented using existing classes in the programming language or designing new classes. The control classes can be identified by looking at processes that are related to each other. From this we may find out base for designing control classes.

The collaboration diagram shows interaction between classes when methods are executed. The collaboration is message passing between objects this can be identified by looking at the dataflow shown between the processes.

**Concluding Remarks**

The method presented in this paper is useful for IT professionals who are shifting from procedure oriented design to object oriented design. This method of course will not give all possible models for object oriented paradigm, but still a major part that comprises of use case model and class model can be built from it. As information from DFD used to extract use case and class model information. User need not have to put extra efforts for building object oriented model. This will result in saving of time and effort as well as the user will be able to understand relationship between procedure oriented and object oriented paradigms.

**References**

[1] Ahmed Jilani, A. A., Nadeem, A., Kim, T. H. and Cho, E. S. (2008). Formal Representations of the Data Flow Diagram: A Survey. *Proc. of the 2008 Advanced Software Engineering and Its Applications*.Washington, USA: IEEE Computer Society. pp. 153-158

[2] B. Alabiso. Transformation of Data Flow Analysis Models to Object Oriented Design. In *OOPSLA '88*, pages 335–53. ACM Press, 1988.

*[3]* Bieliková, M. Softvérové inžinierstvo – *Princípy a manažment, skriptá, Publisher: STU, Bratislava, ISBN 80-227-1322-8, 2000.*

[4] D. Dori. *Object-Process Methodology — A Holistic Systems Paradigm*. Springer Verlag, 2002.

[5] Dennis, A., Wixom, B.H. and Roth, R.M. (2006). *Systems Analysis and Design*. 3rd ed. Hoboken: John Wiley & Sons, Inc.

[6] Dragos Truscan, Jo˜ao M. Fernandes . Johan Lilius Tool Support for DFD-UML Model-based Transformations Proceedings of the 11th IEEE International Conference and

Workshop on the Engineering of Computer-Based Systems (ECBS'04)

[7] H. Gall and R. Kl¨osch. Finding Objects in Procedural Programs: An Alternative Approach. In *2nd Working Conference on Reverse Engineering*, pages 208–16. IEEE CS Press, July 1995

[8] J. M. Fernandes and J. Lilius. Functional and Object- Oriented Modeling of Embedded Software. In *11th Intl Conf. and Workshop on the Engineering of Computer Based Systems (ECBS'04)*, May 2004.

[9] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall International, 1991.

[10] L. B. Becker, C. E. Pereira, O. P. Dias, I. M. Teixeira, and J. P. Teixeira. MOSYS: A Methodology for Automatic Object Identification from System Specification. In *3rd IEEE Intl. Symp. on Object-Oriented Real-Time Distributed Computing*, pages 198–201. IEEE CS Press, Mar. 2000.

[11] M. Peleg and D. Dori. Extending the Object-Process Methodology to Handle Real-Time Systems. *Journal of Object Oriented Programming*, 11(8):53–8, Jan. 1999

[12] P. Shoval and J. Kabeli. FOOM: Functional- and Object-Oriented Analysis & Design of Information Systems — An Integrated Methodology. *Journal of Database Management*, 12(1):15–25, Jan. 2001.

[13] T. Korson and J. D. McGregor. Understanding Object- Oriented: A Unifying Paradigm. *Communications of the ACM*, 33(9):40–60, Sep. 1990.