DESIGN AND IMPLEMENTATION OF A SYSTEM FOR TRANSFORMING FUNCTIONAL MODEL INTO OO MODEL

Yong Yang and Hee Beng Kuan Tan

School of Electrical and Electronic Engineering
Block S2, Nanyang Technological Univeristy
Singapore 639798

Email: yangyong@pmail.ntu.edu.sg

ABSTRACT

This paper describes the design and implementation of a prototype system, F2OO for the enhanced data flow diagram, called data flow net (DF net) (Tan, Yang, & Bian, 2006). The prototype system transforms the software systems designed in DF net to OO design and implementation systematically, precisely and automatically. This has greatly enhanced the OO methodology by synthesizing functional decomposition, which is a well-developed concept in traditional structured methodology. The prototype system facilitates the software designer to construct the functional analysis models for some of the use-cases in their own systems in DF net. Thereafter, together with remaining use-cases using any existing OO software development methods, the whole systems can be implemented following current OO design and implementation. This is very useful in realizing use-cases, especially those with more complex functions.

Keywords: DF net, Functional Decomposition, OO design, F2OO

1.0 INTRODUCTION

Structured methodology (T.DeMarco, 1978), firstly proposed by T.DeMarco in 1970's, is a mature methodology after years of development. It provides a rich set of techniques and tools which facilitate the construction of data models, functional models and dynamic models of large systems.

The data flow diagram (DFD) was first introduced in structured analysis and design (T.DeMarco, 1978). It has been widely used for the development of information systems through providing a visual view on functional refinement. Object-Oriented methodology, emerging in early 1980's, is claimed as very natural system development approach. Based on its strengths on abstraction, encapsulation and inheritance, Object-Oriented (OO) approaches have advantages over structured methods in modeling, maintenance and reuse. However, despite of its strength, OO methodology suffers from the lack of emphasizing on functional decomposition and the benefits of applying it have been discussed in (Wolber, 1997). The need of incorporating functional refinement in OO software development, especially for problems with more complex functions, has also been brought up in (Jalote, 1989). Many OO approaches have attempted to incorporate DFDs into OO paradigm to model the functional aspects of systems (Alabiso, 1988; Gray, 1988; Rumbaugh, Blaha, Premerlani, Eddy & Lorensen, 1991).

However, except the recent approach proposed in Wang (2002) that distributes functions directly into objects in the OO approach, no systematic and well-defined approach or method has been defined so far due to the fundamental difference in the decomposition strategies between DFD and OO approaches. DFD approaches decompose systems according to functionality while OO approaches decompose them according to the static object structure. Tan, Yang, and Bian (2006) on the other hand, motivated by the inconclusive and often conflicting results on comparing the use of OO approach against the structured approach, a study was conducted to compare the two approaches (Pressman, 2005). One main result obtained from this study indicates that the OO approach is not "more natural" than the structured approach. This prototype system applies the DF net theory to enhance the OO methodology by incorporating functional decomposition, taking the advantages of existing techniques and tools in structured methodology.

The rest of the paper is organized as follows: Section 2 briefs through the DF net theory, including some terms which will be used in later sections and workflow of DF net analysis and design process; Section 3 describes the architecture of the F2OO systems and technical specifications are also provided. Finally, the conclusion and future work are in last section.

2.0 DF NET

DF Net theory provides the following analysis artifacts. They are, process, pdfd-sub-process, process attribute, data buffer, output data flow, data flow referenced, output data flow attribute and data flow attribute referenced, in a DF net. Likewise, the term design artifact refers to any class or procedure constituent element as which a DF net artifact is realized. The overall processes have three stages, which are analysis stage, design stage and implementation stage, shown in Fig. 1.

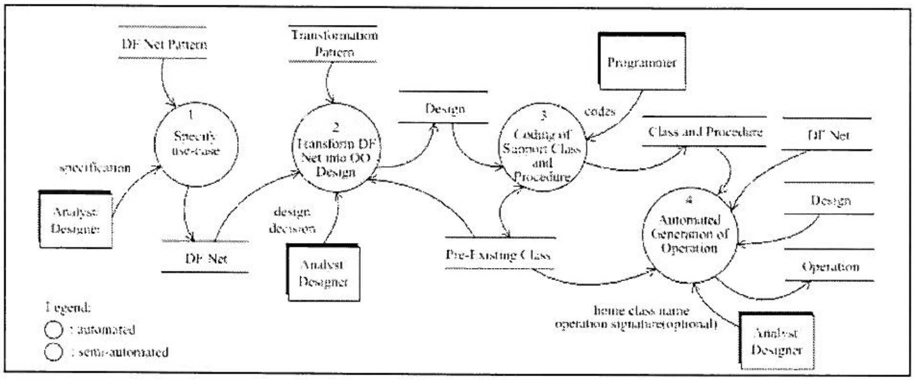

Fig. 1: Workflow of DF net methodology

2.1 Analysis Stage

DF net approach is to be applied from requirements analysis stage onwards. In requirements analysis stage, it realizes a use-case through functional decomposition and specifies it in multi-levels DF nets hierarchically as at DFD until each lowest-level process specifies a meaningful external interaction (including interaction with external entity, data store or data buffer) or externally meaningful computation of data (Tan, Yang, & Bian, 2006).

2.2 Design Stage

In the design stage, for all the use-cases specified in DF nets, designers synthesize and realize each process in the DF nets. Once, this is done, for each use-case, a realization of the use-case as an operation is automatically derived. Designers only need to decide a class to house the operation. For detailed information, please refer to Tan, Yang, and Bian (2006).

2.3 Implementation Stage

In the implementation stage, design model gained in previous step will be transformed to OO code. There are two sub-stages in the implementation of use-cases that are realized using the proposed approach: (1) coding of supporting classes and procedures; (2) automated generation of operations (Tan, Yang, & Bian, 2006).

3.0 F2OO

After brief through some basic knowledge of DF net, in this section, F2OO, a tool for Seamless Transformation of Functional Model into Object-Oriented Design & Implementation, applies Data Flow Net (DF Net) is introduced. The whole prototype system follows Model-View-Control (MVC) architecture, which is widely used in current industrial practice.

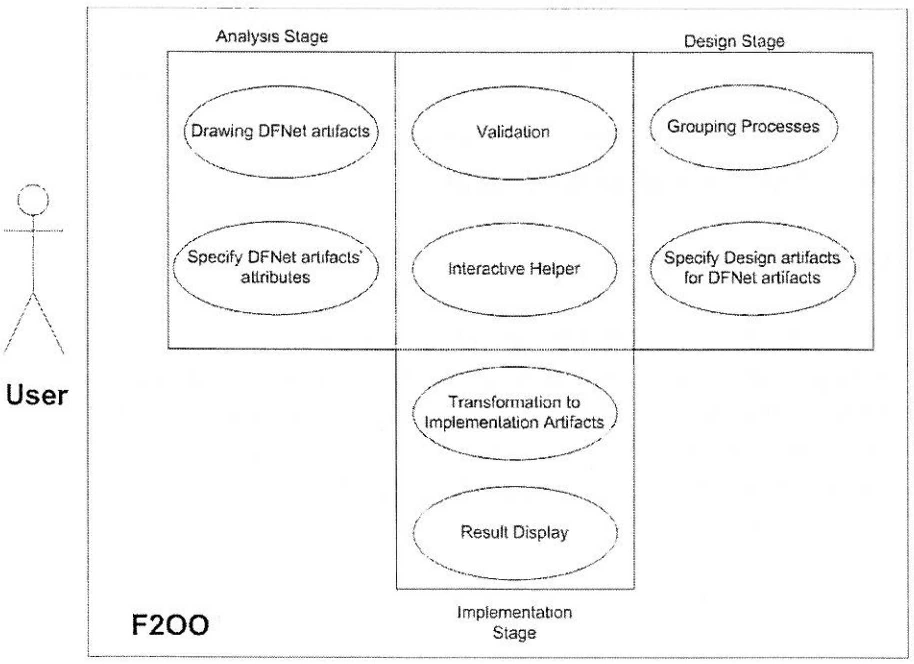

Fig. 2: System's high-level functional feature

3.1 System Overview

F2OO employs a process-based architecture to facilitate the designers to build their functional models and transform them into OO design and implementation through three stages, namely, analysis stage, design stage and implementation stage. F2OO is primarily written in Java.

3.1.1 Component Overview

The prototype system design follows Model-View-Control (MVC) standard. The F2OO system architecture is shown in Fig. 3. It comprises four main components:

- System UI: works as graphical user interfaces of the system.
- System Model: most of the DF net logics are implemented herein. Based on the functionalities, it comprises four sub-components, which are DF net Core Elements, Stage Control, DF net File Management and Algorithm.
- System Control: bridges the System UI with the System Model. It responds to events, typically user actions, and invokes changes on the System Model accordingly.
- Helper: provides helping and guiding the analysis, design and implementation tasks.

3.1.2 Functions Overview

The F2OO system provides users with functions to construct the functional models and transform them into OO design and implementation. The whole process is divided into three stages, namely, analysis stage, design stage and implementation stage. The functional features of the F2OO system are depicted in Fig. 2.

Based on these different stages, functional features provided by the system are summarized as follows:

- Analysis stage functions – In analysis stage, functions are provided to construct the DF net diagram of DF net artifacts for a given use-case. These functions involve drawing DF net diagram and specifying DF net artifacts.

- Design stage functions – In design stage, users are required to map their DF net artifacts with OO design artifacts (classes, methods, attributes etc). Major functions provided in this stage include grouping processes and specifying design artifacts for DF net artifacts.
- Implementation stage functions – functions are provided to implement all the design artifacts in the implementation stage. These include transformation and displaying result. The design model constructed in previous stages will be transformed to OO implementation automatically.
- Other functions – validation function and helper function are provided in all stages. Validation function can be categorized into three types, namely, analysis validation, design validation and implementation validation. Helper function is used for helping and guiding the analysis, design and implementation tasks.

In the rest of this chapter, the design and implementation of each individual component which carries out these functions will be discussed.

3.2 Implementation Details

In this section, the detailed design and implementation of the four major components of the F2OO system, namely, System UI, System Model, System Control and Helper, are discussed.

3.2.1 SystemUI

SystemUI module provides the designers system level interactions with the prototype system.

3.2.2 System Model

System Model Module covers the entire DF net logics. It acts as Model (M) part in MVC structure of the prototype system. The internal structure of System Model Module also follows MVC structure, like the Rational Rose.

3.2.2.1 DF net Core Elements

DF net core elements API works as the skeleton of the prototype system. It covers all the DF net artifacts including DCFProcess, DCFDataflow, DCFDataStore, DCFEntity, DCFSubProcess and DCFDataBuffer etc. shown in Fig. 4.

3.2.2.2 Stage Control

3.2.2.2.1 Analysis Stage

Analysis Stage Control provides the users with all the functionalities needed in Analysis stage. These include, drawing DF net diagram for given use-cases and specifying related attributes of these DF net artifacts. The following sections introduce the detailed design and implementation of these two functions.

3.2.2.2.1.1 DF net Diagram Constructing

Classes in DF net Diagram Constructing inherit from JGraph packages to provide users with drawing function to construct their DF net graphs in requirement analysis stage.

3.2.2.2.1.2 DF net Artifacts Specification

After drawing the corresponding DF net diagrams of the use-cases, users also need to specify the DF net artifacts in the analysis stage. These attributes, such as the name of every process, every dataflow and every dataflow attribute etc, are referred to as the elementary attributes of the DF net core elements. Some of the analysis stage validations are carried out based on these attributes. It contains three functions, namely process editing, data flow editing and others editing.

3.2.2.2.2 Design Stage Control

Design stage is rightly after analysis stage. With its input of the DF net artifacts gained in previous stage, users need to map these inputs with their corresponding design artifacts, before proceeding to the next stage. Design artifacts refer to any class's or procedure's constituent elements as which DF net artifacts are realized. Design Stage Control includes two major functions, namely Process Grouping and Design Artifact Specification. The work flow of this stage is shown in Fig. 5.

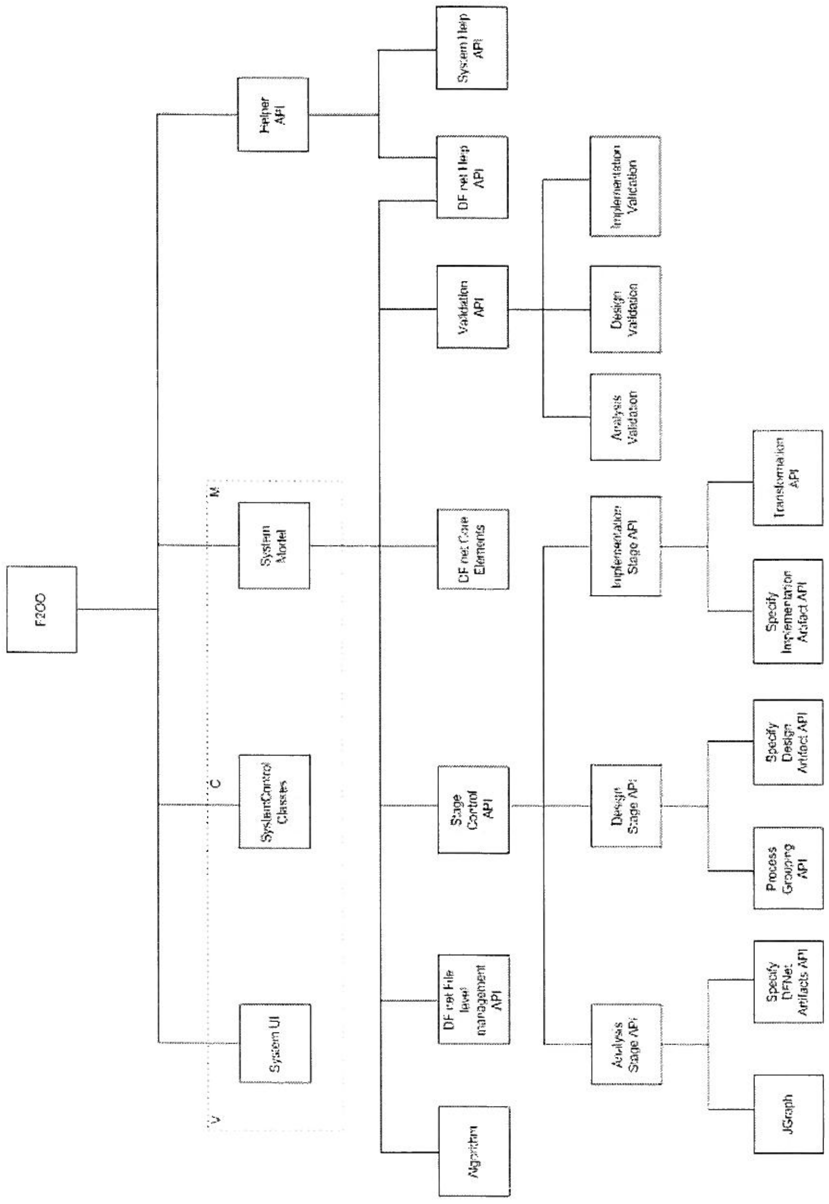

Fig. 3: System's architecture

3.2.2.2.2.1 Process Grouping

Process grouping provides the functions to realize the first step of Design stage, which is to group the processes. In DF net theory, there are three transform types of processes, namely, pdfd-sub-process based transform type, process-based transform type and procedure-based transform type. The first two methods require the designer to group the processes into different classes before moving to their corresponding sub processes design.

3.2.2.2.2.2 Design Artifacts Specification

Designing Artifact Specification provides functions to specify the detailed design artifacts for all the DF net artifacts based on the outputs of the grouping process. For processes which have been grouped, users need to match each of them with its design artifact in OO methodology. For processes which are specified to be realized as procedures, users can provide java source code here. All DF net artifacts need to be mapped with their specific design artifacts correspondingly.

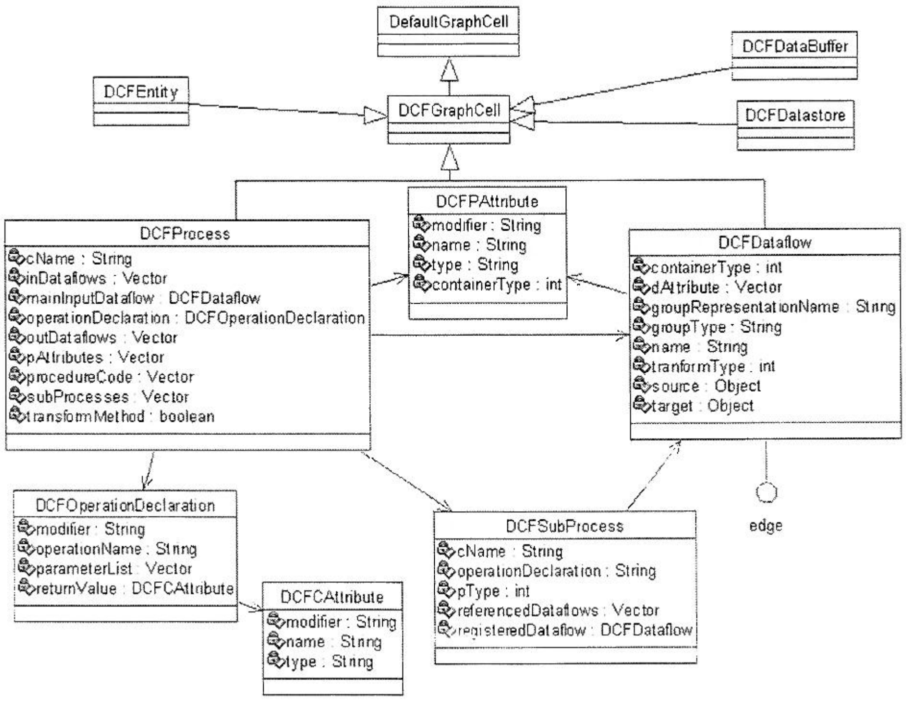

Fig. 4: Class diagram for DF net core elements

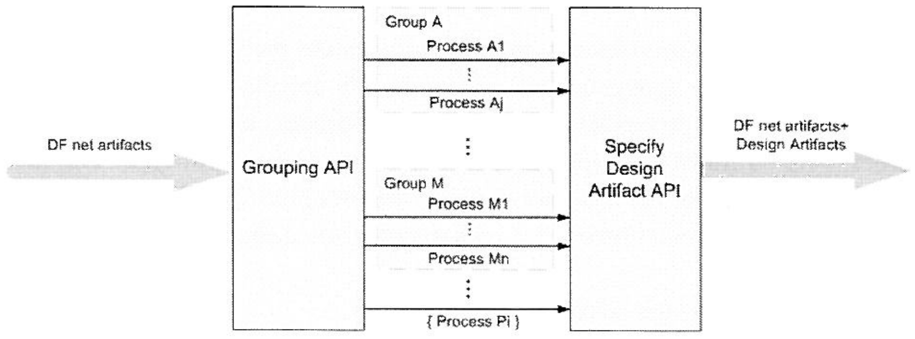

Fig. 4: Class diagram for DF net core elements

In design stage, attributes of the processes which need to be specified are shown as follows:

- Implementation environment: currently the only option is Java.
- Operation signature: if pdfd-sub-process-based method is selected, user needs to input the names of the operations which the sub processes are transformed to. If process-based method is chosen, user needs to input the name of the operation to which this process is transformed.
- Procedure code: if procedure-based method is selected, user needs to input the java code for the procedure to which the current process is transformed.
- Procedure symbols: if procedure-based method is selected, users need to specify the name and type of the variables used in the procedure code segment.

For each individual output data flow of the process, the following attributes need to be specified:

- Design method: output data flows of a process can be transformed to either Class Attribute (CA) or Return Value (RV). In some cases, users do not need to specify them, because there are algorithms for automate deducing. Otherwise, users need to specify.
- Multiplicity: user needs to specify the multiplicity of the output data flow. These include multiplicities: 0…1, 1 or many.
- Variable type and name: in the prototype system, they are referred to as groupType and groupRepresentationName. For the output data flow with

multiple dataflow attributes and its design method is specified as RV, users need to group its data flow attributes into a class and give the class type, name of an instance accordingly.

For each individual referenced data flow of a process, it can be realized as either Class Attribute (CA) or Pass-in Parameter (PI). Prototype system has provided some algorithms for automate deduction.

3.2.2.2.3 Implementation Stage Control

Implementation Stage Control lets the designer to give the implementation artifacts to the design artifacts in the previous stage. The final transformation of the use-case is also carried out here.

3.2.2.2.3.1 Implementation Artifact Specification

Implementation artifacts include two major aspects:

1) Java source code implementation of the operations gained from Design stage.
2) Names of the instances declared for the classes and PI, RV whose multiplicity is many.

3.2.2.2.3.2 Transformation

Transformation function automatically generates the java source code for the use-case specified and its corresponding OO structures are also shown in XML format. Based on different functionalities, transformation function can be divided into three parts, namely, Pre-processing, Transforming and Result Displaying. The following gives the details:

1) Pre-processing
   This part focuses on the pre-processing work for the transformation. This includes two major tasks. One is to get all DCFProcess instances; the other is to sort them according to the traversing requirements.
2) Processing
   This part focuses on the transformation algorithm design and implementation. It is constituted of Transformation.java. Fig. 6 is the pseudo-code for the algorithm for automated generation of operations. List of the conditions used in Fig. 6 comes after the algorithm.

```
1. Declaration section
  1.1 declare and instantiate an instance for the classes that realize process.
  1.2 declare a variable for each output dataflow that are realized as RV
  1.3 declare a variable for each output dataflow attribute and referenced dataflow attribute of
      process p realized as procedure.
  1.4 declare a variable for each partition which satisfies condition 1).
  1.5 Insert port of initialization.
2. Generating code for each process (P) section.
  2.1 Initialization of code segment.
    2.1.1 P is process-based, code segment = empty.
    2.1.2 P is pdfd-sub-process based, its individual sub process' code segment = empty.
    2.1.3 P is procedure-based, code segment = P's procedure code.
  2.2 Invocation of class operation.
    2.2.1 P is pdfd-sub-process based. Append each individual class operation statement to the
          corresponding sub process's code segment.
    2.2.2 P is process-based. Append P's class operation statement to its code segment.
  2.3 Output dataflow realization. For each output data flow e of P,
    2.3.1 Establishing protocol. If P is pdfd-sub-process based, append following statements to
          its main sub process's code segment; if P is process-based, append following
          statements to its process's code segment.
    2.3.2 Making single output instance accessible, if satisfying condition 2).
          2.3.2.1 if e's multiplicity =1, append "// port of "+ e's name
          2.3.2.2 if e's multiplicity = 0..1, append selection construct and "// port of "+ e's name
          2.3.2.3 if e's multiplicity = *, append a loop and "//port of"+ e's name within the loop.
    2.3.3 Instantiating Container Argument Element. If e satisfies condition 3) and e is container
          type, instantiate an instance of container argument element's type. Insert it before
          "//port of "+e's name.
    2.3.4 Assigning Value. If e satisfies condition 3), generate an assigning statement, insert
          before "//port of"+ e's name.
    2.3.5 Inserting container Element: if e is referenced by another process as container type,
          and satisfies condition 3), generate insertion statement and insert it before "//port of"+
          e's name
3. Sequencing code segments of all process. Traversing follows condition 4), for each P,
  3.1 if P'main input dataflow ! =null, insert following code segment before port of its main input
      dataflow, else insert the code segment before port of initialization.
    3.1.1 P is process-based, insert its process's code segment
    3.1.2 otherwise, insert its main sub process's code segment.
  3.2 For each output data flow e of P, if there is initialization code segment registered to it,
      insert it before "//port of" + e's name.
  3.3 For each output data flow e of P, if there is ancestor or input sub process registered to it,
      insert it before "//port of" + e's name.
```

Fig. 6: Transformation algorithm descriptions

Conditions used in the pseudo-code are,

- Condition 1) ---- Partition Criteria: let Q be the set of container operation arguments each of which realizes a data flow d referenced in a process such that excluding d there are data flows in the path from the main input dataflow of the process to d with output multiplicity =*. We partition Q in such a way that container arguments that realize the same dataflow referenced in processes such that the same attribute if the dataflow are referenced in these processes and the paths between the main input data flows of these processes do not include data flows with out put multiplicity = * are put together in the same partition. (as a result, operation arguments in the same partition have identical value).

- Condition 2) : the output dataflow e of process P is registered with some pdfd-sub-process in processes that are not realized using class by applying process-based method or e is referenced by another process R such that f2d(e.Q) is a container operation argument and (f2d(e.P) is not defined or f2dod2i(e.Q)!= f2d od2i(e.P)).
- Condition 3): e is referenced by another process R such that f2dod2i(e.Q)!= f2d od2i(e.P))
- Condition 4) ---- Traversing Criteria: processes will be traversed in such a way that a process will only be traversed if all the processes, which produce its input data flows, have been traversed.

3.2.2.3 Validation

This part provides the validation functions to the users throughout the whole process. Based on the stage in which the validations are performed, it can be divided into three categories, namely, Analysis Validation, Design Validation and Implementation Validation.

3.2.2.4 DF net File Management

In order to support hierarchical decomposition, DF net File Management function is provided to manage different levels of DF net files for a use-case. In DF net, a process in higher level DF net can be decomposed and modeled by a set of lower level DF nets. Managing the relations among different levels of DF net files is needed. In the prototype system, different levels of DF net files are identified by their file names.

3.2.2.5 Algorithm

In summary, Algorithm function provides the algorithms used in the implementation of the prototype system. It comprises two classes, namely Algorithm.java and UserGuid.java from com.dcfnet.core. Some of the algorithms defined inside Transformation.java are also included. Algorithm function comprises the main algorithm like Combination of Process, Get path, Check Synchronization, Referenced data flow's design artifact deduction (pdfd-based) and Output data flow's design artifacts deduction, etc.

3.2.3 System Control

System Control Module API bridges the view with its corresponding model in the prototype system. DCFMain.java is the main class of the software and also the start point of this prototype system. Inside of the main method of DCFMain.java, an instance of DCFBuilder.java is instantiated, which acts as the host for the prototype system.

3.2.4 Helper

Helper functions cover the assistance providing functions to the designer, not only for DF net theory part, but also to show how to use the system.

4.0 CONCLUSION

F2OO provides software designers with a tool to construct their functional models and transform them into OO design and implementation. F2OO's architecture contains four main components, namely, System UI, System Model, System Control and Helper. The following functions are delivered by the system:

- In analysis stage, the F2OO system helps the users to construct their own DF net analysis models and specify the DF net artifacts.
- In design stage, the F2OO system facilitates users to map their DF net artifacts to design artifacts and specify the attributes of these design artifacts.
- In implementation stage, the F2OO system provides the users with the functions to specify the implementation artifacts of those design artifacts gained from previous stage and transform the design model into OO implementation.
- Validation functions and helper functions are provided throughout the whole process. Analysis validation, design validation and implementation validation validate the user input in different stages, based on DF net semantics and rules. Helper functions provide helping and guiding the analysis, design and implementation tasks.

F2OO has also validated the effectiveness of the DF net approach. As for any use-case realized using the proposed approach, once the classes and procedures to realize its processes have been designed and coded, the design and code of an operation to implement the use-case is fully automatically generated by using F2OO. The time and efforts spent for transforming the DF net analysis models to OO implementation have been greatly reduced. The design and coding of an operation to implement a complex use-case will not pose any problem to the use of the proposed approach.

5.0 FUTURE WORK

In DF net, as the control flow between processes is fully implied, a process is no longer required to include information on other processes for the purpose of process interaction. Thus, analysis models for commonly used functions represented by DF nets could be more reusable. The extension of the proposed approach to component-based software development is a natural extension. Reusability receives more and more attention in current practice and many component-based web technologies have been developed to support this, such as Enterprise Java Bean (EJB). EJB has been widely used to promote the components' reusability in various ways. Firstly, it separates business logic from presentation logic, which makes the modeling of the business functions independent from presentation logic. Secondly, the business logic of enterprise beans can be reused through java subclassing. In the analysis stage of these component-based web applications, the proposed approach can be used to construct more reusable EJB analysis models based on the decomposition of the business functions.

Currently, because the implementation of presentation logic involves a lot of non-java technologies, such as the use of xml configuration files in struts application, it makes the analysis of control flows difficult. However, because of the separation of business logic and presentation logic by EJB, the proposed approach can be used in business function modeling without being affected by those non-java technologies in presentation logic modeling. During the design stage of the proposed approach, instead of mapping the DF net analysis models to ordinary classes and operations, they can be mapped to entity beans, session beans and interfaces. This allows the EJB applications to incorporate the advantages of the proposed approach in functional decomposition during the analysis stage. In addition to this, the methods discussed in the proposed approach to deal with the overlapping among use-cases can also be applied in EJB business function modeling to improve the reusability of the business logic of enterprise beans. Therefore, we believe that the proposed approach might help in the development of more reusable analysis models and patterns that can be associated with required design and implementation alternatives and reused on an integrated basis from requirements analysis via design to implementation in component-based web applications. This is a direction for further research.

Another interesting direction would be to use the proposed approach to model distributed database transaction systems. As for such data-intensive systems, the realization of some of the use-cases can be modeled based on process interacting through data flows in DF net. However, because the control flows are fully derived from DF net structure, future research can be carried out to enhance the DF net structure to make it capable of modeling the distributed control flows among the servers and clients.

In the proposed approach, the realization of use-cases is modeled based on processes interacting through data flows in DF nets. The control flows associated with the interaction are derived fully from the structure of the DF nets. This is suitable and natural for a large class of data-intensive systems that are data driven. For systems that are time dependent and processing a lot of signals and events, such as concurrent systems etc, the current DF net might not be sufficient for the analysis modeling of their use-cases. To extend the DF net to represent the control flow and timing of such systems could be a future direction. Software architecture has received much attention recently (Morris, Speier, & Hoffer, 1996) (Shaw, 1996). The bridging between software requirements and architectures is one important issue (Grunbacher & Medvidovic, 2001; Nuseibeh, 2001; Rumpe, Schoenmakers & Radermacher, 1999). Exploring on the possible use of DF net to address this issue could also be a further research direction. F2OO will provide corresponding functions for component-based DF net design and implementation in the future.

REFERENCES

Alabiso, B. (1988). Transformation of data flow analysis models to object oriented design. Conference Proceedings on Object-Oriented Programming Systems, Languages and Application. San Diego, California, United States ACM Press, p. 335.

Gray, L. (1988). Transitioning from structured analysis to object-oriented design. Proceedings of the fifth Washington Ada symposium on Ada. Tyson's Corner, Virginia, United States ACM Press. pp. 151-162.

Grunbacher, P. & Medvidovic N. (2001). Reconciling software requirements and architecture: The CBSP approach. Presented at Proc. 5th Int. Symposium on Requirements Eng.

Jalote, P. (1989, March). Functional Refinement and Nested Objects for Object-Oriented Design. IEEE Transactions on Software Enineering. (Vol. SE-15, pp. 264-270).

Morris, M. G., Speier, C. & Hoffer, J. A. (1996). The impact of experience on individual performance and workload differences using object-oriented and process-oriented systems analysis techniques. Proceedings of the 29th Hawaii International Conference on System Sciences (HICSS) Volume 2: Decision Support and Knowledge-Based Systems. IEEE Computer Society, pp. 232.

Nuseibeh, B. (2001). Weaving together requirements and architectures. Computer (Vol. 34, pp. 115-117).

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. & Lorensen, W. (1991). Object-Oriented Modeling and Design. Englewood Cliffs, NJ: Prince-Hall.

Rumpe, B., Schoenmakers, M. & Radermacher, A. (1999). UML + ROOM as a standard ADL? Proceedings of the 5th International Conference on Engineering of Complex Computer Systems IEEE Computer Society, p. 43.

Shaw, M. (1996). Software Architecture: Perspective on an Emerging Discipline. Prentice Hall.

Tan, H. B., Yang, Y. & Bian, L. (2006). Systematic transformation of functional analysis model into OO design and implementation. IEEE Trans. Softw. Eng. (Vol. 32, pp. 111-135).

Wang, E. Y. (2002). Formalizing and integrating the dynamic model for object-oriented modeling", IEEE Transactions on Software Enineering (Vol. 28 pp. 747-762).

Wolber, D. (1997). Reviving functional decomposition in object-oriented design. Journal of Object Oriented Programming (Vol. 10, pp. 31-38).