Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2003 Proceedings

Americas Conference on Information Systems (AMCIS)

December 2003

Improving Requirements Analysis in OO Software Development

Hee-Beng Tan
Nanyang Technological University

Weihong Li
Nanyang Technological University

Follow this and additional works at: http://aisel.aisnet.org/amcis2003

Recommended Citation

Tan, Hee-Beng and Li, Weihong, "Improving Requirements Analysis in OO Software Development" (2003). *AMCIS 2003 Proceedings*. 147.
http://aisel.aisnet.org/amcis2003/147

This material is brought to you by the Americas Conference on Information Systems (AMCIS) at AIS Electronic Library (AISeL). It has been accepted for inclusion in AMCIS 2003 Proceedings by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.

IMPROVING REQUIREMENTS ANALYSIS IN OO SOFTWARE DEVELOPMENT

Hee Beng Kuan Tan
School of Electrical and Electronic Engineering
Nanyang Technological University
ibktan@ntu.edu.sg

Weihong Li
School of Electrical and Electronic Engineering
Nanyang Technological University

Abstract

Today, most of the Object-Oriented (OO) software development methodologies analyze requirements in terms of objects and their interactions. However, requirements are more naturally to be specified functionally. Our informal survey with the Chinese software engineers in this region also strongly reflects this. This paper proposes an enhanced data flow diagram (DFD) called data flow net (DF net) that resolves the impeding mismatch between DFD based models and OO models. And, it proposes an approach that uses DF net to specify and analyze use-cases functionally during the requirements analysis stage. In the design stage, it maps the resulting DF nets systematically and precisely into OO structure. The proposed approach avoids the need of making decisions on objects and their interactions during the requirement analysis stage.

Keywords: OO software development, DFD, functional decomposition, requirement analysis

Introduction

Most of the OO software development methodologies proposed by Jacobson, Booch and Rumbaugh (1999), adopt OO analysis that identifies objects and their interactions from informal textual requirement description in the early stage of requirements analysis. Turner, Fuggetta and Lavazza (1999) has pointed out the structural mismatch between requirements and object-oriented designs. Practical experiences of Kop and Mayr (1998) show much difficulty and problem in identifying objects and their interactions from informal textual requirements descriptions. These violate the hypothesis of OO analysis proponents that OO model can be used naturally and easily to model requirements.

Ideally, no design decisions should be made during requirements capture and analysis. In addition to correctness problem, Jackson (1995) points out that such premature design decisions will constraint the designs unnecessarily. OO analysis avoids the mismatch between requirements and OO models by forcing analysts to make major design decisions on objects and their interactions in the early stage of requirements analysis through iteration. This, in fact, leads to a trial and error design process without proper analysis. It is a major problem that needs to be addressed.

This paper proposes an enhanced DFD model called **data flow net** (**DF net**) to model realizations of use-cases and functions in requirements analysis. DF net has resolved the impeding mismatch between DFD based models and OO models surveyed by Wieringa (1998) . Through the use of DF net, no decisions on objects and their interactions need to be made before the requirements are analyzed. The paper also proposes an approach that systematically and precisely maps analysis models represented in DF nets into OO designs.

The paper is organized as follows. Section 2 discusses DF net. Section 3 discusses the use of DF net to model realizations of use-cases and the transformation of resulting DF nets into OO designs. Section 4 reports our evaluation. Section 5 compares the proposed approach with related work. Section 6 concludes the paper.

Data Flow Net (DF Net)

A DF net is network of specially structured and augmented DFD processes interconnected by data flows. It associates with an implied formal semantics that defines the computational ordering of its processes. No additional effort is required to define the formal semantics.

General Characteristics of DF Net

In a DF net, an instance of a data flow is a set of values of all its attributes. All data flows in a DF net are un-buffered. At any time, there is at most one instance of each data flow.

A new instance of data flow always overrides its existing instance (if any). We call data flows between processes **inter-process data flows**. If a process in a DF net has an input inter-process data flow, then a selected such data flow is designated as the **main input data flow** of the process.

Let c and d be inter-process data flows in a DF net. If there is a path from c to d in the DF net such that when it passes through a process, it always passes through the main input data flow of the process, and all such maximal paths to d always contain c, then c is called an **ancestor data flow** of d and d is called a **descendant data flow** of c. In Figure 1, subject_id is an ancestor data flow of average, but mark is not an ancestor data flow of average. Note that (mark, 3, average) is not an above-mentioned path.

In a DF net, each non-main input inter-process data flow of a process must be a descendant data flow of the main input data flow of the process. A process may maintain its own internal data as variables called the **variables** of the process. Accessing and updating of data stores and interacting with external entities in a process are modeled as interfaces. This unifies data and processing in a process.

Processes in DF net are classified into two types: data flow process and sink process. This is to improve reusability through separating functions that acquire or compute data from functions that output and update information to external entities and data stores.

A process in a DF net has an implied formal semantics that is represented by a composition of primitive DFD processes each of which augmented with implied state transition diagrams to fully define the control flow of the process. The incorporation of control flow unifies the control of a process into the process. Together with the unification of data and processing in a process mentioned earlier, DF net resolves the data process separation and control separation in DFD based models that make the precise mapping from DFD based models to OO models not possible. Due to space constraint, in this paper, we shall discuss each type of process informally. The formal semantics will be discussed in a paper that will be submitted to a journal shortly.

Data Flow Process

A data flow process in a DF net specifies a function that acquires or computes data for the consumption of other processes. It produces output data flows through computation, retrieving records from data stores, or accepting inputs from external entities or a combination of them. It does not output or update information to data stores or external entities.

The structure of data flow process is inspired from the general structure of functions including aggregate functions. In brief, a data flow process is specified by a set of sub-specifications of the three types: instance sub-specification, descendant sub-specification and main sub-specification. An instance sub-specifications performs initialization with respect to a higher level data flow. A descendant sub-specification performs accumulation with respect to a lower level data flow. And, a main sub-specification performs the final processing for the production of output data flows. From other experience, a large majority of data flows processes can be specified by one sub-specification of each type or a main sub-specification alone. As such, it will not be difficult to understand.

A data flow process is represented by a bubble in a diagram as shown in Figure 1. Main input data flows are indicated by double solid arrowheads. Other inter-process data flows are indicated by single solid arrowhead. A data flow process P is specified by the following sub-specifications together:

(1) A collection of **instance sub-specifications**: An instance sub-specification is a sub-specification that is registered to the main input data flow or an ancestor data flow of the main input data flow, c. It is for the definition and redefinition of P's variables and P's output data flow attributes for the reference by other sub-specifications for P. It may reference to attributes of c and its ancestor data flows. It may also reference to variables defined in other instance sub-specifications that are registered to ancestor data flows of c.

(2) A collection of **descendant sub-specifications**: A descendant sub-specification is a sub-specification that is registered to a non-main input inter-process data flow e of P. It is for the definition and redefinition of P's variables and P's output data flow attributes for the reference by the main sub-specification (discussed shortly) and the subsequent execution of the descendant sub-specification itself. It may reference to attributes of data flow e, the main input data flow and its ancestor data flows. It may also reference to variables defined in the instance sub-specifications and variables defined in the previous execution of the descendant sub-specification itself.

(3) A **main sub-specification**: The main sub-specification produces the instances of all the P's output data flows. Each time, when a new set of values of the attributes of a P's output data flow has been finalized in the main sub-specification, an instance of the data flow is produced. In opposing with current approach, instances of the data flows produced are not put a data structure and returned or sent. Instead, each location in the sub-specification at which an instance of an output data flow is produced is marked. The location is called a **conceptual port** (**c-port**) of the data flow. C-port establishes a conceptual mechanism for other processes to reference to instances of an output data flow of P.

Each instance and descendant sub-specification for P is executed through implicit invocation. That is, it is executed until its completion upon the production of each instance of the data flow to which it is registered.

If the data flow process P has a main input data flow, then the execution of its main sub-specification is commenced when an instance of the main input data flow is produced and all the executions of the processes, such that their main input data flows are identical with P and some of their output data flows or their descendant data flows are non-main input inter-process data flows of P, have been completed. If P does not have a main input data flow, then the execution of P is commenced when it is activated externally. Once an instance of P's output data flow is produced, the execution of P's main sub-specification is paused immediately and it will only be resumed when the executions of all the sub-specifications of other processes that are resulted from the production of the instance directly or indirectly have been completed. The main sub-specification is executed in this to and fro manner until completion.

Sink Process

A sink process references its input data flows to output or update information to a data store or an external entity. The structure of a sink process is inspired from the general structure of outputting and updating information including printing a document with multi-level information. A sink process is specified by a few sink sub-specifications each of which outputs or updates a type of information with respect and reference to one of its input data flow.

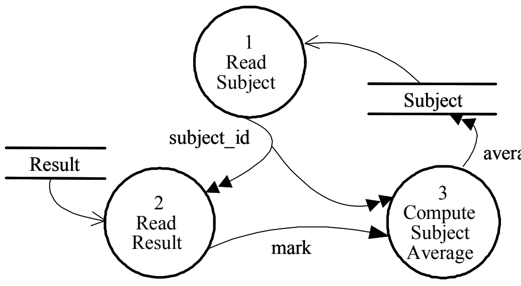
A sink process is not shown in the diagram as a bubble. It is represented by pointing each of its input data flows with double solid arrowheads to the data store updated or the information that is delivered to external entities. The process is specified by registering a sub-specification called a **sink sub-specification**, to each of its input data flows. A sink sub-specification may define, reference and redefine variables of the sink process. It may also reference to the data flow to which the sink sub-specification is registered and its ancestor data flows.

An Example of DF Net

Figure 1 shows a DF net to specify a use-case that processes the result records of each subject record to update the subject average.

The use-case is decomposed into four functions each of which is specified by a process:

(1) Process 1: Read each subject_id from the Subject data store.
(2) Process 2: Read each mark with a given subject_id from the Result data store.
(3) Process 3: Compute the average for the marks with a given subject_id.



```
##Process 1
  #main sub-specification
    Statement stSubject = conn.createStatement();
    ResultSet rsSubject = stSubject.executeQuery("Select Subject_ID from
      Subject");
    while (rsSubject.next()) {
      subject_id = rsSubject.getString("Subject_ID");
      //c-port of subject_id
    } rsSubject.close(); stSubject.close();
```

```
##Process 2
  #main sub-specification
    Statement stResult = conn.createStatement();
    ResultSet rsResult = stResult.executeQuery("Select Mark from Result  where
      Subject_ID='"+subject_id+"'");
    while (rsResult.next()) {
      mark = rsResult.getDouble("Mark");
      // c-port of mark
    } rsResutl.close(); stResult.close();

##Process 3
  #instance sub-specification registered to subject_id
    double total=0; int count=0;
  #descendant sub-specification registered to mark
    total+=mark; count++;
  #main sub-specification
    if (count>0) {
      average = total / count;
      // c-port of average
    }

##Sink Process
#sink sub specification registeried to  average
Statement stUpdate = conn.createStatement();
stUpdate.executeUpdate("Update Subject set Average="+average+"  where
  Subject_ID='"+subject_id+"'");
stUpdate.close();
```

Figure 1. Update Subject Average DF Net

(4) Sink Process: Modify the average of the record with a given subject_id to the average computed.

Note that Process 3 has attributes, total and count as its attributes. Other processes do not have any attribute.

As shown in the Java code in Figure 1, no procedure call or message passing is included in the specification for the DF net. The c-port of each inter-process data flow (the location at which an instance of the data flow is produced) is indicated by a line starting with "//". For example, the c-port for average is in the main sub-specification of Process 3. The interactions between processes are modeled conceptually by the inter-process data flows shown in the DF net.

The Proposed Requirements Analysis

In the proposed approach, once the use-cases are identified, for each use-case, instead of realizing the use-case in terms of objects and their interactions, we realize it through functional decomposition and specify it by a DF net. The use-case is realized and specified by defining each of the following functions that are required in the use-case by a process in the DF net:

(1) Accepting input from an external entity (e.g., user)
(2) Updating or retrieving records from a data store
(3) Delivering a type of information to external entity
(4) Producing a type of externally meaningful data

Once all the use-cases are specified in DF nets, we check and review them and transform the final DF nets into OO designs. In the transformation, we apply the following three mapping to define classes:

(1) Data-Store-Class-or-Association: This mapping is based on data stores used in all the DF nets. For each data store used in the DF nets, a class or an association is defined to model it.
(2) Process-Class: This mapping applies to processes that realize well-defined functions (usually computation-intensive). For each process or each group of related processes that realize a coherent function, a class is defined to model it. For example, Process 3 in the DF net shown in Figure 1 is mapped into Compute_Average class in Figure 2.

(3) External-Interaction-Class: For all the interactions between an external entity (e.g., user) and the processes in a DF net, a class is defined to model the interactions.

The attributes of a class or association are defined according to what the class or association models as follows:

(1) Data Store: If the class or association models a data store, the attributes of all the data flows from and to the data store, in all the DF nets, form the public attributes of the class or association respectively.
(2) Process: If the class models a process or a group of processes, then each variable of each process included is defined as a private attribute of the class. For example, the variables, total and count, of Process 3 in Figure 1 are defined as the private attributes of Compute_Average class in Figure 2. For each output data flow of each process, if an execution of the operation defined (discussed shortly) that produces the data flow, always produces at most one instance of the data flow, then each of its attributes is defined directly as an attribute of the class, otherwise, an attribute C of container type (e.g., array) is defined in the class and each attribute of the data flow is defined as an attribute of C. For the former case, if no instance might be produced in the execution, then include another attribute to indicate the instance existence in the class. For example, the attribute, average, of Process 3's output data flow is defined as a public attribute of Compute_Average class in Figure 2.
(3) External Interaction: If the class models the interactions between the processes and an external entity in a DF net, then all the attributes of the data flows from and to the external entity, form the public attributes of the class.

```
//Compute_Average.java
public class Compute_Average {
   public double average;
   private double total, count;
   public Compute_Average(){ }
   public void isp(){ total = 0;count = 0;}
   public void dsp(double value)  { total+= value; count++;}
   public void msp() {
      if (count>0){ average=total/count; } }
}


// Operation to update average
void Update_Average(){
   try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection conn=DriverManager.getConnection("jdbc:odbc:db");
        Statement stSubject=conn.createStatement();
        ResultSet rsSubject=stSubject.executeQuery("Select Subject_Id
             from Subject");
        while (rsSubject.next()) {
           subject_id = rsSubject.getString("Subject_Id");
           //c-port of subject_id
           Compute_Average ca=new Compute_Average();
           ca.isp();
           Statement stResult = conn.createStatement();
           ResultSet rsResult = stResult.executeQuery("Select Mark
             from Result where Subject_Id='"+subject_id+"'");
           while (rsResult.next()) {
              mark = rsResult.getDouble("Mark") ;
              //c-port of mark
              ca.dsp(mark);
            } rsResult.close(); stResult.close();
           ca.msp();
           //c-port of average
          Statement stUpdate = conn.createStatement();
          stUpdate.executeUpdate("Update Subject set Average=
               "+ca.average+" where Subject_Id='"+subject_id+"'");
          stUpdate.close();
          } rsSubject.close(); stSubject.close(); conn.close();
        } catch(Exception ex) {}
}
```

Figure 2. Compute Average Class and the Operation that Updates Subject Average

The operations of a class or association are defined from the sub-specifications for the processes that define the class or association respectively. They are defined according to what the class or association models as follows:

(1) Data Store: If the class or association models a data store, then each part in a sub-specification (usually the whole sub-specification) that accesses or updates records in the data store defines an operation in the class or association respectively. The attributes of inter-process data flows that are referenced by the part define the arguments of the operation. The attributes of the data flow retrieved from the data store by the part form the attributes of the return value of the operation.
(2) Process: If the class models a process or a group of processes, then either each sub-specification in a process included, defines an operation, or all the sub-specifications in a sub-group of processes combined together define an operation. In the latter case, the sub-group must have a process called its **root process** such that all the main input data flows of other processes in the sub-group are output data flows of the root process or their descendant data flows. For example, each sub-specification for Process 3 in Figure 1 is defined as an operation in Compute_Average class in Figure 2 (these operations are named isp, dsp and msp). More precisely, the operation is defined as follows:
   (a) Arguments: Attributes of each non-internal inter-process data flow in the processes included, which are referenced by sub-specifications for the processes, define the arguments in the similar way as defining class attributes except that in

this case, it depends on the number of instances of the data flow produced as a result of an instance of the main input data flow of the root process.

(b) Specification: A specification of the operation is defined as follows:

(i) If a main sub-specification of a process included produces multiple instances of an output data flow as a result of an instance of the main input data flow of the process (for single process case) or the root process respectively, transform the sub-specification to store the values of each instance attributes in the class attribute of container type defined earlier. If a sub-specification of a process included processes data stores or interacts with external entities, transform the sub-specification to use the corresponding operations defined for the processing and interaction respectively.

(ii) To define the specification for the operation, we traverse all the processes included in such a way that a process will only be traversed if all the processes included, which produce its input data flows, have been traversed and the c-ports of its input data flows are in the specification. For each inter-process data flow that is not produced by the processes included, the attributes of the data flow must have been defined as arguments for the operation. We emulate its c-ports by inserting a procedure in the beginning of the specification to access each attribute value in the arguments. When a process is traversed, if the process has a main input data flow, insert its main sub-specification in each c-port of the main input data flow in the specification, else insert the main sub-specification in the beginning of the specification. And, for each instance, descendant and sink sub-specification for the process, insert the sub-specification in each c-port of the data flow to which the sub-specification is registered. Sub-specifications that are inserted in the same c-port are inserted in such a way that main sub-specifications are always inserted after other sub-specifications. And, for those sub-specifications of the same type, they are always inserted according to the order of traversal.

(3) External Interaction: If the class models all the interactions with an external entity in a DF net, then each part of a sub-specification (usually the whole sub-specification) that deals with such interaction defines an operation in the class. The attributes of data flows that are referenced by the part define the arguments of the operation. The attributes of the data flows that are accepted by the part define the attributes of the return value of the operation.

Note that no classes will be defined for processes that are straightforward or can be realized directly using operations for classes modeling data stores or external interactions. These processes will be realized directly in realizations of DF nets.

Relationships between classes both modeling processes are defined from their main input and output data flows relationships. Relationships between classes modeling data stores and external interactions are defined from the relationships between data flows from and to them. Relationships between classes of other types are defined according to the relationships between the activations of the processes mapped and the data flows from and to the data stores and external interactions mapped.

At most only one class should be designed for processes that are identical. The classes designed so far serves as the lowest level classes. We may group the common attributes and operations of classes together by defining them in a super class. We may also group related classes together to form an assembly class through the use of aggregation.

Finally, we define an operation in a user interface class to realize each DF net that specifies a use-case. The operation is defined and specified in the similar way as we define and specify respectively an operation from all the sub-specifications in a sub-group of processes. For example, the DF net shown in Figure 1 is realized by the operation, Update_Average, in Figure 2. The use of DBMS and GUI are dealt with in the same way as existing OO methodologies.

Evaluation

To evaluate that DF net can be used correctly to specify use-cases and functions, we have developed a prototype system that supports the use of the proposed approach from requirements analysis to implementation. To evaluate the feasibility of using the proposed approach in real projects, we performed five case studies with the use of the prototype system. Due to space constraint, this section can only give an overview of our evaluation and discuss some experiences that we gained from the case studies.

Prototype System

The prototype system is developed in PC under Window NT 4.0. It is implemented in Java language based on Java Development Kits (JDK) 1.3 class library, and developed in an integrated Java development environment. The system has two main sub-systems: a graphical environment for the construction, editing and composition of DF nets; and a transformation sub-system for the transformation of DF nets into OO designs and programs.

In the prototype system, we can define new a DF net from scratch, from composition of existing DF nets or a combination of both. Comprehensive validation has been implemented to ensure that a DF net defined is syntactically correct. In the system, a process in a DF net is defined informally during the analysis stage. At the end of analysis, each sub-specification of each process is refined and specified in Java. At this stage, with the design decision given, DF nets can be automatically transformed into OO designs with the required OO programs generated. As an illustration, the programs produced through the prototype system for the compute average DF net shown in Figure 1 is shown in Figure 2.

Case Studies

The five case studies that we have performed are student result processing, library loan, automated meeting scheduler, order processing and sales analysis. Each case study has two students working on it: one student uses the proposed approach and the other student uses Unified Modeling Language (UML). All the students have same level of knowledge and experience. They have studied UML in details in one of the subject that they have taken. They studied the proposed approach through reading our initial paper. Next, we shall give an overview before proceeding to discuss some more specific experiences.

In the case studies, students who used the proposed approach constructed their OO designs more systematically and encountered lesser problems than students who used UML. Students who used UML had a tendency to go direct to program coding after they had identified the classes, attributes and operations. They were very reluctant to use interaction and activity diagrams for analysis though they have studied in details in a subject. They felt that it is a waste of time in using these diagrams. After much emphasis on the need in following UML properly, students did construct these diagrams for analysis. However, our checking of their outputs revealed that the designs and the codes do not correspond with these diagrams well. In contrast with the use of UML, the outputs from students who used the proposed approach had an exact match with the DF nets.

In terms of resulting designs, all the students produced similar results for classes modeling data stores and user interfaces. However, students who used UML were not able to design classes to model computations such as compute-average, compute-standard-deviation and meeting-scheduler. These computations are designed as operations of classes. This makes these computations less reusable. They also encountered much more difficulty in designing operations.

As a data flow process can have many instance and descendant sub-specifications, it may look difficult to understand from the outlook. When students read the paper on DF net, they felt so. However, from our experiences, data flow processes mostly have at most three sub-specifications: one instance sub-specification, one descendant sub-specification and one main sub-specification. None of the data flow processes in our case studies have more than three sub-specifications. The provision of many sub-specifications is mainly for completeness purposes.

Due to space constraint, we shall only discuss the details of automated meeting scheduler. The automated meeting scheduler problem is obtained from the website of CMU's Composable Systems Group (1995). Upon a request to schedule a meeting given with a preferred room, the scheduler retrieves the required information that is maintained in a database to schedule the meeting automatically. The required information can be classified into five types: meeting requirement, excluded periods, preferred periods and room booking. Meeting requirement comprises of a date range within which the meeting must be conducted, the members who are required to attend the meeting and time required (in hours). Members of the meeting can specify the periods that they cannot attend the meeting, called excluded periods. They can also specify the periods that they prefer the meeting to take place, called preferred periods. A period is represented by a triple that has a date, a time-start and a time-end. All the periods fall within the date range.

The diagram of the DF net that specifies the use-case that schedules meeting is shown in Figure 3. In the DF net, both the initial–calendar and updated-calendar data flows are defined as follows:

$$\{\text{date} + \{\text{all-member-availability} + \text{all-member-preference} + \text{preferred-room-availability}\}^{14}\}$$

In the data structure, for each half-an-hour period from 9.00 hr to 17.00 hr of each day (14 periods), excluding the period from 12.30pm to 13.30 hr, in the date range for the meeting, we have three data items, all-member-availability, all-member-preference and preferred-room-availability, to indicate the availability of all the members, preference for all the important members and availability of the preferred room respectively. These data items are arranged according to their time sequence. The excluded and preferred periods are defined as, meeting-id + member-id + {date + time-start + time-end}.

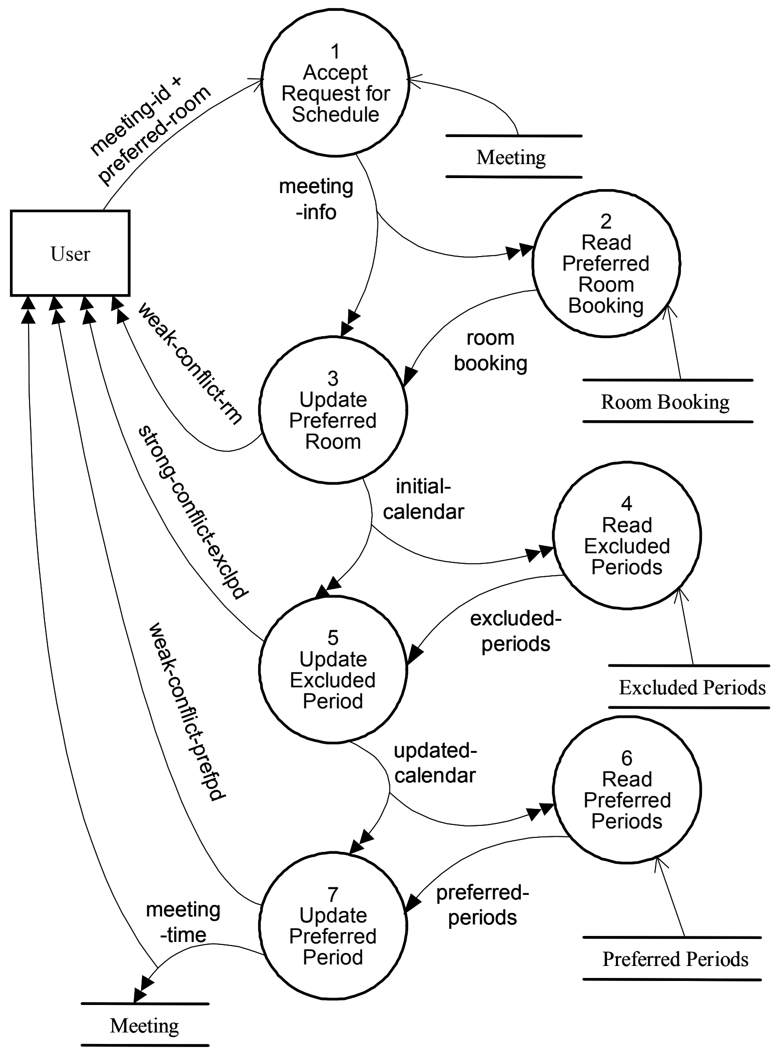


Figure 3. Schedule Meeting DF Net

Applying the process-class mapping to the group of processes that comprises, Process 3, 5 and 7, we constructed the class, Meeting Scheduler, as shown in Figure 4. The class is defined as follows:

(1) All the output data flows of the processes have single attribute that is referenced by some other processes not in the group. As such, they themselves form the public attributes of the class.
(2) The instance sub-specification (registered to meeting-info), the descendant sub-specification (registered to room-booking) and the main-sub-specification of Process 3 form the operations createCalendar, updRmAvailability and rptWConflitRm respectively. The operation updRmAvailability blocks out all the periods in the calendar during which the preferred room is booked by setting their preferred-room-availability to "no".
(3) The descendant sub-specification (registered to excluded-periods) and the main sub-specification of Process 5 form the operations updExcludedPd and rptSConflitExclPd respectively. The operation updExcludedPd blocks out all the periods in the calendar during which some members are not available by setting their all-member-availability to "no".
(4) The descendant sub-specification (registered to preferred-periods) and the main sub-specification of Process 7 form the operations updPreferredPd and rptMeetingTime respectively. The operation updPreferredPd updates all the periods in the calendar, which are not in preferred periods in an instance of preferred-periods data flow to "no".

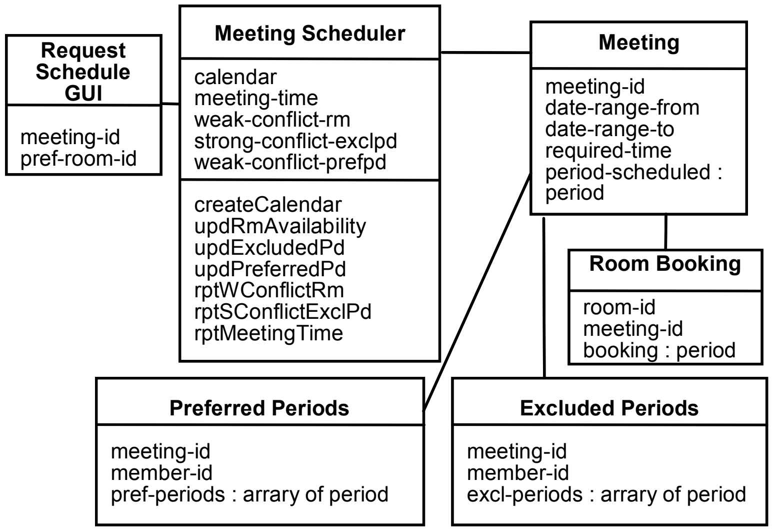


Figure 4. A Class Diagram for Automated Meeting Scheduler

Note that rptWConflitRm and rptSConflitExclPd check the calendar and produce a conflict data flow if there is no period in the calendar can be fixed for the meeting. The classes constructed for the data stores involved in Figure 3 are also shown in Figure 4. The operations required by these classes can be implemented directly through using SQL and are not shown in the figure.

In our sales analysis case study, the proposed approach designed a report writer class for printing a sales analysis report from a sink process that prints the report. Such report writer class can be easily generalized to print reports with many multi-levels of subtotals.

Comparison with Related Work

There is a gulf from requirements to OO model as Turner, Fuggetta, Lavazza and Wolf (1999) had pointed out. Current OO software development methodologies simply avoid the gulf by adopting OO analysis that expresses requirements in terms of objects and their interactions before analyzing the requirements. Expressing requirements in terms of objects and their interactions is a major design process that makes important OO design decisions. As such, OO analysis does not provide a systematic and smooth transformation to deal with the gulf from requirements to OO deigns. It simply avoids the gulf by forcing analysts to make major OO design decisions before analyzing the requirements. Indeed, this leads to trial and error design process without proper analysis. The proposed approach allows requirements at use-case level and below to be modeled more naturally and directly in DF nets through functional decomposition without making the above-mentioned design decisions. In opposing to OO analysis, the proposed approach provides a bridge for the gulf from requirements to OO designs through a precise and systematic mapping that maps requirements in DF nets into OO designs.

Due to the data process separation and control separation mentioned by Wieringa (1998) in DFD based models, the approaches that use DFD based models for requirements analysis in OO software development, which is proposed by Alabiso (1988) and Rumbaugh (1991), cannot systematically and precisely map analysis models into OO designs. The mapping of data stores and data flows to classes and processes to operations discussed in these approaches are at most heuristic rules. They cannot define the operations and their specifications precisely.

The proposed approach shares the use of functional decomposition in representing requirements with the above-mentioned DFD based approaches. In opposing to these approaches, the mapping to OO designs in the proposed approach is precise and systematic, and it includes the precise design of the operations including their arguments and specifications. This is possible as DF net unifies the processing, data and control of a process.

Conclusion

We have proposed DF net for the realization of functions and use-cases. DF net is based on DFD and has all the benefits of DFD. In opposing to DFD that has the problem of data process separation and control separation, DF net unifies processing, data and control of a process. As a result, DF net resolves the incompatibility of DFD based models with OO decomposition principle. And, DF nets can be mapped systematically and precisely into OO designs. We have discussed the mapping.

The only disadvantage of the proposed approach is that when we have too many instance and descendant sub-specifications for a process, it might be difficult to understand a process. However, from our experience, this will be rare. Most processes have at most one instance sub-specification and one descendant sub-specification.

As DF nets can be mapped systematically and precisely into OO designs, we believe that once the mapping is decided, it is feasible to construct collaboration, sequence and statechart diagrams automatically to visualize the resulting OO designs for easier understanding. As DF net has a formal semantics, much automated analysis model checking can also be performed on requirements specified in DF nets. We will be exploring on these areas in the near future.

References

Alabiso, B., "Transforming of data flow analysis models to object oriented design," in Proceedings of OOPSLA, 1988, pp. 335-353.

Composable Systems Group, Carnegie Mellon University, "Calendar Scheduler," 1995, **http://www2.cs.cmu.edu/~Compose/html/ModProb/CS.html**

Demarco, T., Structured Analysis and System Specification, Yourdon Press/Prentice-Hall, Englewood Cliffs, NJ, 1978.

Jacobson, I., Booch, G., and Rumbaugh, J., The Unified Software Development Process, Addison-Wesley, Boston, MA, 1999.

Hatley, D. J., and Pirbhai, I., A., Strategies for Real-Time System Specification, Dorset House Publishing, New York, NJ, 1988.

Jackson, M., Software Requirements and Specification – a lexicon of practise, principles and prejudices, ACM Press, Addison-Wesley, Boston, MA, 1995, pp. 98-100.

Kop, C., and Mayr, H. C., "Conceptual predesign: bridging the gap between requirements and conceptual design," in Proc. 3rd Int. Conf. on Requirements Eng., 1998, pp. 90-98.

Rumbaugh, J., Object-Oriented Modeling and Design, Prentice-Hall, Englewood Cliffs, NJ, 1991.

Turner, C. R., Fuggetta, A., Lavazza, L., and Wolf, A. L., "A conceptual basis for feature engineering," in Journal of Systems and Software, 49, 1999, pp 3-15.

Wieringa, R., "A survey of structured and object-oriented software specification methods and techniques," ACM Computing Surveys, vol. 30, no. 4, pp. 459-527, Dec. 1998.