ISSN: 1545-679X

Information Systems Education Journal

Volume 2, Number 28 **http://isedj.org/2/28/** **May 6, 2004**

In this issue:

Intention-Directed Modeling Technique

Matthew H. S. Kuofie
Illinois State University - The University of Michigan
Normal, IL 61790, USA - Ann Arbor, MI 48109, USA

Christian C. Wagner
Oakland University
Rochester, MI 48309, USA

Abstract: This paper addresses a new object oriented analysis and design technique; the technique, intention-directed modeling (IDM), is based on capturing the abstract intentions of the problem domain specialist who performs analysis and design for a software system. The abstract intentions are represented as knowledge representation schema in what we term the Intentional Class Model (ICM). The ICM has classes that are classified into specification class model (SCM), platform class model (PCM), and engineering class model (ECM). The benefits that accrue from the use of the IDM are high quality software, maintainable object-oriented analysis requirement and design specifications, effort reduction in analysis and design, and reuse of specification.

Keywords: knowledge representation, intentions, problem domain specialist, object oriented analysis and design, requirements specification, design specification, software engineering

---

Recommended Citation: Kuofie and Wagner (2004). Intention-Directed Modeling Technique. *Information Systems Education Journal, 2* (28). http://isedj.org/2/28/. ISSN: 1545-679X. (Also appears in *The Proceedings of ISECON 2003:* §2423. ISSN: 1542-7382.)

This issue is on the Internet at **http://isedj.org/2/28/**

The **Information Systems Education Journal** (ISEDJ) is a peer-reviewed academic journal published by the Education Special Interest Group (EDSIG) of the Association of Information Technology Professionals (AITP, Chicago, Illinois). • ISSN: 1545-679X. • First issue: 8 Sep 2003. • Title: Information Systems Education Journal. Variants: IS Education Journal; ISEDJ. • Physical format: online. • Publishing frequency: irregular; as each article is approved, it is published immediately and constitutes a complete separate issue of the current volume. • Single issue price: free. • Subscription address: subscribe@isedj.org. • Subscription price: free. • Electronic access: http://isedj.org/ • Contact person: Don Colton (editor@isedj.org)

The Information Systems Education Conference (ISECON) solicits and presents each year papers on topics of interest to IS Educators. Peer-reviewed papers are submitted to this journal.

EDSIG activities include the publication of ISEDJ, the organization and execution of the annual ISECON conference held each fall, the publication of the Journal of Information Systems Education (JISE), and the designation and honoring of an IS Educator of the Year. • The Foundation for Information Technology Education has been the key sponsor of ISECON over the years. • The Association for Information Technology Professionals (AITP) provides the corporate umbrella under which EDSIG operates.

Intention-Directed Modeling Technique

Matthew H. S. Kuofie
School of Information Technology, Illinois State University
Normal, IL 61790, USA
mhkuofie@msn.com
mkuofie@ilstu.edu

Matthew H. S. Kuofie is also affiliated with the
School of Information, The University of Michigan
Ann Arbor, MI 48109-1092, USA
mhkuofie@umich.edu

Christian C. Wagner
School of Engineering and Computer Science, Oakland University,
Rochester, MI 48309, USA
wagner@oakland.edu

Abstract

This paper addresses a new object oriented analysis and design technique; the technique, intention-directed modeling (IDM), is based on capturing the abstract intentions of the problem domain specialist who performs analysis and design for a software system. The abstract intentions are represented as knowledge representation schema in what we term the Intentional Class Model (ICM). The ICM has classes that are classified into specification class model (SCM), platform class model (PCM), and engineering class model (ECM). The benefits that accrue from the use of the IDM are high quality software, maintainable object-oriented analysis requirement and design specifications, effort reduction in analysis and design, and reuse of specification.

Keywords: knowledge representation, intentions, problem domain specialist, object oriented analysis and design, requirements specification, design specification, software engineering

1. INTRODUCTION

Some software projects are never completed on time, within budget or meet customers' expectations—quality wise. One of the main reasons software projects fail is that analysts and designers are almost certainly unfamiliar with the problem domain and will have to make decisions about the details of the software to be engineered. Features are added that are not needed, and some needed features are skipped (Rawsthorne and Goodwin, 1999).

Typically, the analyst obtains requirements from the customer who is a problem domain expert of the software to be engineered. It requires tedious effort on the part of an analyst to gather requirements from a customer, a problem domain specialist of the software to be developed. The reasons that make analysis difficult include miscommunication, requirements change, incomplete requirements, and time constraints (Kuofie, 1999; Pressman, 2001; Schach, 2002; Sommerville, 1992). Miscommunication often occurs between the analyst and the domain expert. (Miscommunication could lead

to inaccurate, incomplete, and unclear requirements.) This is because the analyst is unfamiliar with the problem domain. The domain expert often spends lots of time to put his or her requirements across to the analyst (Kuofie, 1999; Rawsthorne and Goodwin, 1999). Changes in requirements may become necessary due to technology change. Incomplete requirements may also be due to the analyst not asking the right questions. Time constraints may also force the analyst to rush to deliver.

Designers of software products use the software requirements specifications documented by analysts to create design specifications (Kuofie, 1999; Rawsthorne and Goodwin, 1999). Designers, too, are almost certainly unfamiliar with the problem domain (Rawsthorne and Goodwin 1999). Designers sometimes misinterpret the software requirements specifications (Kuofie, 1999; Rawsthorne and Goodwin, 1999). Therefore, it requires tedious effort on the part of the designers to accurately design the expected software. Unfortunately, the tedious efforts do not lead to high quality software.

To reduce the problems, this paper describes an intention-directed modeling (IDM) technique for analysis and design of a software system to be developed; the technique is based on object-oriented technology and is a component of the Intentional Directed Software Engineering Methodology (ID-SEM) (Kuofie, 1999). Here the specialist (also referred in this paper as either domain expert or domain specialist) is a problem domain person who understands the specific problem that has to be solved by means of new software. In addition, the domain expert is expected to understand object-oriented technology concepts. The domain expert, instead of the typical software analyst and designer, performs the analysis and the design. The domain expert can spend his or her time to develop complete, accurate and clear software requirements and design specifications. This domain expert can validate and update, if any, his or her intentions in the specifications. Yes, the expert has much better understanding of the problem domain than the typical analyst and designer (Kuofie, 1999; Schach, 2002; Rawsthorne and Goodwin, 1999). The domain expert will not have to spend lots of time to put his or her intentions across to analysts and designers. Therefore, it is reasonable to have the domain expert do the analysis and design right the first time (Kuofie, 1999).

Abstract intentions are presented using a knowledge representation schema: classes in the Intentional Class Model (ICM). Hence, object-oriented concepts involving analysis and design are applied (Booch et al., 1999; Brumbaugh, 1994; Jezequel, 1996; Page-Jones, 1995).

The ICM is an object model in the sense of COM or CORBA. It includes the definition of many classes with properties, methods, and events that are used to represent, store, and process the intentions of the domain specialist. As would be true for any class model, the actual intentions of the domain specialist are stored as instances of the ICM classes. The ICM has three components: the specification class model (SCM), the platform class model (PCM), and the engineering class model (ECM).

The rest of the paper is laid out as follows: The ICM is described in Section 2. In Section 3, we describe the Specification Class Model. Section 4 describes the Platform Class Model. In Section 5, we describe the Engineering Class Model. Section 6 describes the benefits of the intention-directed modeling technique; Section 6 also concludes this paper.

2. INTENTIONAL CLASS MODEL

Typically, the domain expert knows the following: objects, properties, methods, events, business processes, and algorithms.

What the domain expert lacks is the software. In reality, the domain expert lacks the computer knowledge to generate the software needed to solve the domain problem. This paper does not provide all that the specialist's needs; instead, it provides an intention-directed modeling technique, ICM, to perform the analysis and design aspect of engineering software.

Figure 1 shows the graphical representation of the ICM.

The ICM is made up of three components: the specification class model, the platform class model, and the engineering class model.

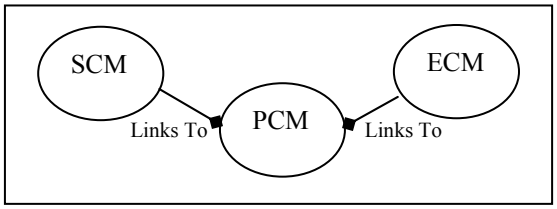

Figure 1. Intentional Class Model

We will start by describing some of the assumptions and constraints that we make pertaining to the ICM; we clarify some of the concepts by using the Microsoft Visual Basic 6 (VB6).

Three-tier Architecture
Generally, an application can be modeled, designed, into an n-tier architecture. However, this paper deals with a three-tier architecture, which has presentation (P), manipulation (M), and storage (S) features. The ICM considers the SCM and PCM as having three architectural layers: the presentation, manipulation, and storage layers. The presentation layer contains intentions that handle visibility or user-interface issues. The manipulation layer addresses computation issues. The storage layer addresses persistent storage issues.

Transfer
Given the three-tired architecture, issues arise concerning the transfer of information between layers. This transfer feature allows for transfer of data from the presentation layer to the manipulation layer and vice versa (P←→M). Similarly, data can be transferred from manipulation layer to storage layer and vice versa (M←→S).

Manipulate
The manipulate feature involves transformation, summarization, and extraction of data from the manipulation layer to the same manipulation layer (M←→M).

A transformation can transform existing data to a new data, for example, transforming length and width data by multiplying them to get an area. In addition, transformation can be achieved by sorting or searching data.

Summarization of statistical intention can be computed for statistical values, such as minimum, maximum, average, standard deviation, and the total of a given collection of data.

Extraction can be performed both in virtual and real basis. In virtual extraction, a subset data is not extracted from the original collection, but instead the subset is summarized or transformed as it is identified to belong to the subset. However, by real extraction, we imply that a subset of collection of data is actually extracted into a subset and the extracted subset is then summarized or transformed.

We will turn to the description of each of the ICM class models: the SCM, the PCM and the ECM. As we describe each class model, we will include the following aspect: (1) the purpose, (2) the major classes, (3) relationships within the class, (4) relationships outside the class and (5) sample entry of a class.

3. SPECIFICATION CLASS MODEL

We will discuss the specification class model in this section. Figure 2 shows the specification class model: the three layers and the relationships.

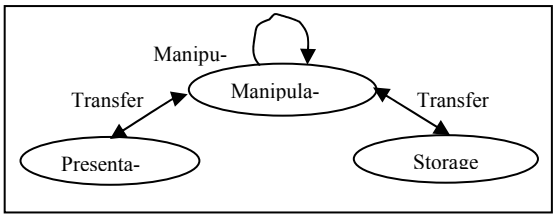

Figure 2. Specification Class Model

Purpose
The specification class model provides abstract classes that are used to represent abstract intentions that serve as the specification for a new software system. The specialist provides real specification intentions.

Table 1: List of Major SCM classes

Presentation	Manipulation	Storage
SCMControl SCMForm SCMWebpage	SCMClass SCMClassEv-entSCMClass-Method SCMClassProp-erty	SCMDatabase SCMDatabase-Table SCMDatabase-TableFields SCMFile

Major Classes
Table 1 is the list of major SCM classes. We

have named the classes with the prefix "SCM" in order to identify the classes to the SCM. The classes are classified based on the P, M, and S layers.

An SCM class has neither method nor event.

The definition of SCMClass, is as follows:

Name: SCMClass class
Properties: Public Name As String Public Description As String Public ClassGroup As String Public Properties As New SCMClassProperties Public Methods As New SCMClassMethods Public Events As New SCMClassEvents Public SubClasses As New SCMClasses Public Parts As New SCMClasses Public SOMTranfers As New SCMTransfers Public SOMFrms As New SCMForms Public SOMDbases As New SCMDatabases
Methods:
Events:

The definition of the SCMClass is a knowledge representation scheme of the abstract intentions in a class that the domain expert wants to see. The properties are primitive abstract intentions. In fact, the properties are the domain expert's intentions of what they should be to this class. For example, the "Name" property is the domain expert's intention of what the name of this class, to be generated in an application, should be. Each property has public access type, property name and its data type. For example, one of the intentions of the SCMClass is "Description." The Description data type is string; therefore, "Description" is the domain specialist's intention of what the description of this Class manipulation should be.

There are some properties, which are instances of other SCM classes. For example, the SCMForm, the SCMClassProperty, the SCMClassEvent and the SCMClassMethod are properties of the SCMClass.

A class is associated with a corresponding class collection for manipulating, such as adding, objects of the class. For example, SCMClasses is a class collection for manipulating SCMClass instances. SOMTransfers hold transfer intentions that the specialist specifies to be used by the SCMClass' object.

SCMClassEvent, a major SCM class, is defined as:

Name: SCMClassEvent class
Properties: Public Name As String Public AccessType As String Public SOMArguments As New SCMArguments
Methods:
Events:

SCMClassEvent has Name, AccessType, and SOMArguments. The SOMArguments is an instance of SCMArguments; it's responsible for manipulation of SCMArgument objects.

Now, we turn to SCMClassProperty class, which is defined as follows:

Name: SCMClassProperty class
Properties: Public Name As String Public Description As String Public DataType As String Public DataArity As String Public ObjectType As String Public IsMultiValued As Boolean Public MinDefined As Boolean Public MaxDefined As Boolean Public Minimum As Double Public Maximum As Double Public DefDefined As Boolean Public DefaultValue As Variant
Methods:
Events:

SCMClassProperty has "DataType" property as string; therefore, it could be assigned "integer," or "string" intention. The "Minimum" and "Maximum" specify the minimum and maximum-- constraints-- respective characters long that the domain expert intends to allow for the property "Name."

We will address SCMForm class, which is in the presentation layer; the current definition of SCMForm is as follows:

Name: SCMForm class
Properties: Public Name As String ' Name Public FType As String ' Type Public Description As String ' Description Public Top As Single ' Position Y Public Left As Single ' Position X Public Width As Single ' Extent X Public Height As Single ' Extent Y Public Caption As String ' Caption Public MDIChild As Boolean ' MDI Flag Public SOMControls As New SCMControls ' controls on form Public UsesClasses As Classes ' classes used Public SOMTranfers As New SCMTransfers ' transfers used
Methods: (none)
Events: (none)

The "Height" property is the domain expert's intention of what the height of this form interface should be. The "Width" is, similarly, the intentional width of the form to be built in the application. However, the "SOMControls" property is itself an instance of another class, the SCMControls class. An SOMControls object holds a collection of control objects, such as menus, command buttons, and list boxes. The "SOMTransfer" property of the SCMForm class is an object of the SCMTransfer class. It holds the "Transfer" intentions of the objects of manipulation classes that can be accessed in the SCMForm.

Name: SCMControl class
Properties: Public Name As String Public Ctype As String ' control type Public Caption As String Public Left As Single Public Top As Single Public Width As Single Public Height As Single Public Index As Single
Methods:
Events:

SCMControl intentions are controls that are displayed on a form to enable intentions to be made possible in an application. The "Name" property is an intention that the expert has to specify.

The definition of SCMDatabase, which is in the storage layer, is as follows:

Name: SCMDatabase class
Properties: Public Name As String ' Database name Public Description As String Public SOMTables As New SCMDatabaseTables Public SOMViews As New SCMDatabaseViews Public SOMQueries As New SCMDatabaseQueries Public SOMTriggers As New SCMDatabaseTriggers Public SOMProcs As New SCMDatabaseProcedures Public SOMUsers As New SCMUsers
Methods:
Events:

The properties inherent in the SCMDatabase object are the kinds you would expect to see in any database design. The properties of this SCMDatabase are Name, Description, Users and SCMDatabaseTables. It seems logically right to describe SCMDatabaseTable class; it's defined as follows:

Name: DatabaseTable class
Properties: Public Name As String 'table name Public Description As String Public SOMDBTFields As New DatabaseTableFields
Methods:
Events:

This SCMDatabaseTable has Name, Description and SCMDatabaseTableFields as current intentions. The Name intention is of string data type.

SCMDatabaseTableField is defined as follows:

Name: SCMDatabaseTableField class
Properties: Public Name As String Public Description As String Public Size As Integer ' length of field Public DataType As String Public Required As String Public IsRequired As Boolean
Methods:
Events:

SCMDatabaseTableField has Name and Description as two of its properties. The data type of "Size" is integer. We describe one other SCM class, the SCMPlatform class,

which is defined as follows:

Name: SCMPlatform class
Properties: Public Name As String ' new system name Public Description As String Public ApplicationType As String 'Exe,dll,ocx Public ProgrammingLanguage As String 'VB6,C++ Public HardWare As String ' PC,Mainframe, Public OperatingSystem As String Public DatabaseSystemType As String
Methods:
Events:

SCMPlatform is a very unique SCM class; it is not in any of the three layers P, M or S. SCMPlatform indicates abstract platform intentions, such as programming language, operating system and hardware platform. The problem domain specialist has to provide the intentions as requirements specification.

Let us march on to the relations within the SCM.

SCM class Name	Has parts relation to other SCM classes
SCM Class	SCMClassProperty SCMClassMethod SCMClassEvent SCMTransfer SCMForm SCMDatabase
SCMDatabaseTable	DatabaseTableField

Table 2: Within SCM relations

Within SCM relations
There are 1- n, where n ≥ 1, relationship (Grossman, 1990). We define a 1-n relationship as one SCM class referencing, as properties or intentions, n other SCM classes. For example, Table 2 shows that the SCMDatabaseTable has a 1- 1 relationship to SCMDatabaseTableField; the SCMDatabaseTable definition shows that the SCMDatabaseTableField instance, SOMDBTField, is the property of SCMDatabaseTable, and SCMDatabaseTableField is a class within SCM.

We call this type of relationship a within relationship since the classes occur within the same ICM class model, the SCM in this case. Table 2 lists some of the within relationships.

Outside SCM relations
An SCM class is linked to some classes in the PCM classes. The type of linkage is referred to as outside relationship. However, we will defer this discussion until we describe the PCM.

Sample Entry of SCM
The domain expert can specify the requirement or the design intention for a class property as an instance of SCMClassProperty, SOMClassProperty, as follows:

Name: SOMClassProperty object
Properties: Name="FirstName"^<- specialist'sintention Description = "first name"`<- designer's `intention DataType="String" '<- specialist's `intention DataArity="Single" '<-specialist's `intention ObjectType = "" '<- ID-SET defined IsMultiValued =FALSE'<- ID-SET defined MinDefined = True '<- ID-SET defined MaxDefined = True '<- ID-SET defined Minimum = 1 '<- specialist's intention Maximum = 25 '<- specialist's intention DefDefined = True '<- ID-SET defined DefaultValue = 0 '<- ID-SET defined
Methods:
Events:

The specialist provides intentions; for example, Maximum first name length as 25 characters long. The intentions that the specialist does not provide are set to default value by the Intention-Directed Software Engineering tool (ID-SET, a CASE tool) (Kuofie, 1999). For example, ID-SET supplied the intended intention for MaxDefined as True.

4. PLATFORM CLASS MODEL

We will discuss the platform class model in this section.

Figure 3 shows the platform class model: the three layers and the relationships.

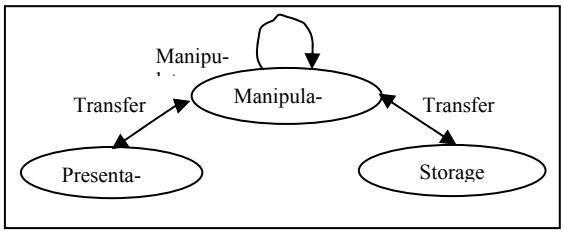

Figure 3 Platform Class Model

Purpose

The PCM is the link between the abstract intentions of the domain expert and the actual implementation of these intentions in a particular language for a particular hardware platform. As such, it mirrors the information contained within the classes of the SCM. In addition, PCM include methods for the generation of actual code in the platform for which the PCM was created. So, where the SCM had only properties for the classes it contained (no specific generation being done), the PCM provides properties to mirror many of those from the SCM plus the methods needed to generate actual code, documentation, help files, training guides, and the like; we are interested in the system documentation—requirements specification.

Of special interest in generation of code for the three-tiered architecture is the implementation of algorithms. Algorithms defined abstractly in the SCM are defined more specifically here in the PCM. But even in the PCM, the algorithms are still abstract until instances are created and the generators are called to create actual lines of code for the application. The differences between these three levels of algorithm specification are shown by the following three statements:

1. General (SCM): Case of X:
2. VB specific (PCM): Select Case X
3. Appspecific(generated): Select Case UserCode

Major Classes

As was true for the SCM, the PCM classes can be placed into three layers: the presentation, the manipulation and the storage layers. The major classes are listed in Table 3.

The PCM classes in Table 3 are named using the prefix VB to associate them to VB6 as the choice of programming language.

Presentation	Manipulation	Storage
VBControl VBForm VBFormMDI	VBClass VBClassEvent VBClass-Method VBClassProperty VBModule	VBDatabase VBDatabase-Table VBDatabase-TableFields VBFile

Table 3: List of Major PCM Classes

Now, we define some of the major classes. The VBClass is defined as follows:

Name: VBClass class
Properties: Public Project As VBProject Public Name As String Public Description As String Public Instancing As String Public AccessType As String Public SOMClass As SCMclass Public SOMForm As SCMForm Public vbClsProps As New VBClassProperties Public vbClsMeths As New VBClassMethods Public vbClsEvents As New VBClassEvents
Methods: Public Function CopyObject() As VBClass Public Sub GenCode() Private Sub GenerateMtoPTransfer(…) Private Sub GenerateMtoSTransfer(…) Private Sub GeneratePtoMTransfer(…) Private Sub GenerateStoMTransfer(…) Private Sub GenerateMtoMTransform(…)
Events:

The VBClass class of the PCM has the GenCode( ) method. This method generates an actual class in a new software application. (The properties of this class are specified by the specialists in SOMClass, which is an instance of SCMClass.) Of special note are the many sub-parts of the GenCode method that generate a wide variety of transfers within and between the three layers of P, M, and S. Also note the MtoMTransform that is the PCM access to generate the algorithms in the SCM.

The definition of the VBDatabase is defined as follows:

Name: VBDatabase class
Properties: Public IsDAO as Boolean Public DAOVersion as String Public IsADO as Boolean Public ADOVersion as String Public SOMDatabase As SCMDatabase
Methods: Public Function CopyObject() As VBDatabase Public Sub GenCode() Public Sub GenDatabase()
Events:

This VBDatabase class has a GenCode method which references the SCMDatabase instance (SOMDatabase) and generates actual DDL to define an SQL relational database and sets up project definitions to include appropriate Microsoft DLLs (dynamic link libraries), and the like.

Name: VBModule class
Properties: Public Project As VBProject Public Name As String Public Description As String 'link IOMProcess to VBModule Public Processes As New SCMProcesses Public vbDecs As New VBDeclarations Public VBSubs As New VBSubroutines Public VBFuncs As New VBFunctions 'link IOM Databases to VBModule Public IOMDBbases As SCMDatabases
Methods: Public Function CopyObject() As VBModule Public Sub GenCode()
Events:

The VBModule has GenCode that is used to generate file, Module that contains subroutines and functions; the VBModule falls in the manipulation layer.

Within PCM relations

The following is a list of some of the PCM classes that have properties that are instances of other PCM classes.

PCM Class Name	Related to other PCM classes
VBClass	VBClassProperties VBClassMethods VBClassEvents
VBForm	VBControl VBDeclaration VBFunction VBMenu VBSubroutine

The VBClass has a 1-3 within relationship to VBProperties, VBMethods, and VBClassEvents.

Outside PCM Relations

A definition of a PCM class, in most cases, includes some SCM classes as properties. For example, the VBClass has SCMClass and SCMForm as properties. This type of relationship is called the outside relationship.

It must have been realized that after stripping off "SCM" and "VB" from the SCM and the VB class names, we are left with some partial names that do match. For example, if we strip off "SCM" and "VB" from SCMClass and VBClass, we get "Class" in each case. However, some of the VB classes do not have corresponding named classes in the SCM and vice versa. For example, VBModule, VBMenu and VBFormMDI, do not have their identical named counterparts, SCMModule, SCMMenu and SCMFormMDI because we consider that these intentions "Module," "menu," and "FormMDI" do not generally occur in programming languages. (MDI denotes multiple document interface). However, "Module," "Menu" and "formMDI" occur in VB6. Similarly, SCMDatabaseTable and SCMDatabseTableField do not have matching counterparts in VB classes.

In Section 3, we deferred description of the outside SCM relationship until after the description of the PCM; it is appropriate to do that now. The definitions of VBForm and VBClass references SCMForm; therefore, SCMForm has an outside relationship to those VBForm and VBClass.

Sample Entry of PCM

We assume that the specialist specified that the programming language for the software system as VB6. Then the VBClass is used;

therefore the GenCode( ) method will reference SOMClass to generate code.

5. ENGINEERING CLASS MODEL

We will discuss the engineering class model in this section.

Purpose
The ECM addresses intentions that the specialist can specify as the engineering processes for engineering software.

Major Classes
All the ECM classes, except the ECMProcess, do not have methods or events.

The ECMAnalysis and the ECMDesign classes are defined as follows:

Name: ECMAnalysis class and ECM-Design class
Properties: Public CheckCompleteness As Boolean ' Do completeness checks Public CheckManipulationUsage As Boolean 'All classes and modules used somewhere Public CheckPresentationUsage As Boolean 'all forms and WebPages used 'somewhere Public CheckStorageUsage As Boolean ' all databases and files used somewhere Public CheckTransferUsage As Boolean ' all transfer used Public CheckTransformUsage As Boolean ' all transforms 'used'
Methods:
Events:

The ECMDesign class has the same properties as the ECMAnalysis because analysis and design are implemented concurrently during specification phase of the software life cycle indicated in (Kuofie, 1999). The specialist specifies intentions. The intentions can include having to check if the requirements for a new application are complete, to check if all classes for the new application are used for manipulations, to check if all forms for the new application are used for presentation purposes, and to check if all database tables are used in the new application

Name: ECMCoding class
Properties: Public UseNamingStandard As Boolean Public UseCommentingStandard As Boolean Public UseCodingStandard AS Boolean
Methods:
Events:

The ECMCoding class specifies intentions for coding of new application. The intentions include conformance to naming, commenting, and coding standards.

The ECMDocumantation class is defined as follows:

Name: ECMDocumentation class
Properties: Public GenerateUserManual As Boolean ' User Manual Public GenerateSystemManual As Boolean ' system document Public GenerateTrainingGuide As Boolean Public GenerateHTMLHelp As Boolean
Methods:
Events:

The documentation phase spells out the type of documentation to generate. A system manual is one such document; the specialist specifies this intention "GenerateSystem-Manual" as a Boolean –True or False -- data type.

The ECMTesting is defined as follows:

Name: ECMTesting class
Properties: Public GenerateUnitTest As Boolean Public RunUnitTestsAtStartup As Boolean Public LogResultOfUnitTestsAtStartUp As Boolean
Methods:
Events:

ECMTesting has properties or intentions, such as generating unit test, GenerateUnit-Test. The GenerateUnitTest has "Boolean" data type, which enable the specialist to indicate true or false as the intention.

Next, the ECMProcess is defined as follows:

Name: ECMProcess class
Properties: Public EOMAnalyses As New ECMAnalyses Public EOMDesigns As New ECMDesigns Public EOMDeliveries As New ECMDeliveries Public EOMDocumentations As New ECMDocumentations Public EOMCodings As New ECMCodings Public EOMTestings As New ECMTestings Public Platform as String Public Language as String
Methods: Public Function AddECMAnalysis() As ECMAnalysis Public Function AddECMDesign() As ECMDesign
Events:

The ECMProcess class properties are instances of the other ECM classes, such as ECMCoding, and ECMTesting. The ECMProcess has a method that sets VB as the default programming language of the program to be generated.

Within ECM relations

The ECMProcess has within relationship to all the other ECM classes. The other ECM classes do not have within relationships.

Outside ECM relations

The ECMProcess is the only ECM class directly related to VBProject, which is used as a driver to generate a VB6 application.

Sample Entry of ECM

As a sample entry, the domain specialist can specify the requirement or the design intention for testing as an instance of ECMTesting, EOMTesting, as follows:

Name: EOMTesting object
Properties: GenerateUnitTest = True RunUnitTestsAtStartup = True LogResultOfUnitTestsAtStartUp = True
Methods:
Events:

In the sample entry above, the specialist specifies that unit tests are to be automatically generated. In addition, the unit tests should be run at Startup of the application and the result the unit tests should be logged.

The instances of the SCM, ECM, and PCM are stored in SOM, EOM, and POM respectively. The SOM, POM and EOM fall under one umbrella called intentional object model (IOM); the IOM therefore contains all the analysis and design specifications that the specialist specifies. The details of IOM will be described in a future article.

6. BENEFITS AND CONCLUSIONS

In this paper we introduced an intention-directed modeling technique called the ICM. The ICM classes follow object-oriented analysis and design approach. The ICM stresses on the three-tiered architecture: presentation, manipulation and storage layer. In addition the ICM has three components: the SCM, the PCM, and the ECM. The SCM and the PCM exhibit the three-tier architecture. The three-tiered architecture ensures that specialists conform to the same process in analysis and design of software system. The same classes are used. Therefore, variation in analysis and design is minimized, and the ICM based specification is easier to maintained by specialists than some other object oriented based techniques.

SCM, PCM, ECM classes provide the means for having the specialist pay particular attention to the requirement specification and the design specification of the three parts—presentation, manipulation and storage-- of an application. Therefore, requirements specification and design specification can be complete, unambiguous and completed in a timely manner than most current object oriented modeling techniques can do.

We indicated that some of the intentions can have default values, which some objected oriented models do not offer.

The actual intentions entered by the specialist can be classified into different groups for an application to be built. Classes for manipulation and transformation can be specified during the analysis and design.

Overall, the use of the ICM reduces effort in analysis and design.

A limitation of the intention-directed model-

ing is that the PCM classes are is dependent on programming languages, hardware platform, and operating systems. At this point in time the PCM classes that we have is for the VB6. Further work has to be put in developing definitions for PCM classes in other programming languages, such as C++ and J++.

7. REFERENCES

Blaha, M. and Premerlani, W. (1998). Object-Oriented Modeling and Design for Database Applications. Prentice Hall, Upper Saddle River, N.

Booch G. et al. (1999). The Unified Modeling Language User Guide. Addison-Wesley, Reading, MA.

Brumbauh, D. (1994). Object-Oriented Development: Building CASE Tools With C++. John Wiley, New York, NY.

Grossman, J.W. (1990). Discrete Mathematics: An Introduction to Concepts, Methods, and Application. Macmillan Publishing Co., New York, NY.

Jezequel, J. (1996). Object-Oriented Software Engineering with Eiffel. Addison-Wesley, Reading, MA.

Kuofie, M. H. S. (1999). An Intention Directed Software Engineering Methodology, Ph.D. Thesis. Oakland University.

Lee, R. C. and W. M. Tepfenhart (1997). UML and C++: A Praactical Approach to Object-Oriented Development. Prentice Hall, Upper Saddle River, New Jersey, NJ.

Page-Jones, M., (1995). What Every Programmer Should Know About Object-Oriented Design. Dorset House Publishing, New York.

Papurt, D. M. (1995). Inside the Object Model: the sensible Use of C++. Sigs Books, NY.

Pressman, R.S., (2001). Software Engineering: A Practitioner's Approach, 5th Edition, McGraw-Hill Inc., New York, NY.

Rawsthorne, D. and G. Phil (1999). "Effective Analysis for Object-Oriented Development." Journal of C++ Report, February, http://www.adtmag.com/joop/crarticle.asp

Schach, S. R. (2002). Classical and Object-Oriented Software Engineering, 5th Edition. McGraw-Hill Inc., Chicago, IL.

Sherry, L. (1998). "Intentional Objects in Business and Manufacturing." PC AI (Artificial Intelligence journal for Personal Computing), Vol. 12, No. 2, March/April.

Sommerville, I. (1992). Software Engineering. 4th Edition. Addison-Wesley, Reading, MA.

Dr. Matthew H. S. Kuofie holds a PhD in Systems Engineering from Oakland University, Michigan, an MS in Computer Science from Old Dominion University, Virginia, and a BS in Statistics with Mathematics, minor, from the University of Ghana. He received his high school education at Fijai Secondary School, Sekondi, Ghana. Dr. Kuofie is currently a visiting professor in the School of Information, Illinois State University. Dr. Matthew Kuofie is also an adjunct professor at the University of Michigan. Dr. Kuofie teaches or has taught business, statistics, computer science, and information systems courses. For a period of over 15 years, Dr. Matthew Kuofie worked on a number of information technology projects for various companies including Electronics Data Systems, General Motors, and Daimler Chrysler. Dr. Kuofie's research interests include software engineering, and alignment of information technology and business strategies. He serves on the Editorial Review Board for the *International Journal of Enterprise Information Systems*.

Christian C. Wagner is with the School of Engineering and Computer Science at Oakland University, Rochester, Michigan.