Malaysian Journal of Computer Science, Vol. 12 No.1, June 1999, pp. 47-54

AN APPROACH TO THE DEVELOPMENT OF HYBRID OBJECT-ORIENTED DESIGN TOOL

Augustina Aegidius Sitiol
Faculty of Computer Science and Information Technology
University of Malaya
50603 Kuala Lumpur, Malaysia
email: wga98031@fsktm.um.edu.my
Fax: 603-7579249

Sai Peck Lee
Faculty of Computer Science and Information Technology
University of Malaya
50603 Kuala Lumpur, Malaysia
email: saipeck@fsktm.um.edu.my
Fax: 603-7579249

ABSTRACT

The design and development of systems are confronted with objects involving a great variety of notations [1]. Objects identified within the problem can be represented in the software solution. Thus object design methods involve constructing an object-oriented model of the problem and mapping this model into a software design [2]. In this paper, Hybrid Object-oriented Method (HOOM) [3] is chosen whose concepts and techniques will be introduced briefly. HOOM is an object-oriented software development methodology [3] with graphical notations[1] representing object-oriented concepts. HOOM focuses on modeling of objects. The use of CASE tools could improve the ability to generate timely, quality software [4] using HOOM. CASE, which is an abbreviation for Computer-Aided Software Engineering, is a technology to automate software development. The need for a development of a new CASE tool, ODT (Hybrid Object-oriented Design Tool) to construct the object diagrams and other diagrams is defined. In contrast to many of the currently available tools, this project is intended to encourage the use of HOOM. Mapping of objects to a common form for integration of the represented knowledge and consistency checking is described.

Keywords: ***Object-oriented method, CASE tool, Graphical tool, Mapping, Consistency checking, Code generation, Document generation***

1.0 INTRODUCTION

Many Meta-CASE and CASE tools [5, 6, 7] like MetaEdit [8], MOOT [9], Graphical Designer, Paradigm Plus [10], etc. vary considerably in their functionality and design [7, 11]. CASE tools evaluations have revealed a number of shortcomings in many of the existing tools in use today. Areas have been identified which require improvement to meet the dynamic nature and growth of the object-oriented paradigm. These include lack of integration between specification tools and construction tools, lack of support for specifying new methodology knowledge, support of a few specific methodologies, lack of customization facilities, and lack of inheritance between the methodologies supported by the tool [10].

The major concern in HOOM and other object-oriented development methodology [3, 12] is also to present an efficient method to detect requirement errors and verify consistency. Consequently, they frequently meet fallacy such as omissions or wrong input of required information [12].

Theoretically, code generation is possible with any method. Under most methodologies, however, once the code has been generated, it is separated from the analysis model, and any changes to the application must be made to the code itself [4, 13]. Most of the present code generating CASE tools generate code from only the object model [14]. Most of the present code generating CASE tools consist of header files [11, 15] containing declaration of classes of the application. To make the application executable, the developer has to convert the dynamic models[2] and functional models into code and combine it with the code that is generated from the object model [14].

It is found that many tools do not support document generation. Typically, a tool that supports document generation would allow the developer to write a script in some scripting language to extract pieces of the model into files for incorporation into document. Unfortunately, the diagrams and text so extracted are often totally unformatted, so the developer is forced to repair the result with a word processor [4].

Several objectives motivated this project. First, to create a graphical tool that would be flexible enough to be used for mapping the design of HOOM's structural model and dynamic model into Java code and assures consistent naming of objects. Second, automatic code generation in Java code [16, 17] can be produced from the object diagram designs. Third, the enforcement of error detection and consistency checking between the diagrams is guaranteed and verified. Finally, the tool should have the ability to do

[1] Notations are textual and diagrammatic. Textual notations have syntax describing them and are 'structure edited'. Diagrammatic notations involve networks of nodes and links with predefined graphical presentations and supported by diagramming operations that respect appropriate connectivity rules.

[2] Dynamic model describes the flow of control and interactions among objects.

automatic generation of design documentation from the model.

The aim of this project is to construct a usable CASE tool, ODT (Object-oriented Design Tool), which will provide a framework[3] within HOOM. ODT will provide mapping process from the notations to the descriptions. In order to develop ODT it is necessary to understand HOOM and its graphical notation.

2.0 OVERVIEW OF HYBRID OBJECT-ORIENTED METHOD (HOOM)

HOOM is *simple* by only using four diagramming techniques. HOOM is also *programming language independent* which gives flexibility to implementers and provides portability of design deliverables across language. It is *complete* by the fact that it has taken and refined the best features of most of the well established object-oriented methods.

HOOM model was defined as grounded on four facilities. These facilities are the four diagramming techniques, the software development process, the system design architecture and the class specifications.

2.1 Diagramming Techniques

A diagramming technique (i.e. notation) is normally described by means of its syntax (i.e. how it looks), semantics (i.e. what it means) and pragmatics (i.e. guidelines and heuristics for its development) [3].

A system in HOOM is described from two different aspects: structural model and dynamic model. Structural model describes the objects in the system and their relationships and the dynamic model describes the flow of control and interactions among objects. The four diagramming techniques capture both the structural and dynamic aspects of the system. Class Relationship Diagram (CRD) and High Level Class Relationship Diagram (HL-CRD) capture the structural aspect of the system, whereas Object Interaction Diagram (OID) and State Transition Diagram (STD) capture the dynamic aspect of the system. Each diagram is described briefly.

2.1.1 Class Relationship Diagram

The Class Relationship Diagram (CRD) is a formal graphic notation used to capture the structure of the system by showing its abstractions in terms of classes with characterizing properties and the relationships between them. It is the core technique of HOOM that it is concise, easy to understand and works well in practice.

[3] Framework provides the infrastructure, the architectural guideline and a mechanism for reliably extending functionality [18].

In a CRD, a box having three compartments represents a class as shown in Fig. 1.

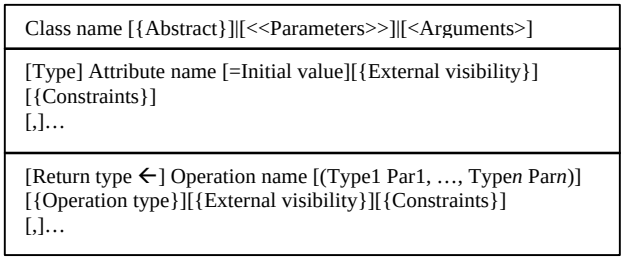


Fig. 1: Graphical Representation of a Class in CRD

The first compartment contains the name of the class. Elements in the square brackets are optional. The specification of {Abstract} indicates that the class is abstract. The specification of <<Parameters>> indicates that the class is generic. The specification of <arguments> indicates that the class is instantiated from a generic class.

The symbol " | " represents that a class can only either be an abstract class, a generic class or an instantiated class, if it is not a concrete class. The " … " indicates that the class may have many attributes and operations.

The second compartment contains attributes declarations, including a type (i.e. integer, character, real, string, etc), the name of the attribute, an initial value, external visibility and constraints.

The third compartment contains operations declarations, including a return type followed by the symbol " ← " which is used to separate between the name of the operation and the return type, the name of the operation, a list of parameters and their types, the operation type, external visibility and constraints. An empty parameter list within parenthesis shows the operation has no parameters, or that no decision has yet been made about the parameters.

2.1.2 High Level Class Relationship Diagram

The High Level Class Relationship Diagram (HL-CRD) shows dependencies between loosely coupled clusters. A cluster is a sub-CRD containing a set of cohesive classes and the relationships between them. A whole CRD of a complex system consists of one or more clusters, which divide it into manageable pieces. The names of classes and associations must be unique within their enclosing cluster. A HL-CRD is drawn to show dependencies between loosely coupled clusters as shown in Fig. 2.

A box in bold illustrates a cluster. A line between the related clusters and writing a {S} beside the server cluster represents a using relationship. A double arrow represents an association, names of the related classes are written beside the double arrow. A layer represents a collection of clusters at the same level of abstraction.

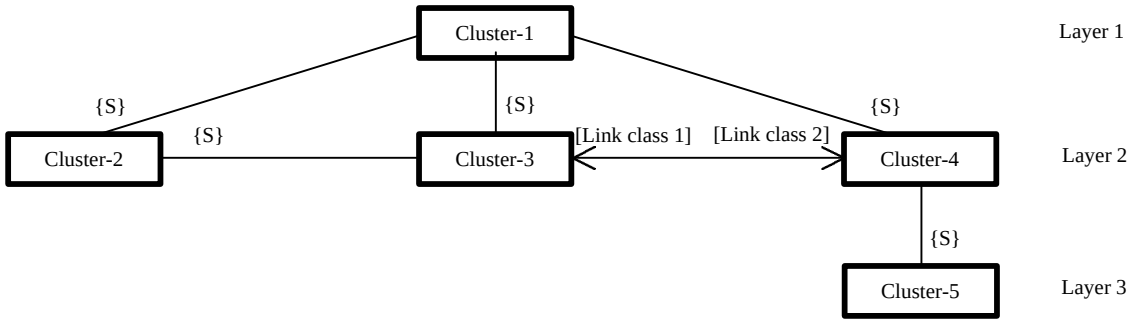
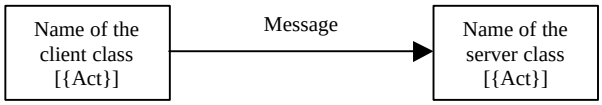


Fig. 2: Diagramming Elements of HL-CRD

2.1.3 Object Interaction Diagram

The Object Interaction Diagram (OID) shows the interactions among a set of objects during one particular execution of the system. It represents a snapshot in time of a stream of messages over certain configuration of interacting objects. It is also used to trace the execution of a scenario, which is a sequence of messages passed between objects during one particular execution of a system. In an OID, objects are represented in boxes and the message passing by arrows from the client object to the server object as shown in Fig. 3.



Act = Active Object

Message = [Return type ←] Operation name [(Arguments)][{Operation sequence number}][{Synchronization}] [{Visibility}][{Constraints}]

Synchronization = {S} (Synchronous), or {B} (Balking), or {T} (Time-out), or {A} (Asynchronous)

Fig. 3: Diagramming Elements of OID

2.1.4 State Transition Diagram

The State Transition Diagram (STD) shows the evolution of objects of a class that exhibits an important dynamic behavior in response to interactions with other objects as shown in Fig. 4.

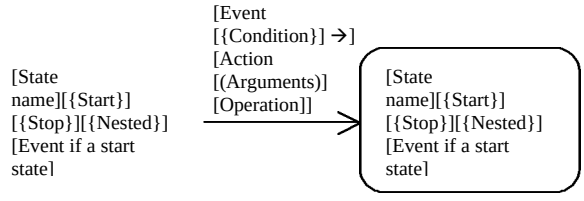


Fig. 4: Diagramming Elements of STD

A state is depicted by a rounded box. It contains the name of the state, an indication in braces of whether it is a start state, a stop state or a nested state, the event causes the entrance to the state if it is a start state, activities preceded by the keyword *Ac:* and recursive transitions (i.e., transitions from a state to itself).

An arrow from one state to another illustrates a transition. A transition is a state change caused by the occurrence of an event. The name of the event, any condition on the event, the symbol " → " separating the event and the action, the resulting action with its arguments and its corresponding operation are labeled on the arrow.

2.2 Software Development Process

The HOOM analysis models (i.e. HL-CRDs, CRDs, OIDs and STDs) aim to give the system a robust and changeable class structure. During the design phase, the analysis models are refined and enhanced and implementation decisions are made. In the implementation phase, the system will be implemented in the platform.

HOOM software development process describes how, with the help of the diagramming techniques, models of different systems can be created. Specific system design architecture is therefore the result obtained after applying the software development process to a system.

HOOM iterative and incremental software development process encompasses three phases: *analysis*, *design* and *implementation*. Each phase is performed in many activities. The order of the steps in each activity needs not be strictly followed, it is rather flexible depending on the nature of the project at hand.

HOOM software development process has taken and refined the best features of existing object-oriented methods and has focused on the analysis phase, which has the most important impact on the whole development process. The analysis deliverables will be smoothly mapped into the design and implementation phases since there is no new diagramming techniques used in these phases.

2.3 System Design Architecture

HOOM system design architecture (SDA) was inspired from the model-view-controller (MVC) developed by the Smalltalk community [3]. The central purpose of SDA is using the analysis models as the heart of the design model.

HOOM is using the same graphical notation in analysis and design. The ability to build compact system design architecture is greatly enhanced. Moreover, one of the principal goals of object-oriented software development is to improve the reusability of software components. Increase reusability of software is considered as a crucial technical pre-condition to improve the overall software quality and reduce production and maintenance costs. Analysis is concerned with what the system under development is intended to do, whereas design is concerned with how the requirements will be implemented. One of the main objectives of design is to introduce consistency and predictability into the software development process.

3.0 DEVELOPMENT APPROACH OF ODT

Object-oriented analysis and design methodologies and the use of CASE technology are extensive in the object-oriented world. One important role of a CASE tool is to serve as a methodology companion [11, 15], i.e. to assist and guide the developer through a particular systems development methodology.

The level of assistance that will be provided by ODT for HOOM includes a graphical tool, error detection and consistency checking, Java code generation and document generation to support the diagramming techniques.

3.1 Overview

The development activity of ODT consists of five steps: modeling, formalizing, verifying, code generating and document generating which are depicted in Fig. 5.

In the modeling step, the system is analyzed and two models are produced: the structural model and the dynamic model. Structural model consists of CRD and HL-CRD, whereas dynamic model consists of OID and STD. ODT will generate Java code directly from the models.

In the formalizing step, these two models are transformed into Atomic Formulae. These models are stored into a Design Specification Language (DSL). DSL is created to provide a textual representation in the form of Atomic Formula of the models. In the verifying step, the rules for error detection and consistency check from a Rule Specification Language (RSL) are applied to check the errors and consistency of the models. In case that an error is detected, the modeling steps, formalizing step and verification step are repeated until the models are consistent.

In the code generating step, the mapping of the models will be translated to Java code. In the document generating step, reports of the models will be generated.

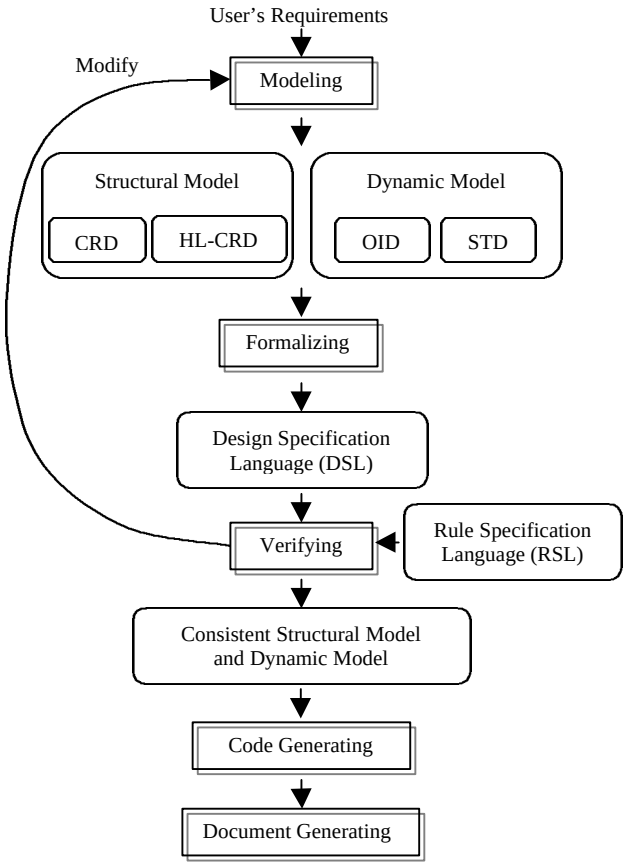


Fig. 5: Development Method of ODT

3.2 Mapping of the Models

The mapping of the models is specified. A graphical editor (GE) will be designed with the intention to support the creation of models and can be easily maintainable. Although some CASE tools already map the notations of various object-oriented methods [5, 8, 11, 19, 20], this has not been done for HOOM. In addition, HOOM is taken in order to create an architecture, which will be easily maintainable and could be extended easily.

A graphical approach should be used in order to take advantage of the diagramming techniques of HOOM. Using this approach also makes GE useful as an educational aid [21, 22] in the development of object-oriented applications, as it will allow novice to create models. GE will facilitate diagramming specification of the system, rather than having to express the system specification in a textual format [20]. The diagramming techniques of HOOM will be captured graphically and mapped to the framework using Java.

The mapping from the models to Java can be described as follows:

- HOOM classes will be mapped to Java classes
- HOOM operations will be mapped to Java methods

- HOOM interfaces will be mapped to Java interfaces
- HOOM inheritance can be implemented in Java through the *extends* and *implements* relationships
- Other kinds of associations and relationships in HOOM can be implemented by reference/composition of an object within another
- HOOM processes will be mapped to Java processes.

3.3 Formalizing with Atomic Formulae

Graphical forms cannot be understood by the system directly. That means error detection and consistency check of graphical models cannot be automated. Therefore, the formal specification of the models is required. The contents of the models will remain intact, which is very important [12, 23, 24].

In this section, formalizing the models will be explained in the form of Atomic Formulae. The Atomic Formula is a name – a predicate – followed by zero or more arguments enclosed in parenthesis and separated by commas.

3.3.1 Formalizing the Structural Model

3.3.1.1 Class Relationship Diagram

The class can be represented as in Expression 1. Attribute and operation in Expression 2 and Expression 3.

Expression 1:
Class (class name, [kind], [parameters or arguments], [attributes], [operations])

Expression 2:
Attribute (attribute name, [type], [initial value], [external visibility], [constraints])

Expression 3:
Operation (operation name, [return type], [type parameters], [operation type], [external visibility], [constraints])

There are four ways which the relationship is specified; associations, inheritance, composition and instantiation. The relationship can be represented as in Expression 4, Expression 5, Expression 6 and Expression 7.

Expression 4:
Association (association name, first associated class, second associated class, [first cardinality], [second cardinality], [first role name], [second role name], [direction])
Expression 5:
Inheritance ([subclass], [superclass])

Expression 6:
Composition (composite class, component class, [cardinality of component], [kind of composition])

Expression 7:
Instantiation ([instantiated class], [generic class])

3.3.1.2 High Level Class Relationship Diagram

The cluster can be represented as in Expression 8. There are two ways by which the relationship is specified; using and association. The relationship can be represented as in Expression 9 and Expression 10.

Expression 8:
Cluster (cluster name, [global], [enclosed classes])

Expression 9:
Using (client cluster, server cluster)

Expression 10:
ClusterAssociation (first associated cluster, second associated cluster, [first link class], [second link class])

3.3.2 Formalizing the Dynamic Model

3.3.2.1 Object Interaction Diagram

The object can be represented as in Expression 11.

Expression 11:
Object (object name, [concurrency])

The message can be represented as in Expression 12.

Expression 12:
Message passing (client object, server object, [signature], [sequence number], [synchronization], [visibility], [constraints])

3.3.2.2 State Transition Diagram

The state can be represented as in Expression 13.

Expression 13:
State (state name, [kind], [event if start], [activity], [reflexive transitions])

The transition can be represented as in Expression 14.

Expression 14:
Transition (source state, destination state, [event], [condition], [action], [operation])

3.4 Error Checking and Consistency Checking

Models are the basic building blocks of the generated product. Thus, good consistency checking [4, 25] is very important to ensure the reliability of the system. Methods

will be developed to perform the verification of consistency between diagrams and assure for consistent naming of objects and methods.

The development of the structural model precedes that of the dynamic model. The HL-CRD, OID and STD are dependent on the CRD. The dependency is closely related to consistency principles [12].

The rules for checking errors in the structural model are as follows:

Rule 1	A class must be unique. Another class with the same identification is not acceptable.
Rule 2	An attribute must be unambiguous in the context of the class and must be unique within the class.
Rule 3	An instantiated class cannot have more than one generic class.
Rule 4	An instantiated class cannot be generic.
Rule 5	Cycles are not allowed in inheritance, composition and association relationships.
Rule 6	Inheritance and composition, inheritance and association, composition and instantiation, association and instantiation are not allowed at the same time between two classes.
Rule 7	The names of the classes and associations must be unique within their enclosing cluster.
Rule 8	Inheritance and composition relationships are not transitive.
Rule 9	All the enclosed classes of a cluster in the HL-CRD must exist in their related CRD.

The rules for checking errors in the dynamic model are as follows:

Rule 10	An activity in a state must be completed before an event causes a transition from that state.
Rule 11	Any state except the terminal state must have at least a transition to other states.
Rule 12	Each state must be unique within its enclosing class.

Rules for checking consistency between the structural model and the dynamic model are as follows:

Rule 13	All the classes of objects in an OID must exist in their related CRD.
Rule 14	If a relationship of association exists between classes, there must be a communication path between the state diagrams that describe the behavior of these classes.
Rule 15	In OID, an operation performed by a server object must exist in its class or in one of its ancestor classes in its related CRD.
Rule 16	In STD, all the actions and activities must exist as operations in their class in the related CRD, otherwise a warning is reported.

These 16 rules above will be programmed in Java and will be stored in Rules Specification Language (RSL).

3.5 Code Generating with Java

Java language has been chosen as the intermediate languages, object-oriented programming is facilitated [26, 27]. From the software diagram, the tool generates the Java code required to interconnect these components [28]. Java code will be generated automatically from the diagram and assures consistent naming of member functions and attributes.

The tool will use the structural models associated with each class operation to generate a complete method function implementation for the operation in Java. The class dynamic models will also be generated. There will be extensive consistency checking at each stage.

3.6 Document Generating

Generating documents was not a single button operation [4]. The functionality of the document generation will be integrated directly into the tool rather than adding it on via scripts. Reports on the diagrams will be automatically generated.

4.0 CONCLUSION

This paper introduces ODT, a CASE tool for an object-oriented method that will be able to provide a framework for HOOM. This is also to encourage the use of this methodology as an educational aid. ODT with the usage of HOOM is applied mainly in order to gain better control of the development activities.

There are five steps described in constructing ODT for HOOM: modeling, formalizing, verifying, code generating and document generating. GE for use in design mapping will support the creation of models in the modeling step. The models then will be formalized in Atomic Formulae expressions in the formalization step. DSL is used to store the expressions. In the verifying step, the enforcement of consistency checking is described. Rules for checking errors and consistency have been identified in RSL. This is to ensure the reliability of the system. In code generation step, a complete method function for the operation of Java will be generated automatically from the models. In the document generation step, reports of the models will be generated.

REFERENCES

[1] W. Cyre, "Mapping Design Knowledge from Multiple Representations". *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 1991.

[2] J. Daniels, "Object Design Methods and Tools". *IEE Colloquium on Distributed Object Management*, pp. 3/1–3/3, 1994.

[3] T. Taibi, "Hybrid Object-Oriented Method (HOOM) for Object-Oriented Software Development". *Master Thesis, Faculty of Computer Science and Information Technology*, University of Malaya, 1998.

[4] T. Church and P. Matthews, "An Evaluation of Object-Oriented CASE Tools: The Newbridge Experience". *Proceedings of the Seventh International Workshop on CASE*, 1995.

[5] A. Alderson, J, Cartmell and T. Elliot, "The Concepts Underlying a Meta-CASE Tool". *Technical Report SOCTR/97/02, School of Computing*, Staffordshire University, 1997.

[6] A. Van Hoff, "The Case for Java as a Programming Language". *IEEE Internet Computing*, Vol. 1, No. 1, pp. 51-56, 1997.

[7] S. C. Stobart, A. J. Van Reeken, G. C. Low, J. J. M. Trienekens, J. O. Jenkins, J. B. Thompson and D. R. Jeffery, "An Empirical Evaluation of the Use of CASE Tools". *Proceedings of the Sixth International Workshop on Computer-Aided Software Engineering*, 1993.

[8] MetaCase Consulting Oy, "MetaEdit Personal 1.2 User Manual". *MetaCASE Consulting*, 1995.

[9] D. Page, D. Griffin, L. Usherwood, and D. Mehandjiska, "Implementation of Semantic Specification Language Interpreter for a Methodology Independent OO CASE Tool". *Proceedings of the IASTED International Conference Software Engineering (SE '97)*, November 2-4, 1997.

[10] D. Mehandjiska, D. Page, D. Griffin and L. Usherwood, "The Methodology Representation Mechanism of an Object-Oriented MetaCase Tool". *Proceedings of the IATED International Conference Software Engineering (SE '97)*, November 2-4, 1997.

[11] D. J. Jankowsi, "CASE Tool Selection". *Journal of Systems Management*, Cleveland, Vol. 46. Issue 4, July/August 1995.

[12] D. Kim and K. Chong, "A Method of Checking Errors and Consistency in the Process of Object-Oriented Analysis". *Proceedings of 1996 Asia-Pacific Software Engineering Conference*, 1996.

[13] E. Heichler, "Development Tools Take on Code Generation". *Computerworld*, Framingham, Vol. 29, Issue 18, May 1, 1995.

[14] J. Ali and J. Tanaka, "Generating Executable Code from the Dynamic Model of OMT with Concurrency". *Proceedings of the IATED International Conference Software Engineering (SE '97)*, November 2-4, 1997.

[15] Spectrum Staff, "The Case for CASE Tools". *IEEE Spectrum*, Vol. 27, Issue 11, pp. 78-81, 1990.

[16] A. Walsh and J. Fronckowiak, "Java Bible". *IDG Books Worldwide, Inc.*, 1998.

[17] B. Maso, "Visual J++ 6 the Ground Up". *McGraw-Hill Companies*, 1999.

[18] K. K. Dong, T. J. Hyo andK. K. Chae, "Techniques for Systematically Generating Framework Diagram Based on UML". *Proceedings of the Asia Pacific Software Engineering Conference*, pp. 203-210, 1998.

[19] N. C. Shammas, "Teach Yourself Visual C++ in 21 Days". *SAMS Publishing, First Edition*, 1993.

[20] V. K. Shanbhag and K. Gophinath, "A C++ Simulator Generator from Graphical Specifications". *Software – Practice and Experience*, Vol. 27 (4), pp. 395-423, April 1997.

[21] F. Dicesare, M. R. Gile and P. T. Kulp, "An Object Oriented Graphical Tool for Creating Petri Nets". *Proceedings of the Fourth International Conference on Computer Integrated Manufactruing and Automation Technology*, 1994.

[22] J. J. Swhwarz, J. J. Skubich, P. Szwed and M. Maranzana, "Real Time Multitasking Design with a Graphical Tool". *Proceedings of the IEEE Workshop on Real Time Application*, 1993.

[23] A. Arnold, D. Begay and J. Radoux, "The Embedded Software of an Electricity Meter: An Experience in Using Formal Methods in an Industrial Project". *Science of Computer Programming*, Vol. 28, pp. 93-110, 1997.

[24] D. Dzierzgowski and E. Gregoire, "Formalizing Software Development Methods". *Design: Concepts, Methods and Tools: CompEuro '88*, pp. 230-239, 1988.

[25] S. Kouno, H. Chang and K. Araki, "Consistency Checking between Data and Process Diagrams Based on Formal Methods". *Proceedings of the Twentieth Annual International Computer Software and Applications Conference: COMPSAC '96*, 1996.

[26] A. Van Hoff, "The Case for Java as a Programming Language". *IEEE Internet Computing*, Vol. 1 No. 1, pp. 51-56, 1997.

[27] C. H. Soo, "A Visual Tool for C++ Program Development". *Proceedings of the TENCON '93. IEEE Region Ten International Conference on Computers, Communications and Automation*, 1993.

[28] R. Janka, "Graphical Tools Enhance Productivity". *Electronic Engineering Times*, Manhasset, Issue 961, July 7, 1997.

[29] K. Arnold, "Why C++?" *UNIX Review*, Mercer Island, February 1992.

[30] K. diamond and K. Pang, "Mapping VHDL Descriptions of Digital Systems to FGPAs". *IEE Colloquium on Software Support and CAD Techniques for FGPAs*, pp. 9/1-9/3, 1994.

[31] D. Mulvaney and C. Bristow, "A Rule-based Extension to the C++ Language". *Software – Practice and Experience*, Vol. 27(7), pp. 747-761, July 1997.

[32] M. A. Rodrigues, C. Loftus, M. Ratcliffe and Y. F. Li, "Structure Notation of Dynamic Systems: A Pictorial Language Approach". *Proceedings of the 1994 International Conference on Computer Languages*, pp. 219-228, 1994.

[33] J. Parsons, "Choosing Classes in Conceptual Modeling". *Association for Computing Machinery, Communications of the ACM*, New York, June 1997.

[34] P. Lang, W. Obermair, W. Kraus and T. Thalhammer, "A Graphical Editor for the Conceptual Design of Business Rules". *Proceedings of the Fourteenth International Conference on Data Engineering*, 1998.

[35] W. Savitch, "Problem Solving with C++, The Object of Programming". *Addison-Wesley Publishing Company, Inc.*, 1996.

[36] D. Xu, "Towards an Object-Oriented Logic Framework for Knowledge Based Systems". *Knowledge-Based Systems*, Vol. 10, pp. 351-357, 1998.

[37] I. A. Zualkernan and R. S. Ordower, "Towards Automating Object-Oriented Analysis". *Proceedings of the Eighteenth Annual International Computer Software and Applications Conference (COMPSAC '94)*, p. 103, 1994.

[38] F. N. Kesim, Member, IEEE Computer Society and M. Sergot, "A Logic Programming Framework for Modeling Temporal Objects". *IEEE Transactions on Knowledge and Data Engineering*, Vol. 85, pp. 724-741, October 1996.

[39] B. Stahl and A. Jansen, "A Conceptual Mapping of the GSM12.20 Managed Objects Templates Defined in ASN.1 Into an Entity-Relationship Datamodel". *Proceedings of the International Zurich Seminar on Intelligent Networks and Their Applications, Digital Coomunications*, pp. E5/1-E5-17, 1992.

[40] E. V. Kortright, "Modeling and Simulation with UML and Java". *Proceedings of the 30th Annual Simulation Symposioum 1997*, pp. 43-48, 1997.

[41] F. Huber, A. Rausch and B. Rumpe, "Modeling Dynamic Component Interfaces". *Proceedings of the Technology Object-Oriented Languages (TOOLS 26)*, pp. 20-32, 1998.

BIOGRAPHY

Augustina Aegidius Sitiol is currently pursuing her Master in Computer Science program in the Faculty of Computer & Information Technology, Universti Malaya.

Sai Peck Lee obtained her Master of Computer Science from University of Malaya in 1990, D.E.A of Computer Science from University of Paris VI Pierre et Marie Curie in 1991 and Ph.D of Computer Science from University of Paris I Pantheon-Sorbonne in 1994. She is a lecturer at Faculty of Computer Science and Information Technology, University of Malaya.