The Software Screenplay Storyboard Model (S3M)

by
Mr. Jia Cherng Hsu

A Thesis of the Twelve-Credit Course
CE 7000 Master Thesis

Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in Computer and Engineering Management
Assumption University

July 2007

The Software Screenplay Storyboard Model (S3M)

by
Mr. Jia Cherng Hsu

A Thesis of the Twelve-Credit Course
CE 7000 Master Thesis

Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in Computer and Engineering Management
Assumption University

July 2007

Thesis Title	The Software Screenplay Storyboard Model (S3M)
Name	Mr. Jia Cherng Hsu
Thesis Advisor	Dr. Chamnong Jungthirapanich
Academic Year	July 2007

The Graduate School of Assumption University has approved this final thesis of the twelve-credit course, CE 7000 Master Thesis, submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer and Engineering Management

Approval Committee:

(Prof.Dr. Srisakdi Charmonman)
Chairperson of Examination Committee

(Assoc.Prof.Somchai Thayarnyong)
CHE Representative

(Dr. Chamnong Jungthirapanich)
Advisor

(Dr. Chanintorn Jittawiriyanukoon)
Member

(Dr. Charnwit Somprakij)
Member

July 2007

ABSTRACT

Software buyers and developers are still experiencing difficulties in communicating functional requirements. A universal software model is needed to bridge the buyer-developer gap.

Using concepts derived from the Unified Modeling Language (UML), software prototypes, film screenplays and storyboards, this research devises a new communication tool called the Software Screenplay Storyboard Model (S3M). It is designed to define and demonstrate functional requirements effectively and efficiently by creating non-technical software models that combine structured narrations with illustrations. S3M is further clarified and exemplified by applying it to a real-world application, the Student Information System (SIS).

With full comprehension of S3M, the new software model is compared with all software models that have been studied in a table. S3M is found to satisfy the need for a non-technical, yet comprehensive and precise software model. However, it is not intended to replace other software models, which may be optionally employed to reflect other aspects, technicalities, or views of a system.

ACKNOWLEDGEMENTS

The accomplishment of this thesis is made possible with the kind guidance and encouragements of my advisor, Dr. Chamnong Jungthirapanich, and the education opportunity, financial support, and patience of my family.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

1.1 Background of the Thesis

Software buyers and developers are still facing communication difficulties during software purchases today. Buyers – especially those inexperienced in software purchasing, and developers – especially those inexperienced in buyer servicing, convey in different communication frequencies. For instance, Figure 1.1 demonstrates how buyers' perception of the software model is impaired due to multiple layers of interpretations before they can begin to comprehend the software. Can there be a universal software model, a true link that is able to synchronize buyers and developers easily as in Figure 1.2? Failing to tune in or communicate well with one another inevitably leads to ineffective requirements engineering – one of the most critical and early phases in all software methodologies.

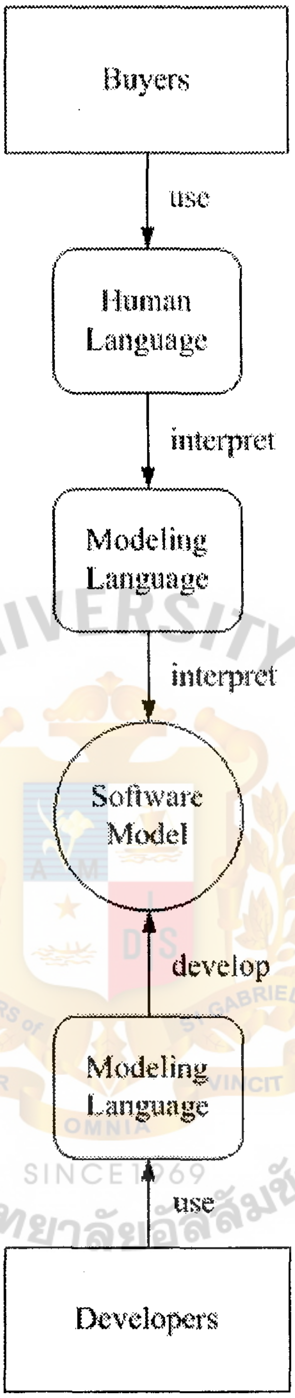


Figure 1.1. Some Software Models Require Multiple Interpretations.

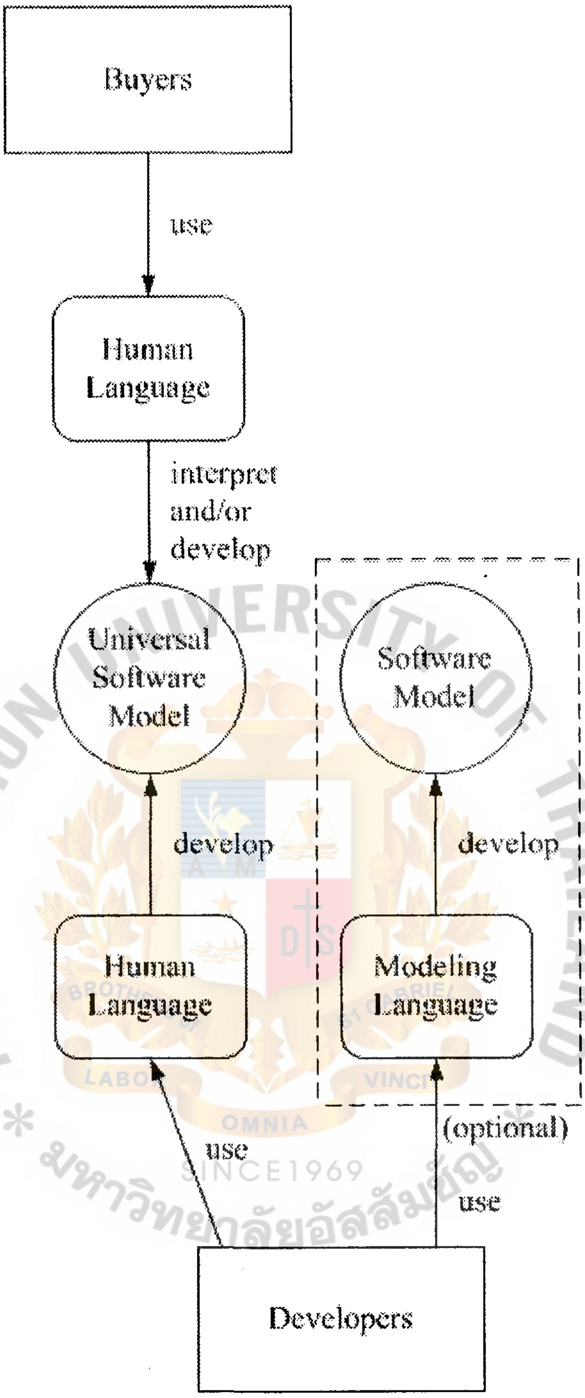


Figure 1.2. Universal Software Model Requires Single Interpretation.

Requirements engineering "(in the context of systems engineering) is concerned with the acquisition, analysis, specification, validation, and management of software requirements" (Aaby, 2000). The first truth topping expert Karl Wiegers' list of five universal truths on software requirements engineering is: “If teams don’t get requirements right, it doesn’t matter how well they execute the rest of the project” (Borland, 2006) – quality assurance professionals can’t agree more. The “Man on the Street” poll conducted by the Borland Software Corporation at the STAR*WEST* conference in Anaheim, California, surveys nearly 250 quality assurance professionals and reveals that, “poorly defined or mismanaged requirements and inadequate time for proper testing are the two biggest problems QA professionals are grappling with today” (Borland, 2006). Problematic requirements also earn high ranks in the Standish Group’s annual CHAOS report, which indicates that three of the top five reasons for project failure are related to requirements (Borland, 2006).

Two prime types of requirements are described in many texts. Functional requirements are actions a software system performs, such as inputs, outputs, and calculations. On the other hand, non-functional requirements are properties a software system possesses, such as performance, availability, and accessibility. As non-functional requirements are also known as technical requirements, quality of service (QoS) requirements, and service-level requirements (Ambler, 2006), it is the functional requirements that chiefly link software buyers and developers, and outnumber non-functional requirements. Moreover, Ambler (2006) firmly believes that purely technical requirements should be minimized due to rapid changes in technology that will ultimately require changes to corresponding technology-dependent requirements. Hence, the challenge is often: how to communicate functional requirements effectively as well as efficiently.

This research devises a new communication tool called the Software Screenplay Storyboard Model (S3M). It is designed to define and demonstrate functional requirements effectively and efficiently by creating non-technical software models that combine structured narrations with illustrations. For uniformity, this research utilizes the same definitions for 'modeling' and 'model' as the Object Management Group (2006) – "Modeling is the designing of software applications before coding . . . A model plays the analogous role in software development that blueprints and other plans (site maps, elevations, physical models) play in the building of a skyscraper."

1.2 Objectives of the Thesis

The research possesses four objectives:

(1) To study the Unified Modeling Language (UML) and software prototypes to find their strengths and weaknesses.

(2) To study film screenplays and storyboards to find their best practices.

(3) To devise and test a new software model that incorporates aforementioned strengths, eliminates aforementioned weaknesses, and reinforced by aforementioned best practices.

(4) To compare all software models that have been studied.

1.3 Scope of the Thesis

S3M builds on concepts derived from the Unified Modeling Language (UML), software prototypes, film screenplays and storyboards. UML is considered to be the de facto standard for building software models. Software prototypes possess the closest resemblance to S3M in terms of using narrations coupled with illustrations to model software. Film screenplays and storyboards have proven best practices that can be incorporated into S3M. UML, software prototypes, and film screenplays and storyboards are studied in the second chapter. Findings from chapter two are then used to develop S3M in chapter three. Chapter four tests S3M by applying it to a real-world application, which also helps clarify any remaining queries regarding S3M. With full comprehension of S3M, chapter five compares all software models that have been studied in a table, and the final chapter concludes findings and recommendations.

II. LITERATURE REVIEW

2.1 Unified Modeling Language (UML)

UML, consisting of graphical notations, is used to fabricate abstract models of software systems as well as non-software systems (Object Management Group, 2006). Keywords such as business process, systems engineering, and organizational structure are covered by its extensive modeling capabilities. UML is able to describe almost any type of software – from its overall structure to internal mechanisms in various views or diagram types.

UML is built on fundamental object-oriented (OO) concepts. For this reason, it works well with OO programming languages such as C++, Java, and C#. However, UML is also flexibly designed allowing it to be hardware, operating system, programming language, middleware and network- independent. In addition, it offers UML Profiles that allows users to define stereotypes to introduce concepts that are unavailable in the base language. These customized subsets of UML help streamline the modeling of Transactional, Real-Time, and Fault-Tolerant systems (Object Management Group, 2006).

UML also enjoys being methodology-independent, which has helped it gain widespread support. UML can model results regardless of how a software project is conducted. And with the XML Metadata Interchange (XMI) technology from OMG, UML models can migrate from one tool to another for refinement or preparation for the next step in the chosen methodology.

Abundant UML-based tools are currently available in the market to analyze requirements and design software solutions. Some tools are able to generate codes from UML diagrams, or reverse engineer existing codes to produce UML diagrams. Some tools are designed for certain industries such as telecommunications or finance, as some generate test and verification suites from UML models.

UML has fueled model-driven technologies such as Model Driven Development (MDD) and Model Driven Engineering (MDE). It is also the foundation of OMG's Model Driven Architecture (MDA) (Object Management Group, 2006). UML is also able to produce platform-independent and platform-specific models. The MDA development process utilizes both features to create Platform-Independent Models (PIM) and Platform-Specific Models (PSM) in the UML language via MDA-enabled development tools. PIM represents business functionalities and behavior accurately, as PSM represents the implementation or running code. Moreover, it is possible to use the new Query-View-Transformations (QVT) standard to transform a UML model, serialized in XMI, into a Java or Enterprise Java Bean (EJB) implementation by using a Model Transformation Language (MTL).

2.1.1 History

UML stems from the efforts by James Rumbaugh, Ivar Jacobson, and Grady Booch to unify several object modeling languages. The best of Rumbaugh's Object-modeling Technique (OMT), Jacobson's Object-oriented Software Engineering (OOSE), and Booch's Booch Method are synthesized to create the more standardized Unified Modeling Language (UML) at Rational Software Corporation in the mid-1990s. In 1996, the UML authors decide to work with professionals from other companies to respond to the Object Management Group's (OMG) request for an object modeling standard. The UML specification draft is submitted to OMG on January 1997, followed by the final proposal in September 1997 (Rumbaugh et al. 1999). The initial semantic integration, however, is still relatively weak and requires several minor revisions and a major revision to conceive UML 2.0, which is the current OMG standard. Even then, UML is still changing and UML 2.1 is scheduled for release in the first semester of 2006 in the form of an XMI 2.1 version.

Despite ongoing changes, the International Organization for Standardization (ISO) has declared UML an international standard, and it is available as ISO/IEC 19501; ITU-T Recommendations Z.100 (SDL) and Z.109 (SDL UML profile) (Object Management Group, 2006).

2.1.2 Diagrams

UML 2.0 defines thirteen diagram types under two major categories and one subset. Six structure diagrams portray the static application structure (what must be in the system), three behavior diagrams represent dynamic object interactions (what must happen in the system), and four special behavior diagrams known as interaction diagrams model the flow of control and data among the things in the system. Diagrams also have the option of nesting inside one another, for example a state machine nests inside another structure diagram. Figure 2.1 depicts the hierarchy of UML 2.0 diagrams.

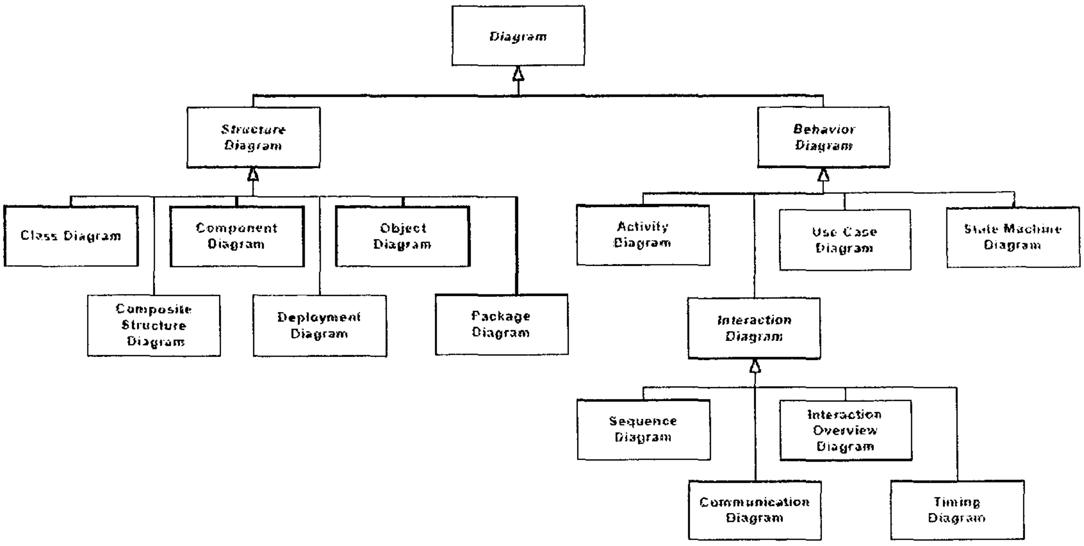


Figure 2.1. Hierarchy of UML 2.0 Diagrams in a Class Diagram (Object Management Group, 2005).

Class diagram, a very common structure diagram, shows classes and their relationships within a system. A class is a distinct concept used to encapsulate related variables and functions or methods. According to Rumbaugh et al. (1999), it can assume many forms: a physical concept such as an airplane, a business concept such as a customer order, a logical concept such as a broadcasting schedule, an application concept such as a cancel button, a computer concept such as a hash table, or a behavioral concept such as a task. Various association notations used to portray class or object relationships may appear from simple connections and ends, to ones with complicated multiplicity and separate classes of their own. Figure 2.2 is an example of a class diagram.

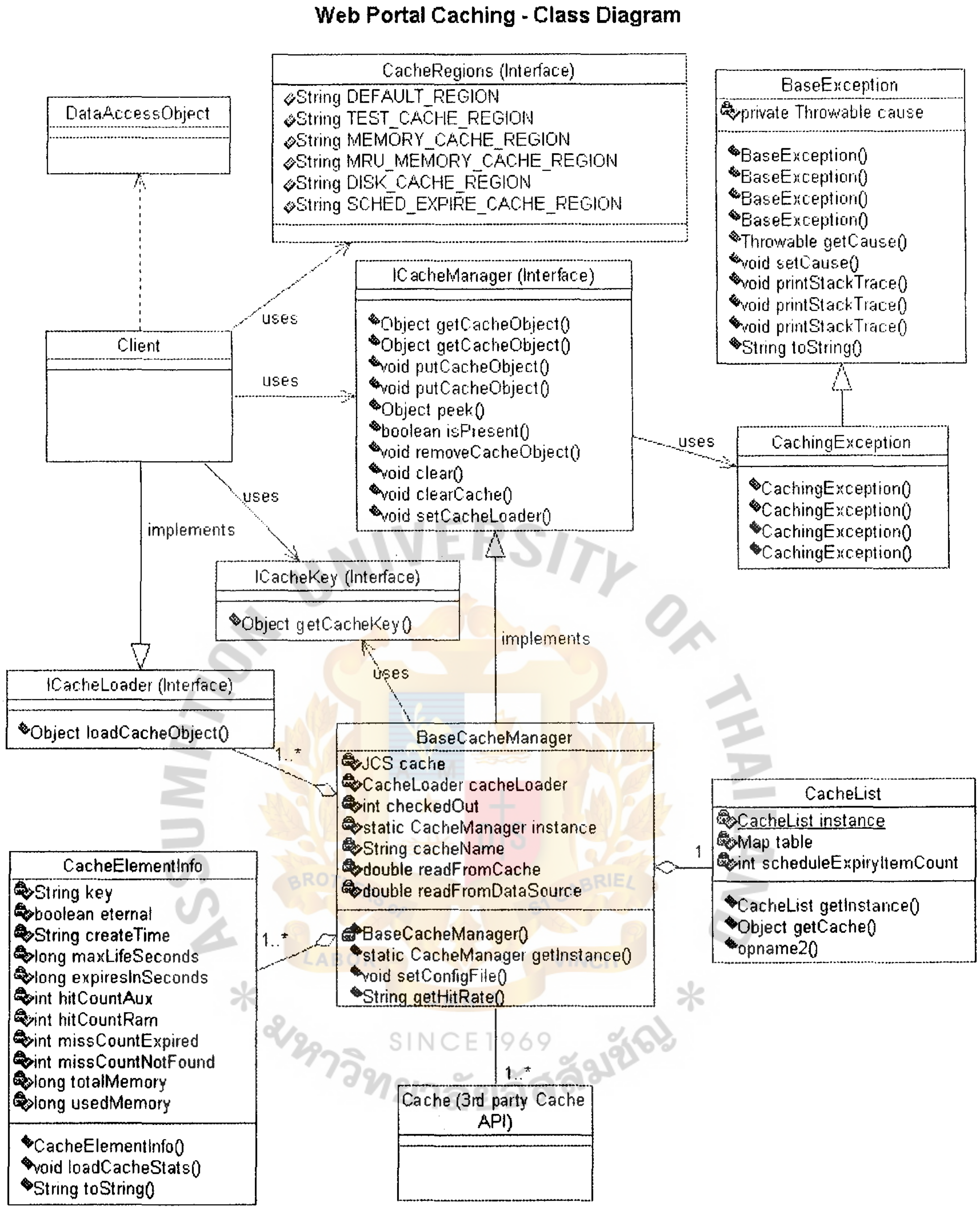


Figure 2.2. “Web Portal Caching” Class Diagram (Penchikala, 2003).

Composite structure diagram, another type of structure diagram, reveals the internal structure of a class, including structured classifiers, parts, ports, connectors, and collaborations. A structured classifier is a class whose behavior is described through parts interactions. Parts are roles played during runtime by an instance or group of instances. A part may name a role, an abstract superclass, or specific concrete class. Ports enable the interaction between a structured classifier and its environment. They may specify the services a classifier provides or requires. Structured classifiers containing ports are called encapsulated classifiers. Connectors are drawn as lines that bind parts, ports, and structured classifiers – allowing them to interact at runtime, and collaborations specify sets of roles working together to represent certain functionalities. Lastly, Ambler (2006) does not find composite structure diagrams to be of much use, and instead, suggests using sequence diagrams which possess more robust notations and are more widely understood. Figure 2.3 is an example of a composite structure diagram.

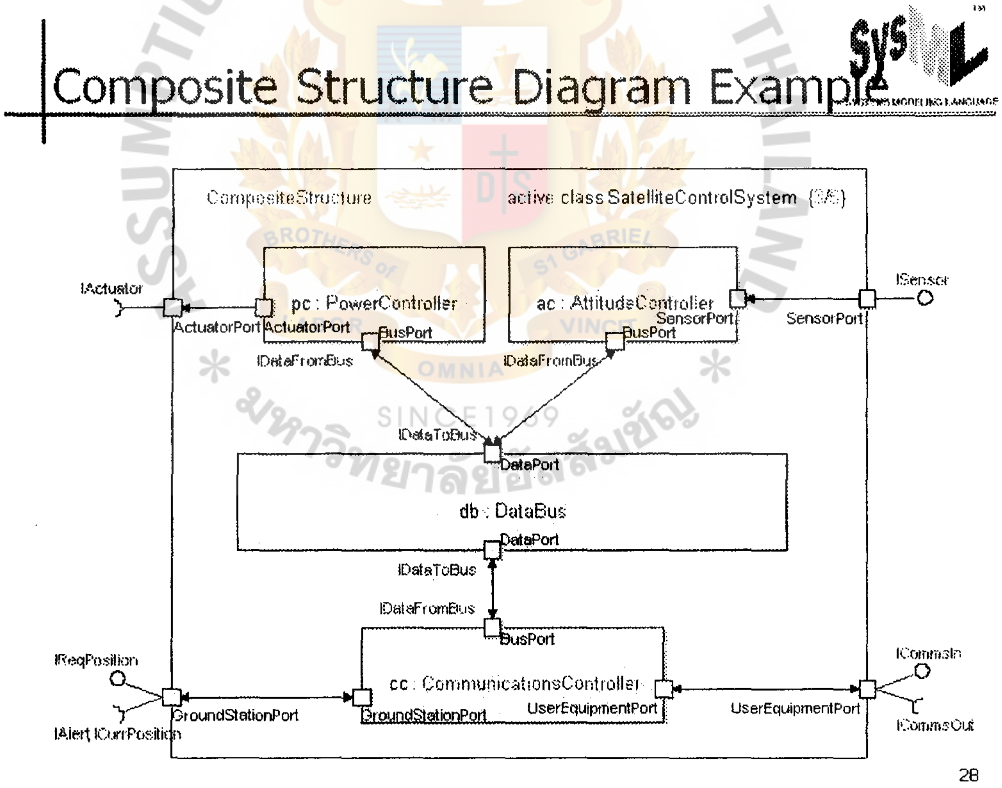


Figure 2.3. Composite Structure Diagram Example (SysML Partners, 2004).

Component diagram, a type of structure diagram, depicts components such as files, headers, link libraries, modules, executables, and packages, and their dependencies or semantic relationships in a system. Components are physical units of implementation of certain classes with interfaces, and they can be replaced by other components with the same interfaces (Rumbaugh et al. 1999). Interfaces, or lists of operations supported by pieces of software or hardware according to Rumbaugh et al. (1999), are used to render dependencies among components. Components generate some interfaces as well as require interfaces from other components. Figure 2.4 is an example of a component diagram.

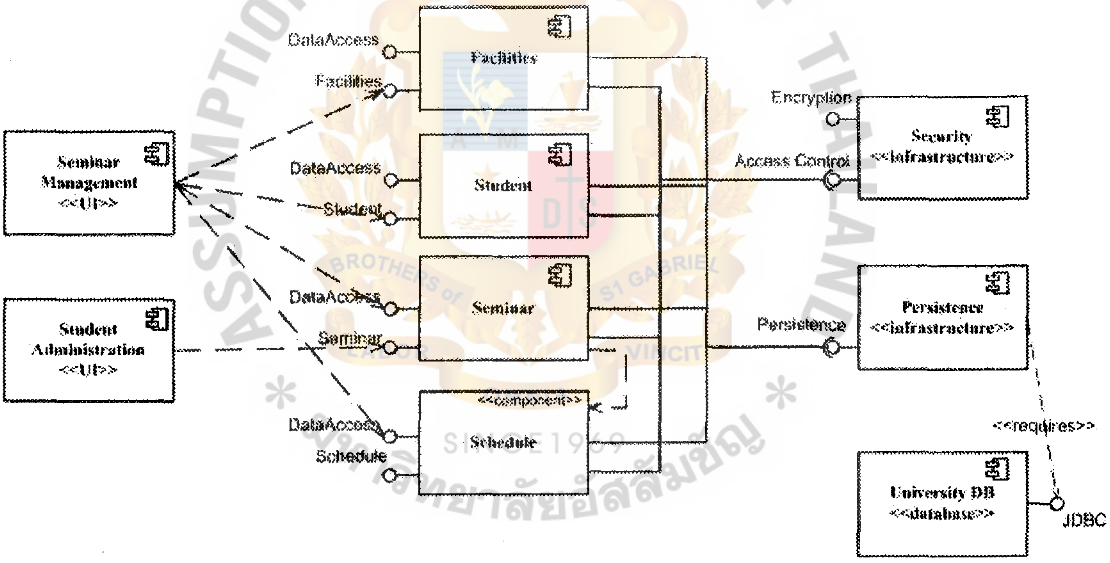


Figure 2.4. UML 2.x Component Diagram (Ambler, 2006).

Deployment diagram, the final type of structure diagram in this study, captures the configuration of run-time hardware nodes and corresponding software components that run on those nodes (Ambler, 2006). Nodes, such as server machines, operating systems, Web servers, application servers, and database servers represent the environment in which a component or set of components execute, and can be linked to demonstrate interdependencies (Chitnis et al. 2006). Basically, deployment diagrams display the hardware of the system being modeled, the software installed in that hardware, and the required middleware to connect disparate machines. Ambler (2006) suggests that deployment diagrams are appropriate in examining system installation, system dependencies with other systems, major deployment configurations, hardware/network infrastructure of an organization, and the hardware and software configuration of an embedded system. Figure 2.5 is an example of a deployment diagram.

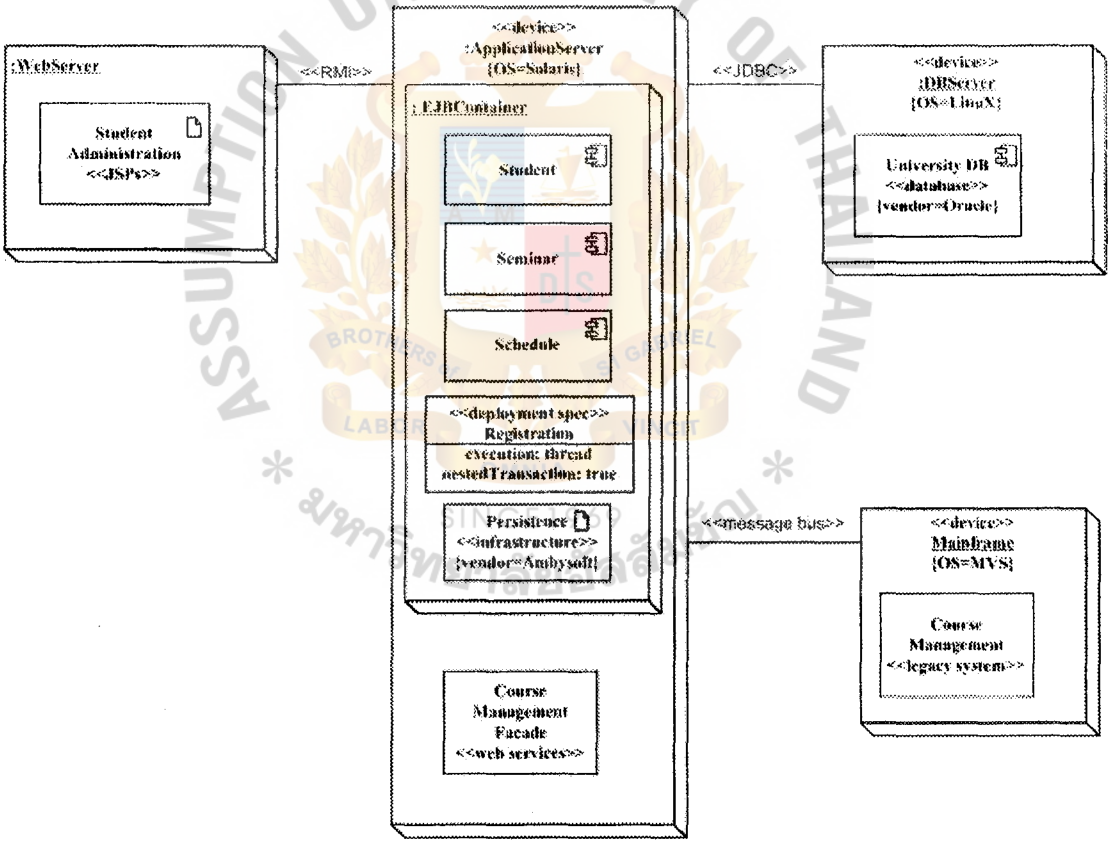


Figure 2.5. UML 2.x Deployment Diagram for University Information System (Ambler, 2006).

Object diagram, a derivative of the class diagram, is a snapshot of an entire or partial view of a system at a point in time (Rumbaugh et al. 1999). The diagram is named after the elements that it contains: objects, attributes, and associations between objects. The founders of UML concisely define an object, "A discrete entity with a well-defined boundary and identity that encapsulates state and behavior; an instance of a class" (Rumbaugh et al. 1999, p. 360). Each set of objects or instances created from a certain class retains similar structure, behavior, and relationships. Object diagrams are useful as illustrated examples of a system, test cases for class diagrams, or to observe how a system behaves through a series of snapshots over time. Figure 2.6 is an example of an object diagram.

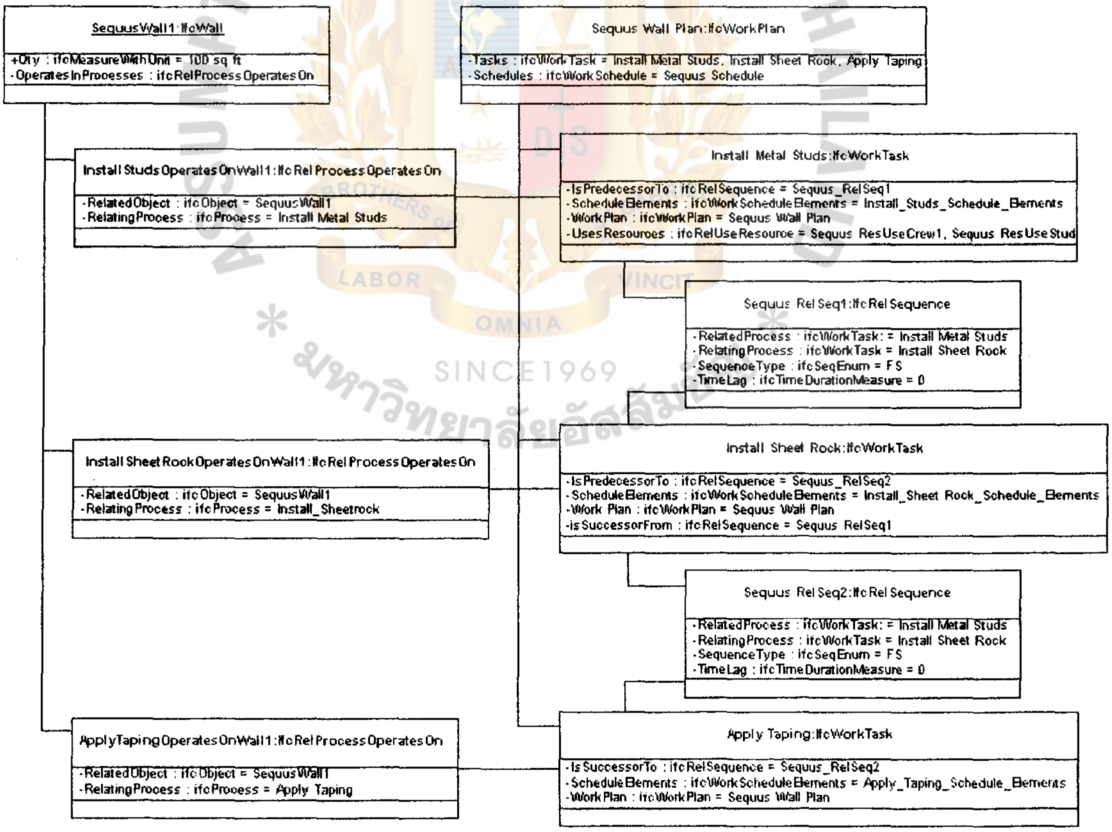

SequusWall1:IfcWall
+Qty : ifcMeasureWithUnit = 100 sq ft
-OperatesInProcesses : ifcRelProcessOperatesOn
Sequus Wall Plan:IfcWorkPlan
-Tasks : ifcWorkTask = Install Metal Studs, Install Sheet Rock, Apply Taping
-Schedules : ifcWorkSchedule = Sequus Schedule
InstallStudsOperatesOnWall1:IfcRelProcessOperatesOn
-RelatedObject : ifcObject = SequusWall1
-RelatingProcess : ifcProcess = Install Metal Studs
Install Metal Studs:IfcWorkTask
-IsPredecessorTo : ifcRelSequence = Sequus_RelSeq1
-ScheduleElements : ifcWorkScheduleElements = Install_Studs_Schedule_Elements
-WorkPlan : ifcWorkPlan = Sequus Wall Plan
-UsesResources : ifcRelUseResource = Sequus ResUseCrew1, Sequus ResUseStud
Sequus RelSeq1:IfcRelSequence
-RelatedProcess : ifcWorkTask: = Install Metal Studs
-RelatingProcess : ifcWorkTask = Install Sheet Rock
-SequenceType : ifcSeqEnum = FS
-TimeLag : ifcTimeDurationMeasure = 0
InstallSheetRockOperatesOnWall1:IfcRelProcessOperatesOn
-RelatedObject : ifcObject = SequusWall1
-RelatingProcess : ifcProcess = Install_Sheetrock
Install Sheet Rock:IfcWorkTask
-IsPredecessorTo : ifcRelSequence = Sequus_RelSeq2
-ScheduleElements : ifcWorkScheduleElements = Install_Sheet Rock_Schedule_Elements
-Work Plan : ifcWorkPlan = Sequus Wall Plan
-isSuccessorFrom : ifcRelSequence = Sequus RelSeq1
Sequus RelSeq2:IfcRelSequence
-RelatedProcess : ifcWorkTask: = Install Metal Studs
-RelatingProcess : ifcWorkTask = Install Sheet Rock
-SequenceType : ifcSeqEnum = FS
-TimeLag : ifcTimeDurationMeasure = 0
ApplyTapingOperatesOnWall1:IfcRelProcessOperatesOn
-RelatedObject : ifcObject = SequusWall1
-RelatingProcess : ifcProcess = Apply Taping
Apply Taping:IfcWorkTask
-IsSuccessorTo : ifcRelSequence = Sequus_RelSeq2
-ScheduleElements : ifcWorkScheduleElements = Apply_Taping_Schedule_Elements
-WorkPlan : ifcWorkPlan = Sequus Wall Plan

Figure 2.6. UML 1.x Object Diagram (Froese et al. 1999).

Package diagram, a type of structure diagram, models logical containers or packages and their relationships at a high-level overview (Sparx Systems, 2006). The diagram helps modularize a complex diagram and organize the source code (Ambler, 2006). Packages are commonly used to group use cases and classes, and can be built to represent either physical or logical relationships. All elements in a package share the same namespace as well as possess a unique identifier. Packages may be applied on any UML diagram, and they are easily recognized as they are drawn as file folders. Figure 2.7 is an example of a package diagram.

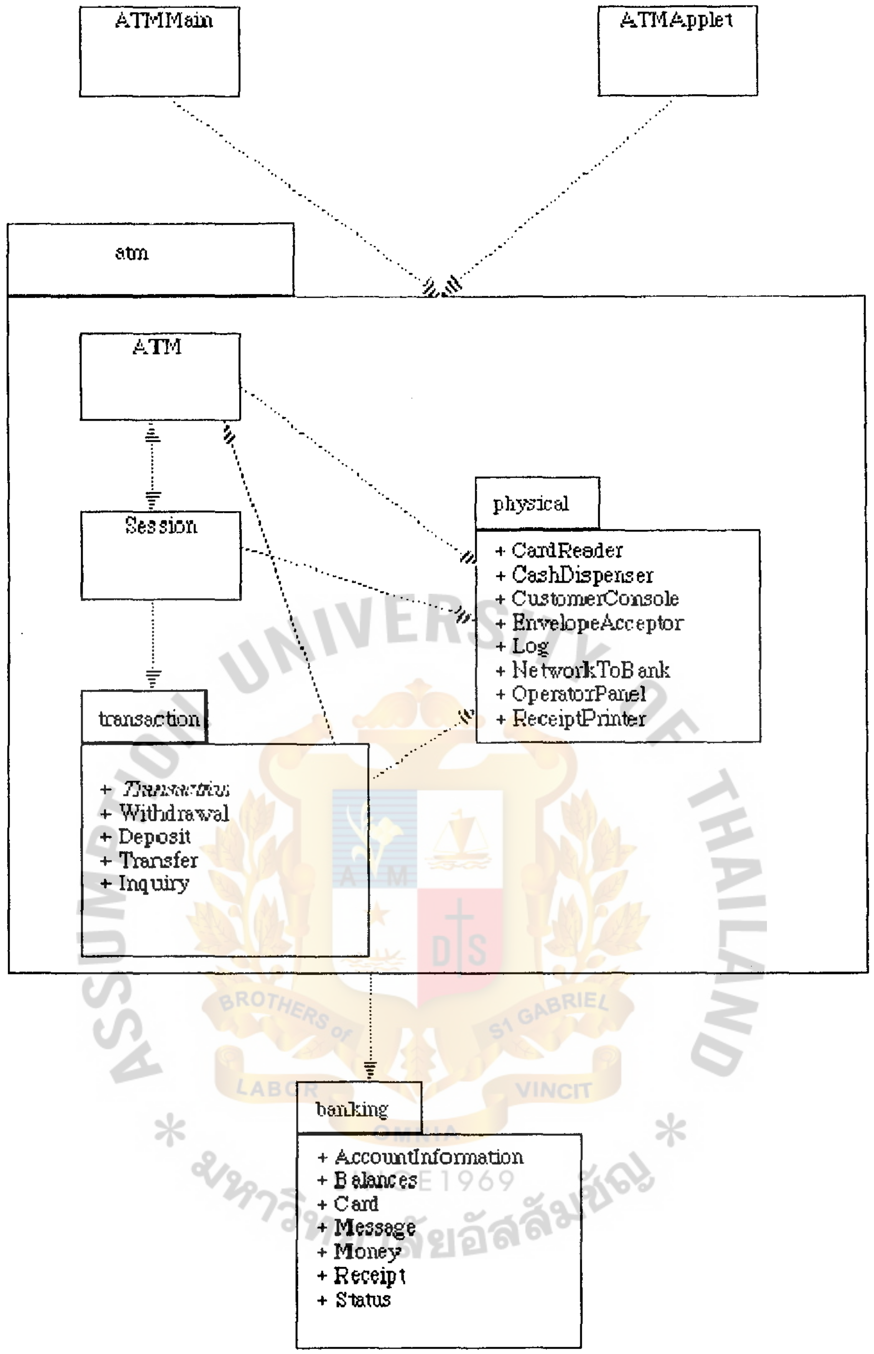


Figure 2.7. Package Diagram for Automated Teller Machine (ATM) (Bjork, 2001).

Activity diagram, a type of behavior diagram, demonstrates the progression of events in a system (Sparx Systems, 2006). From start to finish, decision paths as well as parallel processing may occur. A diagram may involve several use cases, a small portion of a use case, or no use cases at all. Ambler (2006) suggests that activity diagrams are, in many ways, the object-oriented equivalent of flow charts and data flow diagrams (DFDs). Activity diagrams typically model business processes, the logic captured by a single use case or usage scenario, or the detailed logic of a business rule (Ambler, 2006). Figure 2.8 is an example of an activity diagram.

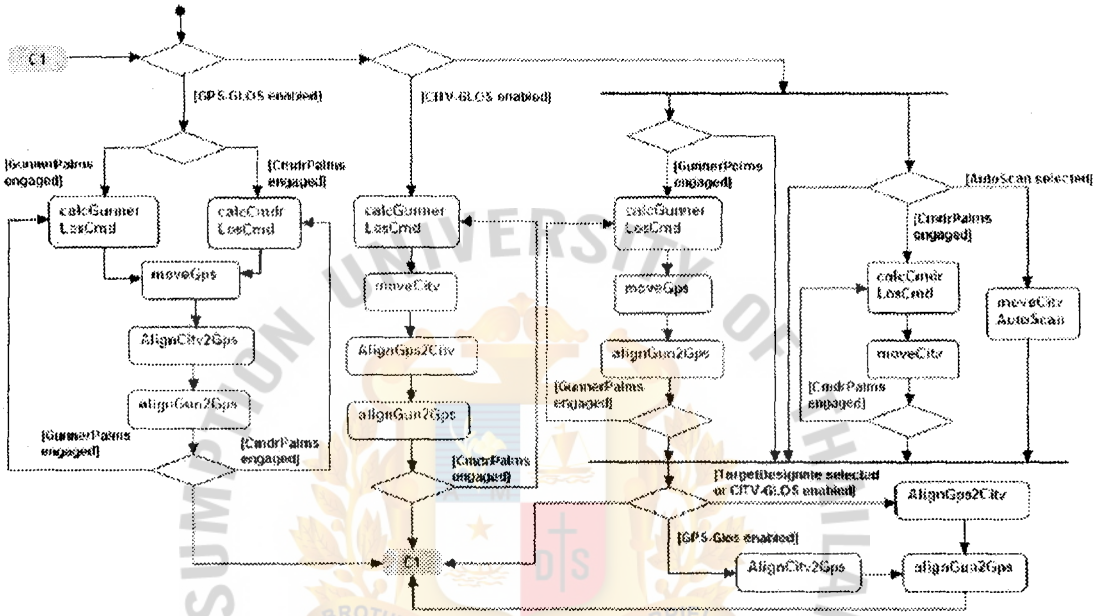


Figure 2.8. 'Black-Box' Activity Diagram (Hoffmann, 2005).

Use case diagram, a common behavior diagram, is used to identify the roles and discrete functionalities in a system. Roles are called actors, and functionalities are known as use cases. Despite its name, actors may assume any entity (or entities), from humans to another system. Actors interact with use cases, as use cases may share different kinds of relationships with one another. Chitnis et al. (2006) believe that use case diagrams help users: discover significant system characteristics, create great storyboard tool for user meetings, and write test scripts for the modeled system. Chitnis et al. (2006) also recommends that technical staffs need not participate in creating use cases. Figure 2.9 is an example of a use case diagram.

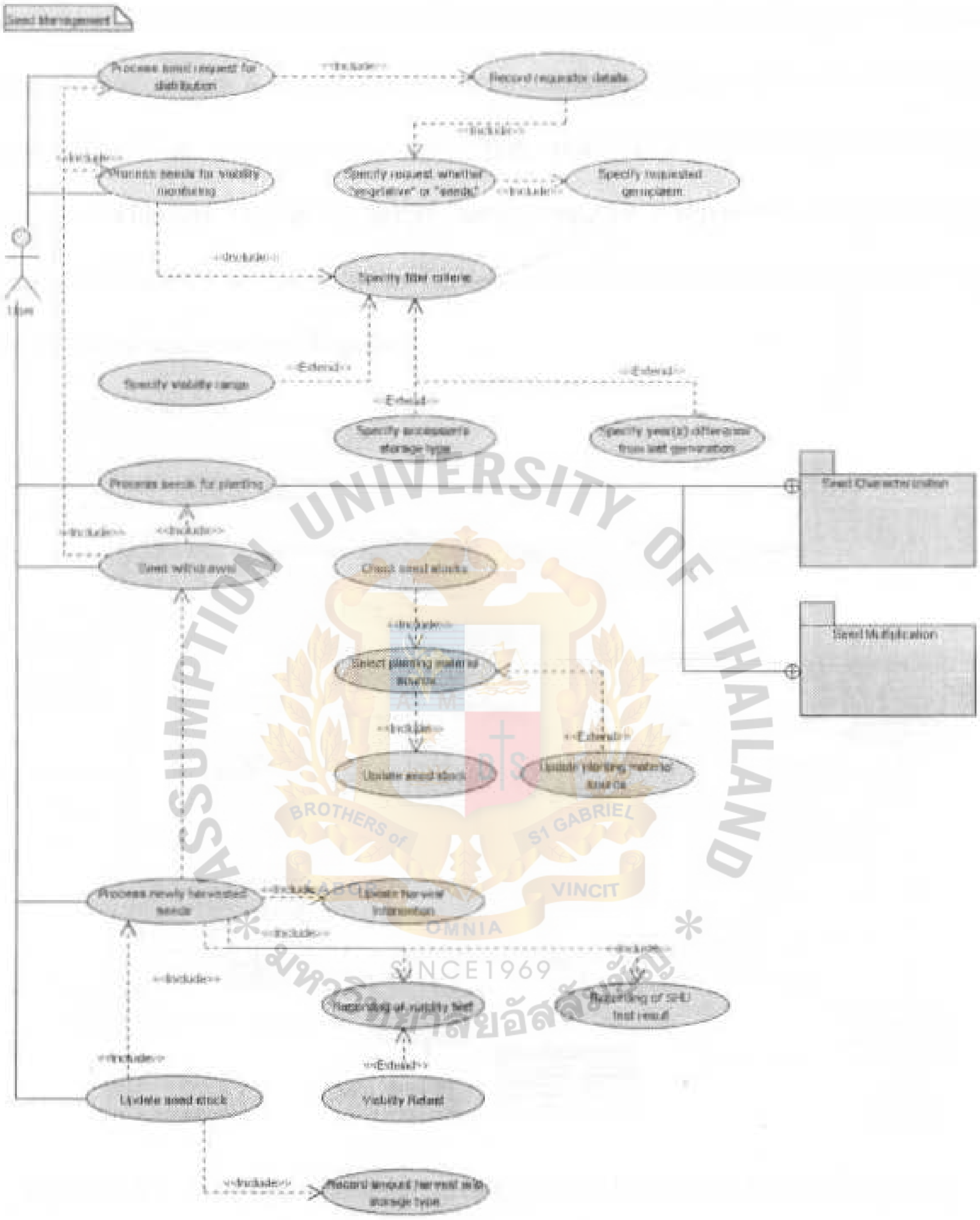

Seed Management
Process (seed) request for distribution
<<Include>>
Record requestor details
Specify request whether "vegetative" or "seeds"
Specify requested germplasm
Process seeds for viability monitoring
Specify filter criteria
User
Specify viability range
<<Extend>>
Specify accession's storage type
Specify year(s) difference from last generation
Process seeds for planting
Seed Characterization
Seed withdrawal
Check seed stocks
Seed Multiplication
Select planting material source
Update seed stock
Update planting material source
Process newly harvested seeds
Update harvest information
Recording of viability test
Recording of SHU test result
Viability Retest
Update seed stock
Record amount harvest and storage type

Figure 2.9. "Seed Management" Use Case Diagram for Crop Industry (Rherrera, 2005).

State machine diagram, formerly known as state chart diagrams in UML 1.x, illustrates how an object responds to various events depending on its current state

(Ambler, 2006). A state is described as a stage in an object's behavior pattern, and there can be initial and final states. Also called a creation state, an initial state is where an object is first created. A final state is where no transitions lead out of it. A transition is a progression from one state to another. State machine diagrams help comprehend complex behaviors of classes, actors, subsystems, or components. Ambler (2006) recommends state machine diagrams for modeling real-time systems. Figure 2.10 is an example of a state machine diagram.

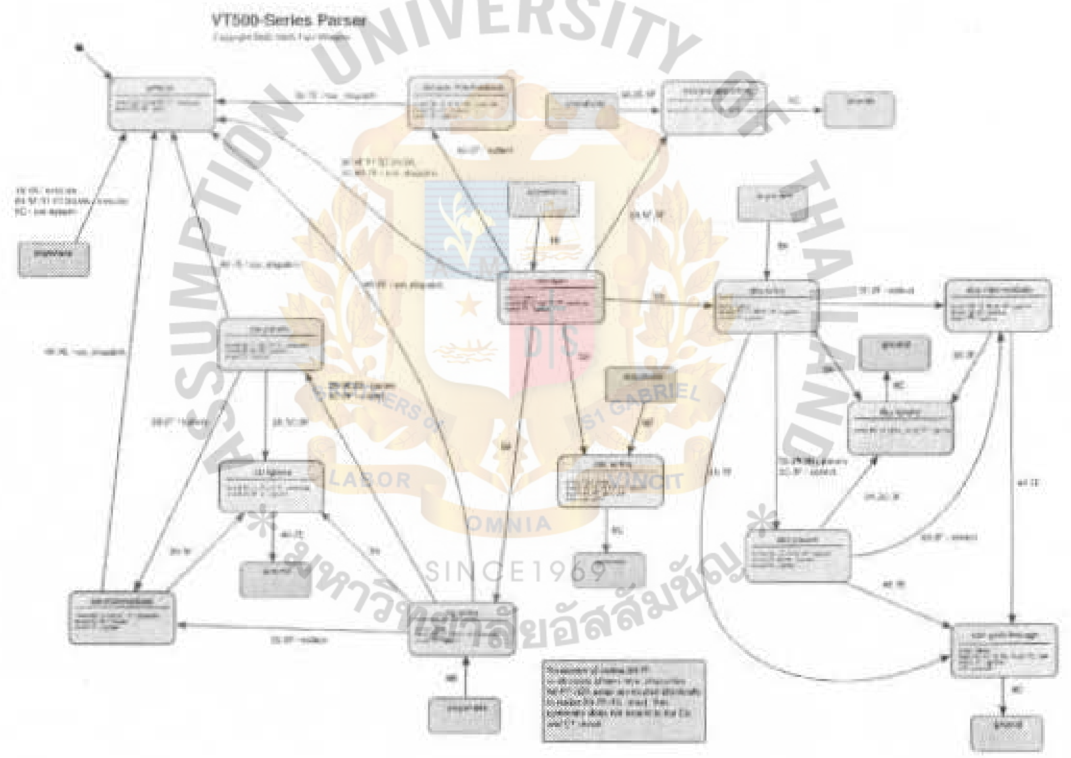


Figure 2.10. State Machine Diagram of Parser for Escape and Control Sequences (Williams, 2005).

Sequence diagram, a prominent interaction diagram, displays two primary items: objects drawn as vertical lifelines, and their interactions drawn as horizontal arrows spanning from the source lifeline to the target lifeline (Sparx Systems, 2006). As the diagram is read from top to bottom, interactions appear over time in the order in which they occur. In a nutshell, sequence diagrams model communications between objects and the messages that trigger those communications. Sequence diagrams are inappropriate for modeling complex procedural logic as well as large number of objects. UML 2.0 sequence diagrams retain similar notations from its predecessors with added support for modeling variations to the standard flow of events. Figure 2.11 is an example of a sequence diagram.

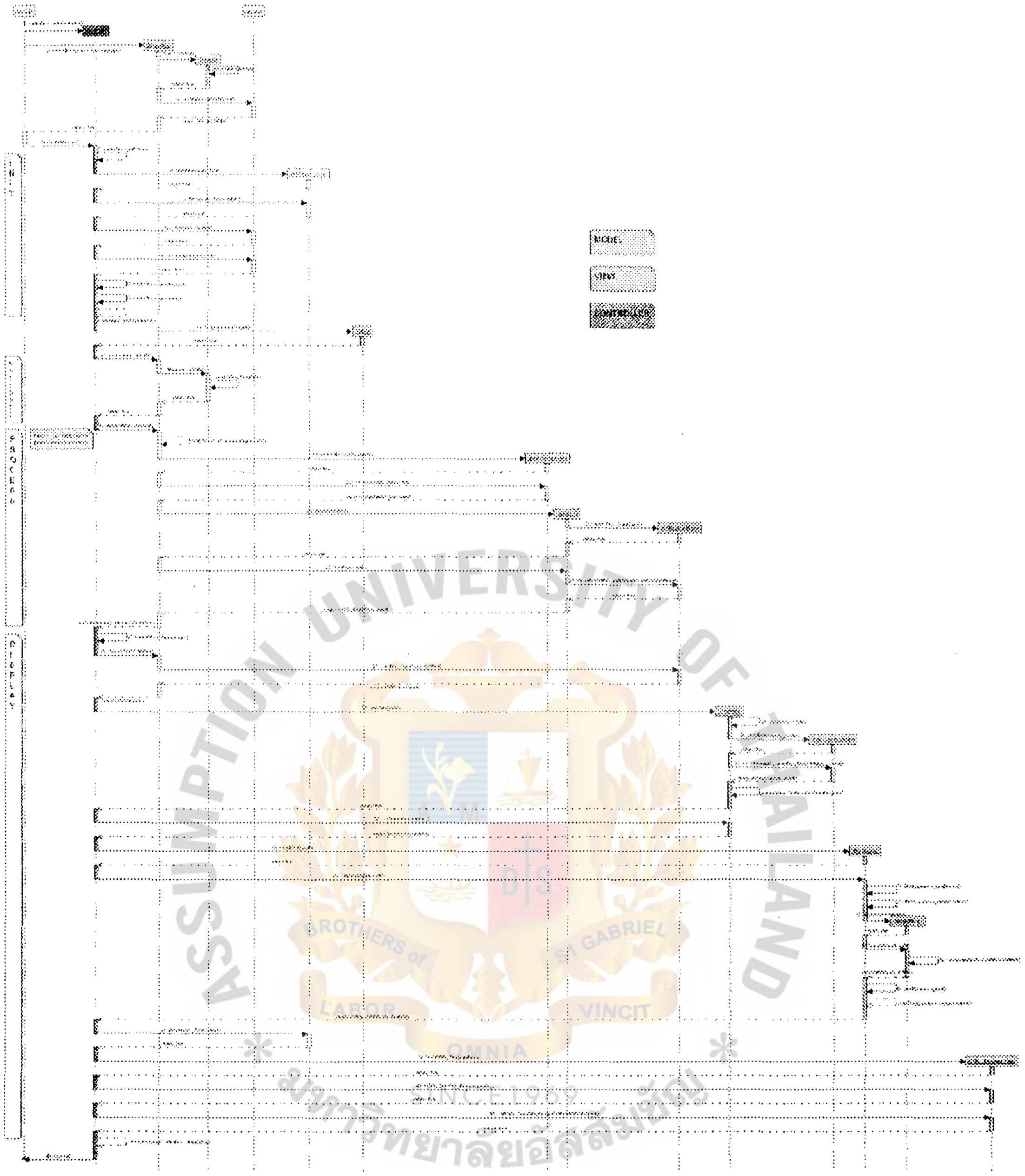


Figure 2.11. Sequence Diagram of Seagull, an OOP Framework (Turner, 2006).

Communication diagram, formerly known as the collaboration diagram in UML 1.x, is the second interaction diagram in this study. Providing similar information to sequence diagrams, communication diagrams focus on object relationships via sequenced messages coupled by arrows pointing in the direction of the message flow (Sparx Systems, 2006). Although communication and sequence diagrams can be transformed into one another, it is not immediately visible. In contrast, it is evident that elements in a communication diagram are arranged in a free-form manner. To read the diagram, always start from message 1.0 and follow the chronological messages from object to object. Figure 2.12 is an example of a communication diagram.

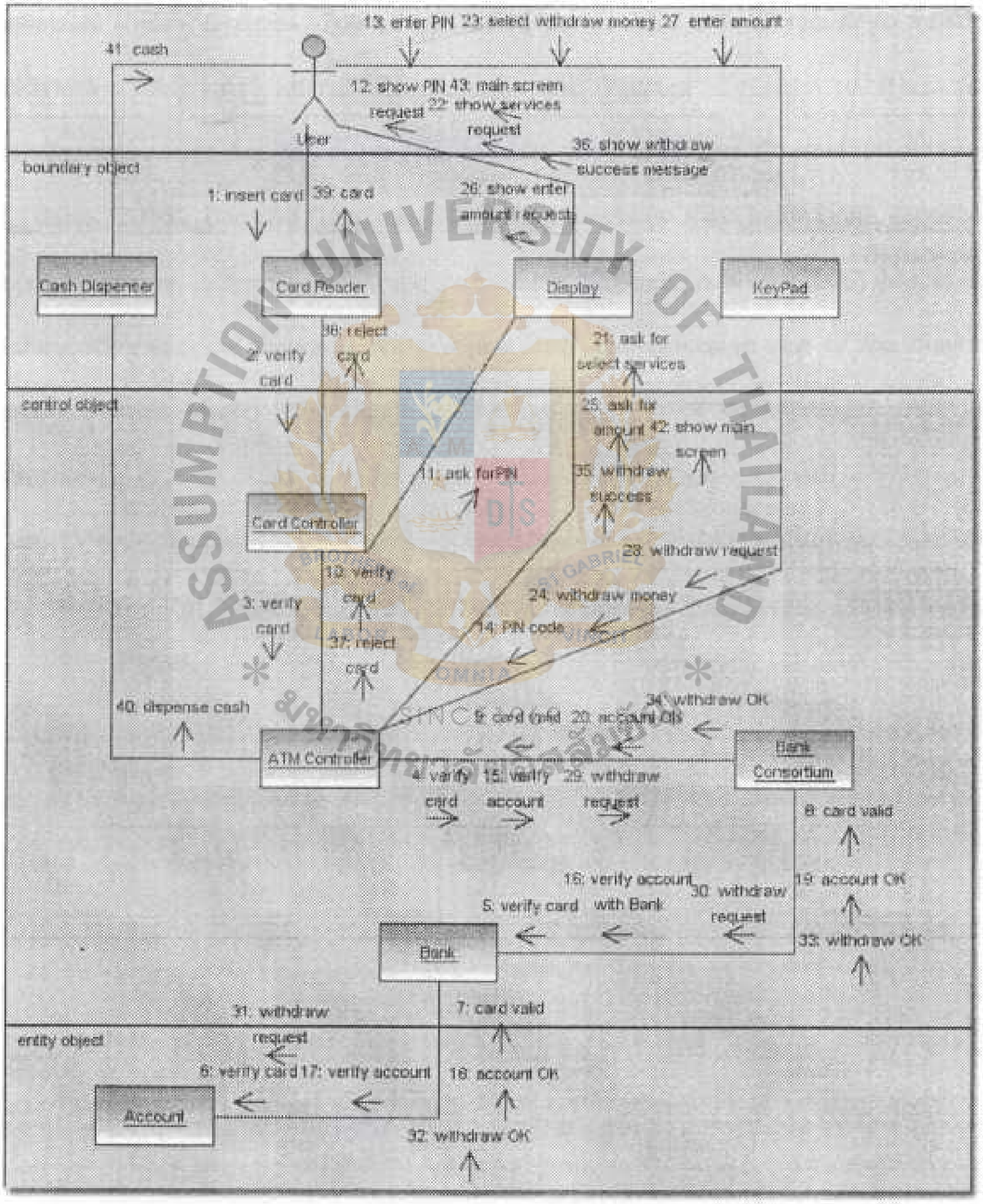


Figure 2.12. Communication (Collaboration) Diagram (Wright, 2005).

Interaction overview diagram, another type of interaction diagram, is closely related to the activity diagram. Although many notations from activity diagrams such as initial, final, decision, merge, fork, and join nodes are shared by interaction overview diagrams, two new elements are introduced: interaction occurrences and interaction elements (Sparx Systems, 2006). Interaction occurrences are references to existing interaction diagrams. Interaction elements are depicted similarly to interaction occurrences except that they display the contents of the references diagram inline (Sparx Systems, 2006). Because interaction overview diagrams may incorporate sequence, communication, interaction overview, and timing diagrams, Ambler (2006) doubts that interaction overview diagrams will be practiced. The extensive size of the diagram easily outruns whiteboards. Figure 2.13 is an example of an interaction overview diagram.

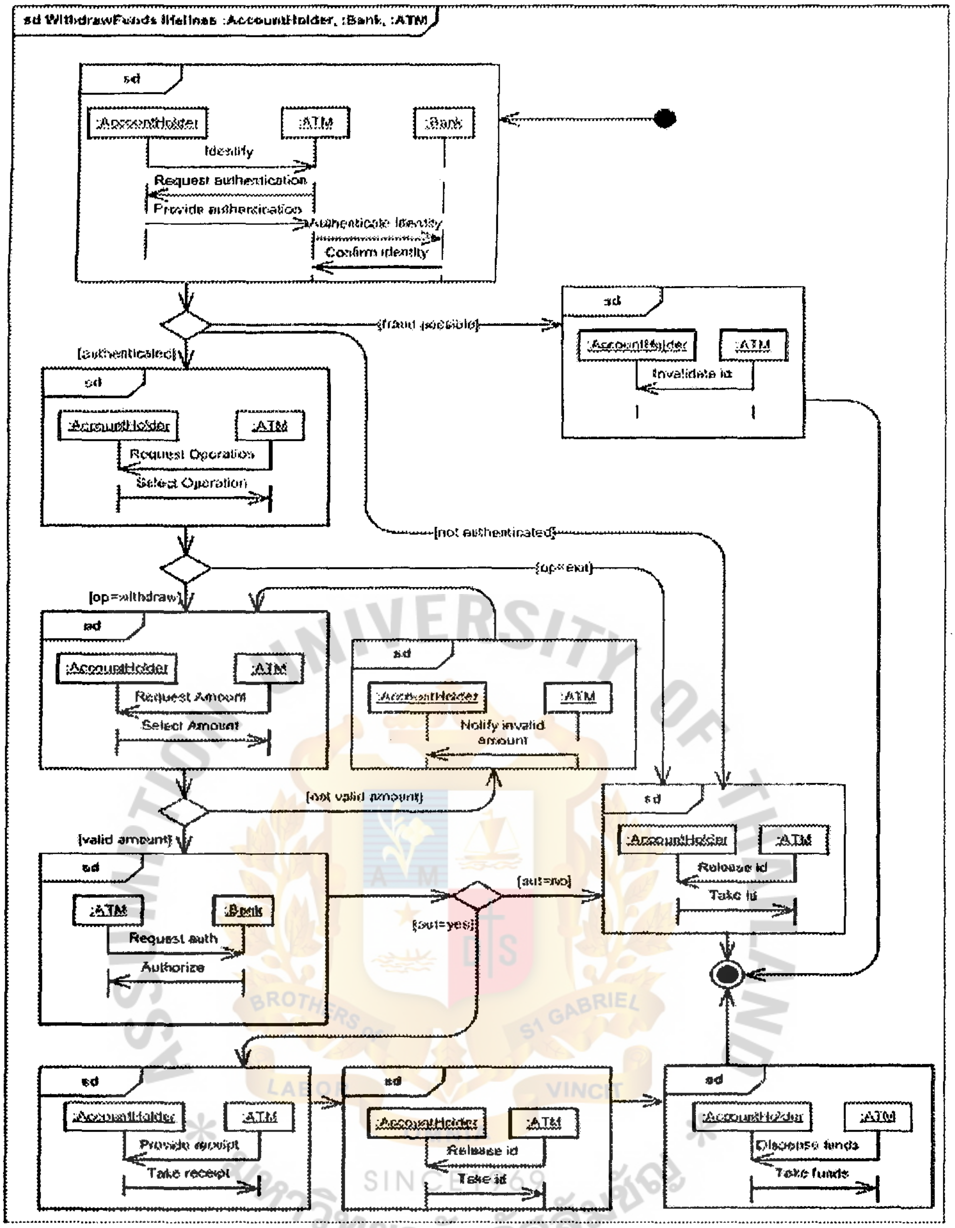


Figure 2.13. Interaction Overview Diagram for Use Case Withdraw Funds (García et al. 2005).

Timing diagram, a new interaction diagram recently added to UML 2.x, shows the change in state or value of one or more objects throughout a given period of time. Moreover, interactions between timed events can also be shown, as well as corresponding time and duration constraints (Sparx Systems, 2006). A timing diagram is drawn with a Y-axis labeled with a list of states, an X-axis labeled with time in a chosen unit, state lifelines to show object state changes over time, and value lifelines to show object value changes over time. State and value lifelines can be stacked in any combination, and lifelines may send messages to another. Because it is difficult to model multiple lifelines at once, Ambler (2006) recommends two lifelines per diagram. He also observes that timing diagrams are often used to design embedded software as well as occasional uses in business software. Figure 2.14 is an example of a timing diagram.

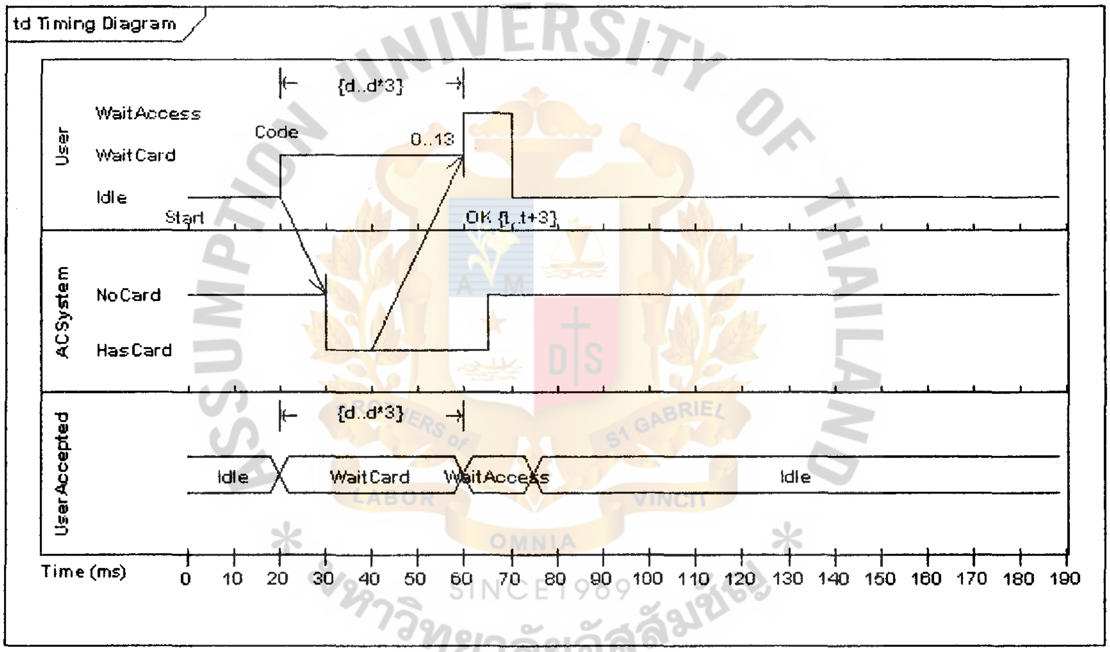


Figure 2.14. Timing Diagram (Sparx Systems, 2006).

2.1.3 Criticisms

UML is enormous. As a result, this requires huge efforts to master the language. The Object Management Group (2006) openly admits that UML 2.x is a "large specification", consisting of four parts: Superstructure, Infrastructure, Object Constraint Language (OCL), and Diagram Interchange. The Superstructure is straightforward – it is a 700+ page document defining the thirteen diagram types and elements that comprise them. The Infrastructure is a 210+ page document defining base classes that form the foundation for the Superstructure and Meta Object Facility (MOF) 2.0. The OCL is a 180+ page document allowing setting of pre- and post-conditions, invariants, and other conditions, and the Diagram Interchange is an 80+ page document providing a supplementary package for graph-oriented information, which allows models to be exchanged or stored/retrieved and then displayed as they were originally. Moreover, the Object Management Group (2006) also documents UML Profile for CORBA®, CORBA Component Model (CCM), Enterprise Application Integration (EAI), Enterprise Distributed Object Computing (EDOC), QoS and Fault Tolerance, Schedulability, Performance, and Time, Testing, and one related specification: the UML Human-Usable Textual Notation (HUTN). Finally, object-oriented knowledge and perhaps, experience are prerequisites for any UML starter.

UML is still changing. This is extremely costly as an army of published materials, especially books, web pages, tools, and projects, need to be revised, rewritten, or redrawn. The Object Management Group (2006) acknowledges that there are more than a 100 book titles on UML. The group has also accumulated more than 40 links on UML information and tools. Inevitably, UML users and management require careful selection of vast UML resources – including old and new, more training to learn both the language and tools, and if necessary, recertification from the OMG Certified UML Professional program for new versions of UML. Botting (2006) provides several publications illustrating concerns for UML changes and ambiguity. He also provides comments made by Rick Bruner, "The OMG has engaged in a courageous effort to keep the size of UML 2.0 to a minimum. Despite the fact that it has retired 25 predefined elements and removed 28 other features, it ADDED 56 NEW features and 5 NEW diagram types (Object, Package, Interaction Overview, Timing, and Composite

Structure Diagrams). The UML documentation went from slightly over 600 pages to a little over 800 pages!"

UML is complex. Although the UML specifications can be downloaded for free, the Object Management Group (2006) agrees that "it's also highly technical, terse, and very difficult for beginners to understand." OMG argues that the specifications are the formal definition of UML, and are written for programmers who implement compliant software products. The specifications, however, are "*not* an instruction book on "How to Model Using UML"" (Object Management Group, 2006), and not intended for application developers and users (Siegal, 2005). OMG recommends that modelers should start with online tutorials, and apply for training from OMG member companies, or purchase a book on modeling with UML.

UML is source code related. This stems from the fact that, given details are in agreement, some UML-based tools are able to compile UML diagrams to generate codes or vice versa. This is a problem if UML diagrams are presented to non-technical software buyers. More confusion can be expected if UML is used to model software written in OO programming languages. Developers will try to conform to OO concepts as accurately as possible for their convenience in code generation.

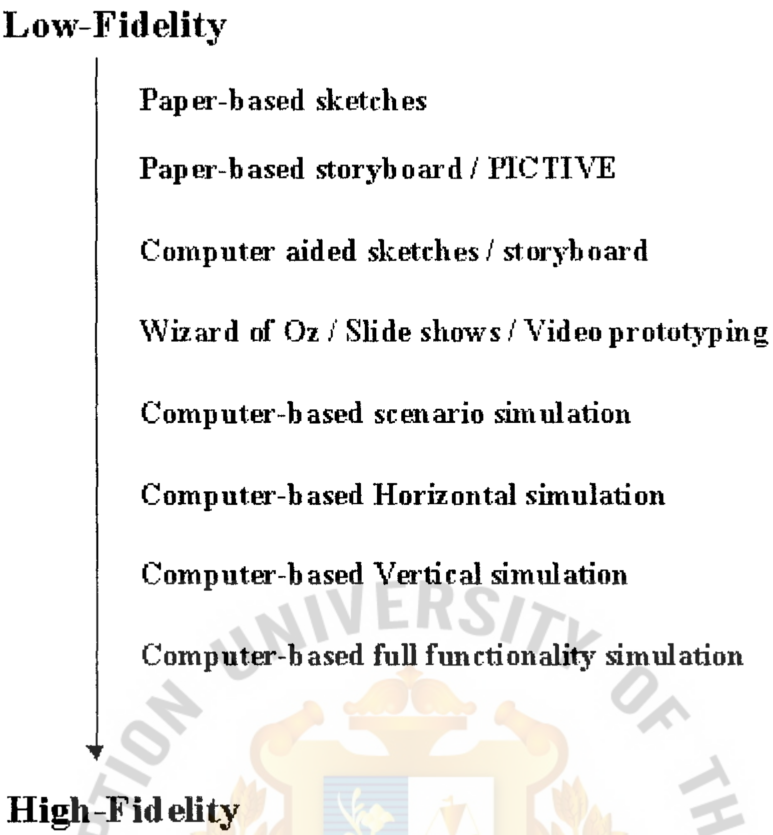
Grady Booch, one of the original developers of UML, asserts that it takes more than blind adoption and usage of UML for its sake to achieve success (Booch, 2004). As Booch acknowledges that many organizations have not enjoyed successes from using UML, he maintains that success with the UML must be coupled with thought, planning, and understanding of UML's purpose, limitations, and strengths. The rush to embrace UML is actually generating cost and schedule trauma on many current software projects. Booch (2004) observes that many organizations simply misunderstand UML by modeling every single line of code, representing a front-end syntax to define the context for comprehensive simulations, drawing insignificant diagrams, and replacing the software development process.

2.2 Software Prototypes

A prototype is a preliminary and incomplete design, serving as a basis from which other designs are developed or copied. In the software industry, software prototypes are created for future full-featured software programs. Software prototypes may allow developers to test the feasibility of certain technical aspects of a system (Carr and Verner, 1997), or allow users to provide feedback for early ideas of the completed software program or portions of it. Users may observe whether or not requirements are met, detect potential errors, or make critical changes or additions before proceeding to full-scale implementations. According to Lu (1998), software prototypes should be cheap and quickly developed. Hoffer et al. (2002) adds that software prototypes are the product of iterative development processes that convert requirements to a working system through close collaboration between analysts and users. This can help avoid the great expense and difficulty in making significant changes to a mature software program during the final stages of software development.

Carr and Verner (1997) agree that there is much confusion in the software prototyping literature, especially in the field of prototyping terminologies. Various researchers attempt to name different prototypes designed to achieve different goals, as well as different prototyping processes and strategies to construct those prototypes. For instance, Carr and Verner (1997) exemplify with Budde et al.'s four types of prototypes: presentation prototypes, prototype proper, breadboard prototypes, and pilot system, which are constructed using either horizontal or vertical prototyping techniques. Software prototypes must be distinguished from the prototyping processes that create them, since software prototypes are the actual agents that provide the communication link between developers and buyers. This research classifies software prototypes according to their fidelity.

Prototype fidelity concerns "the degree to which the prototype accurately represents the appearance and interaction of the product, not the degree to which the code and other attributes invisible to the user are accurate" (Lu, 1998). There are three common degrees of prototype fidelity: low, medium, and high. At one end, low-fidelity prototypes involve chiefly paper-based mock-ups. They are intended to demonstrate the user interface, and require a knowledgeable facilitator to perform the run-through. Sketches, storyboards, and PICTIVE mock-ups are examples of low-fidelity prototypes, which will be further discussed and exemplified. On the other end, high-fidelity prototypes are computer-based simulations of much of the system functionalities. By possessing user interfaces that closely resemble the final product, high-fidelity prototypes are not as quick and easy to create as low-fidelity prototypes. Software prototyping tools and current programming languages may help ease computer-based prototyping. Situated in between the two extremes, medium-fidelity prototypes simulate some system interactions and functionalities. Medium-fidelity prototypes are largely about the approaches to limit or test prototype interactions and functionalities. Figure 2.15 displays the transition of prototyping techniques.



Low-Fidelity

Paper-based sketches

Paper-based storyboard / PICTIVE

Computer aided sketches / storyboard

Wizard of Oz / Slide shows / Video prototyping

Computer-based scenario simulation

Computer-based Horizontal simulation

Computer-based Vertical simulation

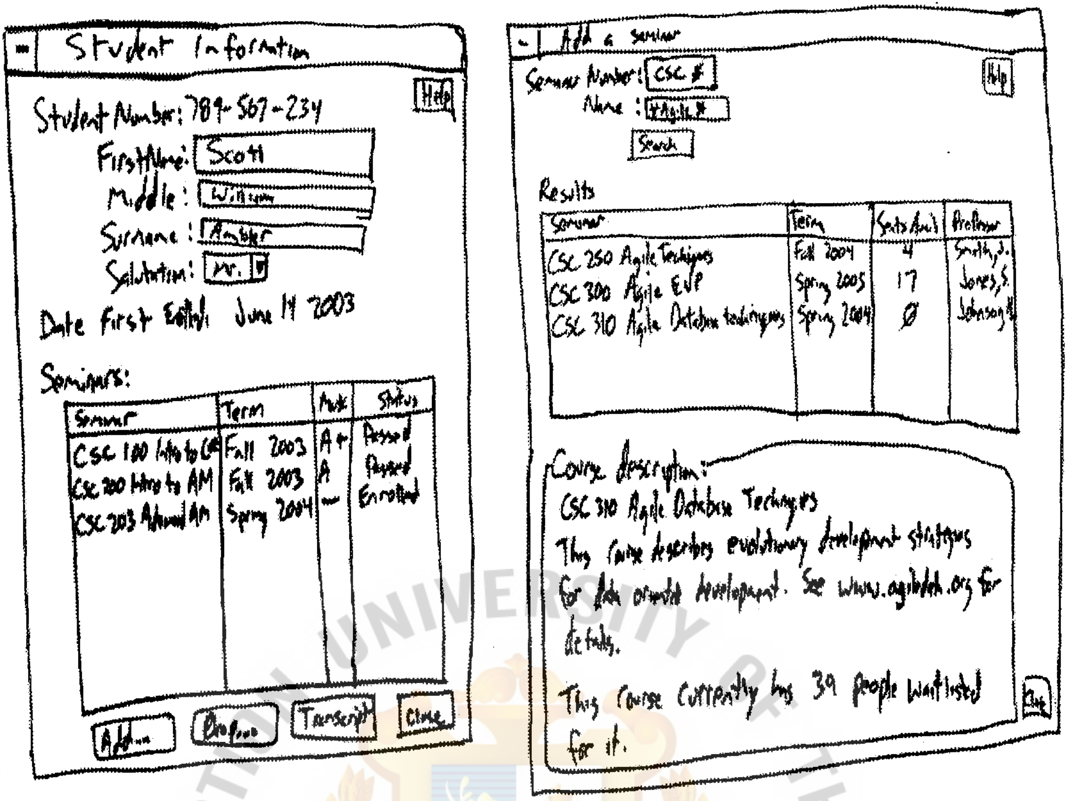
Computer-based full functionality simulation

High-Fidelity

Figure 2.15. Transition of Prototyping Techniques (Lu, 1998).

2.2.1 Sketches

Sketches are quick drawings. Utilizing paper-and-pencil to illustrational software applications, sketches of the software user interfaces are useful for exploring all kinds of design ideas (Lu, 1998). Sketches allow developers to see new ways to refine and revise current ideas in a very simple and cheap manner. However, sketches portray only high level concepts, and are inefficient in showing system progression. Figure 2.16 is an example consisting of two screen sketches.



Student Number: 789-567-234

First Name: Scott

Middle: William

Surname: Ambler

Salutation: Mr.

Date First Enrolled: June 14 2003

Seminars:

Seminar	Term	Mark	Status
CSC 100 Intro to CS	Fall 2003	A+	Passed
CSC 200 Intro to AM	Fall 2003	A	Passed
CSC 203 Advanced AM	Spring 2004	—	Enrolled

Seminar Number: CSC #

Name: *Agile*

Results

Seminar	Term	Seats Avail	Professor
CSC 250 Agile Techniques	Fall 2004	4	Smith, J.
CSC 300 Agile EUP	Spring 2005	17	Jones, S.
CSC 310 Agile Database techniques	Spring 2004	Ø	Johnson, H.

Course description:

CSC 310 Agile Database Techniques

This course describes evolutionary development strategies for data oriented development. See www.agiledata.org for details.

This course currently has 39 people waitlisted for it.

Figure 2.16. Two Screen Sketches (Ambler, 2006).

Ambler (2006) presents a prototype similar to sketches called the essential user interface (UI) prototype. It represents the general ideas behind the user interface in a technology-independent manner by using whiteboards, flip-chart paper, and sticky notes. The essential user interface prototype also differs from traditional user interface prototyping by focusing on users and their usage of the system, not system features (Ambler, 2006). Figure 2.17 is an example of an essential UI prototype.

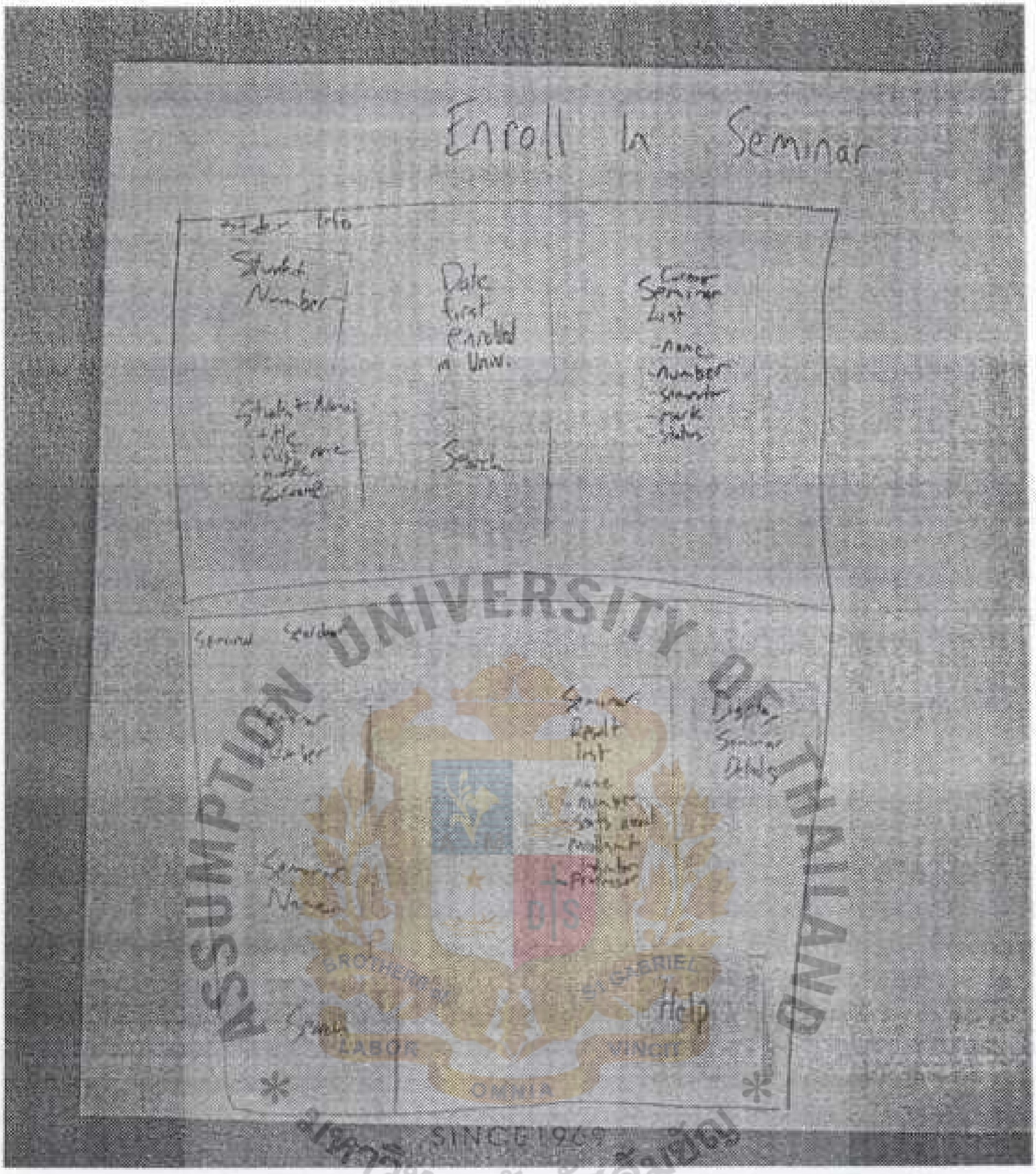


Figure 2.17. Essential UI Prototype for Enrolling in Seminars (Ambler, 2006).

2.2.2 Storyboards

Originating from the film industry, a storyboard is a "graphical depiction of the outward appearance of the intended system without accompanying system functionality" (Lu, 1998). It provides snapshots, with corresponding annotations, of the user interface at different points in an interaction. Storyboards can be constructed using office stationery to modern graphical drawing packages. According to Dalbey (2002), illustrating each interface screen on separate pieces of paper is cheap, fast, and very effective in terms of value gain versus construction time. Figure 2.18 is an example of a storyboard.

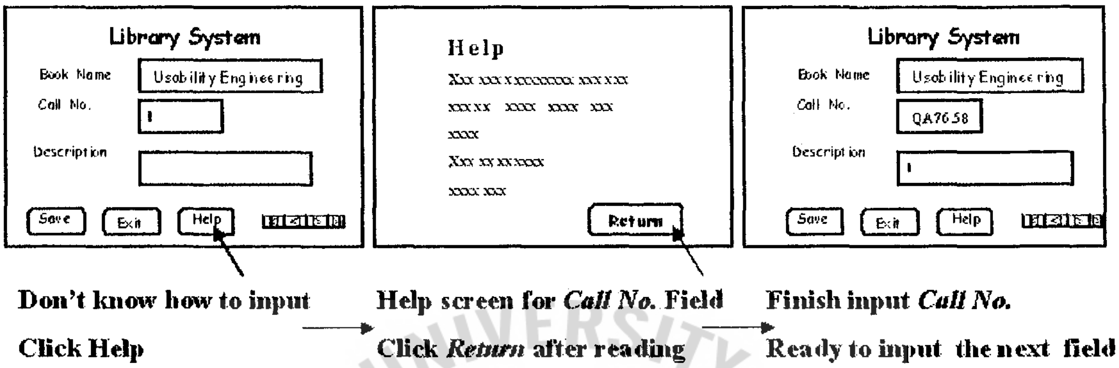


Figure 2.18. Storyboard Representing System Function and Sequence (Lu, 1998).

2.2.3 PICTIVE

PICTIVE (Plastic Interface for Collaborative Technology Initiatives through Video Exploration) is a participatory system design technique developed by Bell Communications Research (Bellcore) in 1990 (Lu, 1998). In a nutshell, the technique makes video and voice recordings of users modifying PICTIVE mock-ups of a system, which are used to determine how the system will look and behave. PICTIVE mock-ups consist of two categories of design objects. The first category is a colorful assortment of simple office materials (e.g. pens, paper, clips, etc), and the second is materials prepared by the developer (e.g. plastic icons of menu bars, dialogue boxes, etc). Figure 2.19 shows some PICTIVE plastic icons, and Figure 2.20 depicts a scene in a PICTIVE session.

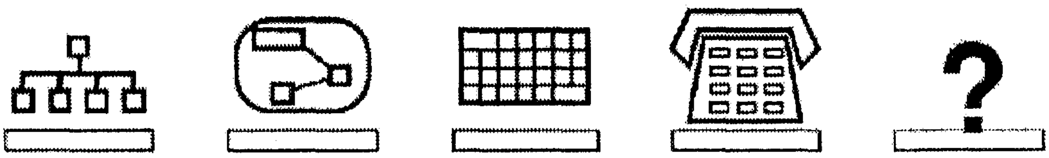


Figure 2.19. Some PICTIVE Plastic Icons (Lu, 1998).

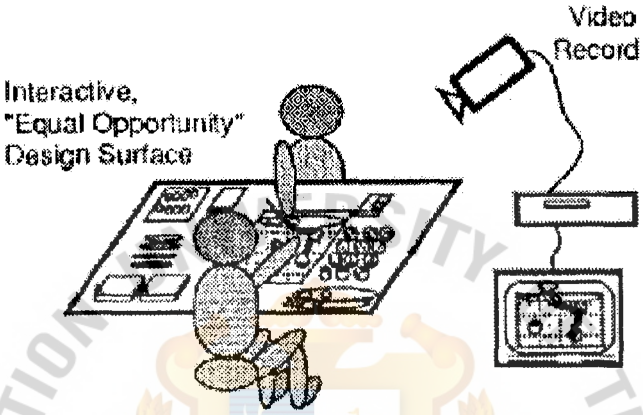
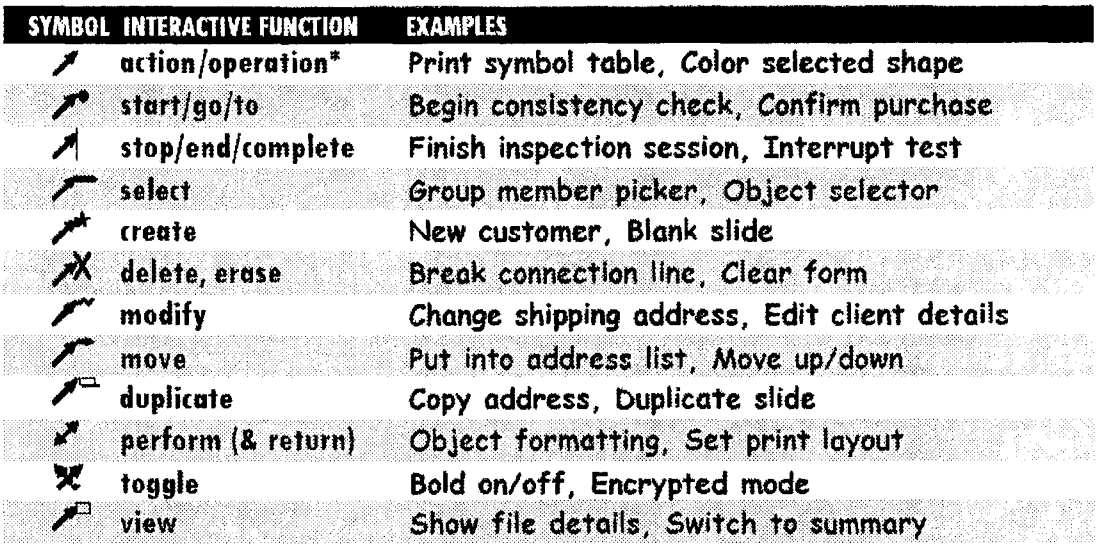


Figure 2.20. Scene in PICTIVE Session (Lu, 1998).

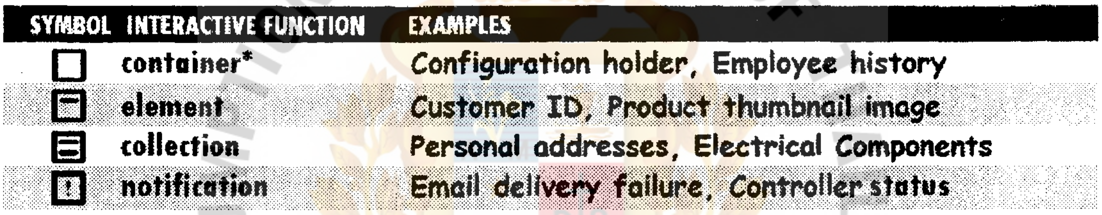
2.2.4 Canonical Abstract Prototypes

Canonical abstract prototypes are designed to bridge between abstraction and realization in user interface design (Constantine, 2003). It is intended to be less abstract and more precise to resolve design issues in very complex user interfaces. Canonical abstract prototypes are constructed from sets of symbols, each with a specific interactive function. Moreover, the symbols also model the position, size, layout, and composition of the user interface features. There are basically two types of symbols: a generic tool or action (Figure 2.21), and a generic material or container (Figure 2.22) – plus extensions and combinations between the two (Figure 2.23). Figure 2.24 is an example of a canonical abstract prototype.



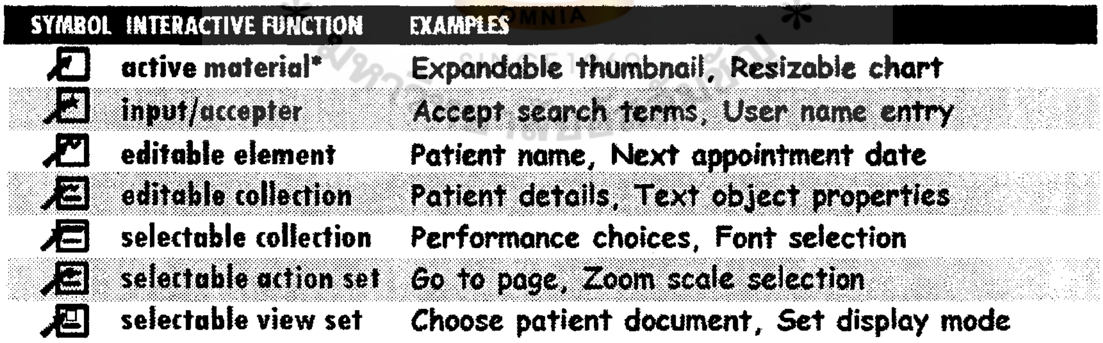
SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	action/operation*	Print symbol table, Color selected shape
	start/go/to	Begin consistency check, Confirm purchase
	stop/end/complete	Finish inspection session, Interrupt test
	select	Group member picker, Object selector
	create	New customer, Blank slide
	delete, erase	Break connection line, Clear form
	modify	Change shipping address, Edit client details
	move	Put into address list, Move up/down
	duplicate	Copy address, Duplicate slide
	perform (& return)	Object formatting, Set print layout
	toggle	Bold on/off, Encrypted mode
	view	Show file details, Switch to summary

Figure 2.21. Generic Tool or Action (Constantine, 2003).



SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	container*	Configuration holder, Employee history
	element	Customer ID, Product thumbnail image
	collection	Personal addresses, Electrical Components
	notification	Email delivery failure, Controller status

Figure 2.22. Generic Material or Container (Constantine, 2003).



SYMBOL	INTERACTIVE FUNCTION	EXAMPLES
	active material*	Expandable thumbnail, Resizable chart
	input/accepter	Accept search terms, User name entry
	editable element	Patient name, Next appointment date
	editable collection	Patient details, Text object properties
	selectable collection	Performance choices, Font selection
	selectable action set	Go to page, Zoom scale selection
	selectable view set	Choose patient document, Set display mode

Figure 2.23. Extensions and Combinations Between Generic Actions and Containers (Constantine, 2003).

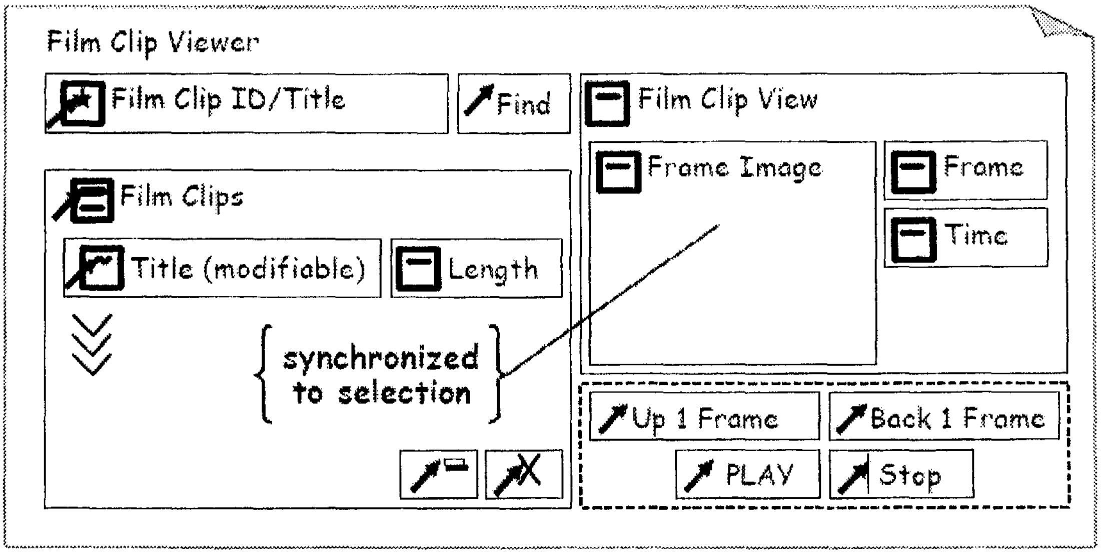


Figure 2.24. Canonical Abstract Prototype with Key Notational Elements (Constantine, 2003).

2.2.5 User Interface Flow Diagrams

The user interface flow diagram allows stakeholders to model high-level relationships and interactions between major user interface elements of a system (Ambler, 2006). Also called storyboards, interface-flow diagrams, windows navigation diagrams, and context-navigation maps, the user interface flow diagram uses boxes to represent major user interface elements, and arrows to represent the flow between them. The diagram may span a single use case to model the interactions that users have with a system, or several use cases to gain a high-level overview of a system's user interfaces. Figure 2.25 is an example of a user interface flow diagram.

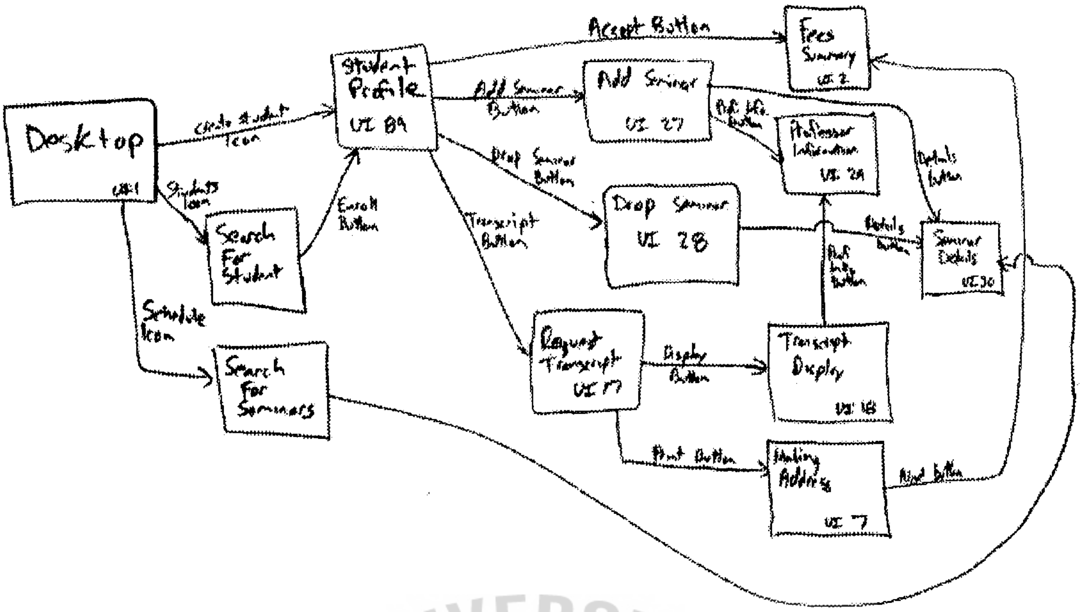


Figure 2.25. User Interface Flow Diagram for University System (Ambler, 2006).

2.3 Filmmaking

The film or movie industry is a vigorous multi-billion dollar business. A film can be translated into multiple languages allowing its customer base to span the entire world. Films can also be sold and resold, as the shelf life of a timeless film is virtually endless. Such lucrative possibilities invoke astronomical investments, which includes the willingness to spend extensive periods of time, massive funds and effort, and employ diverse teams of professionals. The entire complexity of filmmaking, however, sources from two vital pieces of compositions that serve as the foundation for a film: screenplays and storyboards.

2.3.1 Film Screenplay

A screenplay is a written description of a film. Also known as a script, it is composed in the earliest stage of filmmaking: the Development Stage. In this stage, an idea or story is developed into a working script. Based on a concept, previous literary compositions or films, or an original work in itself, a screenplay is a vital blueprint of the film and may comprise of everything from character names, descriptions, dialogue and actions, to scene descriptions. Although there are no single format for writing a screenplay, Craig (2005) argues that screenplays must be kept very simple since any investor, in or out of the film industry, a banker, or lawyer must be able to understand the story without having to learn another special language. As thousands of screenplays are circulated at any given time, a writer should capture the imagination of readers in the first few pages than scare them off with jargons and cluttered complex scripts. Leave the production versions of a screenplay, such as shooting scripts, to production staffs. Alice (2001) provides the following example of a screenplay:

▼▼▼▼▼▼▼▼▼ Start screenplay example ▼▼▼▼▼▼▼▼▼

10. INT. ALBERT'S LIVING ROOM. DAY

Seventy-year-old ALBERT JOHNSON is sitting in an armchair wearing worn but comfortable shoes. He is an amiable man out with a well -worn face exhibiting a lifetime of drudgery.

STEWART lies on the carpet coloring in a picture book. The room is well furnished and well maintained but the first thing to grab your attention is the tick-tock of a mantel piece clock. On top of the drinks cabinet there is an abundance of shining trophies for Albert's prized fishing catches.

LOUISE enters carrying a tray of food.

ALBERT
(sarcastic)
I told you it was no good for you.

LOUISE smiles and hands ALBERT the tray.

LOUISE
Dad, Jim's a good man. Now, eat your tea.

LOUISE picks up an ouster and tin of polish from a table and begins to clean the room.

LOUISE

How's your chest?

ALBERT
Not so bad since I started on the pills. (to Stewart) Fuss, fuss, fuss.

STEWART
Fuss, fuss, fuss.

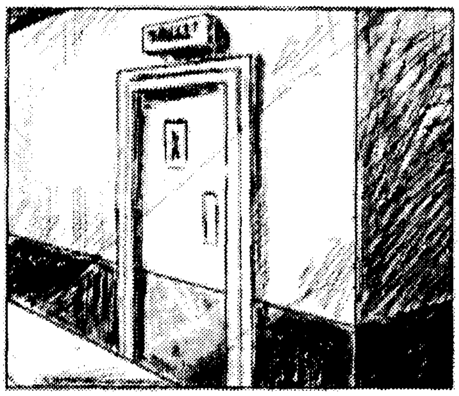
▲▲▲▲▲▲▲▲ End screenplay example ▲▲▲▲▲▲▲▲

The first draft of a screenplay may require several months to complete, and it is rewritten until all stakeholders agree that it is sufficient for filming. Even then, the writer is often summoned during film production to rewrite sections of the screenplay. To avoid film crews having different updates of the screenplay, page-locking is employed. Changes are added or replace pages of the page-locked screenplay by means of colored pages.

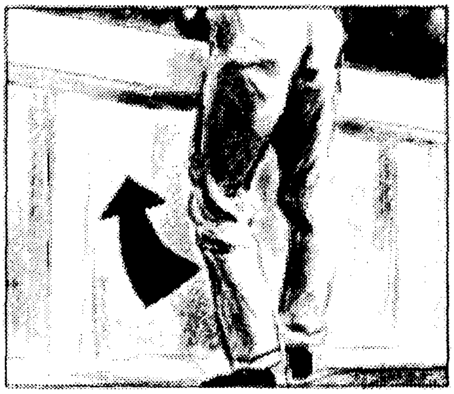
2.3.2 Film Storyboard

A film's storyboard is a sequence of drawings designed to visualize a film or sections of a film beforehand. Its appearance resembles that of a comic strip, illustrating the location, characters, props and settings of each camera shot. Each drawing is further adorned with technical instructions and arrows describing actions, camera and lighting directions, and occasionally basic dialogue. Basically, the storyboard represents what the camera lens or viewers will see. Figure 2.26 is an example of a film storyboard.

'P' is for Psycho



mix to...



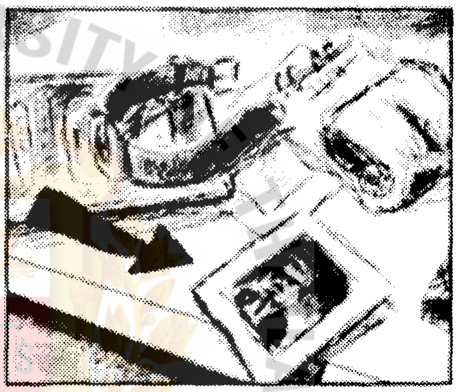
1 Est. wide shot (WS) (Low key lighting)

low angle. zoom in - med. shot

2 Int. low angle (MS)

Slow tilt up to 3...

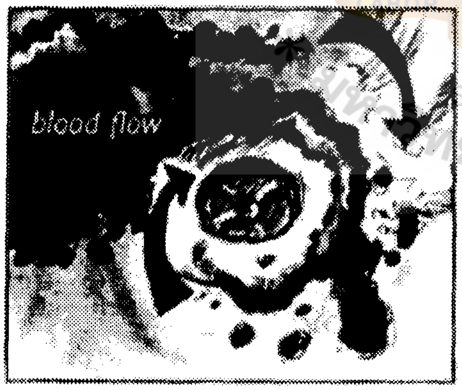
cut



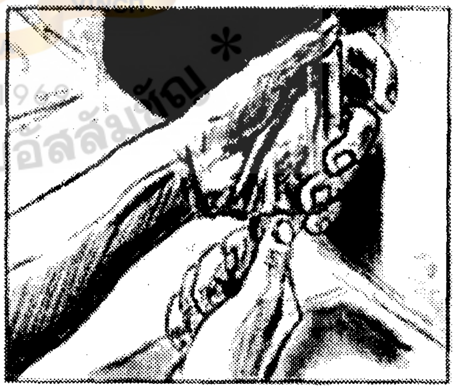
cut

3 Low angle (MS)

4 High angle (CU) Pan across objects L-R (very slow)



cut



cut

5 (CU) plug - hole!

6 (CU) "bandage - wrapping"

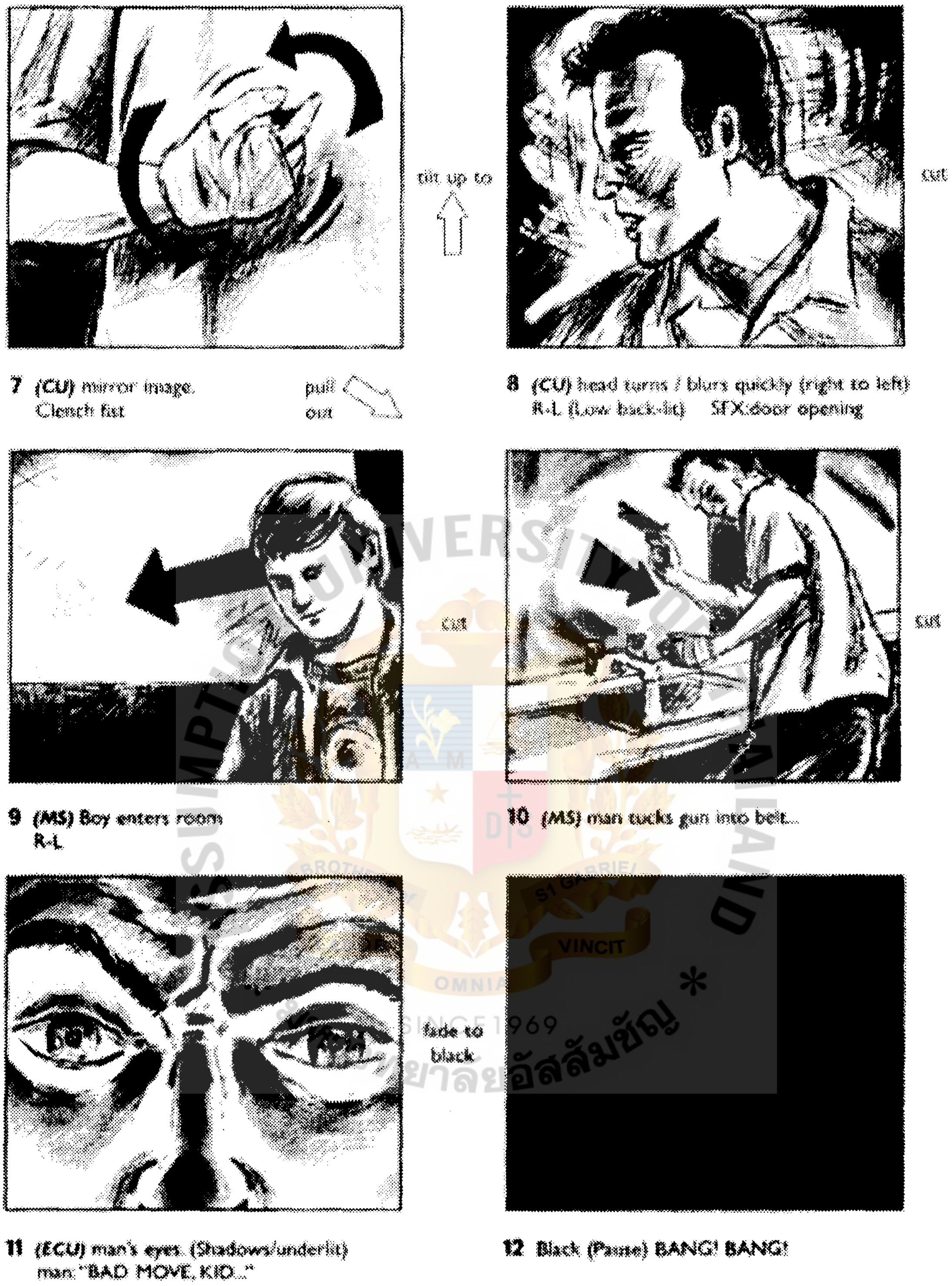

tilt up to
cut
7 (CU) mirror image.
Clench fist
pull
out
8 (CU) head turns / blurs quickly (right to left)
R-L (Low back-lit) SFX:door opening
cut
cut
9 (MS) Boy enters room
R-L
10 (MS) man tucks gun into belt...
fade to
black
11 (ECU) man's eyes. (Shadows/underlit)
man: "BAD MOVE, KID..."
12 Black (Pause) BANG! BANG!

Figure 2.26. 'P' is for Psycho Storyboard (Alice, 2001).

Based on a shooting script, a storyboard is first drawn as a rough sketch in the second stage of filmmaking: the Preproduction Stage. At this stage, a film is planned, designed, and the storyboard is extensively used until the completion of the film. The storyboard matures as the director discusses the 'look' of the film with the director of photography. During film production, copies of the storyboard are distributed to the crew and casts, virtually as job descriptions for each camera shot. Finally, during post-production, the storyboard visually reminds film editors of the original intentions, sequence, and timing of the film.

Lower budget or smaller films may discard the process of storyboarding altogether. However, storyboards are time worthy and indispensable if there are hundreds if not thousands of film crews and casts. Sometimes it replaces a screenplay entirely. The benefits of storyboards extend far beyond a visual consensus or framework which changes can be judged. It is an essential piece of communication, a tool used to inspire more ideas or discover potential problems, and costs estimator as well as saver. It can accurately position actors in an imaginary world, as well as help them to perform convincingly with an imaginary character. It also defines the scope and number of shots for each event. A close-up shot of an object, for instance, may only require an indoor backdrop rather than a long strenuous trip to the actual location. Without the storyboard, ten different directors may even produce ten different interpretations of how to shoot the same written event.

Matthew Jones, a script editor, concisely expresses his gratitude for storyboards, "In a production meeting, a picture really is worth a thousand words. You can script a sequence in words as clearly as you like, and there will always be some misunderstanding. But if you use storyboards, it's so much easier to communicate your visual and dramatic ideas" (Alice 2001, p. 1).

Storyboards find its uses in business as well such as ad campaigns, commercials, and computer games. It is used to convince investors of a potential project, or as a

"Quality Storyboard" that paints the Quality Control process within an organization. Human resources involved in a storyboard can now learn how and when to explain their part in a story.

2.3.3 Best Practices

The study of screenplays and storyboards reveals valuable best practices pertaining to blueprints:

Create narrative blueprints. Narrative blueprints, such as screenplays, are written plans based on original concepts or works. They allow elaborate details to be documented, and may provide the basis for visual blueprints. Narrative blueprints' appearances in the earliest project stage indicate their profound importance.

Create visual blueprints. Visual blueprints, such as storyboards, are consensuses of how the final product may appear to the audience or users beforehand. Commonly deriving from narrative blueprints, they work to bring narrative blueprints to life. Without the visual blueprint, visual interpretations may vary from person to person.

Create non-technical blueprints. Larry Sherby, a digital camera owner said, " . . . I have to consult a manual anytime I try other features and then I forget how to do it" (The Associated Press, 2004). Larry Sherby has demonstrated that too many products are built for technicians. Similarly, there is a dire need for non-technical, intuitive blueprints that do not depend on other special bodies of knowledge or experts to understand.

III. SOFTWARE SCREENPLAY STORYBOARD MODEL (S3M)

3.1 Introduction

S3M models are intended to be design plans that can be distributed to any software buyers and developers, and all will still be able to perceive the same exact software system. To make this happen, S3M employs two crucial features. First, S3M levels usability to a common ground – one that is not favorable for, or operational by only particular groups of people and vendors. Special tools, languages, and expertise are only hindrances to the reconciliation between software buyers and developers. S3M models are described using only human languages, preferably English, which is extensively used as a second language and as an official language in many countries. Visual sections of the model can be illustrated using the simplest of office tools: the pen and paper, or virtually any illustrational software. Secondly, S3M is self-contained. Each individual S3M model or group of S3M models are accompanied by standard items consisting of folders and documents that are designed to be extremely portable, and contain all the *essentials* to create, store, and guide project stakeholders through an entire group of S3M models. This produces three momentous benefits for stakeholders who are involved with S3M models: anyone can learn S3M, learn by themselves, and learn instantly. Moreover, S3M's standard items are highly flexible. They are editable, or translated if necessary, to allow optimal fitting for each individual software system. Altered standard documents, however, should at least continue to realize the objectives of each document.

This research refers to the set of S3M models and corresponding standard items simply as a 'blueprint'. S3M blueprints have three formats to select from: paper-based, digital, or both (for example, paper-based narrations with digital illustrations). Figure

3.1 suggests one way to store paper-based S3M blueprints, and Figure 3.2 suggests one way to store digital S3M blueprints. Because S3M blueprints can be entirely in paper-based form, it has been extensively scrutinized and simplified to minimize software-dependency, albeit modern system designers being more likely to opt for word processors and stencil-based drawing software. Lastly, S3M blueprints are photocopy-friendly – meaning every element has been designed in only three colors: white, black, and gray.

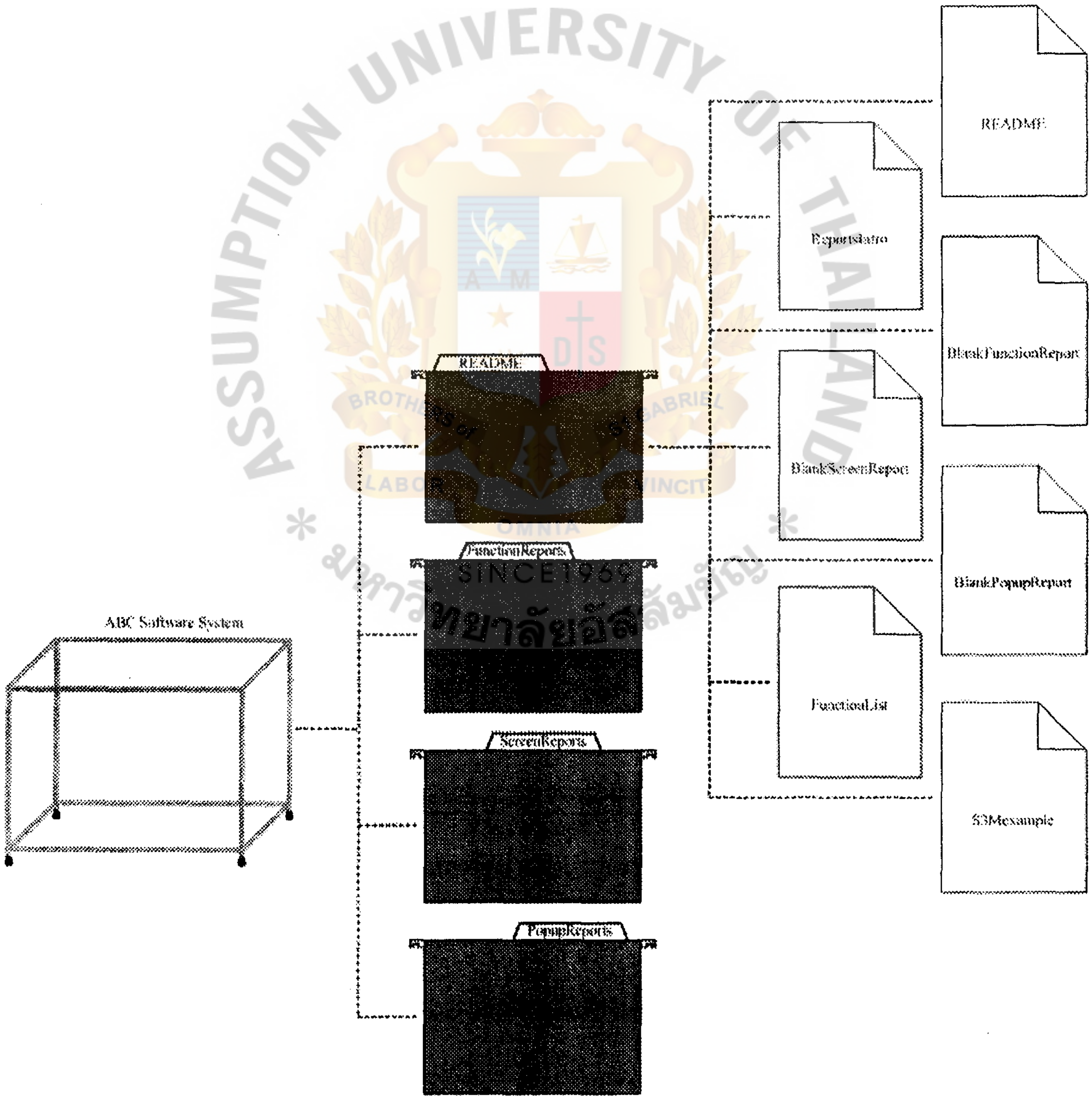


Figure 3.1. Paper-based S3M Blueprint with Standard Items in Suspension Folders.

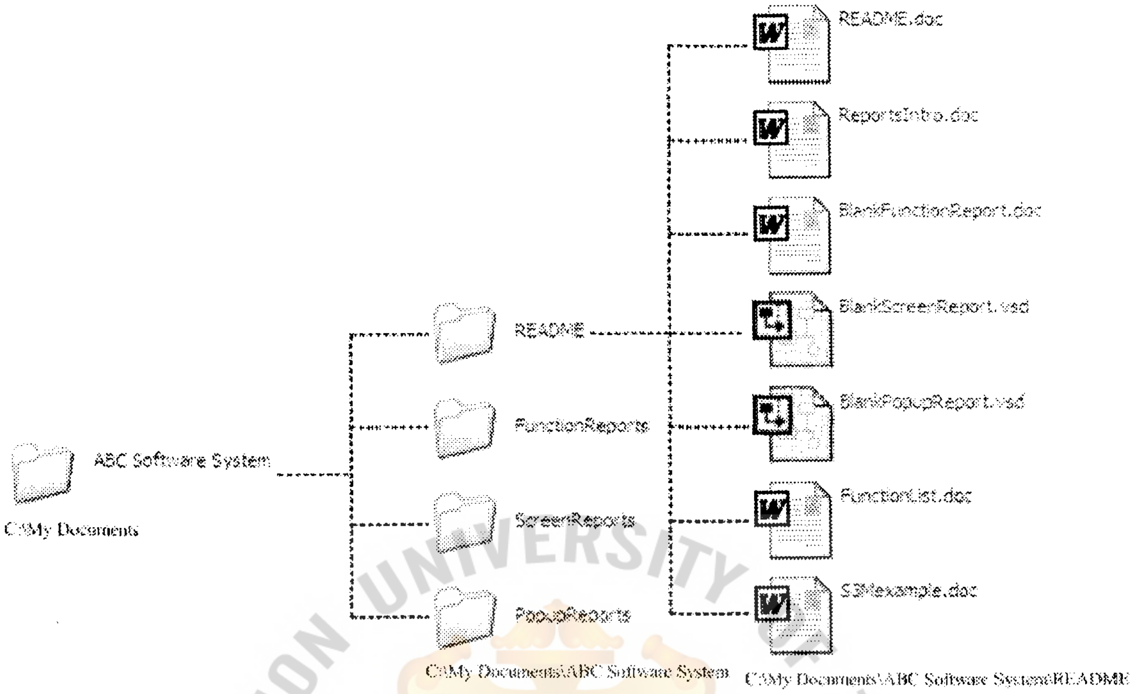


Figure 3.2. Digital S3M Blueprint with Standard Items in Computer Hard Drive.

3.2 The README Document

README is a commonly distributed file containing information concerning other files in a directory or archive. The capitalized name is chosen to attract users to read the file first. Similarly, all S3M blueprints are delivered with a single README document in a README folder. It should also be the first standard item created, as everything else branches from here. The S3M README document aims to meet the following objectives:

(1) To introduce S3M concisely.

(2) To introduce S3M standard items.

(3) To provide other essentials S3M users need to know before proceeding.

The following is an example of the README document:

▼▼▼▼▼▼▼▼ Start README document ▼▼▼▼▼▼▼▼

README

This software system is designed using the Software Screenplay Storyboard Model (S3M), which narrates and visualizes the system before actual development. The narrative and visual aspects of S3M models are documented in separate reports. S3M is also self-contained. Each individual S3M model or group of S3M models are accompanied by standard items consisting of 4 folders and 7 documents that are designed to be extremely portable, and contain all the *essentials* to create, store, and guide project stakeholders through an entire group of S3M models. The first folder, called README, stores all 7 documents named respectively: README, ReportsIntro, BlankFunctionReport, BlankScreenReport, BlankPopupReport, FunctionList, and S3Mexample. These documents should also be read in the same sequence. The other 3 folders are named according to their contents: FunctionReports, ScreenReports, and PopupReports.

S3M, as well as this point henceforth, utilizes two literary conventions. First, nouns enclosed in square brackets "[ ]" mean to insert contents, as indicated by the given nouns, at the same location as the square brackets. For example, "[system name]" means: insert a system name here. Any word formatting such as bold, italic, or underline applied to the bracketed nouns also applies to the corresponding insertion. In some cases where insertion is optional or unavailable at the time, system designers may either leave the insertion intact, or if possible, simply erase the square brackets and its contents. Lastly, three periods separated by spaces ". . ." mean: so on so forth.

▲▲▲▲▲▲▲▲ End README document ▲▲▲▲▲▲▲▲

3.3 The ReportsIntro Document

The modern user interface of a computer is a remarkable demonstration of simplicity. Virtually anything can be achieved in three actions: type, click, and drag. S3M reports are substantially based on these actions. They document what system users will type, click, and drag in the user interface. A forthcoming S3Mexample document will evidently corroborate the sufficiency of these actions in modeling a software system.

The ReportsIntro document is essentially the heart of S3M. S3M users will be able to create the reports after reading this one single document. The ReportsIntro document aims to meet the following objectives:

(1) To introduce S3M reports and their key elements.

(2) To explain how to use the reports.

The following is an example of the ReportsIntro document:

▼▼▼▼▼▼▼▼ Start ReportsIntro document ▼▼▼▼▼▼▼▼

ReportsIntro

S3M models consist of three types of report: function reports constitute the narrative aspect of the model, and screen and popup reports constitute the visual aspect of the model. S3M standard items also include a blank function report, a blank screen report, and a blank popup report as templates to provide convenience in creating the reports. The following sections are introductions to each type of report:

Function Report Introduction

In S3M, a function is a set of steps or actions users perform on a computer system to achieve a particular end. Steps can be described narratively as well as pictorially. Smaller steps may be fused into or described in a single step, and vice versa.

A function report provides full description of a particular function performed by users of a computer system. A function report is read alongside screen reports that it calls. Function reports mainly describe steps performed by users, unless developers decide they need a technical version of the function report that also details steps performed by the system. The following is the layout and full description of a function report:

[function name] Function Report

Function Name: < Function names are preceded by a letter 'f'. All other words that the name comprises begin with a capitalized letter to distinguish each individual word.

Category Path: < This lists the category and subcategories that the function belongs to. Categories and subcategories are separated by the "greater than" symbol with spaces " > ".

Description: < This is a description of the function.

Functions Involved: < This lists other functions used by this particular function, and helps designers retrieve relevant function reports beforehand. Disabling functions listed here will affect the current function that requires those other functions in a system.

Screens Involved: < This lists all screens used by this function, and helps designers retrieve relevant screen reports beforehand.

Popups Involved: < This lists all popups used by this function, and helps designers retrieve relevant popup reports beforehand.

Designers: < This lists all developers or buyers that create or edit this report.

Function:

Steps*	**Description**
1	
2	
. . .	

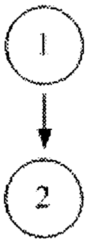
Exceptions: < These are things that can go wrong when users perform the function.

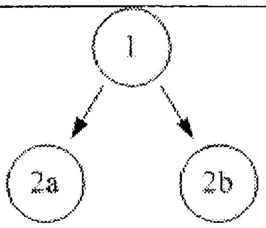
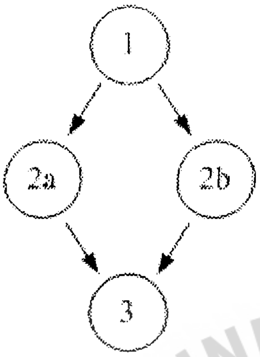
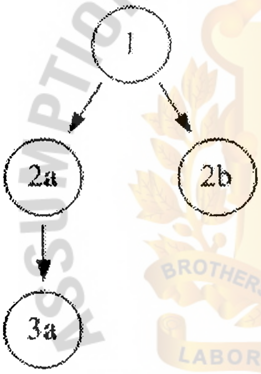
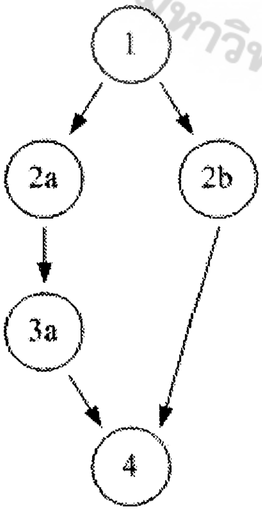
#	**Descripton**	**Response** < How the system respond to an exception that is detected.
1		
2		
. . .		

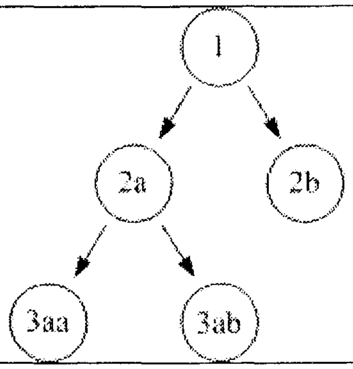
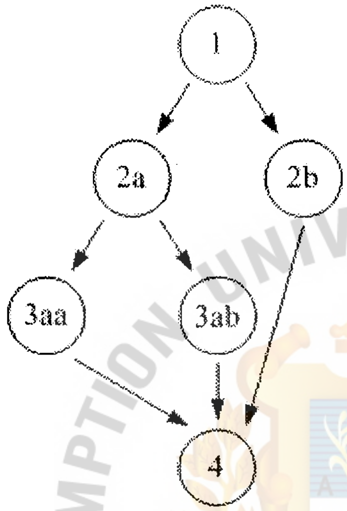
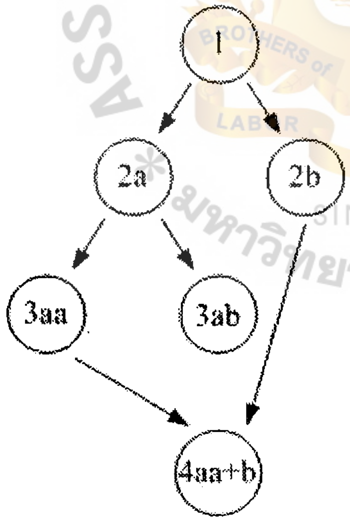
*Please see the Step Tutorial section on how to name steps.

Steps Tutorial

Steps must be uniquely named for reference purposes. For example in a recursion, a later step tells the user to perform an earlier step. There are sequential steps and alternative steps. Sequential steps are normally formulated by adding one to the previous step. Alternative steps are normally formulated by adding one and a non-capitalized English alphabet to the previous step. However, S3M recommends that designers avoid multiple alternative steps simply by creating separate function reports for each alternative step path, or create a function report for the most significant step path. The step naming convention is demonstrated by the following table. Notice that there is always a single first Step 1 as the starting point. Also, notice how step paths influence the step sequence. It is more natural to think step paths one by one, than all alternatives at once.

Step Sequence	**Steps Illustration**	**How to Read**	**Step Paths**
Steps 1 2		Perform Step 1, perform Step 2 (stop)	1>2
Steps 1 2a 2b		Perform Step 1, perform either Step 2a (stop) or Step 2b (stop)	1>2a 1>2b

			
Steps 1 2a 2b 3		Perform Step 1, perform either Step 2a or Step 2b, perform Step 3 (stop) *Notice that there may be as many alternatives necessary, but Step 3 succeeds all.	1>2a>3 1>2b>3
Steps 1 2a 3a 2b		Perform Step 1, perform either Step 2a or Step 2b (stop), perform Step 3a (stop) *Notice that Step 3a only ends the alternative path 'a'.	1>2a>3a 1>2b
Steps 1 2a 3a 2b 4		Perform Step 1, perform either Step 2a or Step 2b, perform Step 3a, perform Step 4 (stop) *Notice that there may be as many alternatives and paths necessary, but Step 4 succeeds all.	1>2a>3a>4 1>2b>4
Steps		Perform Step 1, perform either Step	1>2a>3aa 1>2a>3ab

1 2a 3aa 3ab 2b		2a or Step 2b (stop), perform either Step 3aa (stop) or Step 3ab (stop)	1>2b
Steps 1 2a 3aa 3ab 2b 4		Perform Step 1, perform either Step 2a or Step 2b, perform either Step 3aa or Step 3ab, perform Step 4 (stop) *Notice that there may be as many alternatives and paths necessary, but Step 4 succeeds all.	1>2a>3aa>4 1>2a>3ab>4 1>2b>4
Steps 1 2a 3aa 3ab 2b 4aa+b		Perform Step 1, perform either Step 2a or Step 2b, perform either Step 3aa or Step 3ab (stop), perform Step 4aa+b (stop) * Notice that Step 4aa+b only ends the alternative paths 'aa' and 'b'. Try to avoid this.	1>2a>3aa>4aa+b 1>2a>3ab 1>2b>4aa+b

Screen Report Introduction

A screen is basically an instance of a system's user interface (UI). All widgets, or UI components a system user interacts with such as buttons and text boxes, are uniquely tagged for reference in function reports. Multiple widgets can also be tagged as a group. Tags are formed according to the following convention: [screen name]-# (for example, sLogin-1, sLogin-2, and so on). Tag numberings usually start from left to right and top to bottom of the screen.

A screen report describes and illustrates a particular screen. Screen reports are used with function reports that call them. The following is the layout and full description of a screen report:

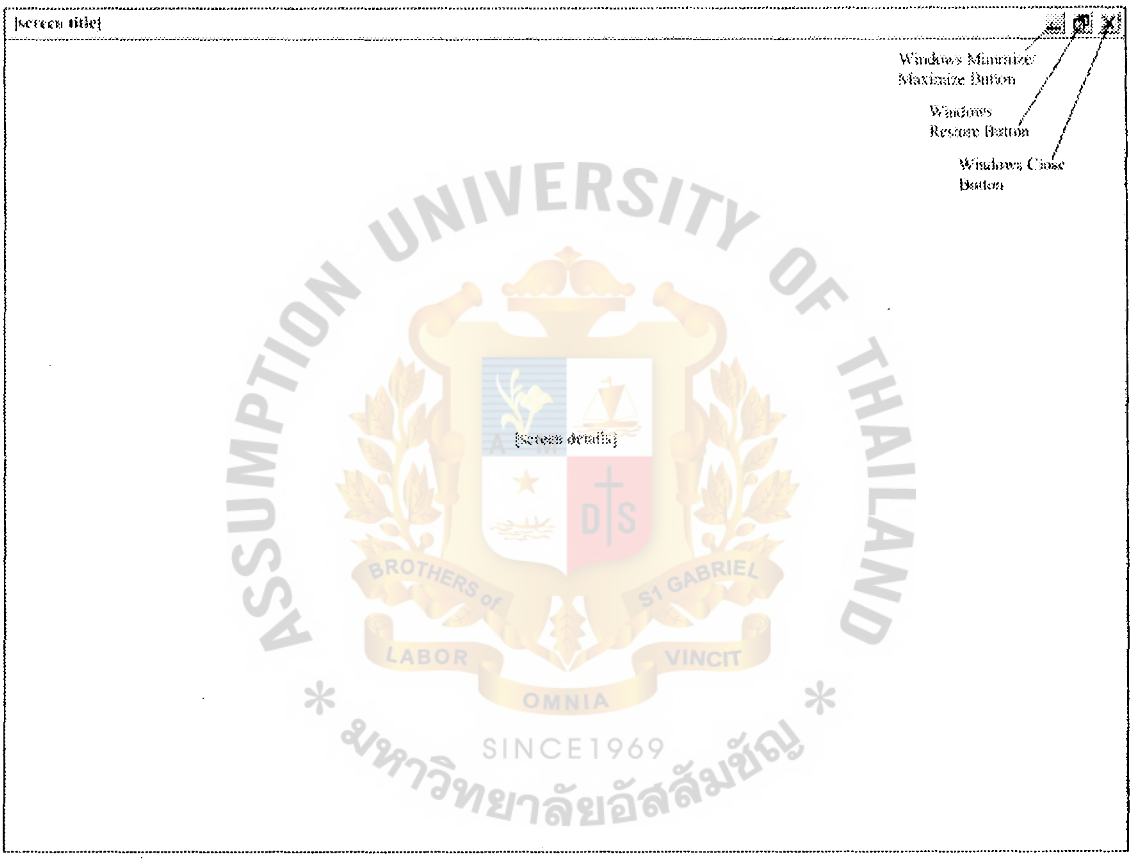
[screen name] Screen Report

Screen Name: < Screen names are preceded by a letter 's', as all other words that the name comprises begin with a capitalized letter to distinguish each individual word.

Description: < This is a description of the screen.

Functions Involved: < This identifies functions involved with (use) this screen, and helps designers retrieve relevant function reports beforehand. Editing the current screen may affect the functions listed here.

Designers: < Designers, or authors of this report.



Popup Report Introduction

A popup is a window that appears over a screen, and calls the system user to perform a specific function. For example, a popup may require the user to confirm the deletion of a piece of critical information, or inform the user of exceptions and errors. A popup may call another popup, although rarely practiced. A popup disappears and allows the user to return to the screen or popup that calls the popup *only* when the user performs the specified function or cancels the popup. This distinguishes popups from screens, which are less strict than popups. To decrease documentation, S3M recommends that popup reports are only composed for complex functions. Simple popups can be directly described in function reports. The following is the layout and full description of a popup report:

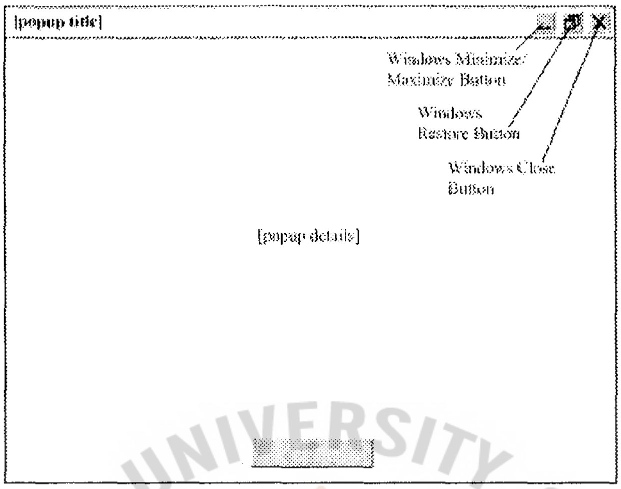
[popup name] Popup Report

Popup Name: < Popup names are preceded by the letters 'p', as all other words that the name comprises begin with a capitalized letter to distinguish each individual word.

Description: < This is a description of the popup.

Functions Involved: < This identifies functions involve with (use) this popup, and helps designers retrieve relevant function reports beforehand. Editing the current popup may affect the functions listed here.

Designers: < Designers, or authors of this report.



▲▲▲▲▲▲▲▲ End ReportsIntro document ▲▲▲▲▲▲▲▲

3.4 The BlankFunctionReport Document

The following is a typical BlankFunctionReport document created with and designed for use with word processors:

▼▼▼▼▼▼▼▼ Start BlankFunctionReport document ▼▼▼▼▼▼▼▼

Function Report

Function Name:
Category Path:
Description:
Functions Involved:
Screens Involved:
Popups Involved:
Designers:

Function:

Steps	Description
1	
2	

Exceptions:

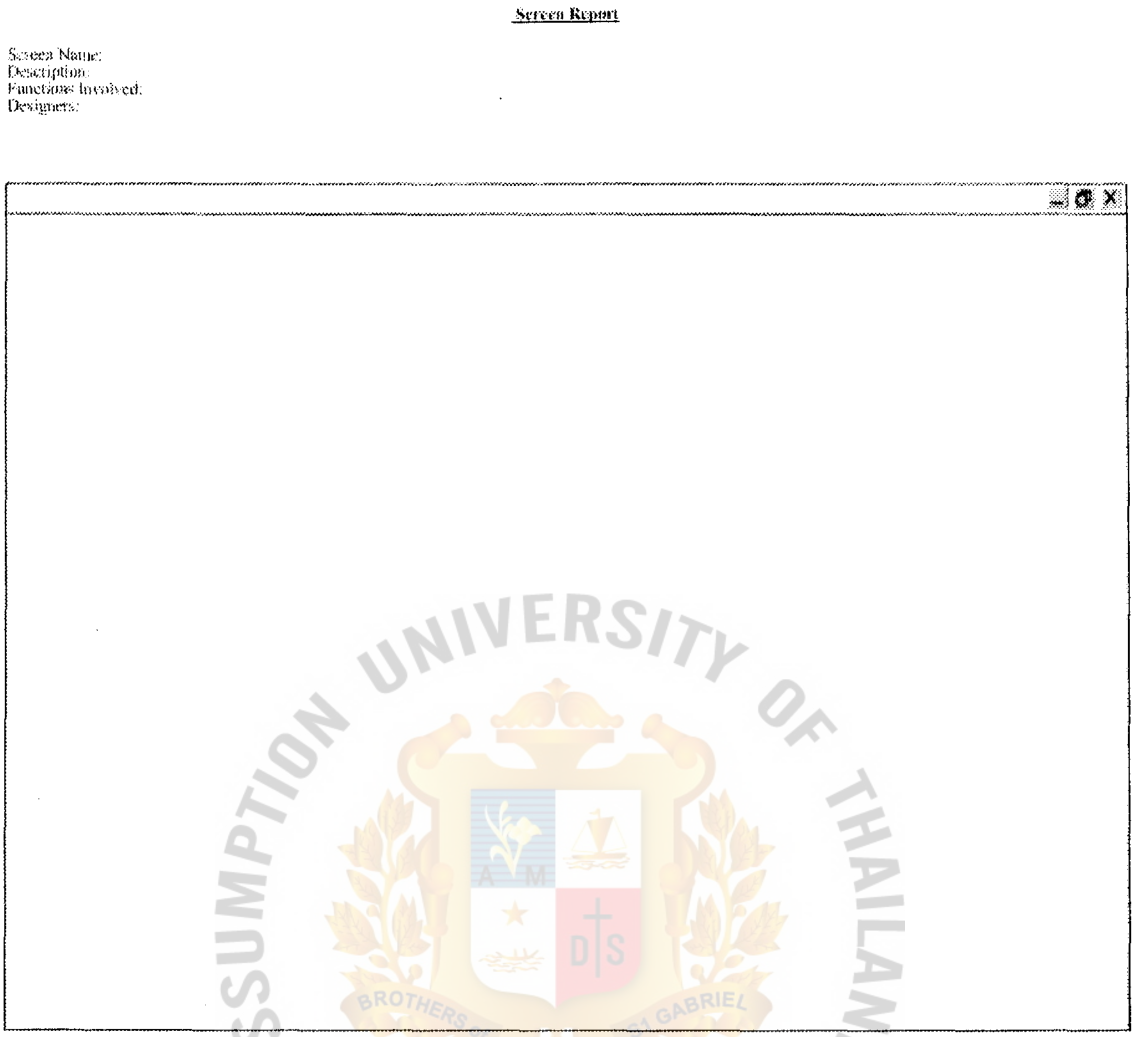
#	Description	Response
1		
2		

▲▲▲▲▲▲▲▲ End BlankFunctionReport document ▲▲▲▲▲▲▲▲

3.5 The BlankScreenReport Document

The following is a typical BlankScreenReport document created with and designed for use with stencil-based drawing software:

▼▼▼▼▼▼▼▼ Start BlankScreenReport document ▼▼▼▼▼▼▼▼



Screen Report

Screen Name:
Description:
Functions Involved:
Designers:

▲▲▲▲▲▲▲▲ End BlankScreenReport document ▲▲▲▲▲▲▲▲

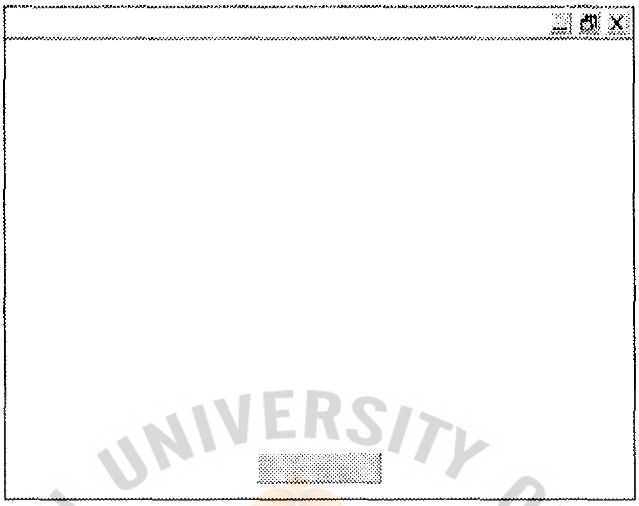
3.6 The BlankPopupReport Document

The following is a typical BlankPopupReport document created with and designed for use with stencil-based drawing software:

▼▼▼▼▼▼▼▼ Start BlankPopupReport document ▼▼▼▼▼▼▼▼

Popup Report

Popup Name:
Description:
Functions Involved:
Designers:



▲▲▲▲▲▲▲▲ End BlankPopupReport document ▲▲▲▲▲▲▲▲

3.7 The FunctionList Document

Function reports are gateways to screen and popup reports because both types of reports are designed to be called by function reports. However, when there can be hundreds, perhaps thousands of functions, the FunctionList document is vital in aiding S3M users in function searches as well as creation. The document aims to meet the following objectives:

(1) To explain the important purpose of the document.

(2) To aid S3M users in function search and creation.

The following is an example of the FunctionList document:

▼▼▼▼▼▼▼▼ Start FunctionList document ▼▼▼▼▼▼▼▼

FunctionList

This document provides an introduction to the software system and a portal to its S3M reports. The portal is virtually a table of contents of functions and their descriptions. Function categories and subcategories may also be described as well to aid function search and creation. For convenience, function descriptions may be copied into corresponding reports. This document initially provides a blank outline of the table of contents, which will eventually be filled as the document is open for updates throughout the system model development.

[software system name]

[software system introduction]

1. [category1 name]

[category1 description]

Function reports in this category include:

1.1. [function report1 name]

[function report1 description]

1.2. [function report2 name]

[function report2 description]

1.3. . . .

2. [category2 name]

[category2 description]

Function reports in this category include:

2.1. [function report1 name]

[function report1 description]

2.2. [function report2 name]

[function report2 description]

2.3. . . .

3. . . .

▲▲▲▲▲▲▲▲ End FunctionList document ▲▲▲▲▲▲▲▲

3.8 The S3Mexample Document

The S3Mexample document is intended to be read last. It demonstrates S3M with a classic example found in many introductory texts for teaching programming languages: a simple software program that displays "Hello world!" With slight additions to the program, the S3Mexample document aims to meet the following objectives:

(1) To show how prime standard items and reports may appear in practice.

(2) To show how prime standard items and reports work together.

(3) To prove the feasibility of S3M.

The following is an example of the S3Mexample document:

▼▼▼▼▼▼▼▼ Start S3Mexample document ▼▼▼▼▼▼▼▼

S3Mexample

This document does not present a complete S3M solution, but provides a demonstrative S3M example. It has excluded the README, ReportsIntro, BlankFunctionReport, BlankScreenReport, and BlankPopupReport documents because their contents normally remain the same. Designed for a software system called Hello World, the example consists of a FunctionList document, a function report, a screen report, and a popup report. Separated by rows of asterisks, the document and reports are intended to demonstrate how descriptions from the FunctionList document can be used to generate functions, how steps and exceptions may be described, how widgets may be tagged, how the reports relate to one another, and clarify other queries new S3M users may have after reading through the standard items. New S3M designers may also adopt the design sequence and logic implied by the example, although S3M does not impose any in particular.

****

FunctionList

This document provides an introduction to the software system and a portal to its S3M reports. The portal is virtually a table of contents of functions and their descriptions. Function categories and subcategories may also be described as well to aid function search and creation. For convenience, function descriptions may be copied into corresponding reports. This document initially provides a blank outline of the table of contents, which will eventually be filled as the document is open for updates throughout the system model development.

Hello World

Hello World is a software system that calls a popup to display the message, "Hello world!"

1. Main Functions

Hello World has two types of users: authorized and unauthorized. Authorization is checked by the system when users attempt to log in by entering a username and password. Only users with valid combinations of username and password are able to view the popup that displays the "Hello world!" message.

Function reports in this category include:

1.1. fLogin

All users log into the system here.

****

fLogin Function Report

Function Name:	fLogin
Category Path:	Hello World > Main Functions
Description:	All users log into the system here.
Functions Involved:	-
Screens Involved:	sLogin
Popups Involved:	pDisplay
Designers:	Jia

Function:

Steps	Description
1	In sLogin, type in username and password in sLogin-1, and click sLogin-2 to attempt to log into the system. If username and password is valid, system calls pDisplay.
2	In pDisplay, click pDisplay-1 to return to sLogin.

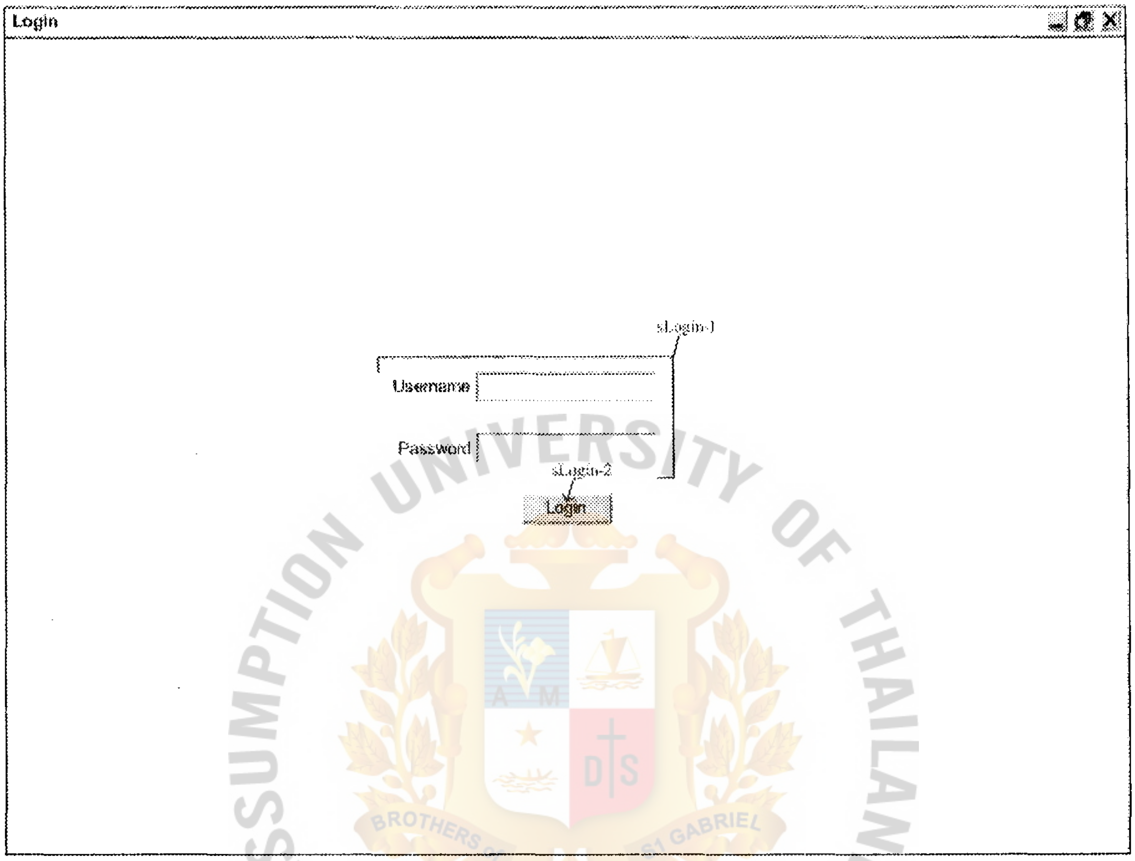
Exceptions:

#	Description	Response
1	The username and password are not found.	System calls popup that states: "The username and password are not found."

****

sLogin Screen Report

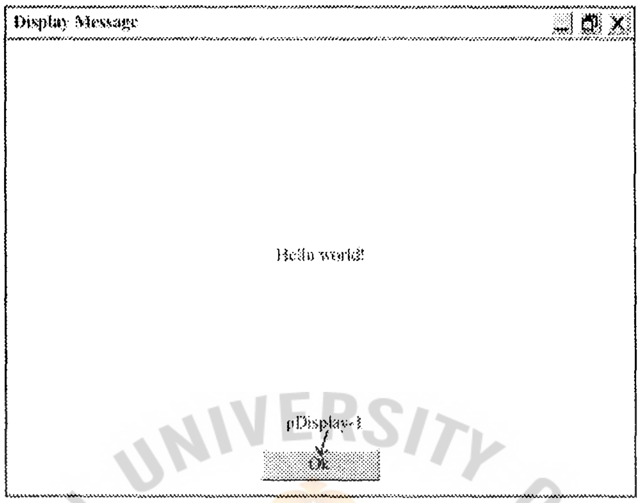
Screen Name:	sLogin
Description:	All users log into the system here.
Functions Involved:	fLogin
Designers:	Jia



****

pDisplay Popup Report

Popup Name: pDisplay
Description: Display the message, "Hello world!"
Functions Involved: fLogin
Designers: Jia



Display Message

Hello world!

▲▲▲▲▲▲▲▲ End S3Mexample document ▲▲▲▲▲▲▲▲

IV. TESTING S3M

4.1 Student Information System (SIS)

In contrast to the "Hello World" software program in the S3Mexample document, SIS is a practical real-world application – demonstrating actual scenarios that require the creative harness of S3M. The SIS blueprint excerpt presented here not only abides by the S3M framework, but incorporates narration-illustration principles in various areas of the blueprint to maximize users' comprehension. Again, this section is not intended to provide the entire blueprint, but presents only a set of related functions concerning academic program setup. The README, ReportsIntro, BlankFunctionReport, BlankScreenReport, BlankPopupReport, and S3Mexample documents are excluded, as their contents normally remain the same in all blueprints. The SIS blueprint excerpt will start with the FunctionList document, and work its way through the reports that build upon the FunctionList document. The excerpt is created using only a word processor and stencil-based drawing software.

4.2 SIS Blueprint Excerpt

To eliminate clutter, individual documents and reports of the SIS blueprint excerpt will each start on new pages, starting from the next page. Moreover, reports will not be regrouped according to report types, but they will remain in the same sequence they are created to provide some insights to the excerpt's design sequence and logic.

FunctionList

This document provides an introduction to the software system and a portal to its S3M reports. The portal is virtually a table of contents of functions and their descriptions. Function categories and subcategories may also be described as well to aid function search and creation. For convenience, function descriptions may be copied into corresponding reports. This document initially provides a blank outline of the table of contents, which will eventually be filled as the document is open for updates throughout the system blueprint development.

Student Information System (SIS)

SIS is a generic system concerning all functions surrounding student information such as grades and attendance. Four primary types of users of SIS are system administrators, which are simply referred as administrators, instructors, students, and parents. Administrators typically perform technical functions such as system or user access configurations. Instructors perform functions related to students such as grading and issuing progress reports. Students and parents share similar functions, which mostly pertain to viewing student information. All user information, despite their status, is stored permanently and requires authorized personnel to perform any deletion.

1. Generic Functions

For security reasons, each type or individual user may be configured to allow access to only certain information or functions. User access rights are checked each time the user attempts to log into the system with his/her username and password.

Function reports in this category include:

1.1. fLogin

All users log into the system here.

2. Academic Program Functions

Most educational institutes possess more than one academic program. For example, a typical grade 1 – 12 school may be divided into Elementary School, Middle School, and High School. As for universities, the academic program hierarchy may branch as far as 3 to 4 tiers. For example, a university may possess Bachelor, Master, and Doctorate Degree Programs. Master Degree Programs may be divided into several schools such as Business, Computer, and Engineering. Finally, the School of Computer may be further divided into Computer Information Systems, Computer and Engineering Management, and Internet & E-Commerce Technology.

Each academic program may possess its own set of calendars, daily schedules, and students, as well as address (academic programs are not necessarily located at the same place). A branch of academic programs, however, may possess a single publicly-accessible master calendar that oversees the entailing set of publicly- and privately-accessible calendars. School day indications (full, partial, and no school) and events added or edited in a master calendar will be inherited by all publicly- and privately-accessible calendars of the corresponding branch of academic programs, unless inhibited by the user. As a consequence, a master calendar cannot be overseen by another master calendar as shown in Figure 1, nor can two or more master calendars reside in the same node. If a master calendar is deleted, information previously conveyed to all publicly- and privately-accessible calendars of the corresponding branch of academic programs will also be deleted. Lastly, all calendars display the current month by default when accessed.

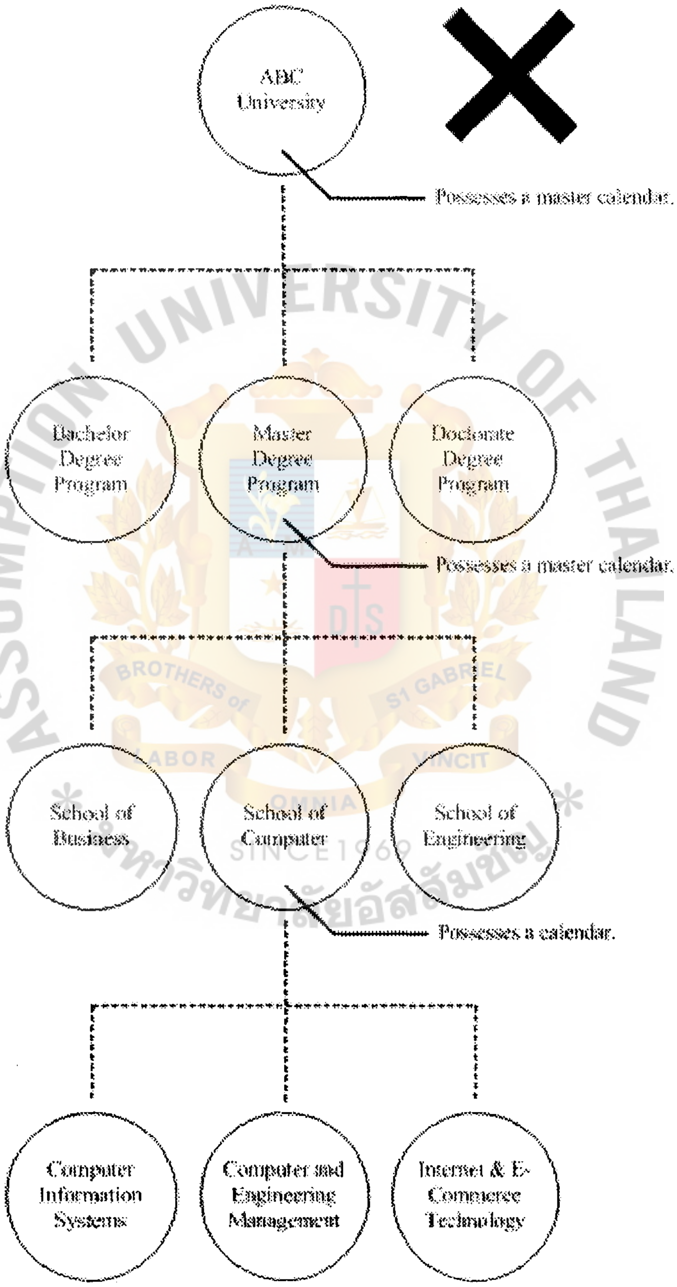


Figure 1. A master calendar overseeing another master calendar is illegal and confusing.

Academic programs need to organize marking periods to inform students of their grades or progress throughout an academic year. Marking periods should be named meaningfully, and possess no more than three tiers. For example, Academic Year 2007 (1st tier) has 2 Semesters (2nd tier), and each Semester has 2 Quarters (3rd tier). Marking periods appear as non-recurring events in calendars hence, they are recreated for each academic year.

On a daily basis, academic programs abide by daily schedules. An academic program may possess multiple daily schedules, for instance, one for full school days and another for partial school days. The daily schedule displays information such as class periods and breaks.

Function reports in this category include:

2.1. fAddAcad

Add a new academic program.

2.2. fViewAcad

Added academic programs may be viewed, edited, deleted, or saved as new academic programs.

2.3. fAddMarkPer

Add up to three tiers of marking periods.

2.4. fViewMarkPer

Added marking periods may be viewed, edited, or deleted.

2.5. fAddCal

Add a calendar for an academic program.

2.6. fViewCal

Added calendars may be viewed or deleted.

2.7. fSchDayCal

Indicate full, partial, or no school days on a calendar.

2.8. fEventCal

Add, edit, or delete events in calendars.

2.9. fAddDlySchd

Add a daily schedule for an academic program.

2.10. fViewDlySchd

Added daily schedule may be viewed, edited, or deleted.

fLogin Function Report

Function Name:	fLogin
Category Path:	Student Information System > Generic Functions
Description:	All users log into the system here.
Functions Involved:	-
Screens Involved:	sLogin, sIndex
Popups Involved:	-
Designers:	Jia

Function:

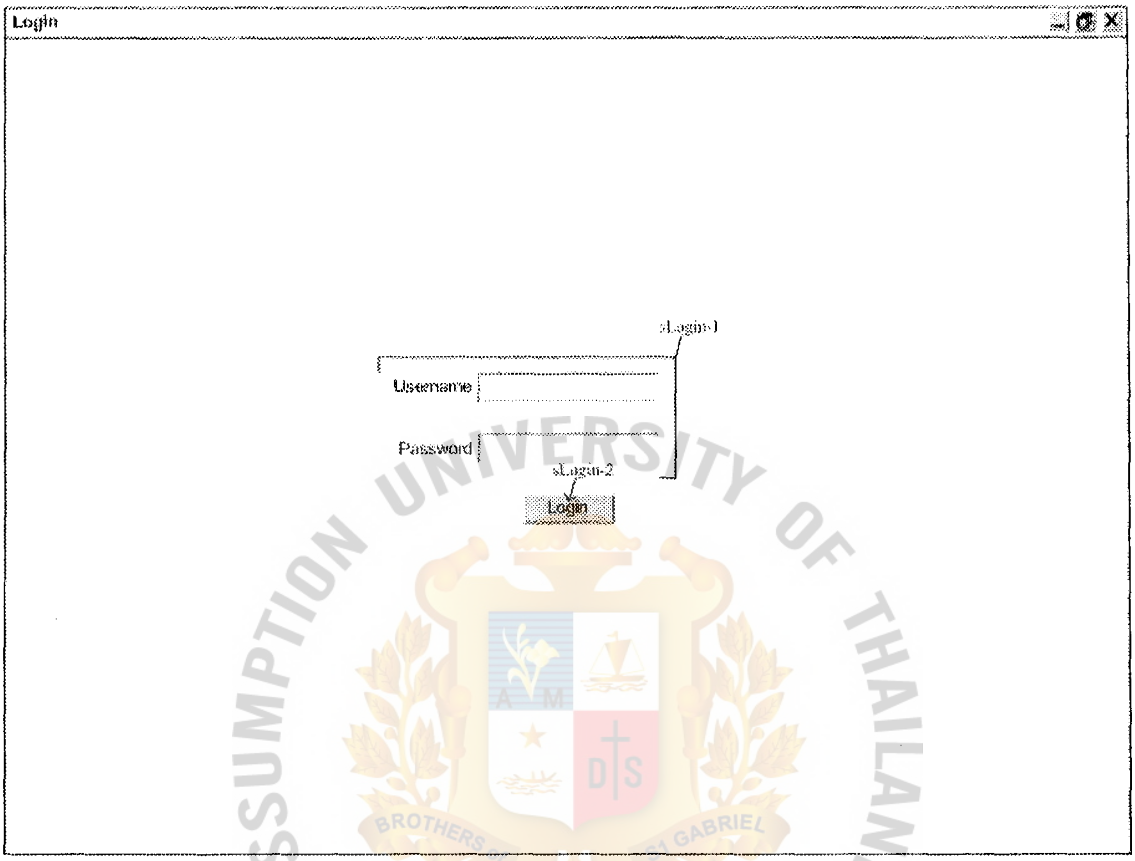
Steps	Description
1	In sLogin, type in username and password in sLogin-1, and click sLogin-2 to attempt to log into the system. If username and password is valid, system loads the user's access rights and displays sIndex.

Exceptions:

#	Description	Response
1	The username and password are not found.	System calls popup that states: "The username and password are not found."

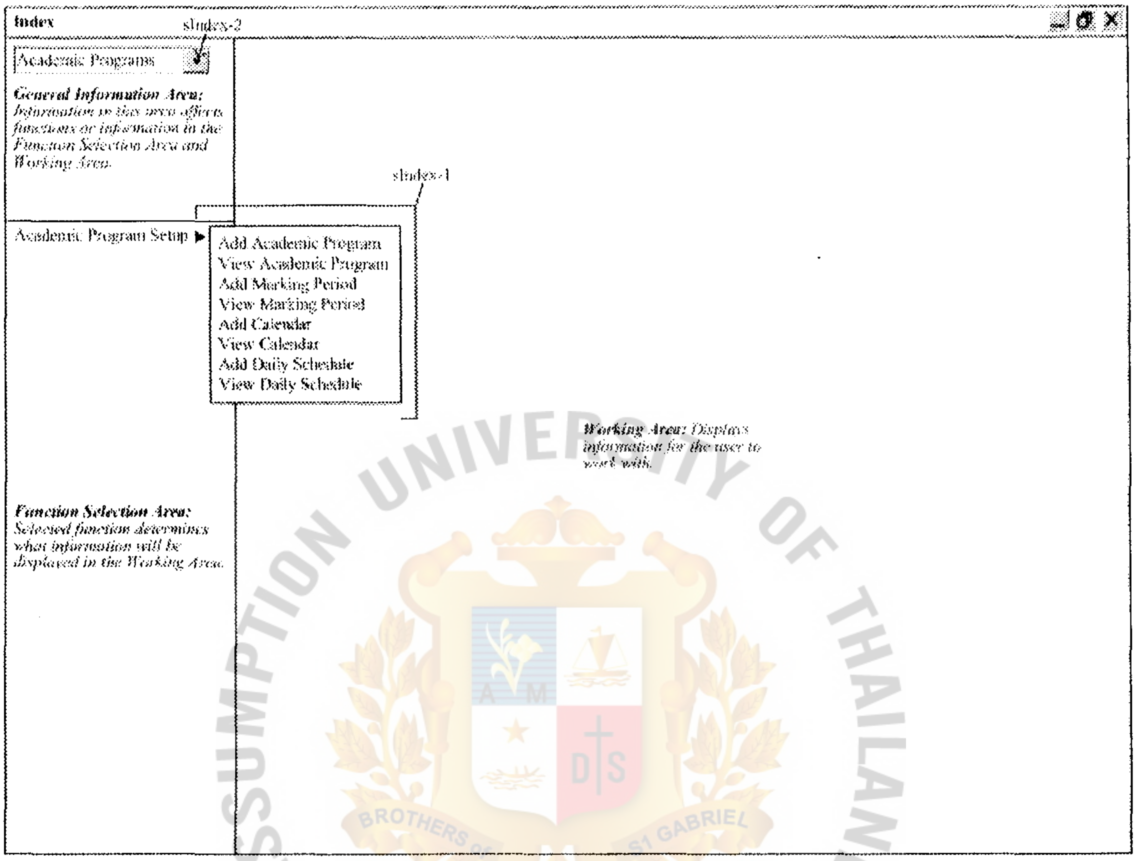
sLogin Screen Report

Screen Name:	sLogin
Description:	All users log into the system here.
Functions Involved:	fLogin
Designers:	Jia



sIndex Screen Report

Screen Name:	sIndex
Description:	This is the first screen that appears after the user has log into the system. This screen is also used to describe the layout commonly employed in most SIS screens.
Functions Involved:	fAddAcad, fViewAcad, fAddMarkPer, fViewMarkPer, fAddCal, fViewCal, fAddDlySched
Designers:	Jia



General Information Area: *Information in this area affects functions or information in the Function Selection Area and Working Area.*

Working Area: *Displays information for the user to work with.*

Function Selection Area: *Selected function determines what information will be displayed in the Working Area.*

fAddAcad Function Report

Function Name:	fAddAcad
Category Path:	Student Information System > Academic Program Functions
Description:	Add a new academic program.
Functions Involved:	-
Screens Involved:	sIndex, sAddAcad
Popups Involved:	-
Designers:	Jia

Function:

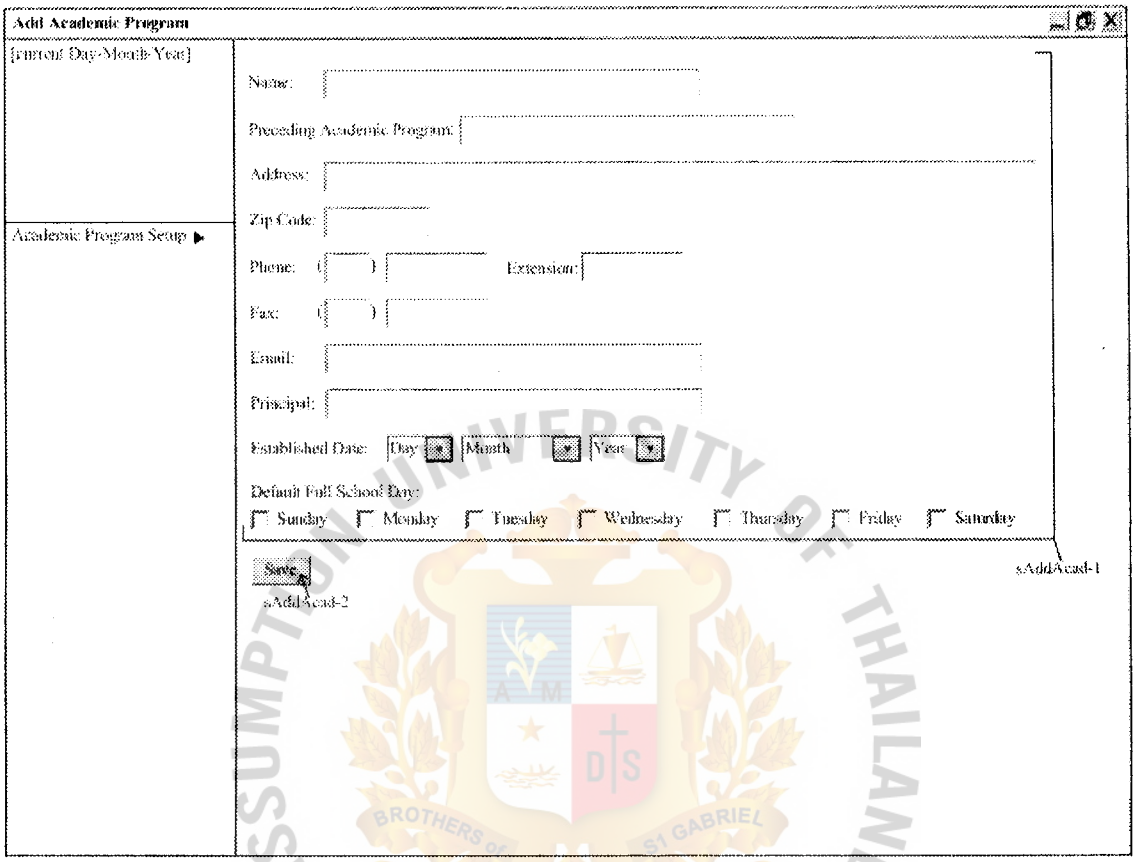
Steps	Description
1	In sIndex, place the cursor on "Academic Program Setup ▶" until a function menu appears.
2	Click on "Add Academic Program" in sIndex-1 to display sAddAcad.
3	In sAddAcad, complete the form for a new academic program in sAddAcad-1.
4	Click on sAddAcad-2 to save the new academic program. The form will reappear blanked to indicate to the user that the new academic program has been successfully saved.

Exceptions:

#	Description	Response
1	Academic program name must be unique.	Depending on which, or both exceptions are detected at once, system selectively state the followings in a popup: 1. Academic program name must be unique. 2. Established date may not exceed current date.
2	Established date may not exceed current date.	

sAddAcad Screen Report

Screen Name:	sAddAcad
Description:	Add a new academic program.
Functions Involved:	fAddAcad
Designers:	Jia



fViewAcad Function Report

Function Name:	fViewAcad
Category Path:	Student Information System > Academic Program Functions
Description:	Added academic programs may be viewed, edited, deleted, or saved as new academic programs.
Functions Involved:	-
Screens Involved:	sIndex, sViewAcad
Popups Involved:	-
Designers:	Jia

Function:

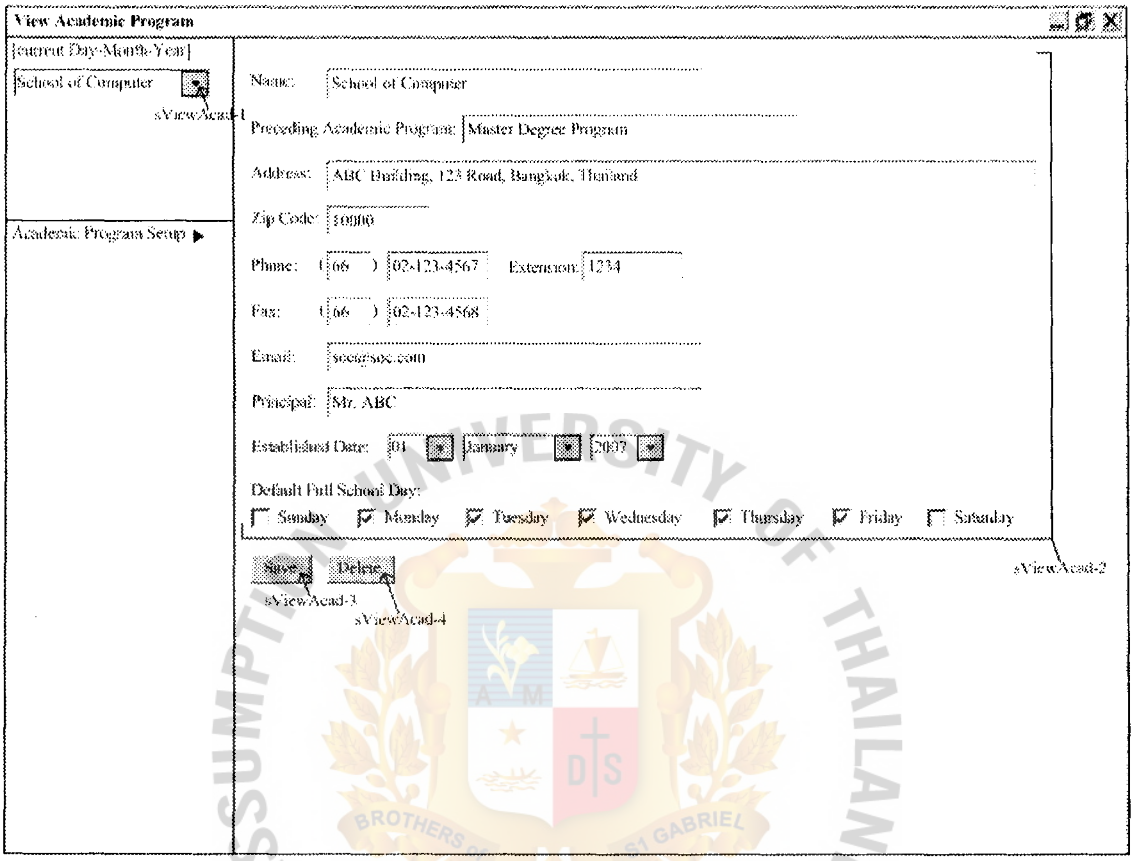
Steps	Description
1	In sIndex, click on sIndex-2 and select an academic program.
2	Place the cursor on "Academic Program Setup ▶" until a function menu appears.
3	Click on "View Academic Program" in sIndex-1 to display information of the selected academic program in sViewAcad.
4a	In sViewAcad, to edit the selected academic program, revise the form in sViewAcad-2 except "Name". Click on sViewAcad-3, to save any changes made to the selected academic program. A popup appears to inform of the successful save.
4b	In sViewAcad, to save as new academic program, change the "Name" of the academic program. Click on sViewAcad-3 to save as a new academic program. A popup appears to inform of the successful "save as".
4c	In sViewAcad, to delete the current academic program, click on sViewAcad-4 to display a popup that warns the user that a deletion is about to be performed. Click "Ok" in the popup, and a new blank form will reappear to indicate to the user that the academic program has been successfully deleted.

Exceptions:

#	Description	Response
1	In both cases of editing the selected academic program and saving as a new academic program, established date may not exceed current date.	System calls popup that states: "Established date may not exceed current date."

sViewAcad Screen Report

Screen Name: sViewAcad
Description: Added academic programs may be viewed, edited, deleted, or saved as new academic programs. Sample data is also shown.
Functions Involved: fViewAcad
Designers: Jia

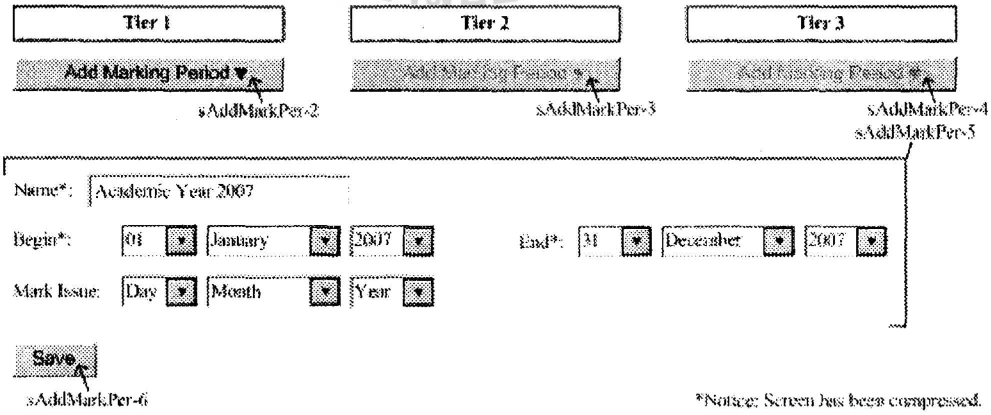


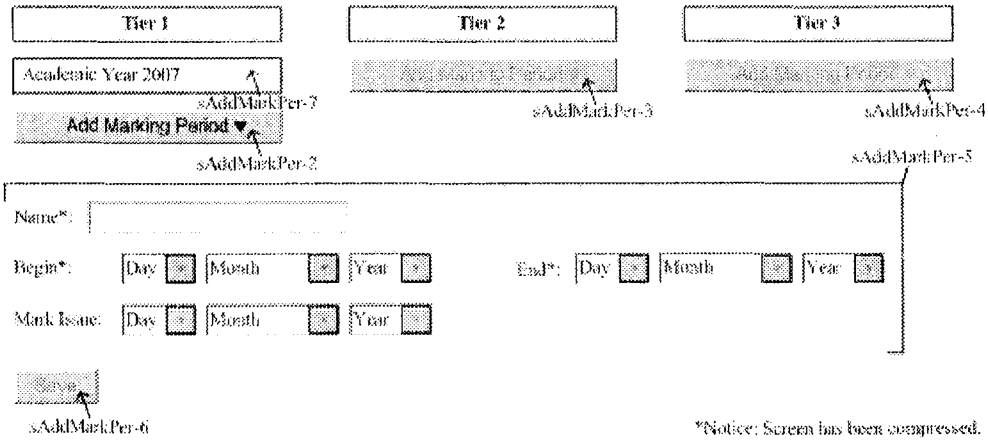
Name: School of Computer
Preceding Academic Program: Master Degree Program
Address: ABC Building, 123 Road, Bangkok, Thailand
Zip Code: 10000
Phone: (66) 02-123-4567 Extension: 1234
Fax: (66) 02-123-4568
Email: soc@soc.com
Principal: Mr. ABC
Established Date: 01 January 2007
Default Full School Day:
□ Sunday ☑ Monday ☑ Tuesday ☑ Wednesday ☑ Thursday ☑ Friday □ Saturday

fAddMarkPer Function Report

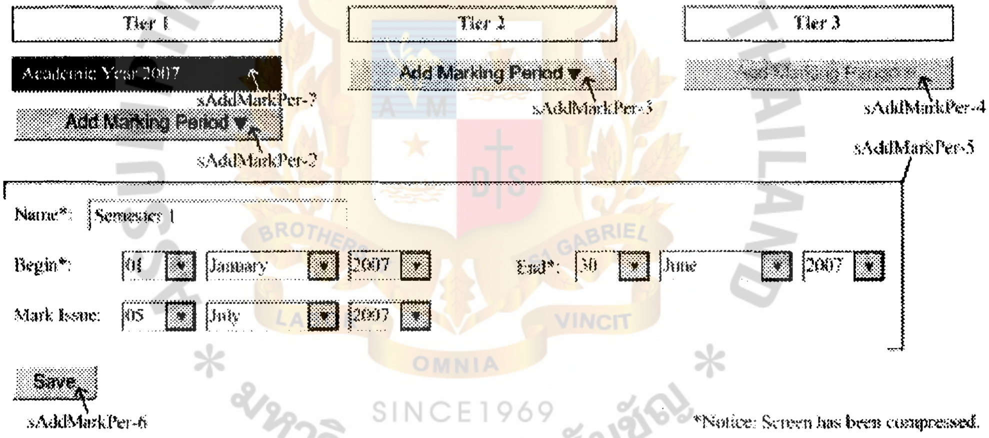
Function Name:	fAddMarkPer
Category Path:	Student Information System > Academic Program Functions
Description:	Add up to three tiers of marking periods. To demonstrate the function, the following marking periods are created for the School of Computer: Academic Year 2007 (1st tier) has 2 Semesters (2nd tier), and each Semester has 2 Quarters (3rd tier).
Functions Involved:	-
Screens Involved:	sIndex, sAddMarkPer
Popups Involved:	-
Designers:	Jia

Function:

Steps	Description
1	In sIndex, click on sIndex-2 and select an academic program, which in this case is the "School of Computer".
2	Place the cursor on "Academic Program Setup ▶" until a function menu appears.
3	Click on "Add Marking Period" in sIndex-1 to display sAddMarkPer for the selected academic program.
4	In sAddMarkPer, the initial state of the screen is shown, which no marking periods have been added yet. sAddMarkPer-3 and sAddMarkPer-4 are both disabled, and can only be enabled when a marking period of the previous tier is selected. sAddMarkPer-5 is blanked and disabled, and can only be enabled when any "Add Marking Period ▼" button is clicked. Lastly, sAddMarkPer-6 is disabled, and can only be enabled when required fields of sAddMarkPer-5, indicated by asterisks '*', are typed in or selected. To add a marking period in the first tier, for instance "Academic Year 2007", click on sAddMarkPer-2 to enable sAddMarkPer-5. Complete the form in sAddMarkPer-5 like the following figure, and click on sAddMarkPer-6 to save.  Tier 1 Tier 2 Tier 3 Add Marking Period ▼ Add Marking Period ▼ Add Marking Period ▼ sAddMarkPer-2 sAddMarkPer-3 sAddMarkPer-4 sAddMarkPer-5 Save sAddMarkPer-6 Name*: Academic Year 2007 Begin*: 01 January 2007 End*: 31 December 2007 Mark Issue: Day Month Year *Notice: Screen has been compressed.
5	The following figure portrays the screen after "Academic Year 2007" has been successfully added. sAddMarkPer-5 is again blanked and disabled, as well as sAddMarkPer-6.



To add a child marking period in a succeeding tier, for instance "Semester 1", click on a parent marking period in the preceding tier, for instance "Academic Year 2007", to highlight it. Then click on the "Add Marking Period ▼" button in the succeeding tier, for instance "sAddMarkPer-3", to enable sAddMarkPer-5. Complete the form in sAddMarkPer-5 like the following figure, and click on sAddMarkPer-6 to save.



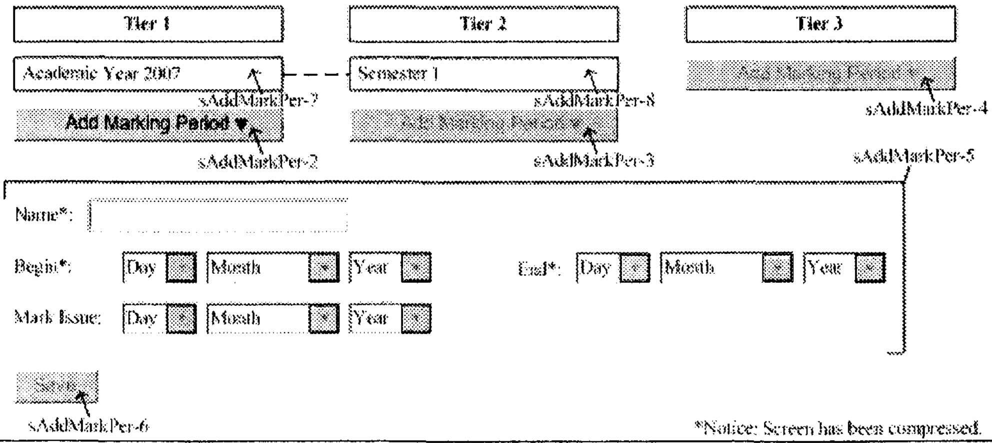
Name*: Semester 1

Begin*: 01 January 2007

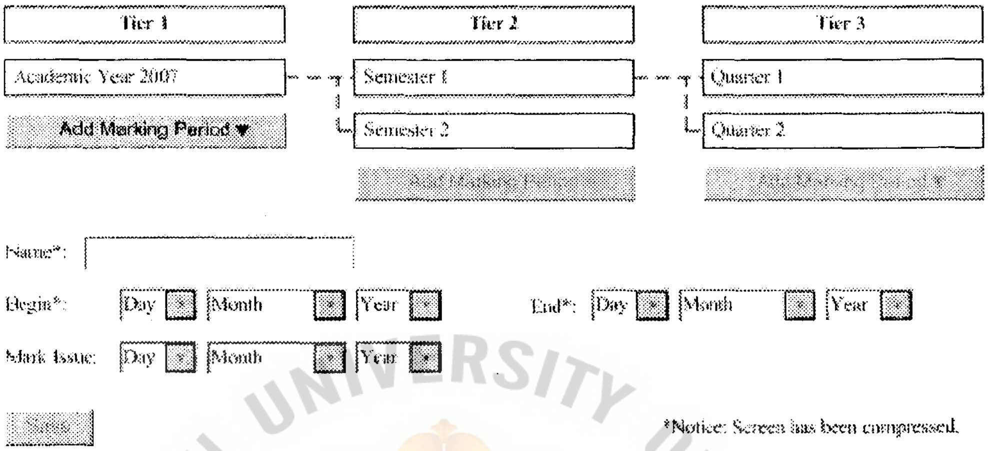
End*: 30 June 2007

Mark Issue: 05 July 2007

6 The following figure portrays the screen after "Semester 1" has been successfully added, and connected to "Academic Year 2007". sAddMarkPer-5 is again blanked and disabled, as well as sAddMarkPer-6.



To add the entire set of marking periods, follow similar instructions in Step 4 if you want to add a marking period in the first tier, or follow similar instructions in Step 5 if you want to add a child marking period in a succeeding tier. The following figure portrays the screen after the entire set of marking periods has been successfully added.



Academic Year 2007
Semester 1
Semester 2
Quarter 1
Quarter 2

Name*:
Begin*: Day Month Year
End*: Day Month Year
Mark Issue: Day Month Year

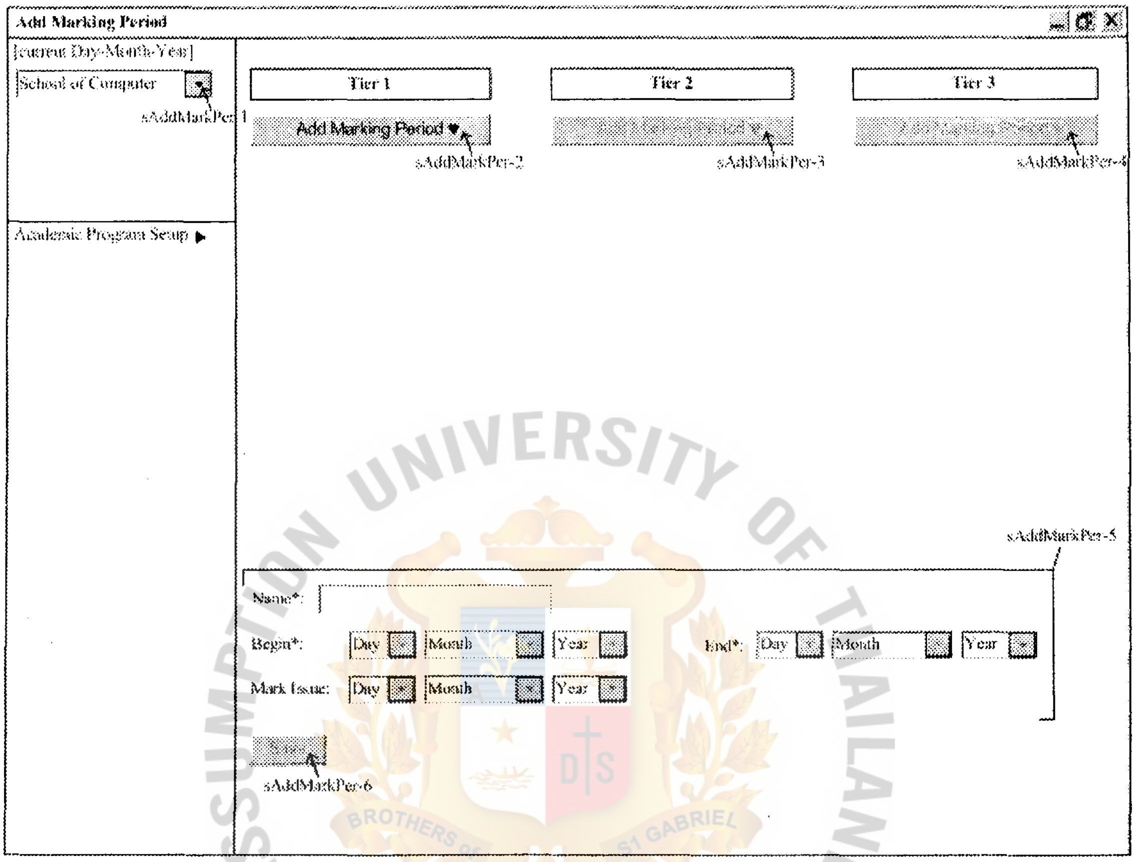
*Notice: Screen has been compressed.

Exceptions:

#	Description	Response
1	A child marking period may not possess a time frame that exceeds the time frame of parent marking period.	Calls popup that states: "A child marking period may not possess a time frame that exceeds the time frame of parent marking period."
2	Marking periods in same tier may not overlap in time.	Calls popup that states: "Marking periods in same tier may not overlap in time."
3	"Mark Issue" may not occur before "End" date.	Calls popup that states: "Mark Issue" may not occur before "End" date."

sAddMarkPer Screen Report

Screen Name: sAddMarkPer
Description: Add up to three tiers of marking periods. To demonstrate the function, the following marking periods are created for the School of Computer: Academic Year 2007 (1st tier) has 2 Semesters (2nd tier), and each Semester has 2 Quarters (3rd tier).
Functions Involved: fAddMarkPer
Designers: Jia



fViewMarkPer Function Report

Function Name:	fViewMarkPer
Category Path:	Student Information System > Academic Program Functions
Description:	Added marking periods may be viewed, edited, or deleted. To demonstrate the function, the following marking periods from the School of Computer are used: Academic Year 2007 (1st tier) has 2 Semesters (2nd tier), and each Semester has 2 Quarters (3rd tier).
Functions Involved:	-
Screens Involved:	sIndex, sViewMarkPer
Popups Involved:	-
Designers:	Jia

Function:

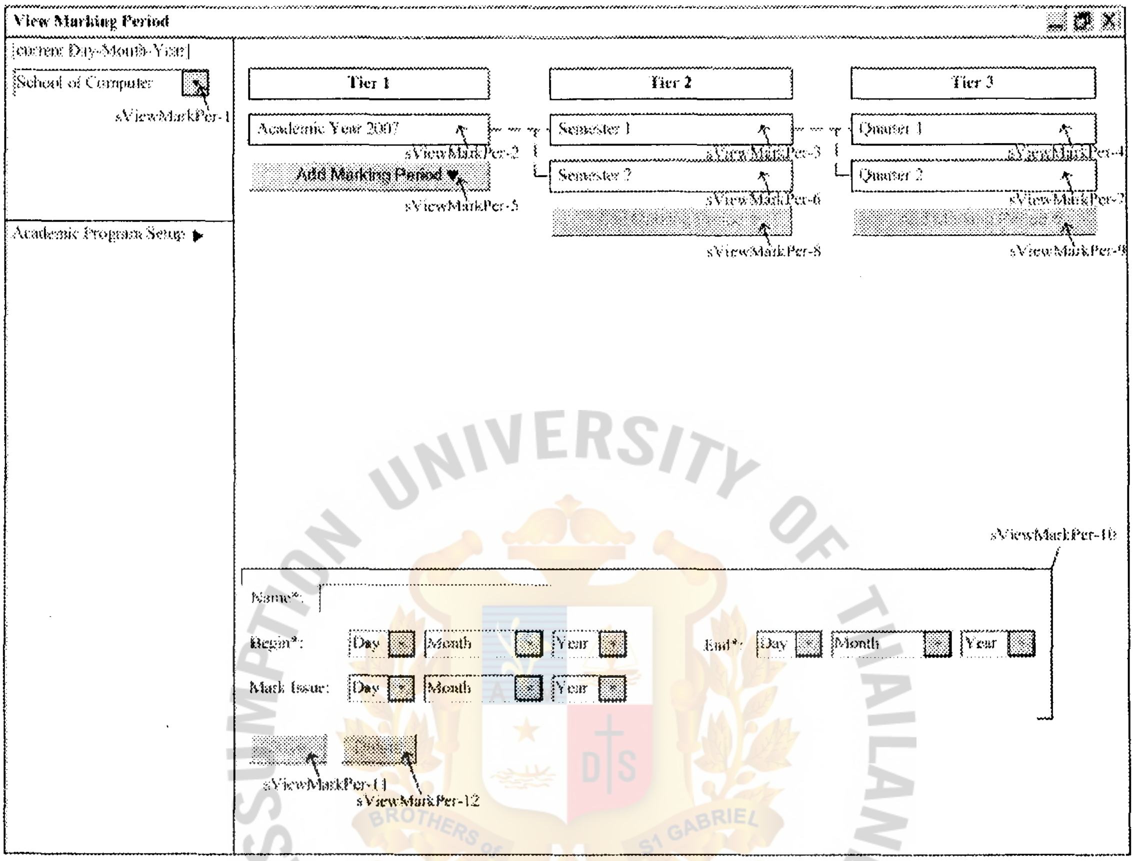
Steps	Description
1	In sIndex, click on sIndex-2 and select an academic program, which in this case is the "School of Computer".
2	Place the cursor on "Academic Program Setup ▶ " until a function menu appears.
3	Click on "View Marking Period" in sIndex-1 to display sViewMarkPer for the selected academic program.
4a	In sViewMarkPer, to view the details of a marking period, click on the marking period to highlight it. For instance, click on sViewMarkPer-2, -3, -4, -6, or -7. Details of the marking period will appear in sViewMarkPer-10.
4b	In sViewMarkPer, to edit the details of a marking period, perform step 4a to view the details of a marking period. Revise the form in sViewMarkPer-10, and click on sViewMarkPer-11 to save. sViewMarkPer-10 is blanked and disabled, as well as sViewMarkPer-11 and -12, to indicate to the user of the successful save.
4c	In sViewMarkPer, to delete a marking period, perform step 4a to view the details of a marking period. Click on sViewMarkPer-12 to delete the marking period. A popup appears to warn the user that child marking periods associated with the marking period that is about to be deleted will also be deleted. The user confirms the deletion by clicking 'Ok'. sViewMarkPer-10 is blanked and disabled, as well as sViewMarkPer-11 and -12, to indicate to the user of the successful deletion.

Exceptions:

#	Description	Response
1	Child marking periods may not possess time frames that extend beyond the time frame of parent marking period.	Calls popup that states: "Child marking periods may not possess time frames that extend beyond the time frame of parent marking period."
2	Marking periods in same tier may not overlap in time.	Calls popup that states: "Marking periods in same tier may not overlap in time."
3	"Mark Issue" may not occur before "End" date.	Calls popup that states: "Mark Issue" may not occur before "End" date."

sViewMarkPer Screen Report

Screen Name: sViewMarkPer
Description: Added marking periods may be viewed, edited, or deleted. To demonstrate the function, the following marking periods from the School of Computer are used: Academic Year 2007 (1^{st} tier) has 2 Semesters (2^{nd} tier), and each Semester has 2 Quarters (3^{rd} tier).
Functions Involved: fViewMarkPer
Designer(s): Jia



fAddCal Function Report

Function Name:	fAddCal
Category Path:	Student Information System > Academic Program Functions
Description:	Add a calendar for an academic program, for instance, the "School of Computer".
Functions Involved:	-
Screens Involved:	sIndex, sAddCal
Popups Involved:	-
Designers:	Jia

Function:

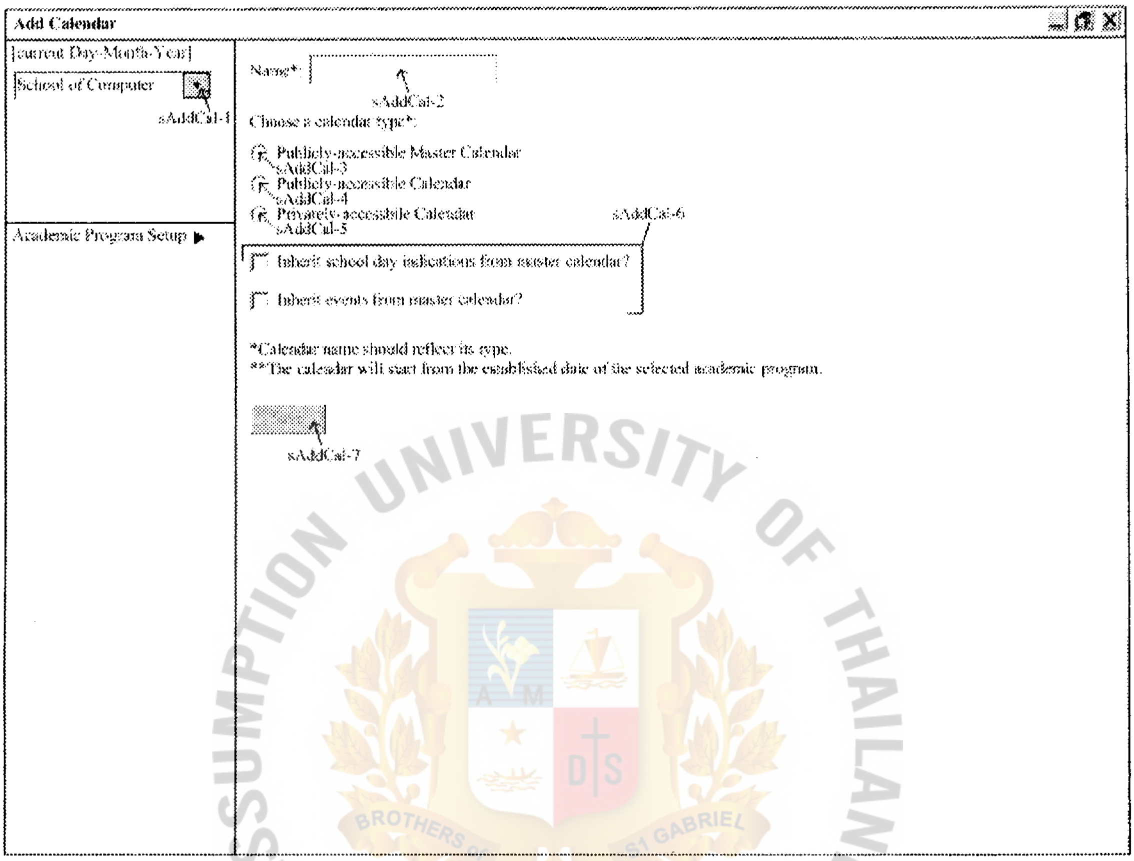
Steps	Description
1	In sIndex, click on sIndex-2 and select an academic program, which in this case is the "School of Computer".
2	Place the cursor on "Academic Program Setup ▶" until a function menu appears.
3	Click on "Add Calendar" in sIndex-1 to display sAddCal for the selected academic program.
4	In sAddCal, the followings are possible initial states of the screen: • sAddCal-3 is only enabled when no master calendar has been created for the branch of academic programs, and user is authorized to create one. • sAddCal-4 is only enabled when user is authorized to create one. • sAddCal-7 is disabled, and can only be enabled when required fields, indicated by asterisks '*', are typed in or selected. These authorized users can also edit and delete the calendars that they have added. Complete the form, spanning from sAddCal-2 to -6, for a new calendar. If sAddCal-3 is selected, sAddCal-6 becomes unchecked and disabled.
5	Click on sAddCal-7 to save the new calendar. The form will reappear blanked to indicate to the user that the new calendar has been successfully saved.

Exceptions:

#	Description	Response
1	Calendar name must be unique.	System calls popup that states: "Calendar name must be unique."

sAddCal Screen Report

Screen Name: sAddCal
Description: Add a calendar for an academic program, for instance, the "School of Computer".
Functions Involved: fAddCal
Designers: Jia



Add Calendar

[current Day-Month-Year]

School of Computer ▾

Academic Program Setup ▶

Name*:

Choose a calendar type*:

○ Publicly-accessible Master Calendar
○ Publicly-accessible Calendar
○ Privately-accessible Calendar

□ Inherit school day indications from master calendar?

□ Inherit events from master calendar?

*Calendar name should reflect its type.
**The calendar will start from the established date of the selected academic program.

fViewCal Function Report

Function Name:	fViewCal
Category Path:	Student Information System > Academic Program Functions
Description:	Added calendars may be viewed or deleted. To demonstrate the function, the School of Computer is selected.
Functions Involved:	-
Screens Involved:	sIndex, sViewCal
Designers:	Jia

Function:

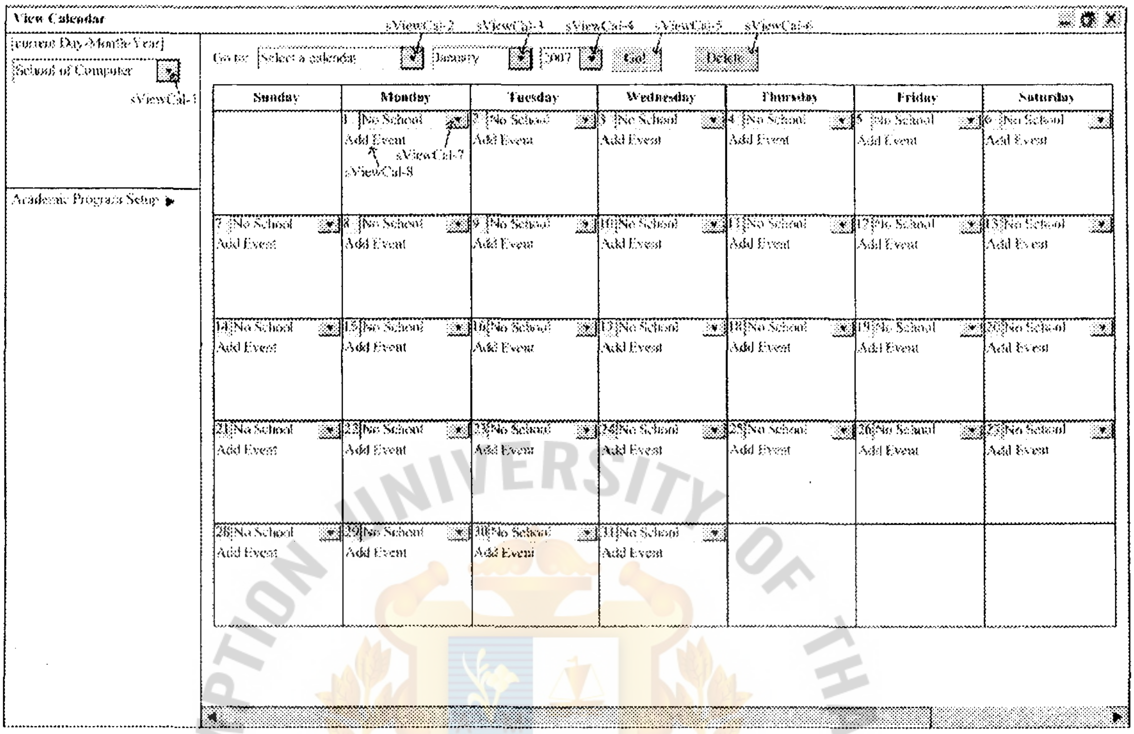
Steps	Description
1	In sIndex, click on sIndex-2 and select an academic program, which in this case is the "School of Computer".
2	Place the cursor on "Academic Program Setup ▶ " until a function menu appears.
3	Click on "View Calendar" in sIndex-1 to display sViewCal for the selected academic program.
4	In sViewCal, the initial state of the screen is shown: a calendar waits to be selected in sViewCal-2, sViewCal-3 and -4 automatically selects the current month and year by default, and a calendar is shown in default values. To view a calendar that has been added, click on sViewCal-2 to select a calendar, click on sViewCal-3 and -4 to select a month and year respectively that the calendar will display, and click on sViewCal-5 to display the calendar. All saved information such as school day indications and events will also be displayed in the calendar.
5	To delete the calendar in display, click on sViewCal-6 to display a popup that warns the user that a deletion is about to be performed. Click "Ok" in the popup, and the screen will return to the initial state as described in Step 4 to indicate to the user that the calendar has been successfully deleted.

Exceptions:

#	Description	Response
1	Deletion is attempted before a calendar is selected.	System calls popup that states: "A calendar must be selected before deletion."

sViewCal Screen Report

Screen Name:	sViewCal
Description:	Added calendars may be viewed or deleted. To demonstrate the function, the School of Computer is selected.
Functions Involved:	fViewCal, fSchDayCal, fEventCal
Designers:	Jia



[current Day-Month-Year]

School of Computer ▾

Academic Program Setup ▶

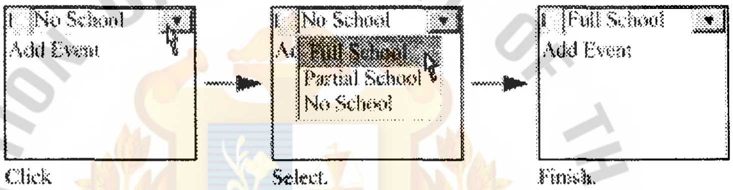
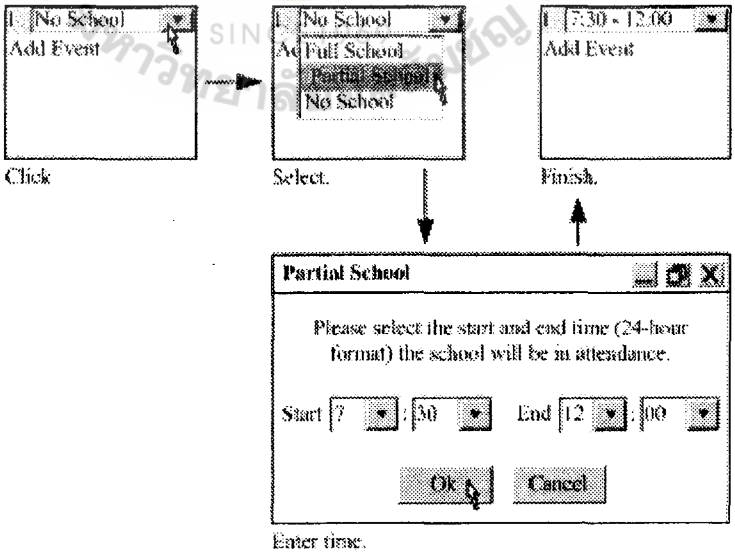
Go to: Select a calendar ▾ January ▾ 2007 ▾ Go! Delete

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	1 No School ▾ Add Event	2 No School ▾ Add Event	3 No School ▾ Add Event	4 No School ▾ Add Event	5 No School ▾ Add Event	6 No School ▾ Add Event
7 No School ▾ Add Event	8 No School ▾ Add Event	9 No School ▾ Add Event	10 No School ▾ Add Event	11 No School ▾ Add Event	12 No School ▾ Add Event	13 No School ▾ Add Event
14 No School ▾ Add Event	15 No School ▾ Add Event	16 No School ▾ Add Event	17 No School ▾ Add Event	18 No School ▾ Add Event	19 No School ▾ Add Event	20 No School ▾ Add Event
21 No School ▾ Add Event	22 No School ▾ Add Event	23 No School ▾ Add Event	24 No School ▾ Add Event	25 No School ▾ Add Event	26 No School ▾ Add Event	27 No School ▾ Add Event
28 No School ▾ Add Event	29 No School ▾ Add Event	30 No School ▾ Add Event	31 No School ▾ Add Event			

fSchDayCal Function Report

Function Name:	fSchDayCal
Category Path:	Student Information System > Academic Program Functions
Description:	Indicate full, partial, or no school days on a calendar.
Functions Involved:	fViewCal
Screens Involved:	sViewCal
Popups Involved:	-
Designers:	Jia

Function:

Steps	Description
1	Perform fViewCal to view a calendar in sViewCal.
2a	In sViewCal, to indicate full or no school for a particular day, click on the school day combo box such as sViewCal-7. Select "Full School" or "No School", and the selection will be displayed in the school day combo box. The following figure provides a demonstration:  1 No School Add Event Click 1 No School Full School Partial School No School Select. 1 Full School Add Event Finish.
2b	In sViewCal, to indicate partial school for a particular day, click on the school day combo box such as sViewCal-7. Select "Partial School", and a popup will appear asking for the start and end time the school will be in attendance. Select the start and end time and click "Ok". The start and end time will be displayed in the school day combo box. The following figure provides a demonstration:  1 No School Add Event Click 1 No School Full School Partial School No School Select. 1 7:30 - 12:00 Add Event Finish. Partial School Please select the start and end time (24-hour format) the school will be in attendance. Start 7 : 30 End 12 : 00 Ok Cancel Enter time.

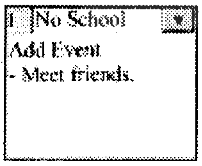
Exceptions:

#	Description	Response
1	No exceptions.	-

fEventCal Function Report

Function Name:	fEventCal
Category Path:	Student Information System > Academic Program Functions
Description:	Add, edit, or delete events in calendars.
Functions Involved:	fViewCal
Screens Involved:	sViewCal
Popups Involved:	pEventCal
Designers:	Jia

Function:

Steps	Description
1	Perform fViewCal to view a calendar in sViewCal.
2a	In sViewCal, to add an event for a particular day, click on "Add Event" such as sViewCal-8 to display pEventCal.
3a	In pEventCal, the initial state of the popup is shown: • pEventCal-4 to -6 are only enabled when pEventCal-3 is anything but "Never". • pEventCal-9 is only enabled when required fields, indicated by asterisks '*', are typed in or selected. • pEventCal-7 and -8 are only enabled when a recurring event that has been saved is displayed. • pEventCal-10 is only enabled when an event, recurring or non-recurring, that has been saved is displayed. To add an event, type in the name and description of the event in pEventCal-1 and pEventCal-2 respectively. Click on pEventCal-3 to select whether to never repeat the event, repeat daily, weekly, monthly, or yearly, from the current day onwards. If select to repeat event, you may choose to select the end date of the recurring event in pEventCal-4 to -6. The end date before the current day is unavailable. Click on pEventCal-9 to save the event. The event's name appears on the calendar similar to the following figure:  1 No School Add Event - Meet friends. *Scroll bars will appear if space is insufficient for displaying events in a particular day.
2b	In sViewCal, to edit an existing event, click on the event to display pEventCal that contains details of the event.
3b	In pEventCal of the event, edit the name and description of the event in pEventCal-1 and pEventCal-2 respectively. If the event is a non-recurring event and the user wants to change it into a recurring event, click on pEventCal-3 to select whether to repeat daily, weekly, monthly, or yearly, from the current day onwards. You may choose to select the end date of the recurring event in pEventCal-4 to -6. The end date before the current day is unavailable. Click on pEventCal-9 to save the

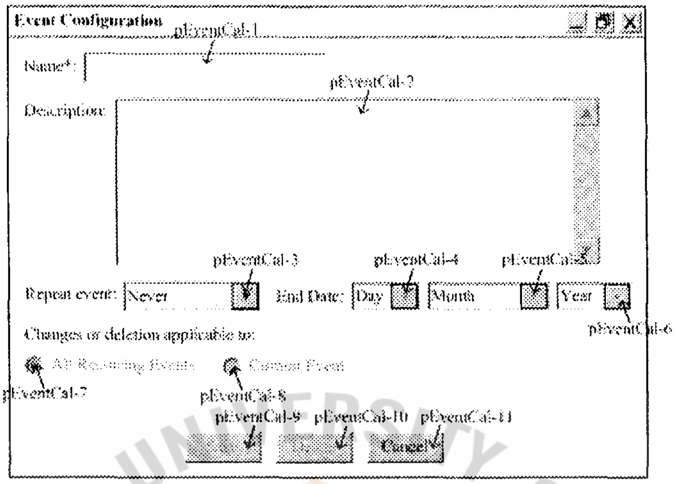
	event and return to sViewCal to indicate to the user that the event has been successfully saved. If the event is a recurring event and the user wants to apply the changes only to the current event, click on pEventCal-8 and the system ignores and disables pEventCal-3 to -6. Click on pEventCal-9 to save the event and return to sViewCal to indicate to the user that the event has been successfully saved. If the event is a recurring event and the user wants to apply the changes to all recurring events, click on pEventCal-7, optionally change pEventCal-3 to -6, and click on pEventCal-9 to save the event and return to sViewCal to indicate to the user that the event has been successfully saved.
2c	In sViewCal, to delete an existing event, click on the event to display pEventCal that contains details of the event.
3c	In pEventCal of the event . . . if the event is a non-recurring event, click on pEventCal-10 to delete the event and return to sViewCal to indicate to the user that the event has been successfully deleted. if the event is a recurring event and the user wants to delete only the current event, click on pEventCal-8, click on pEventCal-10 to delete the current event, and return to sViewCal to indicate to the user that the event has been successfully deleted. if the event is a recurring event and the user wants to delete all recurring events, click on pEventCal-7, click on pEventCal-10 to delete all recurring events, and return to sViewCal to indicate to the user that the events has been successfully deleted.

Exceptions:

#	Description	Response
1	No exceptions.	-

pEventCal Popup Report

Popup Name:	pEventCal
Description:	Add, edit, or delete events in calendars.
Functions Involved:	fEventCal
Designers:	Jia



Event Configuration

Name*:

Description:

Repeat event: Never

End Date: Day Month Year

Changes or deletion applicable to:

All Recurring Events

Current Event

fAddDlySchd Function Report

Function Name:	fAddDlySchd
Category Path:	Student Information System > Academic Program Functions
Description:	Add a daily schedule for an academic program, for instance, the "School of Computer".
Functions Involved:	-
Screens Involved:	sIndex, sAddDlySchd
Popups Involved:	-
Designers:	Jia

Function:

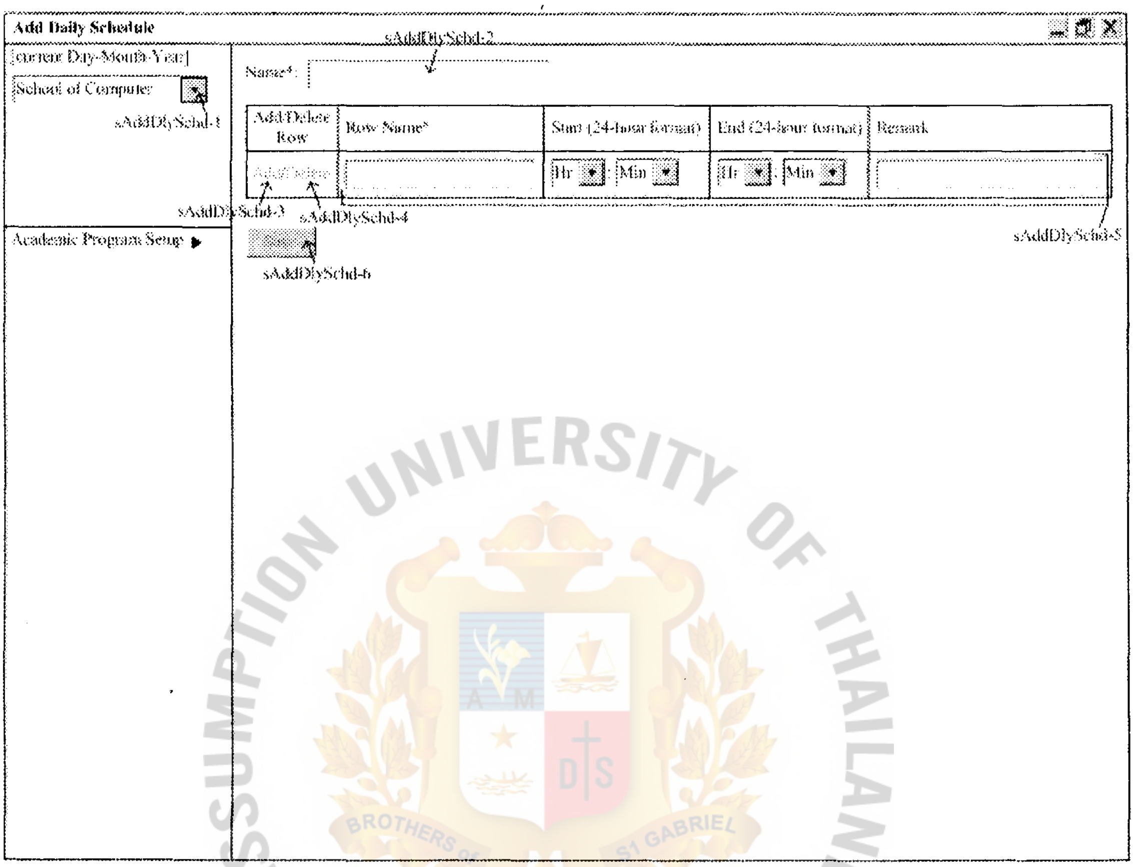
Steps	Description
1	In sIndex, click on sIndex-2 and select an academic program, which in this case is the "School of Computer".
2	Place the cursor on "Academic Program Setup ►" until a function menu appears.
3	Click on "Add Daily Schedule" in sIndex-1 to display sAddDlySchd for the selected academic program.
4a	In sAddDlySchd, the initial state of the screen is shown: • sAddDlySchd-3 is only enabled when "Row Name" in sAddDlySchd-5 is typed in. • sAddDlySchd-4 is only enabled when the corresponding row has been added. However, there must be at least one remaining row. • sAddDlySchd-6 is only enabled when sAddDlySchd-2 has been typed in. These conditions also similarly apply in sViewDlySchd. To add a row in the daily schedule, type in and select row details in sAddDlySchd-5, and click sAddDlySchd-3 to add the row. A new row appears to inform the user that the previous row has been successfully added.
4b	To delete a row in the daily schedule, click on sAddDlySchd-4 of a row that has been added.
5	To save the daily schedule, type in daily schedule name in sAddDlySchd-2, and click sAddDlySchd-6. A popup appears to inform the user that the daily schedule has been successfully saved.

Exceptions:

#	Description	Response
1	Daily schedule name must be unique.	System calls popup that states: "Daily schedule name must be unique."
2	'Start' and 'End' time cannot be the same.	Depending on which time is selected first, system prohibits the user from selecting the same time in the later set of combo boxes.

sAddDlySchd Screen Report

Screen Name: sAddDlySchd
Description: Add a daily schedule for an academic program, for instance, the "School of Computer".
Functions Involved: fAddDlySchd
Designers: Jia



Add Daily Schedule

[current Day-Month-Year]

School of Computer

Name*:

Add/Delete Row	Row Name*	Start (24-hour format)	End (24-hour format)	Remark
Add/Delete		Hr : Min	Hr : Min	

fViewDlySchd Function Report

Function Name:	fViewDlySchd
Category Path:	Student Information System > Academic Program Functions
Description:	Added daily schedule may be viewed, edited, deleted, or saved as new daily schedule. To demonstrate the function, the School of Computer and its "Full School" daily schedule are selected.
Functions Involved:	fAddDlySchd
Screens Involved:	sIndex, sAddDlySchd, sViewDlySchd
Popups Involved:	-
Designers:	Jia

Function:

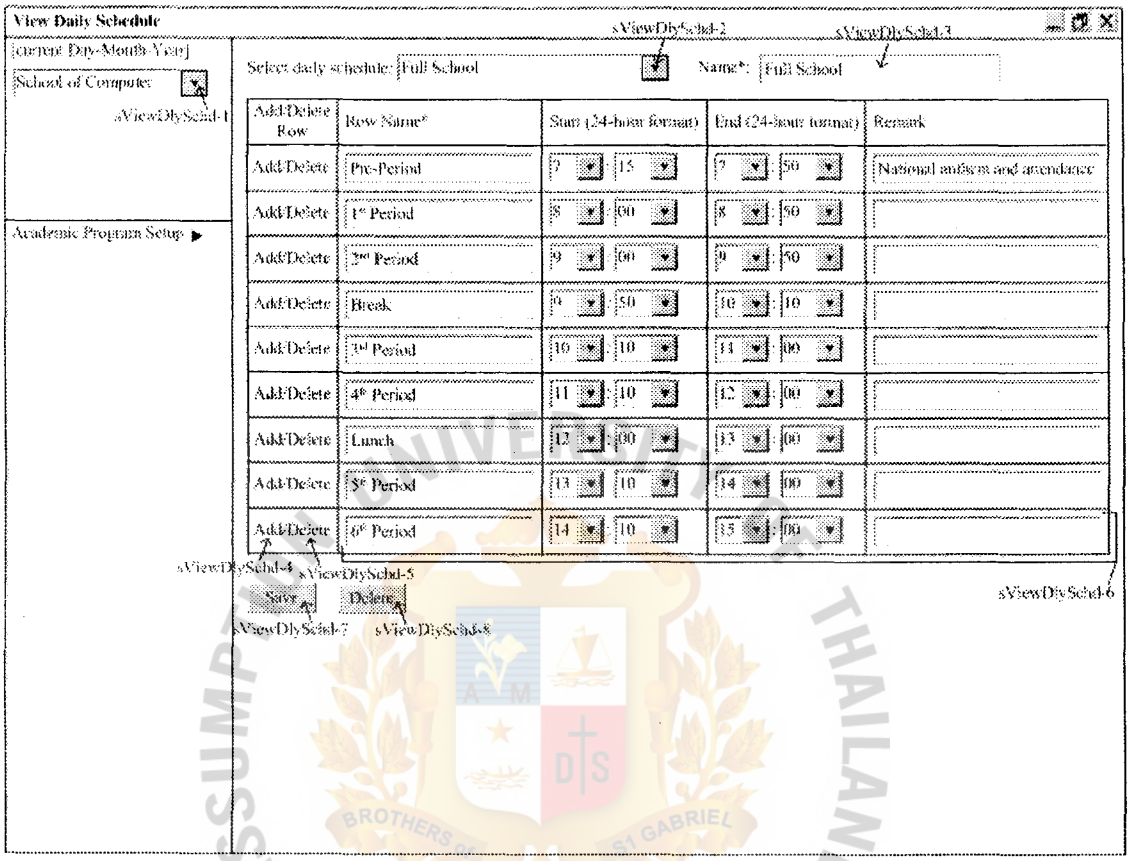
Steps	Description
1	In sIndex, click on sIndex-2 and select an academic program, which in this case is the "School of Computer".
2	Place the cursor on "Academic Program Setup ►" until a function menu appears.
3	Click on "View Daily Schedule" in sIndex-1 to display sAddDlySchd for the selected academic program.
4	In sAddDlySchd, to view a schedule, select a daily schedule such as "Full School" in sAddDlySchd-2. Details of the daily schedule will be loaded onto the screen, and the selected daily schedule's name will appear in sAddDlySchd-3. The screen now becomes sViewDlySchd.
5a	In sViewDlySchd, to edit row details, type in or select row details such as those found in sViewDlySchd-6.
5b	In sViewDlySchd, to add or delete rows, consult Step 4a or 4b in fAddDlySchd.
5c	In sViewDlySchd, to save the edited daily schedule, click on sViewDlySchd-7. A popup appears to inform the user that the daily schedule has been successfully saved.
5d	In sViewDlySchd, to save as new daily schedule, change the "Name" of the daily schedule in sViewDlySchd-3. Click on sViewDlySchd-7 to save as a new daily schedule. A popup appears to inform of the successful "save as".
5e	In sViewDlySchd, to delete a daily schedule, click on sViewDlySchd-8 to display a popup that warns the user that a deletion is about to be performed. Click "Ok" in the popup, and sAddDlySchd will be displayed to indicate to the user that the daily schedule has been successfully deleted.

Exceptions:

#	Description	Response
1	Daily schedule name must be unique.	System calls popup that states: "Daily schedule name must be unique."
2	'Start' and 'End' time cannot be the same.	Depending on which time is selected first, system prohibits the user from selecting the same time in the later set of combo boxes.

sViewDlySchd Screen Report

Screen Name:	sViewDlySchd
Description:	Added daily schedule may be viewed, edited, or deleted. To demonstrate the function, the School of Computer and its "Full School" daily schedule are selected.
Functions Involved:	fViewDlySchd
Designers:	Ea



View Daily Schedule

[current Day-Month-Year]

School of Computer ▼

Select daily schedule: Full School ▼

Name*: Full School

Add/Delete Row	Row Name*	Start (24-hour format)	End (24-hour format)	Remark
Add/Delete	Pre-Period	7 ▼ : 15 ▼	7 ▼ : 50 ▼	National anthem and attendance
Add/Delete	1st Period	8 ▼ : 00 ▼	8 ▼ : 50 ▼	
Add/Delete	2nd Period	9 ▼ : 00 ▼	9 ▼ : 50 ▼	
Add/Delete	Break	9 ▼ : 50 ▼	10 ▼ : 10 ▼	
Add/Delete	3rd Period	10 ▼ : 10 ▼	11 ▼ : 00 ▼	
Add/Delete	4th Period	11 ▼ : 10 ▼	12 ▼ : 00 ▼	
Add/Delete	Lunch	12 ▼ : 00 ▼	13 ▼ : 00 ▼	
Add/Delete	5th Period	13 ▼ : 10 ▼	14 ▼ : 00 ▼	
Add/Delete	6th Period	14 ▼ : 10 ▼	15 ▼ : 00 ▼	

4.3 Change Management

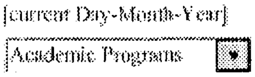
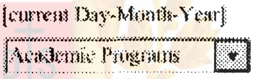
S3M blueprints also face edits, deletions, or save as new. For instance, SIS reports are created sequentially according to function report lists in the FunctionList document. This causes earlier reports to miss some features found in subsequent reports, features are added to earlier reports on request, or widget numberings do not start with the usual convention from left to right and top to bottom of the screen. Although earlier reports are sufficient in their own rights, they still need to synchronize with subsequent reports for screen transition uniformity.

System designers may perform minute changes instantly, and document the more significant potential blueprint alterations simply as "change requests". Change requests may simply serve as reminders to system designers, or require the decisions of project stakeholders to proceed. In the later case, formal documentations are required.

In S3M, change requests can be documented in the README document. If the README document belongs to a new folder containing the updated version of the blueprint, it can be used to inform stakeholders of "what is new" in the latest blueprint. Table 4.1 is one possible change requests table that adopts the familiar layout from the exceptions table found in function reports. The table comprises of SIS change requests, which provides more insights into S3M. The 'Description' column describes the concepts behind each change requests, the 'Response' column suggests specific actions required to realize each change requests, and the "Designer/Request Status" column identifies the system designer responsible for each change requests and status of the job. Again, by applying narration-illustration principles, change requests can be communicated accurately. Inevitably, the system designer also needs to be able to assess the impact of each change request, and revise other documents and reports associated with each change request.

Table 4.1. SIS Change Request

Change Requests:

#	Description	Response	Designer/ Request Status
1	In sIndex, there is no indication of current date above sIndex-2 in the General Information Area. This causes variance among screens.	Include the current date like the following figure:  [current Day-Month-Year] Academic Programs	Jia Work in progress
2	In sIndex, widget numberings does not start from left to right and top to bottom of the screen.	Retag widgets in the screen.	Jia Rejected because sIndex is used by many functions.
3	sAddAcad and sViewAcad have similar compositions therefore, synchronize the screens.	3.1 Change both screen titles to "Academic Program". 3.2 Include the "Academic Programs" combo box in sAddAcad like the following figure:  [current Day-Month-Year] Academic Programs 3.3 Insert sViewAcad-4 into sAddAcad. 3.4 Change "Add Academic Program" and "View Academic Program" in sIndex-2 to "Academic Program".	Jia Work in progress
4	sAddMarkPer and sViewMarkPer have similar compositions therefore, synchronize the screens.	4.1 Change both screen titles to "Marking Period". 4.2 Insert sViewMarkPer-12 into sAddMarkPer. 4.3 Change "Add Marking Period" and "View Marking Period" in sIndex-2 to "Marking Period".	Jia Work in progress
5	In sAddCal, there is no check box for the calendar to inherit default full school days from sAddAcad.	Insert a check box with the following description: "Inherit default full school days from the academic program".	Jia Work in progress
6	Calendar have no place to be	6.1 Change both screen titles	Jia

	edited and save-as. To solve this, synchronize sAddCal and sViewCal.	to "Calendar". 6.2 Insert sAddCal-2 to -7 into sViewCal. 6.3 Insert sViewCal-2 to -8 into sAddCal. 6.4 Change "Add Calendar" and "View Calendar" in sIndex-2 to "Calendar".	Work in progress
7	sAddDlySchd and sViewDlySchd have similar compositions therefore, synchronize the screens.	7.1 Change both screen titles to "Daily Schedule". 7.2 Insert sViewDlySchd-2 and -8 into sAddDlySchd. 7.3 Change "Add Daily Schedule" and "View Daily Schedule" in sIndex-2 to "Daily Schedule".	Jia Work in progress
8	Unanswered questions concerning marking periods: • Are there limits in the number of marking periods in a tier? • How long are marks that are issued, posted?	-	Jia Pending
9	Unanswered questions concerning calendars: • How will a master calendar influence other calendars residing in the same node? • What are publicly-accessible calendars? • What are privately-accessible calendars?	-	Jia Pending

V. MODELS COMPARISON

This section compares S3M, UML models, and software prototypes in Table 5.1. The table has six columns: Name, Short Description, Modeling Category, Prerequisite, Narrative Aspect, and Visual Aspect. Name and Short Description are straightforward, but remaining columns require more discussions.

The Modeling Category column asks the question: which modeling category does the model fall into? Receiving multiple categories reflect the model's comprehensive capabilities. Eight modeling categories and their corresponding models are based on Ambler's Iterative Modeling figure (Figure 5.1). The Usage Modeling category identifies how people work with the system, and questions to ask are: what will users do with the system, and how will the system support that usage? Similar to Usage Modeling, Process Modeling also identifies how people work with the system, but takes into account the flow of activities being performed. User Interface (UI) Modeling identifies UI requirements and addresses system usability issues. The Supplementary Requirements Modeling category recognizes details that usage or UI modeling activities may not be able to identify effectively. The Conceptual Domain Modeling, also called conceptual modeling or domain modeling, identifies the entities, their responsibilities, and their relationships within the problem domain. Architectural Modeling identifies the high-level design or "general landscape" of a system. Dynamic Object Modeling identifies the behavioral aspects of an object system, and finally, Ambler (2007) did not define the Detailed Structural Modeling category but provides only the modeling activities in this category.

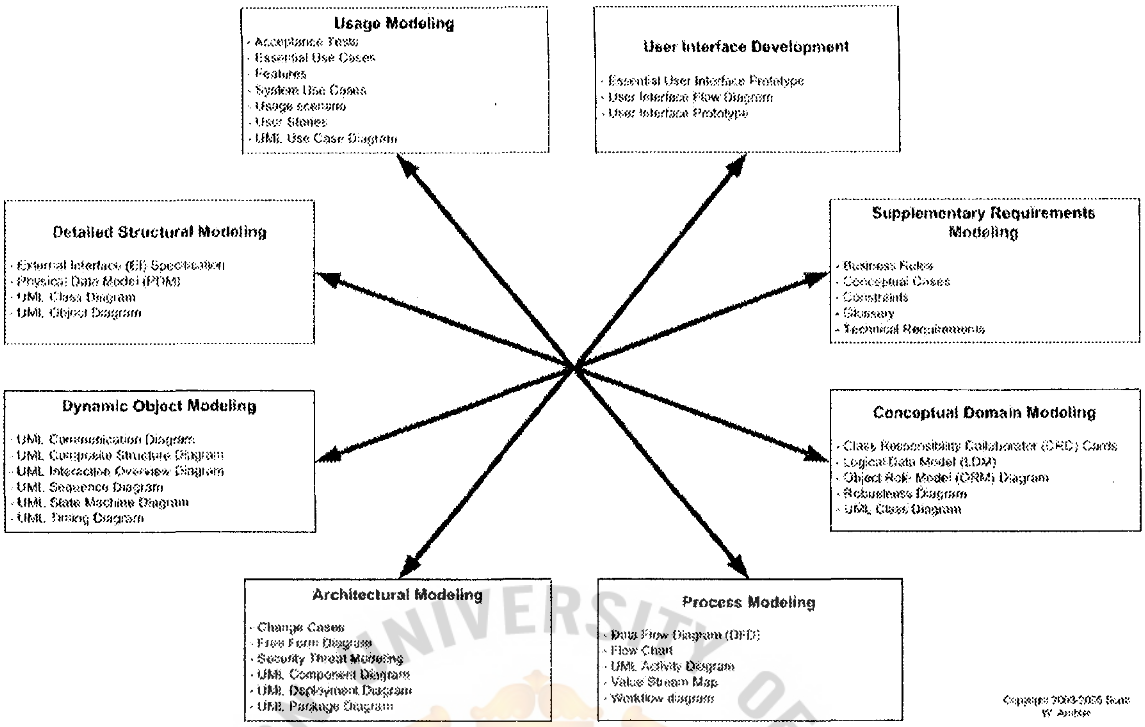

Usage Modeling
· Acceptance Tests
· Essential Use Cases
· Features
· System Use Cases
· Usage scenario
· User Stories
· UML Use Case Diagram
User Interface Development
· Essential User Interface Prototype
· User Interface Flow Diagram
· User Interface Prototype
Detailed Structural Modeling
· External Interface (EI) Specification
· Physical Data Model (PDM)
· UML Class Diagram
· UML Object Diagram
Supplementary Requirements Modeling
· Business Rules
· Conceptual Cases
· Constraints
· Glossary
· Technical Requirements
Dynamic Object Modeling
· UML Communication Diagram
· UML Composite Structure Diagram
· UML Interaction Overview Diagram
· UML Sequence Diagram
· UML State Machine Diagram
· UML Timing Diagram
Conceptual Domain Modeling
· Class Responsibility Collaborator (CRC) Cards
· Logical Data Model (LDM)
· Object Role Model (ORM) Diagram
· Robustness Diagram
· UML Class Diagram
Architectural Modeling
· Change Cases
· Free Form Diagram
· Security Threat Modeling
· UML Component Diagram
· UML Deployment Diagram
· UML Package Diagram
Process Modeling
· Data Flow Diagram (DFD)
· Flow Chart
· UML Activity Diagram
· Value Stream Map
· Workflow diagram
Copyright 2003-2006 Scott W. Ambler

Figure 5.1. Iterative Modeling (Ambler, 2007).

The Prerequisite column asks what the user needs to master before he/she can build the model. A prerequisite is not part of the method, but something external. It is also not part of the software system being studied. Having prerequisites reflect the technicality or difficulty of the model. Technical prerequisites, especially, hinder users from building the model effectively. The Prerequisite column allows two possible answers: a listing of outstanding prerequisites, or None.

The Narrative Aspect column asks whether the model includes narrations or not, and the Visual Aspect column asks whether the model includes illustrations, graphics, or not. The response to these columns is a simple Yes or No. However, it is most beneficial for the model to possess both structured narrations and illustrations. This is heavily influenced by best practices from filmmaking. Narrative aspect complements the visual aspect by clarifying illustrations, as visual aspect complements the narrative aspect by representing each set of narrations with a single visual interpretation to minimize ambiguity. In this way, precision is achieved.

Table 5.1. Comparison of S3M, UML Models, and Software Prototypes.

Name	Short Description	Modeling Category	Prerequisite	Narrative Aspect	Visual Aspect
Activity Diagram	Demonstrates the progression of events in a system.	Process Modeling	Object-oriented concepts	No	Yes
Canonical abstract prototype	Bridges abstraction and realization in user interface design by using specially designed sets of symbols.	User Interface Modeling	None	No	Yes
Class Diagram	Shows classes and their relationships within a system.	Detailed Structural Modeling	Object-oriented concepts	No	Yes
Communication Diagram	Focus on object relationships via sequenced messages coupled by arrows pointing in the direction of the message flow.	Dynamic Object Modeling	Object-oriented concepts	No	Yes
Component Diagram	Depicts components such as files, headers, link libraries, modules, executables, and packages, and their dependencies or semantic relationships in a system.	Architectural Modeling	Object-oriented concepts	No	Yes
Composite Structure Diagram	Reveals the internal structure of a class, including	Dynamic Object Modeling	Object-oriented concepts	No	Yes

	structured classifiers, parts, ports, connectors, and collaborations.				
Deploy-ment Diagram	Captures the configuration of run-time hardware nodes and corresponding software components that run on those nodes.	Architec-tural Modeling	Object-oriented concepts	No	Yes
Essential User Interface Prototype	Represents the general ideas behind the user interface in a technology-independent manner by using whiteboards, flip-chart paper, and sticky notes.	User Interface Modeling	None	No	Yes
Interac-tion Overview Diagram	Closely related to the activity diagram, but introduces two new elements: interaction occurrences and interaction elements.	Dynamic Object Modeling	Object-oriented concepts	No	Yes
Object Diagram	Provides a snapshot of an entire or partial view of a system at a point in time.	Detailed Structural Modeling	Object-oriented concepts	No	Yes
Package Diagram	Models logical containers or packages and their relationships at a high-level overview.	Architec-tural Modeling	Object-oriented concepts	No	Yes
PICTIVE	A participatory system design technique that makes video and	User Interface Modeling	None	Yes	Yes

	voice recordings of users modifying PICTIVE mock-ups of a system.				
Sequence Diagram	Models communications between objects and the messages that trigger those communications.	Dynamic Object Modeling	Object-oriented concepts	No	Yes
Sketches	Are quick drawings of software user interfaces.	User Interface Modeling	None	No	Yes
Software Screen-play Story-board Method (S3M)	Creates non-technical software models that combine structured narrations with illustrations to define and demonstrate functional requirements.	Usage Modeling, User Interface Modeling	None	Yes	Yes
State Machine Diagram	Illustrates how an object responds to various events depending on its current state.	Dynamic Object Modeling	Object-oriented concepts	No	Yes
Story-board	Provides snapshots, with corresponding annotations, of the user interface at different points in an interaction.	User Interface Modeling	None	Yes	Yes
Timing Diagram	Shows the change in state or value of one or more objects throughout a given period of time.	Dynamic Object Modeling	Object-oriented concepts	No	Yes
Use Case Diagram	Used to identify the roles and discrete functionalities in	Usage Modeling	Object-oriented concepts	No	Yes

	a system.				
User Interface Flow Diagram	Models high-level relationships and interactions between major user interface elements of a system.	User Interface Modeling	None	No	Yes

VI. CONCLUSIONS AND RECOMMENDATIONS

6.1 Conclusions

The suitability of software models depends on users' need. If the case is the need for a non-technical, yet comprehensive and precise software model, UML models and software prototypes may not entirely satisfy users. Hence, S3M is designed to fill in this missing link and bridge the communication gap between software buyers and developers. It is not intended to replace other software models, which may be optionally employed to reflect other aspects, technicalities, or views of a system.

S3M is built with some of the strengths of UML and software prototypes, and avoids some of the weaknesses of the two. Some of the strengths of UML that S3M retains are the ability to describe almost any type of software, ability to be hardware, operating system, programming language, middleware, methodology and network-independent, and the ability to allow introductions of concepts that are unavailable. Some of the strengths of software prototypes that S3M retains are the ability to allow users to provide feedback for early ideas of the completed software program or portions of it, ability to be cheap and quickly developed, and the ability to focus on users and their usage of the system. However, S3M avoids the use of graphical notations, dependency of external concepts, enormity, ongoing changes, complexity, and the close relation to source code that can be found in either UML or software prototypes.

S3M discards the huge investments that buyers require to learn UML before they can communicate with developers, and produces *structured* narrations and illustrations that software prototypes lack. Although some models can be vigorously used, exceeding its original intention to portray in-depth details of a software system, S3M is designed from the start to be a comprehensive solution by serving at least two modeling categories. For instance, S3M function reports are very similar to Usage Scenario models of the Usage Modeling category. They both delineate events and accompanying steps users take to achieve those events. Moreover, S3M screen and popup reports allow S3M to be clearly categorized under the User Interface Modeling category.

6.2 Recommendations

The non-technical nature of S3M discloses many possibilities. S3M users, including software buyers and developers, can now focus on functional requirements and technical requirements separately. Background in software development may be helpful to S3M users, but is no longer a necessity. Buyers are now empowered to be able to create their own S3M blueprints, or check and edit developers' blueprints. As a consequence, buyers need to understand their increased share of responsibilities in verifying the blueprints' accuracy that arrives with the empowerment.

S3M also allows software buyers and developers to evaluate and select one another. If a developer's S3M blueprint does not meet buyer standards or the developer is incapable of implementing a buyer's S3M blueprint, the buyer may approach other developers without losing the know-how since S3M is technology-independent and does not generate source codes. On the other hand, a buyer's S3M blueprint allows developers the opportunity to assess their capabilities against the software project. Developers may then choose to reject, accept, or accept and outsource the project. Developers may also want to reject buyers who are highly uncertain about their requirements to avoid the risks of excessive or ongoing requirements change. All in all, buyers can now save costs by avoiding incompetent developers, as developers save costs by avoiding overdue fines.

The comprehensiveness and precision of S3M blueprints may allow them to be included in contracts. Although S3M blueprints are good representations of the final product, they may be further enhanced by studying remaining software models, and introduce their strengths into S3M as well as avoid their faults. The end product may then be tested with a complete software system, or customized for various industries.

BIBLIOGRAPHY

References

1. Aaby, Anthony. 2000. Requirements Engineering. Walla Walla College. http://cs.wwc.edu/~aabyan/435/Requirements.html (accessed January 12, 2007).

2. Alice. 2001. Screenplay. Film Education. http://www.filmeducation.org/secondary/StudyGuides/screenplay.pdf (accessed July 28, 2006)

3. Alice. 2001. The Storyboard. Film Education. http://www.filmeducation.org/secondary/StudyGuides/storyboard.pdf (accessed July 28, 2006)

4. Ambler, Scott W. 2007. Development Phases Examined: Why Requirements, Analysis, and Design No Longer Make Sense. Ambysoft Inc. http://www.agilemodeling.com/essays/phasesExamined.htm (accessed March 9, 2007).

5. Ambler, Scott W. 2006. Essential (Low Fidelity) User Interface Prototypes. Ambysoft Inc. http://www.agilemodeling.com/artifacts/essentialUI.htm (accessed September 27, 2006).

6. Ambler, Scott W. 2006. Technical ("Non-Functional") Requirements. Ambysoft Inc. http://www.agilemodeling.com/artifacts/technicalRequirement.htm (accessed December 25, 2006).

7. Ambler, Scott W. 2006. UML 2 Activity Diagrams. Ambysoft Inc. http://www.agilemodeling.com/artifacts/activityDiagram.htm (accessed August 30, 2006).

8. Ambler, Scott W. 2006. UML 2 Composite Structure Diagrams. Ambysoft Inc. http://www.agilemodeling.com/artifacts/compositeStructureDiagram.htm (accessed March 14, 2007).

9. Ambler, Scott W. 2006. UML 2 Deployment Diagramming Guidelines. Ambysoft Inc. http://www.agilemodeling.com/style/deploymentDiagram.htm (accessed August 30, 2006).

10. Ambler, Scott W. 2006. UML 2 Interaction Overview Diagrams. Ambysoft Inc. http://www.agilemodeling.com/artifacts/interactionOverviewDiagram.htm (accessed September 6, 2006).

11. Ambler, Scott W. 2006. UML 2 Package Diagramming Guidelines. Ambysoft Inc. http://www.agilemodeling.com/style/packageDiagram.htm (accessed August 30, 2006).

12. Ambler, Scott W. 2006. UML 2 State Machine Diagramming Guidelines. Ambysoft Inc. http://www.agilemodeling.com/style/stateChartDiagram.htm (accessed September 1, 2006).

13. Ambler, Scott W. 2006. UML 2 State Machine Diagrams. Ambysoft Inc. http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm (accessed September 1, 2006).

14. Ambler, Scott W. 2006. UML 2 Timing Diagrams. Ambysoft Inc. http://www.agilemodeling.com/artifacts/timingDiagram.htm (accessed September 5, 2006).

15. Ambler, Scott W. 2006. User Interface Flow Diagrams (Storyboards). Ambysoft Inc. http://www.agilemodeling.com/artifacts/uiFlowDiagram.htm (accessed September 27, 2006).

16. Booch, Grady. 2004. The Fever is Real. Association for Computing Machinery, Inc. http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=131&page=1 (accessed September 12, 2006).

17. Borland. 2006. Borland Addresses the Leading Cause of Software Project Failure with New Requirements Definition and Management Solution. Borland Software Corporation. http://www.borland.com/us/company/news/press_releases/2006/04_17_06_borland_addresses_the_leading_cause.html (accessed December 22, 2006).

18. Borland. 2006. Borland QA Professionals Survey Reiterates the Impact of Requirements on Software Quality. Borland Software Corporation. http://www.borland.com/us/company/news/press_releases/2006/10_31_06_borland_qa_professionals_survey.html (accessed December 22, 2006).

19. Borland. 2006. Software Requirements Management Processes. Borland Software Corporation. http://www.borland.com/us/company/newsletter/issue3/strategies_more_effective_requirements.html (accessed December 22, 2006).

20. Botting, Richard J. 2006. Changes in the Unified Modeling Language. California State University. http://www.csci.csusb.edu/dick/papers/20050502Outline.html (accessed August 30, 2006).

21. Botting, Richard J. 2006. Directory. California State University. http://www.csci.csusb.edu/dick/papers/ (accessed September 12, 2006).

22. Carr, Mahil and June Verner. 1997. Prototyping and Software Development Approaches. City University of Hong Kong. http://www.is.cityu.edu.hk/Research/WorkingPapers/paper/9704.pdf (accessed October 20, 2006).

23. Chitnis, Mandar, Pravin Tiwari, and Lakshmi Ananthamurthy. 2006. Creating Use Case Diagrams. Jupitermedia Corporation. http://www.developer.com/design/article.php/2109801 (accessed August 30, 2006).

24. Chitnis, Mandar, Pravin Tiwari, and Lakshmi Ananthamurthy. 2006. Deployment Diagram in UML. Jupitermedia Corporation. http://www.developer.com/design/article.php/3291941 (accessed August 30, 2006).

25. Constantine, Larry L. 2003. Canonical Abstract Prototypes for Abstract Visual and Interaction Design. Constantine & Lockwood, Ltd. http://foruse.com/articles/abstract.pdf (accessed September 27, 2006).

26. Craig, Bill. 2005. Slug Lines (Master Scene Lines). Screenwriting Help. http://www.screenwritinghelp.com/z-sluglines.html (accessed July 28, 2006).

27. Craig, Bill. 2005. The Dreaded Screenplay Format. Screenwriting Help. http://www.screenwritinghelp.com/z-dreadedformat.html (accessed July 28, 2006).

28. Dalbey, John. 2002. User Interface Prototype Document Format. California Polytechnic State University. http://www.csc.calpoly.edu/~jdalbey/205/Deliver/prototype.html (accessed September 27, 2006).

29. Hoffer, Jeffrey A., Joey F. George, and Joseph S. Valacich. 2002. *Modern Systems Analysis & Design*. 3rd ed. New Jersey: Prentice-Hall International, Inc.

30. Lu, Guang. 1998. Prototyping for Design and Evaluation. University of Calgary. http://pages.cpsc.ucalgary.ca/~saul/681/1998/prototyping/survey.html (accessed October 12, 2006).

31. Object Management Group. 2005. Unified Modeling Language: Superstructure. Object Management Group, Inc. http://www.omg.org/docs/formal/05-07-04.pdf (accessed August 3, 2006).

32. Object Management Group. 2006. Catalog of OMG Modeling and Metadata Specifications. Object Management Group, Inc. http://www.omg.org/technology/documents/modeling_spec_catalog.htm (accessed September 7, 2006).

33. Object Management Group. 2006. Introduction to OMG's Unified Modeling Language™ (UML®). Object Management Group, Inc. http://www.omg.org/gettingstarted/what_is_uml.htm (accessed July 31, 2006).

34. Object Management Group. 2006. UML® Resource Page. Object Management Group, Inc. http://www.uml.org/#UML2.0 (accessed September 7, 2006).

35. Rumbaugh, James, Ivar Jacobson, and Grady Booch. 1999. *The Unified Modeling Language Reference Manual.* Massachusetts: Addison Wesley Longman.

36. Siegel, Jon. 2005. Getting Specifications and Products. Object Management Group, Inc. http://www.omg.org/gettingstarted/specsandprods.htm#HardToRead (accessed September 7, 2006)

37. Sparx Systems. 2006. UML 2.0 Tutorial. Sparx Systems Pty Ltd. http://sparxsystems.com.au/resources/uml2_tutorial/ (accessed August 30, 2006).

38. Sparx Systems. 2006. UML 2 Activity Diagram. Sparx Systems Pty Ltd. http://sparxsystems.com.au/resources/uml2_tutorial/uml2_activitydiagram.html (accessed August 30, 2006).

39. Sparx Systems. 2006. UML 2 Communication Diagram. Sparx Systems Pty Ltd. http://sparxsystems.com.au/resources/uml2_tutorial/uml2_communicationdiagram.html (accessed September 5, 2006).

40. Sparx Systems. 2006. UML 2 Interaction Overview Diagram. Sparx Systems Pty Ltd. http://sparxsystems.com.au/resources/uml2_tutorial/uml2_interactionoverviewdiagram.html (accessed September 6, 2006).

41. Sparx Systems. 2006. UML 2 Package Diagram. Sparx Systems Pty Ltd. http://www.sparxsystems.com.au/resources/uml2_tutorial/uml2_packagediagram.html (accessed August 30, 2006).

42. Sparx Systems. 2006. UML 2 Sequence Diagram. Sparx Systems Pty Ltd. http://sparxsystems.com.au/resources/uml2_tutorial/uml2_sequencediagram.html (accessed September 5, 2006).

43. Sparx Systems. 2006. UML 2 State Machine Diagram. Sparx Systems Pty Ltd. http://sparxsystems.com.au/resources/uml2_tutorial/uml2_statediagram.html (accessed September 1, 2006).

44. Sparx Systems. 2006. UML 2 Timing Diagram. Sparx Systems Pty Ltd. http://sparxsystems.com.au/resources/uml2_tutorial/uml2_timingdiagram.html (accessed September 5, 2006).

45. The Associated Press. 2004. Newest Electronics Short on Simplicity. Cable News Network LP, LLLP. http://edition.cnn.com/2004/TECH/ptech/01/30/unfriendlier.electronics.ap/ (accessed September 15, 2006).

Image References

1. Alice. *'P' is for Psycho*, 2001. Film Education, London. http://www.filmeducation.org/secondary/StudyGuides/storyboard.pdf (accessed July 28, 2006)

2. Ambler, Scott W. *componentDiagramUML1.jpg*, 2006. Ambysoft Inc., Ontario. http://www.agilemodeling.com/artifacts/componentDiagram.htm (accessed August 25, 2006).

3. Ambler, Scott W. *componentDiagramUML2.jpg*, 2006. Ambysoft Inc., Ontario. http://www.agilemodeling.com/artifacts/componentDiagram.htm (accessed August 25, 2006).

4. Ambler, Scott W. *deploymentDiagram.jpg*, 2006. Ambysoft Inc., Ontario. http://www.agilemodeling.com/artifacts/deploymentDiagram.htm (accessed August 30, 2006).

5. Ambler, Scott W. *modelingOverview.jpg*, 2005. Ambysoft Inc., Ontario. http://www.agilemodeling.com/essays/phasesExamined.htm (accessed March 9, 2007).

6. Ambler, Scott W. *uiEssential.jpg*, 2006. Ambysoft Inc., Ontario. http://www.agilemodeling.com/artifacts/essentialUI.htm (accessed September 27, 2006).

7. Ambler, Scott W. *uiFlow.jpg*, 2006. Ambysoft Inc., Ontario. http://www.agilemodeling.com/artifacts/uiFlowDiagram.htm (accessed October 11, 2006).

8. Ambler, Scott W. *uiSketches.jpg*, 2006. Ambysoft Inc., Ontario. http://www.agilemodeling.com/artifacts/uiPrototype.htm (accessed September 27, 2006).

9. Bjork, Russell C. *Packages.gif*, 2001. Gordon College, Massachusetts. http://www.math-cs.gordon.edu/courses/cs211/ATMExample/Package.html (accessed August 29, 2006).

10. Constantine, Larry L. *Example of a Canonical Abstract Prototype show examples of key notational elements*, 2003. Rowley, Massachusetts. http://foruse.com/articles/abstract.pdf (accessed September 27, 2006).

11. Froese, Thomas, Martin Fischer, Francois Grobler, John Ritzenthaler, Kevin Yu, Stuart Sutherland, Sheryl Staub, Burcu Akinci, Ragip Akbas, Bonsang Koo, Alex Barron, and John Kunz. *paper03.gif*, 1999. Kruisplein, Rotterdam. http://www.itcon.org/1999/2/paper.htm (accessed August 25, 2006).

12. García, José Daniel, Jesús Carretero, José María Pérez, Félix García, and Rosa Filgueira. *figure3.gif*, 2005. Journal of Object Technology (JOT), Zurich. http://www.jot.fm/issues/issue_2005_11/article5 (accessed September 6, 2006).

13. Hoffmann, Hans-Peter. *0511Hoffman_fig6.jpg*, 2005. Ogden Air Logistics Center, Hill AFB. http://www.stsc.hill.af.mil/crosstalk/2005/11/0511Hoffman.html (accessed August 30, 2006).

14. Lu, Guang. *low.jpg*, 1998. University of Calgary, Alberta. http://pages.cpsc.ucalgary.ca/~saul/681/1998/prototyping/survey.html (accessed October 12, 2006).

15. Lu, Guang. *pictive1.jpg*, 1998. University of Calgary, Alberta. http://pages.cpsc.ucalgary.ca/~saul/681/1998/prototyping/survey.html (accessed October 12, 2006).

16. Lu, Guang. *pictive2.jpg*, 1998. University of Calgary, Alberta. http://pages.cpsc.ucalgary.ca/~saul/681/1998/prototyping/survey.html (accessed October 12, 2006).

17. Lu, Guang. *story.jpg*, 1998. University of Calgary, Alberta. http://pages.cpsc.ucalgary.ca/~saul/681/1998/prototyping/survey.html (accessed October 12, 2006).

18. Object Management Group. *The taxonomy of structure and behavior diagram*, 2005. Needham, Massachusetts. http://www.omg.org/docs/formal/05-07-04.pdf (accessed August 3, 2006).

19. Penchikala, Srini. *ClassDiagram.gif*, 2003. North Sebastopol, California. http://www.onjava.com/onjava/2003/12/23/graphics/ClassDiagram.gif (accessed August 25, 2006).

20. Rherrera. *SMT_USE_CASE.png*, 2005. International Crop Information System (ICIS). http://cropwiki.irri.org/icis/images/7/73/SMT_USE_CASE.PNG (accessed August 30, 2006).

21. Sparx Systems. *com01.gif*, 2006. Sparx Systems, Victoria. http://sparxsystems.com.au/resources/uml2_tutorial/uml2_communicationdiagram.html (accessed September 5, 2006).

22. Sparx Systems. *com02.gif*, 2006. Sparx Systems, Victoria. http://sparxsystems.com.au/resources/uml2_tutorial/uml2_communicationdiagram.html (accessed September 5, 2006).

23. Sparx Systems. *td03.gif*, 2006. Sparx Systems, Victoria. http://sparxsystems.com.au/resources/uml2_tutorial/uml2_timingdiagram.html (accessed September 5, 2006).

24. SysML Partners. *vpict-28.jpg*, 2004. Consultative Committee for Space Data Systems (CCSDS). http://www.ccsds.org/docu/dscgi/ds.py/GetRepr/File-1514/html/vpict-28.jpg (accessed August 29, 2006).

25. Turner, Demian. *seagull_uml_sequence_diagram.png*, 2006. Seagullproject.org. http://seagullfiles.phpkitchen.com/images/seagull_uml_sequence_diagram.png (accessed September 5, 2006).

26. Williams, Paul. *vt500_parser.png*, 2005. West Sussex, England. http://vt100.net/emu/vt500_parser.png (accessed September 1, 2006).

27. Wright, Nikki. *collaboration.png*, 2005. University of Illinois at Urbana-Champaign. http://ilabs.inquiry.uiuc.edu/ilab/wqdl/documents/889/home/uml+design (accessed September 5, 2006).