Course Outline
CS371S
Object-oriented Software Development
(*Actually – Model Driven Software Development*)

Instructor - J.C. Browne
Fall 2005

Course Approach and Goal

This course will introduce a model of software system development where an executable program is derived directly from an executable specification called an analysis model. This development process for which the name "Model Driven Development" or "Model Driven Architecture" has recently emerged, is a major innovation in software development. The software system is developed in an executable design language, xUML, and the code in C or C++ or Java is compiled from the executable design. No "code" is written except for a reusable software architecture.

Model driven development moves software development away from programming into design and creation of intellectual property. The instructor has had numerous students who have taken the course in the past report that having mastered this material was a significant plus in applications for positions in information technology organizations. Model driven development transforms software development from a commodity skill to a professional skill.

The steps in the model driven development cycle are:
a) The system is defined as an executable specification which is an object-oriented model. The executable specification is written in an executable version of UML.
b) The program is validated and verified at the model level.
c) A software and execution architecture is defined as a set of class templates in an object-oriented programming system.
d) The executable system is realized by compilation of the validated analysis model to the software execution architecture.

This method of software development is now being used for high-reliability long-lived systems by leading embedded systems vendors such as Motorola, Xerox and Kodak, Ford and others in the automotive industry.

Web Background Material

The philosophy of model driven development can be found on this web site.

http://www-106.ibm.com/developerworks/rational/library/3100.html

The two textbooks available can be found at:

http://www.executableumlbook.com/

and

http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=0521537711

Course Materials

The course materials include a textbook and some supplementary materials including papers and software system manuals.

The texts are "Executable UML" A Foundation for Model-Driven Architecture" by S.J. Mellor and M.J. Balcer and "Model Driven Architecture with Executable UML" by C. Raistrick, et.al.

The lecture notes will be distributed for the first few weeks and will then be available over the web and in hard copy through a distribution service.

Work Statement

This is essentially a laboratory class. The lectures will cover the xUML and the executable specification based development method in detail and other methods as alternatives. The main goal of the course will be to carry through a complete development of a small software system using object-oriented development methods. There will be one class examinations but no final examination. The examinations is open-book and open-notes. The course grade will be two thirds on the project and one third on the examination. Use will be made of commercial software tools which are used in industry.

Project Specifications

Project Structure - The project will be development of a small software system through the executable specification development methodology. The projects will be executed by small teams of co-workers. I have a set of possible projects. Each team will do a different project. A team can suggest a project of their own definition by preparing a requirements specification and getting it approved. The scale and scope should be similar to the requirements I will circulate.

Communication between Students and Instructor

The instructor is email oriented. He answers email almost every morning. This is by far the best way to communicate. Formulate your questions in writing and send them to browne@cs. Announcements concerning the class will be sent by email. Read your mail every day.

Standards for Conduct

Standard University of Texas rules for conduct of classes will be followed. Please make yourself familiar with those rules.

Lecture Schedule
CS371S –Fall 2005

There follows an approximate schedule of lectures for the fall semester. There may be minor variations if some topics take more time than anticipated and/or some additional guest lecturers of interest become available.

Lecture Date	Lecture Topic	Reference Material
9/1/05	Model Driven Development – What, Why and How	Mellor-Chapter 1 Raistrick – Chapters 1 and 2
9/6/05	Introduction to Executable UML	Mellor – Chapter 2 Lecture Notes and Web Materials
9/8/05	Design Principles and Process	Mellor – Chapter 3 Raistrick – Chapter 3, 5 Lecture Notes
9/13/05	Project Specifications and Responsibilities	Lecture Notes
9/15/05	Requirements Specifications – Use Cases	Mellor – Chapter 4 Raistrick – Chapter 4
9/22/05	Domains and the Modeling of Classes	Mellor – Chapter 5 Raistrick – Chapter 6
9/24/05	Modeling of Relationships and Interactions	Mellor – Chapter 6 Raistrick – Chapter 7
9/29/05	Modeling of Control Flow and Data Flow	Mellor - Chapter 10 Raistrick – Chapters 8 and 9
10/4/05	State Machines and Dynamic Behavior	Mellor – Chapters 9, 10 and 11 Raistrick – Chapters 9 and 10
10/6/05	Project Presentations	
10/11/05	Project Presentations	
10/13/05	Case Studies and Review or Project Presentations	
10/18/05	Class Examination	
10/20/05	Action Language	Mellor – Chapter 7 Raistrick – Chapter 10
10/25/05	Action Language	
10/27/05	Constraints and Behavioral Specifications	Mellor – Chapter 8
11/1/05	Verification and Validation	Mellor – Chapter 15
11/3/05	Verification and Validation	Lecture Notes
11/8/05	Multiple Domains and Bridges	Mellor – Chapters 16 and 17 Rasitrick – Chapter 12
11/10/05	Code Generation	Mellor – Chapter 18

		Raistrick – Chapter 13 Lecture Notes
11/15/05	Code Generation	
11/17/05	Project Presentations	
11/22/05	Project Presentations	
11/29/05	Project Presentations	
12/1/05	Project Presentations	
12/6/05	Case Study – Guest Lecture	
12/8/05	Case Study – Guest Lecture	

Appendix A – Analysis of Software Development Methods

Why Software Development by the Conventional Process is Difficult

Development of complex software systems has always been a challenging task. (We assume the reader is familiar with the conventional software development process based on manual translation of application designs to implementations in conventional procedural programming languages such a C or C++. The steady increase in functional complexity required for competitive capabilities in software products is compounded by implementation of these systems on distributed and networked systems. But the root causes of the problems of developing software by the convention process of manually programming application operations in procedural programming languages are:

(a) The wide conceptual gap between the operations defined in typical application domains and the operations defined in conventional programming languages. The results of this gap are: high complexity for the manual translations from application concepts to programming language concepts and high complexity and low readability of the programs which result. This conceptual gap also impedes validation.

(b) In the conventional development process end-user operations of the application are not validated until the programs in procedural programs are have been completed. When complex application operations are realized through complex transformations to complex programs in procedural programming languages, errors of translation are inevitable and execution behaviors become unpredictable. The level of complexity of the program in the procedural language precludes detailed understanding of the entire system while at the same time the system must be validated in terms of application level operations. Additionally there is no provision for validation of the increasing important requirements for performance.

(c) Each software development project can make little use of artifacts from past development projects except at the lowest level of data structure library routines.

(d) Modifications of functional requirements expressed in application concepts must be done by modification of the procedural program in a different conceptual framework and at a much higher level of complexity.

(e) Modifications in execution environment lead to complex ports of the procedural program.

It is often difficult to estimate the effort necessary to realize products when the conventional process is used. The products often contain many defects due the difficulty of validation of the procedural program representations. The costs of modifications are high and error prone because they must be done on the procedural program.

It is apparent that a qualitative improvement in software development must automate translations from application specifications to procedural languages and that validation must be done in terms of application concepts.

Foundations of a New Paradigm

The preceding section makes it clear that qualitative improvement in software development process cannot be expected to arise in evolutionary enhancement based on the conventional process. And the analysis given in Section b identifies the steps in the development process which must be replaced.

(a) Manual translation from application operations to procedural programs must be automated.

(b) The application system must be validated in the conceptual basis of the application. Validation must include conformance to performance specifications as well as functional specifications.

The innovations which enable a process which removes these fundamental barriers are:

Separation of Concerns – Specifications for the application operations are done separately from specification of the execution environment.

Executable Specifications – The operations of the application are defined in a specification language with an executable semantics in the application conceptual domain.

Software Architectures – A software architecture is a specification of a set of operation and data structure templates to which the operations of an application can be translated. A software architecture is a virtual machine to which application level operations can readily be compiled. Execution environments for the applications are defined as software architectures in a standard procedural programming language.

Associative Objects – Associative objects are conventional objects extended and encapsulated to support automation of reuse and composition of systems from components. Associative interfaces replace and extend the concept of relationships in conventional object models.

The steps in the development process based upon these concepts are as follows:

(a) Requirements are captured in a traceable form in a database.
(b) An executable representation of the application system is captured in application concepts as an analysis level associative object model in which the constructs of the analysis model all have an operational semantics.
(c) A software architecture is selected or constructed.
(d) A simulation model of the execution environment is selected or constructed.
(e) The executable specification of the application is validated by execution over a workload (test suite) defined in terms of invocations of "end user" operations of the application.
(f) The functionally validated executable specification of the application is used as a workload generator for the simulation model of the execution environment to validate the performance behavior of the application operations.
(g) The fully validated executable specification of the application is translated to the software architecture by a compiler.
(h) The resulting program is compiled and linked by the standard compilers for the procedural languages to realize the production system.

This development process leads to a qualitative improvement over conventional development processes because it avoids the sources of difficulty listed in above. The characteristics of the new development process are:

(a) The application is validated in application concepts and with application based test cases. The immense complexity of validation of the procedural program is avoided.
(b) Modifications driven by changes in functional requirements are made to the executable representation of the application. Error-prone modifications of procedural program representations are avoided.
(c) Modifications driven by changes in execution environment are made to the software architecture and the simulation model of the execution environment. Production systems for new platforms are accomplished by compiling the application specification to the new software architecture. Error-prone ports of procedural programs to new platforms are avoided.
(d) Each software development task is accomplished by an appropriate expert. Application experts construct the executable specification of the application. Software designers develop the software architectures.
(e) Reuse of application level objects is enabled by defining the application objects as associative objects.
(f) The software architecture can often used for multiple projects with little or no change, particularly in the common case of a family of software products with similar characteristics.

Appendix B

Example Project for CS371S – Fall 2005

Reservation System for Avis Rent a Car

Avis has a set of offices in a set of cities. Each city/office pair has a set of cars available for rental. Each car is belongs to a rental class. A rental class has a model name, a manufacturer and a base daily rental price. Each car belongs to some rental location but may at any given time be in some other location.

Avis is putting in a web-based reservation system. Customers request a reservation for a specific class of car in some city at some location for some span of dates and specifying a location to which to return the car. Many customers may be simultaneously attempting to make reservations. Requests are accepted in FIFO order. A customer either gets the class of car desired or a notification that a car in the rental class he/she requested is not available. The customer will sometimes then request another class of car. If the customer accepts the available rental car then he/she then presents a credit card against which the rental charges will be made. The system obtains an authorization from the credit card company for the expected charge. The rental is denied if the credit card company refuses to authorize the charge.

If a customer has requested to return the rental car to a location different from the renting location then the system will choose a car belonging to the return site if one is available at the rental site.

A reservation may be cancelled before its initiating date and time. When the customer picks up the car a rental is initiated. A rental may be extended while the customer is in possession of the car. If a car has not been returned 24 hours after the scheduled date and the customer has not requested an extension the car is reported to the local Police Department of the location as being missing or stolen.

Each customer belongs to an organization. Each organization has a discount percentage for each car class. Rental rates are determined by the organization to which a customer belongs. The discounts are revised fairly often as a function of contracts and competitive pressures.

The software system you are to develop should support making and canceling of reservations in a fair and orderly manner. It should also keep track of when cars are returned and create a bill for the rental of each car and send copies of that bill to the organization and the customer. Your software should cover the cases when a car is not available and where a given class of car is not available. The software should keep track of the number of rental days for each class of cars by manufacturer to enable optimization of the mix of cars in the rental fleet. The GUI is a separate domain with which your system just exchanges messages.