Uniform object modeling methodology and reuse of real-time system using UML

Bui Minh Duc, Professor, CSSE Dept.
Laval University, Québec, Canada
duc.bui@ift.ulaval.ca
http://www.ift.ulaval.ca/`bui

Abstract
The recent release of UML 2.0 has corrected a lot of design difficulties encountered in the 1.x revisions. The biggest change which allows UML to really attack embedded systems is the implementation of true object diagram and business process diagram. For embedded systems, at analysis stage, use cases and business processes express system requirements. At design time, class diagrams store operations of generic objects and object diagrams show all instantiated objects participating in macroscopic processes. Normally, dynamic studies are supported by sequence suite, activity and state diagrams. Unfortunately, dynamic support undergoes only cosmetic changes. Based on UML version 2.0 diagrams, uniform object modeling methodology shown hereafter handles indifferently any object in the model: a user, a mechanical button or a piece of software program. This uniform abstraction is necessary to implement easily simulation and test. A design of a very simple load elevator going through two levels with security system is used here to illustrate the uniform process and serve as a basic design for discussion.

Categories and Subject Descriptors
D.2.13 **[Reusable software]** Reuse models

Keywords: Embedded real-time systems, reactive systems, UML design, object technology, modeling methodology, induced energy, object message.

General Terms: Theory

1. INTRODUCTION
Complex software development would require minimum development effort and is a fastest "time to market" design if reuse mechanism making use of available designs and components is planned carefully. The basic question is how to design "true" reusable objects. To satisfy large scale reuse of heterogeneous systems, objects must be designed uniformly across domains without ever questioning what kind of object (mechanical, electrical, software, biological…) we are using.

EMSOFT'05, September 19–22, 2005, Jersey City, New Jersey, USA.

The term "object" used in this paper does not mean only "software object" in object paradigm. Traditionally, software frontiers are traced on the I/O interfaces as they connect computer systems to real world. This frontier is somewhat artificial as all hard/soft systems can be now simulated (e.g. fly simulator). A mechanical button is replaced by a software button in a windows environment. A pressure on a mechanical button is equivalent to a click on a software button. An electrical, biological, mechanical device… can be assimilated to objects with attributes and operations/methods. Messages are unique channels of communication between objects and the way we interpret messages must be revised.

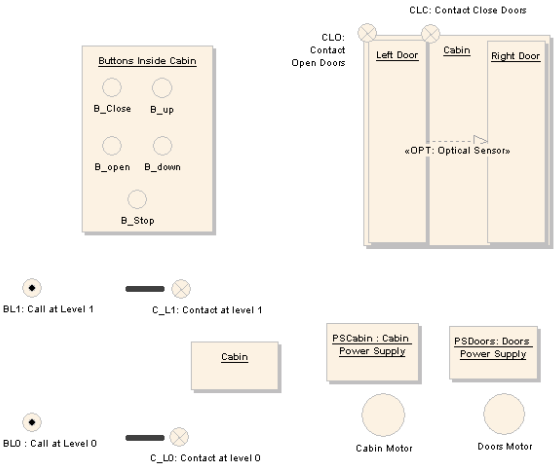
2. BACKGROUND AND CASE STUDY

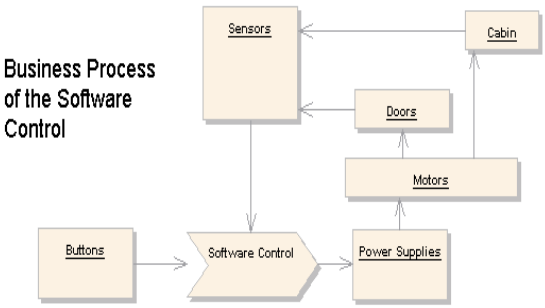
2.1 UML version 2.0

Uniform Modeling Language (UML) is an object oriented modeling language [1] standardized by Object Management Group (OMG) mainly for software system development. Its development is due to the effort of researchers and industrial partners [2] [3]. UML consists of a set of diagrams whose graphical elements represent concepts and abstractions, rules that dictate their use and common mechanisms that enhance or extend models through stereotypes. As a language, UML does not impose any methodology. Embedded specialists are interested to UML [4] and real time UML-RT [5] is a profile that extends core UML with stereotyped active objects, called capsules, to represent system components.

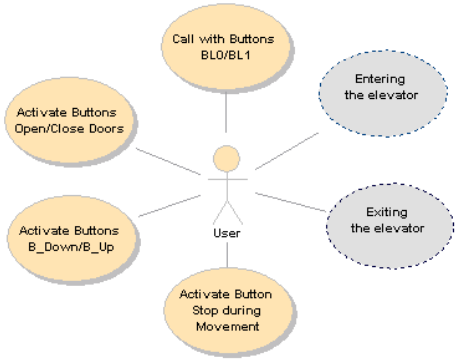
2.2 Case study

Our purpose aims to illustrate the Uniform Object Modeling Methodology of a system made of components taken from various disciplines. Buttons, contacts sensors of the load elevator are inputs, power supplies of the cabin and doors of the elevator are electrical outputs with digital commands for power on/off and direction, the optical detector is the security element. User is biological object, buttons and sensors are mechanical objects; cabin and doors power supplies are electrical objects and software are computer objects. The cabin and the doors watch events on sensor contacts to operate between two levels L0 and L1. When the cabin reaches a level, it urges the doors to open and the doors remain always opened. When the user gets into the cabin, he must act on B_Close/B_Open to close/open the doors and on B_Up/B_Down to change level. If an obstacle hinders the door during closing, the door changes to opening. During the movement of the cabin, a movement sensor inhibits all buttons except the Stop button. So doing, we have most ingredients of real time and embedded system. Hereafter, we make use of a subset of UML diagrams for the two phases: analysis and design.

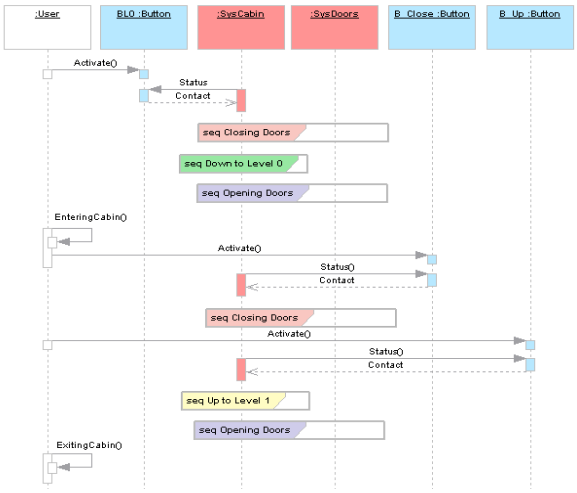




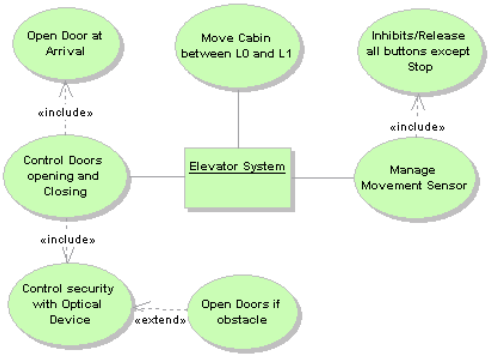
Use Cases around User



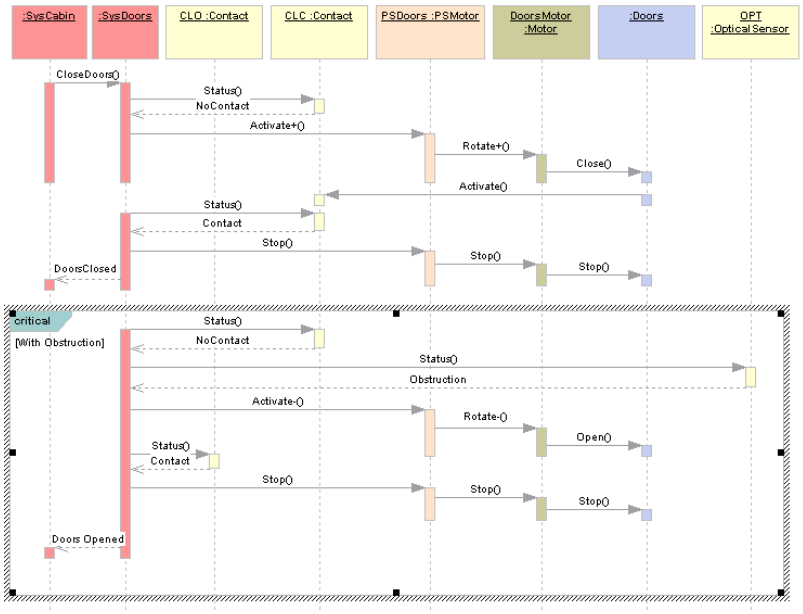
User at level 0, calling elevator at level 1, entering then going upstairs

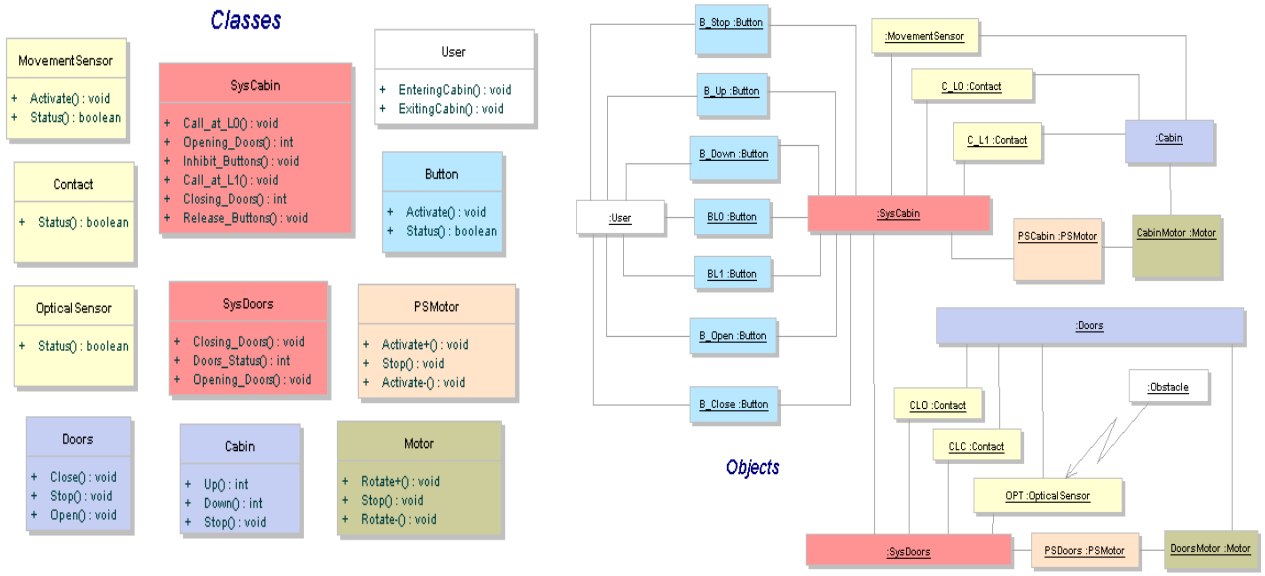


Use Cases of the system to be build



Closing Doors Sequence





3. DESIGN WITH UNIFORM OBJECT MODELING METHODOLOGY

3.1 Use cases in analysis phase

In the analysis phase, use cases (UC) diagram and business process (BP) diagram [8] are used to express functional constraints of a project and as such, they must not contain any design flavors unless they are constraints. If reusable components are required, they become constraints. These diagrams are invaluable tools to understand "what to do" before starting the "how to do" design phase. The UC diagram shown hereafter is actor-centric (actor can be human or not).

3.2 Business Process in analysis phase

BP diagram gives new life to older bubble diagrams of functional methodologies [6] and as such, there is a potential danger to start a functional study. If we avoid making early designs in the requirement analysis phase, BP diagram is a plus as it can nail down all I/O and events out of a "macroscopic" process. The difference with the functional approach relies mainly in the fact that almost everything is object now. If data are used at this phase, they must be assigned to existing objects at the design phase. In the requirement analysis, everything is not "still" object as developers work with non object specialists at the interface of the development process.

3.3 Design phase with sequence and interaction overview diagrams

The rationale of the design phase is to get a logical definition of all objects/classes. Class methods must be defined at such a granularity that people involved at the implementation phase should not have any question to ask while coding. Even private methods must be named, though not coded. As for attributes, things are hazier. Evident object attributes can be listed at this design phase for good documentation and comprehension but the full set will emerge only at the implementation phase.

As all systems are based on three principal views [7], structural, functional and dynamic, class diagrams deal mainly with structural and functional views (definition of classes and operations inside classes). Sequences are mainly dynamic but the object diagram is hybrid. Object diagram, instance of class diagram, describes all the objects involved in a particular high level process. Each object diagram relates a "story". The whole complex system has a lot of stories to tell us through high-level and mostly public operations. Each operation involves object subsets. UML 2.0 has corrected a major deficiency of its previous 1.x versions by deleting the dynamic collaboration diagram generated from a sequence diagram and authorizing a real hybrid object diagram.

Sequence diagrams are used at the beginning of a process to identify classes, objects and operations. The main interesting addition in the "interactions" suite in UML 2.0 was the Interaction Overview Diagram with graphical "fragments" to organize dynamic sequences, to represent alternatives, cases, exceptions, etc.

In a true design, we cannot establish all the sequences because the task is incommensurable. We need most representative sequences to start a study but once objects/classes are identified, we can add operations directly to classes by representing sequences "mentally" while reasoning.

Hereafter is a user at level 0 calling the cabin at level 1, entering the cabin and trying to access L1 level. It is the longest sequence, so we start with an interaction overview to identify "fragments". We detail only the "closing doors sequence" to illustrate the procedure of identifying objects, classes and operations through sequences and Interaction Overview.

3.4 Design phase with state and activity diagrams

State and activity diagrams are used to specify algorithms. As these diagrams take a long time to establish, they typically deter a lot of people. We must find some means to work around the generation of algorithms. In this sense, UML 2.0 has shortcomings as the dynamic parts only underwent cosmetic changes. In [9], some

efforts are done in the direction of Petri-like networks to address the problem of designing graphical algorithms. Due to the limited format of this paper, we cannot show them.

3.5 Design phase with Classes/Objects Diagram

Opposite to database practice, classes are presented in real-time systems without any relationship. Only heritage/derivation is shown in a class diagram. Object diagram is more informative. Linked objects can communicate signals and data.

4. DISCUSSIONS

In the object diagram, we can close the loop by expressing the fact that the cabin lifts the user and the obstacle can be also his leg. All objects in this system are modeled uniformly as software objects. They act through messages. When the user puts his finger on the button, he sends a message to the button and urges the button to make a contact. The user execute an internal operation Activate-CallButtonAtLevel0(), not represented, and the button has also an internal service Activate(). Messages in a software program convey activation signals and data through communication channels to wake up processes, messages in non computer systems can convey energy. Energy is then another form of data and is mostly "induced". Roof tiles do not have energy but the wind can induce energy to tiles so they can hurt people. The energy transmitted by the user is sufficient to allow the button to activate itself and make a contact. We can model easily the domino effect.

If we model a message from the button to SysCabin, we think of interruption. We can do so but, at the implementation phase, programmers will poll the button as this process needs less hardware. Buttons are able to give its status by transforming their mechanical contacts to signal when the button is powered on. At this time, energy is induced by the power supply. With this reasoning, we are able to uniformly consider any mechanical, physical, biological objects with attributes and methods.

Uniform object modeling methodology and reuse are very sensitive to naming. Never name the method of the motor as "GoUp" "GoDown". So doing, we interpret what the motor is doing in a specific application and we destroy the reuse mechanism. In the nature, motors "Rotate+" or "Rotate-". With a mechanical coupling that we can model either, this rotational movement is transformed in a specific application to translational movements to act on doors and cabin.

Software simulation and test of real time and reactive systems can be done with this uniform modeling since every pieces of physical parts found along the reaction loop can now be modeled as objects. Properties of objects, structural, functional or dynamic find their replica in attributes while processes are materialized as operations. The model of communication in an object program is demonstrated in the past [7] as a reduced model of a more general interaction between objects; object interaction cannot be considered as simple object call in programming language but as a complex sequence of object calls/exchanges of data, controls, energies.

The impact of this uniform view has considerable consequences on the way software and models are designed with reuse in sight. In fact, reuse patterns are not limited only on software chunks but can now be designed as more generalized components which include, besides software, physical, electronic, mechanical, biological, etc. parts. In this sense, the notion of components must be revised in the future to allow more generalized reuse frameworks.

5. CONCLUSIONS

The substance of this methodology is resumed as follows: "Uniform modeling to facilitate integration" and "Molding objects as they are naturally in the nature for reuse". In reactive systems, objects must react to stimuli and act on real world. The reaction loop includes several objects of various nature and disciplines. A uniform modeling concept is needed for not to change the way we view, reason or model objects along the reactive loop. To make objects fully reusable, the key concept is to give to objects only natural properties in its "own domain" and never give them properties oriented towards any specific application.

In this paper, we approach a hypothetic system with UML 2.0 through a uniform object modeling methodology considering all objects of various natures with a same view and modeling technique. The concept of message developed in object technology can be extended to ease design of heterogeneous systems. Attributes, methods and messages are strong abstractions that govern universe systems. We suspect that this uniform methodology can profit to intelligent systems, problems in human sciences. Despite its weaknesses in dynamic modeling, UML 2.0 has overcome a lot of youth errors and turns into a more interesting tool with the arrival of the true and expressive object diagram where we can bind objects with associations freely interpreted as communication channels. Embedded systems can benefit from these progresses waiting for more formal methods to expressing algorithms to terminate the object design phase.

REFERENCES

[1] Official OMG website http://www.omg.org. The current norm of UML is 2.0. It replaces older revisions 1.x

[2] J. Rumbaugh. I. Jacobson and G. Booch. The Unified Modeling language User Guide, Addison-Wesley, 1998

[3] B. Selic, A generic Framework for modeling resources with UML, IEEE Comp. Soc., June 2000, pp 64-69

[4] G. Martin, L. Lavagno, J. Louis Guérin, Embedded UML: a merger of real-time UML. And co-design, Proceedings of CODES 2001, Copenhagen, April 2001, pp 23-28

[5] B. Selic, J. Rumbaugh, Using UML for modeling complex real-time systems, White Paper, Rational (Object Time) 1998

[6] H. Gomaa, A software design method for real-time systems, Comm. ACM, vol 27, No 9, Sept 1984, pp. 938-949

[7] Bui Minh Duc, Analyse et conception objet des systèmes temps réel, second edition, Eyrolles 1998, France, ISBN 2-212-09027-7

[8] Official SparX Systems website http://www.sparxsystems.com. All diagrams in this paper are drawn with Enterprise Architect version 4.51 implementing UML 2.0

[9] Bui Minh Duc, SEN State Event Net, proposal to enrich the arsenal of UML dynamic diagram, 2005 World Congress On Applied Computing, Las Vegas