# Elevator Software Development Process using Feature-Oriented Analysis & Modeling

B.W. Choi*, K.B. Jang*, C.H. Kim*, K.C. Kang**, S.J. Roh+, C.W. Kim+

*LG Industrial Systems, R&D Center, Core Technology Laboratory

**Pohang University of Science and Technology

+LG Industrial Systems, Building Systems Research Laboratory

*Abstract*

*Elevator systems are very complicated, hard real-time systems that have various functions and features. It makes difficult to develop new models especially on requirements change. In order to make our elevator development faster, easier and more reliable, a new software development process focused on software reuse is introduced. This process has been applied from the early phase of the software lifecycle in addition to conventional software engineering techniques. It is called Feature-Oriented Reuse Method (FORM). Using FORM, we could describe those complicated elevator systems in more understandable form, utilize reusability of requirement specifications and easily adopt new requirements. In this paper, we describe the elevator feature model based on FORM, and also briefly introduce design and implementation process and tools.*

## 1. Introduction

As well as other software, the elevator control software is also getting bigger and more complicated due to adding new functions and adopting customers' demands. In order to achieve faster and easier development for bigger, more complicated, but more robust software systems, software engineering should be properly applied. Especially when considering productivity, software reuse is the most important. Software reuse was meant by code-level reuse in its early days, but now it extends its range to the most of the entire software life cycle from the user requirements to final codes. The reuse of the products in earlier phases of the software life cycle is more effective than the reuse of the source codes since the former is independent of programming languages [2] and advantageous for improving software quality. But there is few software development process which really considers the reusability in the user requirements specification and analysis phase. Feature-Oriented Reuse Method (FORM) proposed by K.C. Kang [1] is the software development method that considers those aspects.

The elevator control system software has been developed from the assembly era so a lot of software developers are still familiar with function-oriented development style that is not quite adequate to the modern reuse scheme. Also, since a lot of decision-making logic and real-time requirements are involved, it has been very difficult to change the system according to new requirements caused by the users or the operating environment. And the elevator is basically the order-based product so the specifications of each elevator can be different from each other. That causes a necessity of a better method to apply user's requirements to the system effectively. With all these

reasons, a new software development method has been developed and applied to our new elevator control system development. The method is based on FORM method mentioned above, but is slightly modified considering the characteristics of the elevator systems and our development process. Also, we have developed CASE (Computer-Aided Software Engineering) tools for automating the process.

This paper presents the concept of FORM method in a simple form, a brief introduction to our software development process, from the requirement analysis to the final code generation, including the tools for the process. Even though the each of the entire process is important, we are concentrated on the analysis and feature modeling. So the feature model of the elevator control system is also presented. Feature-oriented analysis and modeling has been proven as a very useful and effective method to model the complicated elevator control system.

## 2. FORM (Feature-Oriented Reuse Method)

FORM is a systematic method that looks for and captures commonalities and differences of applications in a domain in terms of *features* and using the analysis results to develop domain architectures and components. The model that captures the commonalities and differences is called the *feature model* and it is used to support both engineering of reusable domain artifacts and development of applications using the domain artifacts. The use of *features* is motivated by the fact that customers and engineers often speak of product characteristics in terms of features the product has and/or delivers. They communicate requirements or functions in terms of features and, to them, features are distinctively identifiable functional abstractions that must be implemented, tested, delivered, and maintained [1].

Figure 1. FORM Process Overview.

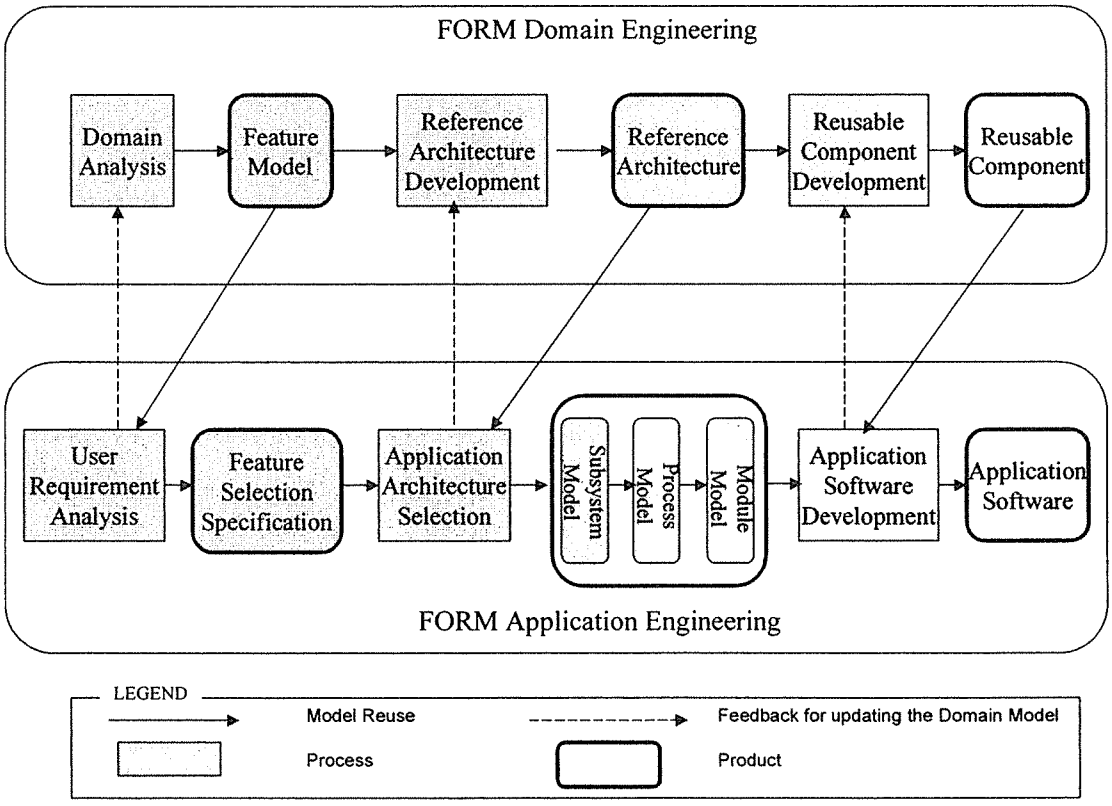
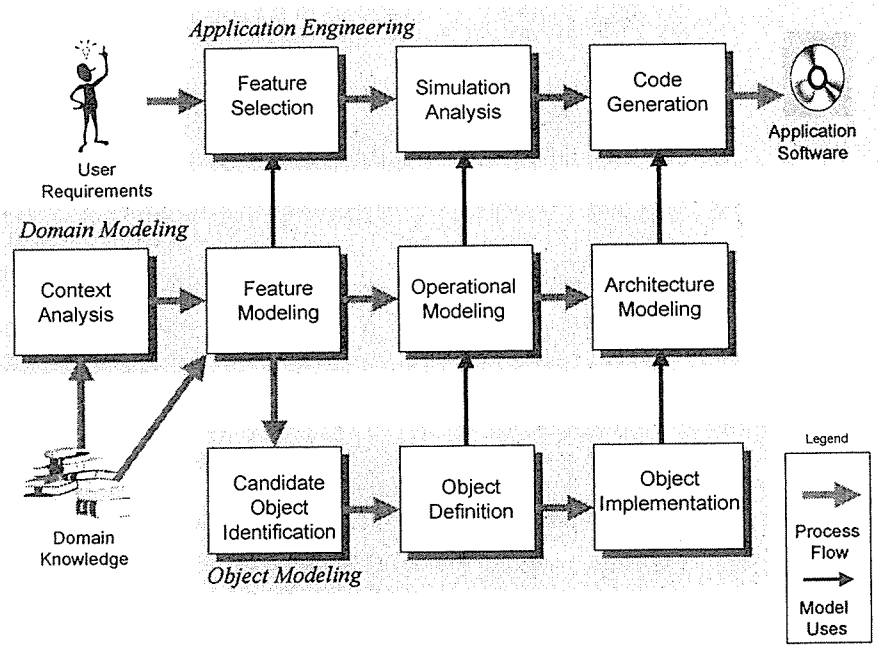


Figure 1. shows the FORM process. FORM process consists of domain engineering and application engineering flows. Through domain engineering, a feature model, a reference architecture and a reusable components are produced. After the feature model is developed, domain architecture should be made from it. The domain architecture is used as a reference architecture for creating architectures for different systems. It consists of three different models, each of which represents the architecture at a different level of abstraction. Those are subsystem model, process model, and module model [1]. The subsystem model defines the overall system structure by grouping functions. The process model represents the dynamic behavior of each subsystem. The module model describes the appropriate module structure of the system.

Domain engineering can be simply said to produce reusable resources such as features, architectures, and reusable components. On the other hand, application engineering actually makes products using the resources produced by domain engineering. Application engineering starts from the feature selection. This is done by first analyzing user's requirements for the target application, and finding a matching set of features from the feature model. Then appropriate architecture is identified from the reference architecture. Once the reference architecture is identified, reusable components are easily found by following the specifications and methods encoded in the modules. With the skeletons made through the application engineering, adding small amount of codes into the components will complete the application development. As you see, the strong point of FORM method is that FORM makes user's requirements reusable format. And selecting features leads to the final application very systematically.

## 3. Software Development Process for Elevator Control System

Figue 2. Elevator Control System Development Process.

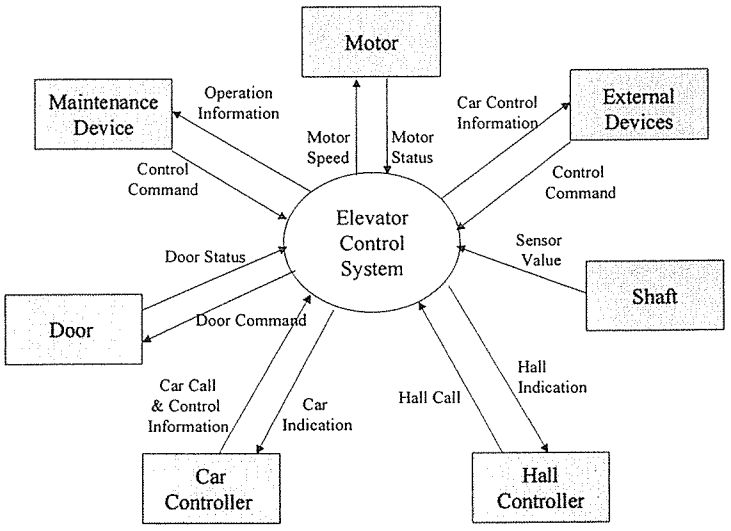



The process applied to develop elevator control system is based on FORM method, but it is slightly modified fitting into our development culture and domain characteristics. Elevator control system is a rather special than general application

because it is embedded, hard real-time system and has quite complicated decision-making logic. On the other hand, the original FORM had been applied to rather normal applications such as electronic bulletin board, so modification and extension to the FORM were necessary.

First of all, the entire development process is divided into three major flows - domain modeling, object modeling, and application engineering. Since the elevator control system has been decided to be object-oriented which is more efficient to develop complicated systems, object development process should be added. That is, reusable components mentioned in FORM are objects in this process.

Figure 3. Context Diagram.



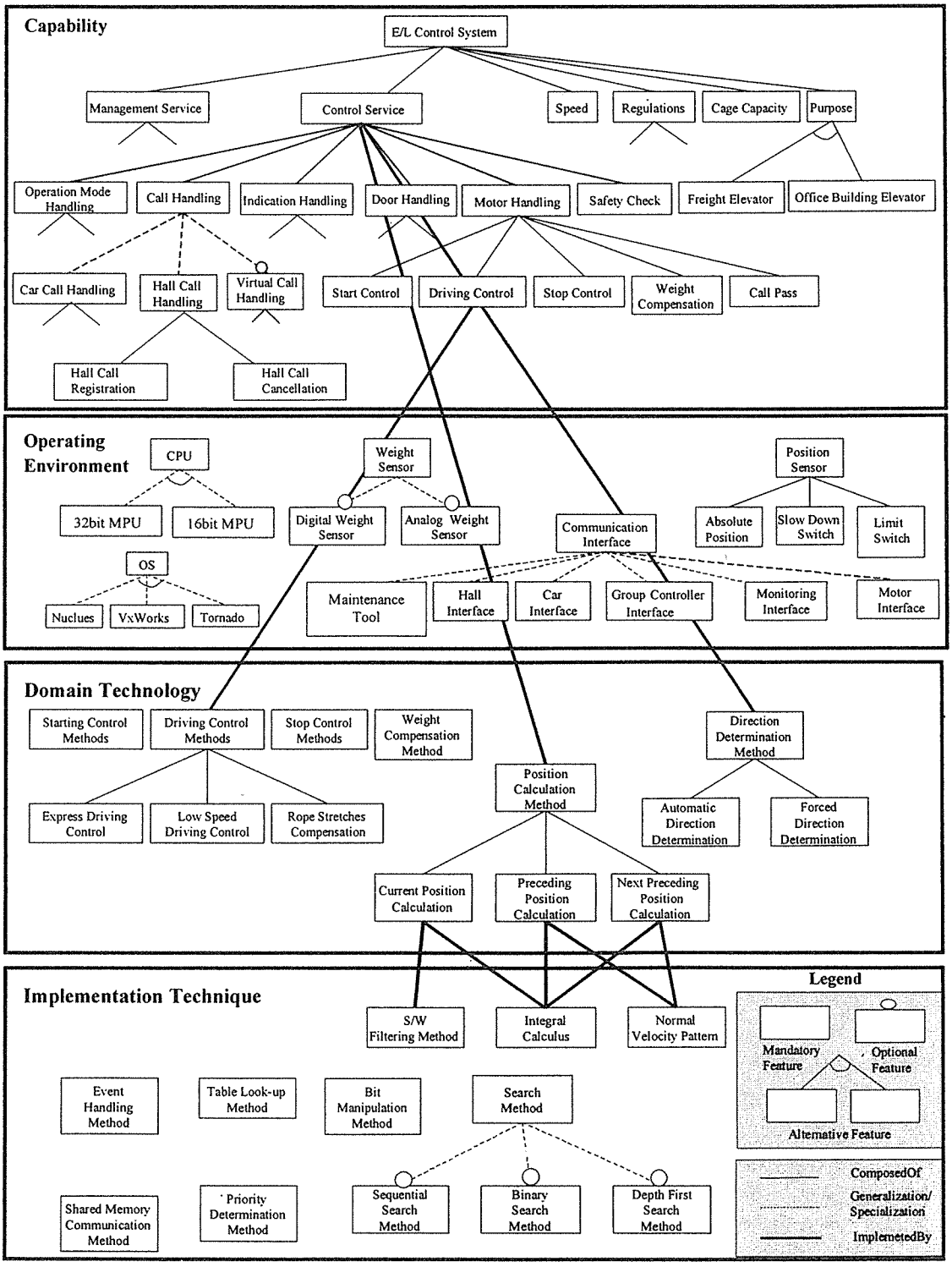

Domain modeling begins with context analysis. Context analysis specifies development requirements. That is, by defining external objects which communicate with the target system, we come to know what should be developed, what should be interfaced, and also which constraints the system has. Figure 3 shows the context diagram for elevator control system. We should get the rough picture about the operation of the system from the context diagram and it will be used as base knowledge for feature modeling and the message sequence diagram.

The next step is feature analysis and modeling. It is the same as the feature modeling in FORM described in previous chapter. Simply speaking, the developers identify the features of the analyzing domain in four categories and make relations among them. The simplified elevator control system feature model is shown in Figure 4. As we can see in Figure 4, *Capability* features are services or operations the system provides. *Operating Environment* features are the hardware and the software on which the system operates. *Domain technology* features are unique technologies that are usually used only in that domain. And *Implementation technique* features are general-purpose techniques used to develop the system.

The major contribution of feature modeling is to identify the mandatory and alternative/optional features so that we can have a system model which can be changed. *Mandatory* feature should be included in every system in the domain. If two children features are *Alternatives*, then only one of them must be included in a system. If a feature is *Optional*, then including that feature depends on user's requirements and omitting this feature does not affect the basic system functions. As we can see in figure 4, the entire structure of the elevator control system is shown in a feature model. In the feature selection phase, which is the first phase of the application engineering, the user

will select desired features among the features defined as optional or alternatives. Then the tool will validate whether the selection is valid. In addition to, feature selection apparently shows which part of the system can be re-used and which part of the system should be newly added.

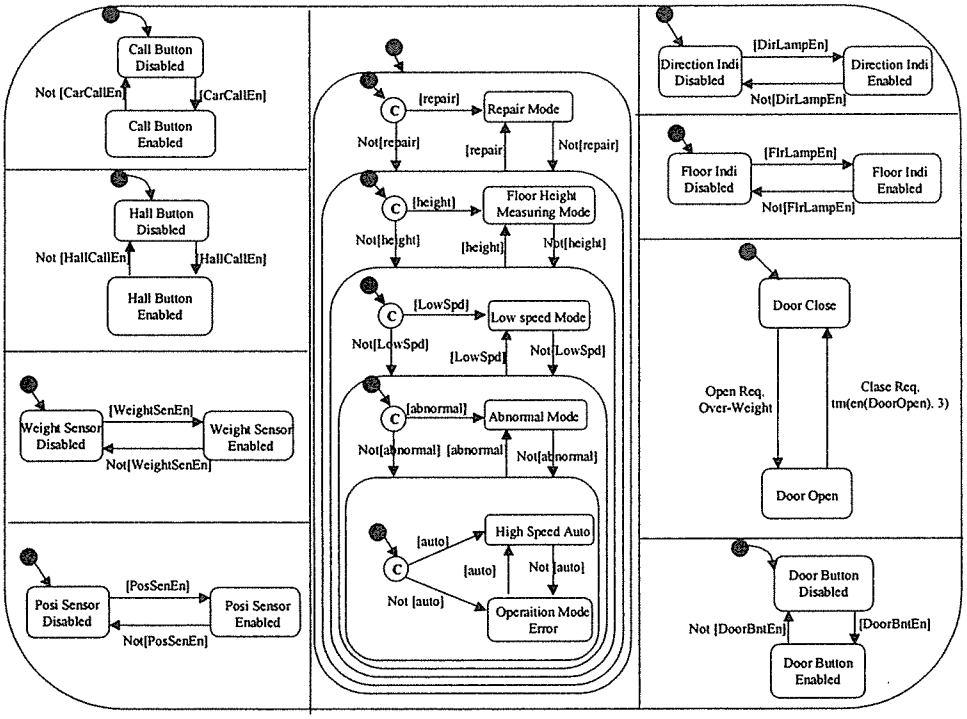
Figure 4. Elevator Control System Feature Model



The next phase is operational modeling. Since the elevator system has a lot of decision-making logic for the operation modes, the internal behaviors should be analyzed in the model. But in the feature model that is merely a static view of the system, the dynamic aspects can only be described in textual format. So we adopted message-sequence diagram, statechart, and data-flow diagram from the conventional software engineering to show those dynamic characteristics that input and output produced from the functions and how related with each other. MSD and DFD are generally well known so not presented here, but we briefly introduce statecharts since it is useful to describe dynamic behaviors. [4]

The statecharts show the behavioral aspects of the system. It can express sequential and parallel behaviors at the same time so that the simulation can be done correctly. The elevator system has many operational modes that need to check very complicated critical conditions to transform one mode to another. The complicated decision making logic has been expressed in ladder chart which is usually very hard to understand unless people has a lot of discipline. Using statecharts, mode transformation can be presented in much more understandable form as shown in figure 5

Figure 5. Statechart

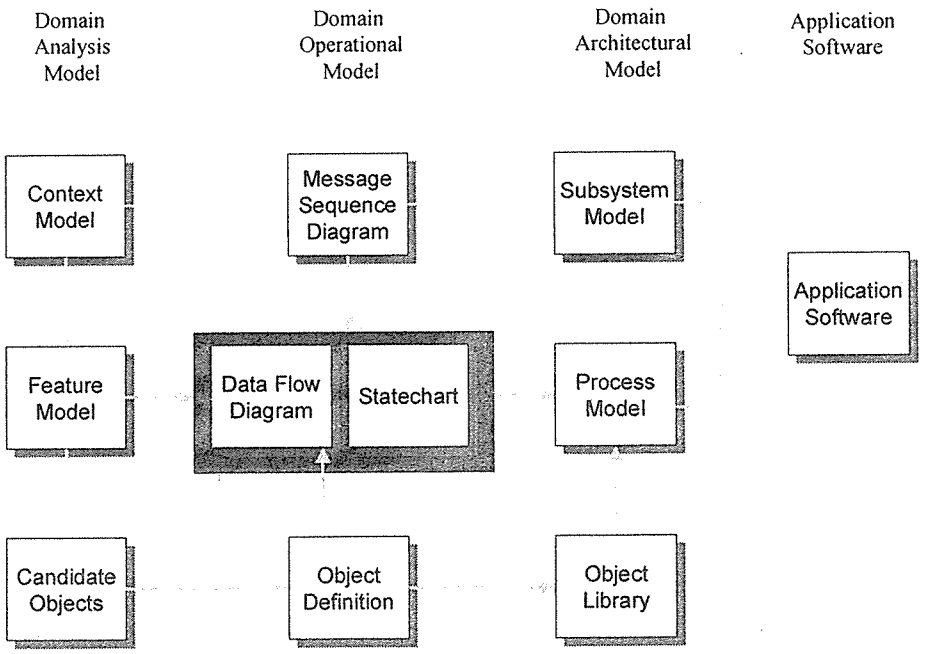



The first phase of object modeling is candidate object identification. It usually goes with operational modeling at the same time. Among the feature layers, operating environment, domain technology and implementation technique features are easy to be changed due to user's new requirements or the technology evolution. These features can generally be identified as objects. And by separating those as objects, when the change is needed for new requirements or new technology, the system maintains easily by modifying appropriate objects or just adding new objects. After the object identification is done, defining those objects is the next process. It means to define interfaces of methods that the object has. Once object definition is done, this definition is used to write functional specifications for data flow diagram in operational modeling.

Architecture modeling is the next. It is the same as the reference architecture modeling in FORM except that module modeling is replaced by object modeling process that actually started before the architecture modeling in a new process. Simply speaking, architecture modeling in domain engineering results to the physical subsystems and processes. Feature model and operational model can be called 'logical model' and then the architecture model is to practically locate that logical model into the physical model. According to the information and results from the previous steps, developers decide how to organize the system to be made. This architecture model is related to objects through the object interface, so to implement each object according to its interface should be done also. It is the object implementation process.

After completion of these two processes, a domain analysis model which includes all characteristics of various systems existed in a domain is done and it is a reusable library for producing practical application software. So we need to follow the application engineering process to produce the actual system software. It is producing a platform–dependent system automatically from the reusable library. The first step is feature selection phase. According to user's requirements, appropriate features are selected and refined. Then a model that is going to be made from the selected features is analyzed and simulated. Since the elevator system carries people, safety should be the critical factor. The simulation analysis validates the selected software system in terms of functional and behavioral aspects using visual simulation tools. If simulation is successfully finished, then the application software is produced using automatic code generator. Since we have built the reusable object library, and functional and behavioral models, the most of the codes can be generated automatically. Only the small amount of codes should be added for some details and then, we can obtain reliable software components. Figure 6 shows the meta model for the elevator software development. The overall development process can be represented in this meta model, too.

Figure 6. Meta Model



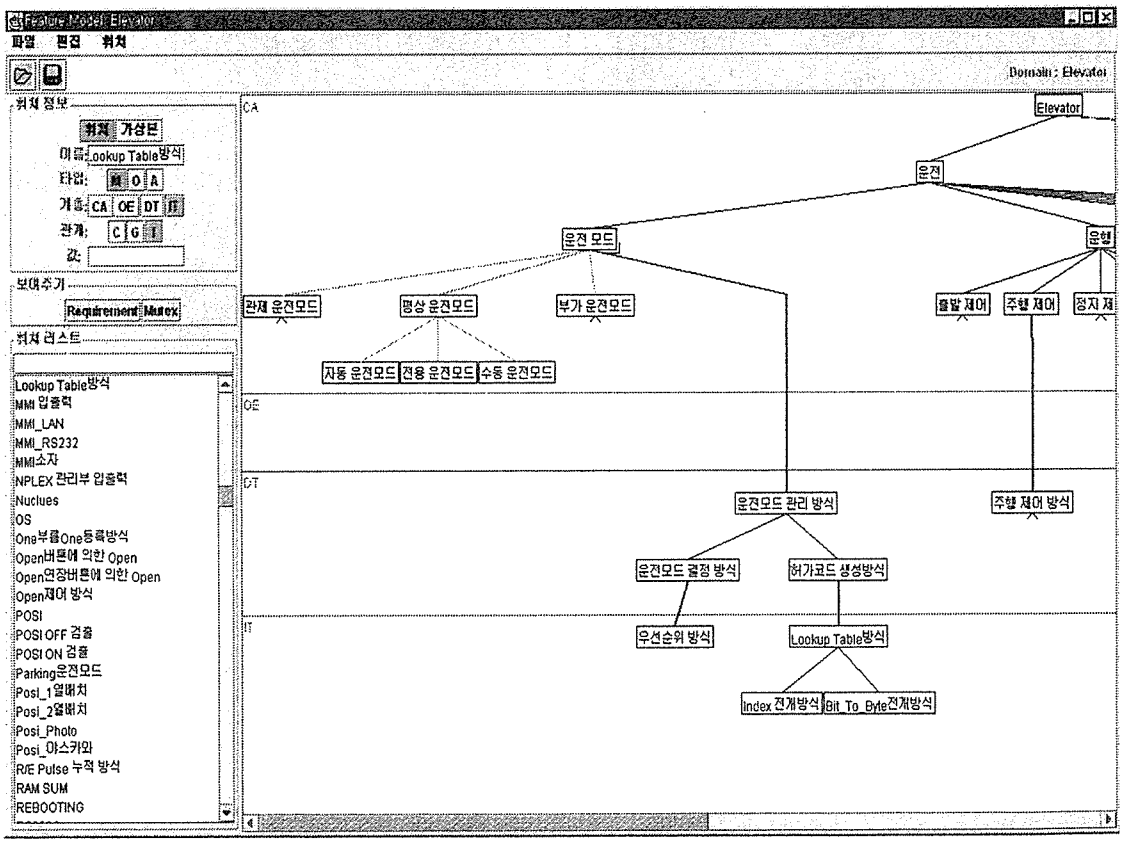
## 4. CASE Tools in the development process

Successful software engineering needs convenient environment as well as organizational support. Especially, to establish the automated development environment can greatly improve possibility of success since software engineering usually requires a lot of labor work. Also, models produced should be evaluated and validated before proceeding to the next step. So the necessity of proper tools is clearly inevitable. So we come up with developing new tools for the new method and the result was very successful.

As the most of other CASE tools do, our tool provides all necessary model drawing facilities such as feature model diagram, MSD, DFD, statechart, subsystem model diagram and process model diagram, and supports automatic data transformation and transfer between models according to the meta model. Also, it provides feature model validation, operational model simulation, as well as target model simulation. For the elevator control system, we produced a feature model with over 400 features, many statecharts, DFD's and MSD's. So it is nearly impossible to evaluate and validate all those feature relationships and statechart transitions manually, and this tool does those works correctly in short time. In addition to that, the advanced visualized simulation tool development using 3D modeled objects is going on.

Automatic code generator generates the codes not in fixed language, but in canonical form that means the codes generated from the auto code generator can be used in any platform. Only thing needed is a proper interpreter between canonical language and target language.

Figure 7. CASE Tool Screen



## 5. Conclusion

In the paper, the new software development process is successfully applied to the elevator control system development. First of all, the new process really extends the reusability range into the requirement specification phase. Feature model is used to describe complicated elevator control system along with dynamic models such as statechart, so the entire elevator control system is now in much more understandable form. Systematic relationships and data transformation among the models makes the entire process quite automatic and as a result, application development from user's feature selection to the final product can be done faster, easier and more reliable while saving time and labor.

CASE tools do the most of works needed to develop a system. Especially validation of models and simulation of the selected systems can greatly reduces the logical error before the actual codes generated. CASE tools are programmed in JAVA language which can run the application in any platform, so we could almost achieve the platform-free tool that produces platform-free codes.

## 6. References

[1] Kyo C. Kang, Sajoong Kim, (1998) Annals of Software Engineering Vol.5, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures", Virginia Tech, VA

[2] Mari Natori, Akira Kagaya, Shinichi Honiden, (1996) Software Reuse, "Reuse of Design Processes Based on Domain Analysis", IEEE, California, pp.31-40

[3] Kyo C. Kang, (1993) Proceedings of JCSE '93, "Feature-Oriented Domain Analysis for Software Reuse", pp. 389-395

[4] Gomma, Hassan, (1993) Software Design Methods for Concurrent and Real-Time Systems, Addison Wesley, MA

[5] Martin, Robert C. (1995) Designing Object-Oriented C++ Applications Using the Booch Method, Prentice Hall, NJ

[6] Marilyn Keller, Ken Shumate, (1992) Software Specification and Design A Disciplined Approach for Real-Time Systems, John Wiley & Sons, NY

## 7. Authors' Biography

Byung Wook Choi received his Ph.D from KAIST in Seoul, Korea, in 1992. From 1992, Dr. Choi joined LG Industrial Systems Co. Ltd., where he is currently working as a senior engineer in Core Technology Laboratory. His current research interests include real-time systems, embedded systems programming, and software engineering.

Chang Hee Kim received B.A. in Computer Science from New York University(NY) in 1994 and got M.S. in Computer Science from Columbia University(NY) in 1996. He

joined LG Industrial Systems Co. Ltd. in 1996 and has been working on software engineering for Core Technology Laboratory.

Ki Byung Jang received M.S. in Electrical Engineering from Inha University in 1990. He joined LG Industrial Systems in 1990 and has been working on Software Engineering for Core Technology Laboratory. His current research interests are object-oriented design of embedded system, formalism for real-time simulation.

Kyo-Chul Kang received his Ph.D. from the University of Michigan in 1982. Since then he was worked as a visiting professor at the University of Michigan, a member of technical staff at Bell Communications Research and AT&T Bell Laboratories, and a senior member at the Software Engineering Institute, Carnegie Mellon University. He is currently a professor at the Pohang University of Science and Technology. His areas of research interest include requirements engineering, real-time systems development, and software reuse.

Seng Jin Roh was graduated from Kyung Hee University in Seoul, Korea in 1986. He joined LG Industrial Systems Co. Ltd. and has been working on elevator control systems as a senior engineer.