Proceedings of the 14th
KACC, October 1999

UML을 이용한 원전 디지털 제어시스템의 소프트웨어 설계 및 검증

Design, Verification and Validation for Digital Control System S/W for Nuclear Power Plant Using UML

°이 병 윤*, 신 창 훈**, 문 홍 주, 이 성 우, 윤 명 현

* 전력연구원 원자력연구실(Tel : 81-042-865-5664; Fax : 81-042-865-5504 ; E-mail: bylee@kepri.re.kr)
** 전력연구원 원자력연구실(Tel : 81-042-865-5643; Fax : 81-042-865-5504 ; E-mail: hoony@kepri.re.kr)

Abstract : For the development of digital I&C(instrumentation and control) system in the nuclear power plant, its safety and reliability are the most critical properties to be required. In this paper, the feasibility of developing the digital I&C system in the nuclear power plant with OOM(Object Oriented Method) using UML was studied from the view point of the safety and reliability.

Keywords : UML(Unified Modeling Language), Nuclear Power Plant, Software Design, Software Verification and Validation, I&C(Instrumentation and Control)

1. 서론

원자력발전소의 디지털 제어계측계통을 위한 소프트웨어는 고신뢰성 및 안전성을 요구한다는 면에서 기존의 타 분야 제어시스템 소프트웨어와 차별화 된다[3]. KINS(한국 원자력 안전 기술원)에서 요구하는 안전요건을 적정 수준으로 만족하기 위해서는 신뢰성과 안전성을 고려한 설계개념 및 개발과정의 확인 및 검증이 필수적이다[6]. 현재까지 원자력 디지털 제어계통의 소프트웨어 개발을 위한 표준적인 절차(Process)와 방법(Methodology) 그리고 확인 및 검증 방법(Verification and Validation)은 정립되지 않고 있으며 최근에 들어서야 디지털 시스템에 대한 안전 심사 지침을 마련해 가고 있다.

본 논문에서는 객체지향개발 기법을 사용한 디지털 제어 시스템 소프트웨어의 개발의 적합성을 고려해 보고 객체 지향 소프트웨어 개발의 표준 모델링 언어인 UML(Unified Modeling Language)을 사용한 소프트웨어 개발 방법을 고찰해보고자 한다. 또, 이에 따른 확인 및 검증 활동의 용이성을 타진해 보고자한다.

2. 본론

1. 디지털 제어시스템 소프트웨어의 객체지향 기법으로의 개발

일반적으로 디지털 제어시스템은 실시간성(Real Time)을 요구하는 내장형 디지털 시스템(Embedded Digital System)이다. 전통적으로 내장형 디지털 시스템은 구조적 설계 방법(Structural Development Method)을 기반으로 하여 실시간 운영체제(Real Time OS)위에 어셈블리 언어나 C언어를 사용한 구조적 프로그래밍의 방법을 통해 구현해왔다.

상당수의 내장형 디지털 시스템 개발자들은 객체기술의 미성숙, 객체기술의 비효율성, 객체기술 지원 도구의 부재 등을 이유로 구조적 방법을 계속 사용하고 있으나 최근에 다수의 내장형 시스템이 객체기술을 기반으로 하여 성공적으로 개발되고 있다.

객체기술의 비효율성은 기존의 구조적 개발 방법에 익숙한 개발자에게는 근거가 있다. 추상화(abstraction), 캡슐화(encapsulation) 등과 같이 잘 정의된 소프트웨어 공학적 개념을 요구하기 때문에 경험이 충분하지 않은 개발자들은 구조적 방법으로 개발하는 것보다 비효율적인 결과물을 생산하기 쉽다. 객체기술을 위한 컴파일러와 같은 지원도구들도 과거의 8-bit나 16-bit 프로세서에는 지원되는 컴파일러가 흔하지 않았지만 최근의 32-bit 프로세서에는 다양한 컴파일러가 제공되고 있다. 객체기술은 현재 범용 컴퓨터의 소프트웨어 개발 방법으로 광범위하게 응용되고 있으며 내장형 디지털 시스템으로 점차 확대되고 있는 추세이다[1].

객체 기술은 구조적 개발 방법의 여러 가지 문제점을 보완하기 위해 그 개념이 고안되었고 아래와 같은 상당한 장점들을 갖고 있다.

1.1 모델 관점의 일관성(Consistency of model View)

구조적 설계방법의 문제점 중의 하나는 분석 관점(Analysis View)과 설계 관점이 달라서 서로간의 연관관계를 찾기가 힘들다는 것이다. 구조적 분석 모델(Data Flow Diagram, Entity Relation Diagram)과 구조적 설계 모델(Structure Flow Chart)의 연관관계를 보여주는 것은 쉽지 않다. 따라서, 분석단계의 결과가 설계에 정확하고 빠짐없이 반영되었는지 확인하는 작업이 연속적이고 일관되게 이루어지지 못한다. 하지만, 객체 기법은 전 개발 단계를 통해서 일관된 모델링 기법을 사용하여 상위 단계의 결과물의 연속적인 사용을 통해서 개발하게 되므로 각 개발 단계의 결과물이 일관된 모델 속에서 나타난다. 객체(Object)와 클라스(Class)는 분석 모델에서부터 실제 코드에까지 그 모델이 나타나게 되므로 실제 코드와 분석 모델의 관계가 보다 명확하다.

1.2 보다 나은 문제영역 추상화(Improved problem-domain abstraction)

모델링 언어의 추상화능력은 매우 중요하며 시스템내의 독립적인 부분들은 추상화 된 모델을 통하여 자연스럽게 분리되며 실물대상의 변화가 있을 때 타 부분에 변화를 최소화 할 수 있으며 모델 자체가 매우 안정적이다. 구조적 개발 방법론은 추상화 능력에 다소 한계가 있으며 이로 인한 인위적인 구조의 분리로 인해 실물대상 변화 시 전체시스템에 상당한 영향을 주는 등의 비효율성을 야기할 수 있다.

1.3 모델 재사용에 대한 편이성(Improved model facilities for reuse)

객체지향 기술은 모델 재사용을 위해 집성(aggregation), 일반화(generalization), 형태 매개화(type parameterization)의 3가지 방법을 제공한다. 집성(aggregation)은 재사용 가능한 부분들을 모아서 새로운 것을 만들어내는 것을 말하며 이것을 통해서 하위 구성요소들을 포괄하는 상위 요소를 만들어낼 수 있다. 일반화(Generalization 혹은 Inheritance)는 이미 존재하는 요소로부터 확장(Extension) 혹은 특별화(Specialization)를 통해서 새로운 것을 만들어내는 것을 말한다. 확장(Extension)은 새로운 동작이나 정보를 추가해서 새로운 요소를 만드는 것을 말하고 특별화(Specialization)는 동작을 새롭게 변경하는 것을 말한다. Type parameterization은 요소의 데이터를 기본적인 구조만 정의하고 불완전하게 정의함으로서 재사용을 지원한다.

1.4 시스템 크기 조절의 용이성(Improved Scaleability)

구조적인 개발 방법은 해당 문제의 복잡성과 밀접한 관계를 갖고 개발하게 되고 문제의 복잡성의 변화에 따라 적절하고 쉽게 시스템의 변화를 가져오기 힘들다. 객체 지향 개발 방법은 잘 정의된 추상화와 캡슐화를 지원함으로서 문제의 복잡성의 변화를 적절히 반영할 수 있다.

1.5 신뢰성과 안전성의 향상된 지원(Better Support for Reliability and Safety Concerns)

추상화와 캡슐화의 지원으로 객체간의 상호작용이 잘 정의된 인터페이스를 제공하여 객체간 상호 작용을 제어할 수 있고 시스템이 제대로 동작하기 위한 사전 사후 조건을 강화할 수 있으므로 신뢰성을 증대시킬 수 있다. 그리고, 재사용이 용이하므로 이미 사용 경험이 있는 검증된 요소들을 사용하여 안전성과 신뢰성을 증대시킬 수 있다.

1.6 원천적인 동시성 제공(Inherent Support for Concurrency)

표준적인 구조적 방법(Structured Method)에서는 실시간 내장형 시스템의 중요한 특징인 동시성(Concurrency), 임무 관리(Task Management), 임무 동기(Task Synchronization)등을 표현하기가 힘들다. 하지만, 객체 기법은 원천적으로 동시성(Concurrency)을 제공하며 Statechart, Active Object, Object Messaging 등과 같은 방법의 강력한 도구를 제공한다.

객체지향 기법은 위와 같은 장점이 있지만 모든 시스템에서 성능, 신뢰성, 안전성 및 경제성 면에서 향상된 시스템을 개발하게 되는 것은 아니다. 시스템의 복잡성, 타스크 관리 중요성, 재사용으로 인한 경제성, 신뢰성 및 안전성을 요구하는 시스템일수록 구조적인 방법보다는 객체지향의 개발 방법이 보다 효과적이라고 말할 수 있다.

원자력 발전소의 제어시스템 개발에 객체 기법을 사용하는 것은 장단점이 있다. 하지만 현재의 기술 수준으로 볼 때 객체지향기술은 성숙된 기술이며 원전에서 요구하는 안전성과 신뢰성을 보장하기 위한 설계개념을 포함시키기에도 비교적 용이하다. 또한 재사용의 용이성으로 인해 이미 개발되고 검증된 모델을 사용하기 쉽게 하기 때문에 보다나은 신뢰성을 확보할 수 있다. 객체기법을 통한 개발은 소프트웨어 공학에 기초한 분석과 설계단계에서의 철저한 모델링을 요구하기 때문에 소프트웨어의 고품질을 확립할 수 있다. 객체기법 자체가 엄격한 개발 절차와 확인 및 검증 절차에 보다 순응할 수 있는 개발 과정 및 도구들을 사용하는 것은 분명한 사실이다.

2. UML의 소개

UML(Unified Model Language)은 Booch, OMT(Object Modeling Technique) 그리고 OOSE(Object-Oriented Software Engineering)로 대표되는 객체지향 모델 표시방법을 합쳐 놓은 것으로서 90년대 중반 이후에 등장하기 시작하였으며 OMG(Object Model Group)에서 표준으로 채택하고 있고 현재까지 UML1.3이 발표되었다.

2.1 UML의 목적

UML은 아래와 같은 목적으로 고안되었다.

- 쉽게 사용될 수 있는 객체기법의 비쥬얼 모델링 언어 제공
- 핵심 개념의 확장(Extension), 구체화(Specialization)등의 구조(Mechanism) 제공
- 개발과정과 구현 언어에 독립성 제공
- 모델링 언어를 이해할 수 있는 정형적 기반제공
- 객체지향도구의 시장성 확대
- 군집(Collaboration), 프레임워크(Framework), 패턴(Pattern), 콤포넌트(Component)와 같은 상위의 개발 개념 제공
- 기존의 좋은 모델링 관행의 통합

2.2 UML의 구성

UML은 여러개의 그래픽 표현 수단을(Graphical Notation) 통합적으로 제공하며 개발 프로세스와 목적에 맞게 적절히 선택해서 사용하게 된다.

2.2.1 Class Diagrams

클라스(Class)들과 그들 사이의 다양한 관계(Association, Aggregation, Composition, Generalization, Dependency 등)를 표현한다.

2.2.2 Use Case Diagrams

외부 사용자(Actor)와 사용예(Use Case) 간의 관계를 나타내며 사용예(Use Case)는 시스템의 내부구조를 반영한 것이 아니고 외부적으로 나타나는 기능을 나타낸 것이다. 따라서, 사용예(Use Case)는 클라스(Class)와 대응되는 것이 아니며 객체간의 상호작용으로 이루어진 군집(Collaboration)으로 구체화 될 수 있고 이때 각 클라스(Class)는 다른 군집(Collaboration)을 이루어 다른 사용예(Use Case)를 표현한다.

2.2.3 Sequence Diagram

Sequence Diagram은 Use Case Diagram의 실제 시나리오를 표현할 수 있고 내부적인 객체(Internal Object)의 상호작용을 나타내는 데 쓰일 수 있다. Sequence Diagram은 시간적인 순서영역에서 표현하고자하는 다양한 요소들의 시나리오를 표현한다.

2.2.4 Collaboration Diagram

사용예(Use Case)와 같은 공통 목적 달성을 위한 객체들간의 상호작용을 표현한다.

2.2.5 Statechart Diagram

FSM(Finite State Machine)를 기반으로하여 구조적 상태(Nested Hierachical State)와 직교 영역(Orthogonal Region)을 추

가하여 표현력을 확장한 것으로 각종 사건(Event)나 입력에 반응하여 동작하는 것을 표현한다[4].

2.2.6 Activity Diagram

시스템의 복잡한 활동모델을 나타내기 위해서 쓰인다. Activity Diagram은 절차적 활동을 표현하는 것과 동시성(Concurrency)를 표현하는데 유용하게 쓰인다.

2.2.7 Implementation Diagram

Implementation Diagram은 Component Diagram과 Deployment Diagram으로 구성되며 다른 표현 방법에서 제공하는 논리적인 모델링에 보완하여 물리적으로 각 소프트웨어 콤포넌트(Component)를 나타내고 실제 어디에 배치되어 전체 시스템을 구성하는가를 나타낸다.

3. UML을 이용한 원전 디지털 제어 시스템의 개발

원자력발전소의 계측제어 시스템은 고 신뢰성 및 안전성을 포함하는 설계개념 뿐만 아니라 엄격한 확인 및 검증 절차(Verification and Validation)가 요구된다. 확인 및 검증은 소프트웨어 생명주기 모델(Software Life Cycle Model)에 기초한 소프트웨어 확인 및 검증 계획 및 조직에 따른 체계적인 단계별 확인 및 검증활동의 수행을 요구한다. 앞에서 언급한 바와 같이 UML은 소프트웨어의 객체지향 분석 및 설계를 위한 비쥬얼 모델링 표기법을 제공하는 것이며 이 UML을 이용한 표준적인 개발 절차 및 방법은 아직 정립되지 않은 실정이므로 UML을 사용한 적절한 개발절차를 작성하는 것은 개발자들의 몫이다.

원전 소프트웨어 개발을 위해서는 우선적으로 각 개발 단계를 잘 정의하여 단계별 확인 및 검증 작업이 효과적으로 이루어지도록 하는 것과 개발 개념과 각 개발 활동들을 실시간성, 안전성과 신뢰성을 확보하도록 잘 정의하는 것이다. 시스템을 효과적이고 경제적으로 개발하기 위해서는 개발 순서에 적절하게 각 활동들을 적절하게 배열해야한다. 이를 위하여 소프트웨어 개발 계획 작성시 해당 시스템에 적합한 소프트웨어 생명주기 모델(Software Life Cycle)을 선택하여 이에 기초하여 모든 계획을 수립하게된다. 소프트웨어의 개발 모델 혹은 생명주기 모델은 폭포수 모델(Waterfall Model), 나선형 모델(Spiral Model), 프로토타이핑 모델(Prototyping Model)등 여러 가지 있으며 이들 중 해당 시스템에 적합한 개발 모델을 선정하거나 혹은 위에서 언급한 프로세스의 요구사항을 만족하는 적절한 프로세스를 별도 정하면 된다.

소프트웨어 개발 계획은 일정한 품질을 유지하며 시스템을 개발할 수 있도록 해야하며 복잡한 기능을 가진 시스템을 신뢰성있게 개발 할 수 있도록 해야하며, 시스템 개발의 종료시점을 예측할 수 있어야 하며, 개발비용을 예측할 수 있어야 하며, 중간 단계별 확인이 가능해야 하며, 대규모 혹은 중간규모의 시스템개발을 위한 효과적인 팀 활동을 가능하게끔 해야한다. 공통적으로 소프트웨어 개발계획은 분석(Analysis), 설계(Design), 구현(Implementation), 시험(Testing)등의 단계를 포함하게 되며 각 단계의 모든 작업은 계획에서 명시한 정도로 상세하고 구체적으로 다양한 관점의 시스템의 모델을 구성함으로서 이루어지고 서로 다른 관점의 모델들이 적절한 연관관계를 유지, 발전시키면서 개발단계를 진행시킨다. 여기서, UML은 각 단계별 활동에서 구축하는 모델을 표시하는 표기법으로서의 역할을 하며 실시간성과 안전성이 요구되는 시스템에 적합한 개념적 요소들을 적절하게 표현할 수 있는 도구들을 제공한다.

3.1 분석(Analysis)

분석단계는 요건분석(Requirement Analysis), 시스템 분석(System Analysis) 및 객체분석(Object Analysis)으로 이루어지며 요건 분석과 시스템요건분석은 시스템의 기능(Function)과 동작에 대한 것으로서 시스템 사양(System Specification)을 이룬다. 객체분석은 구조객체분석(Structural Object Analysis)과 동작 객체 분석(Behavioral Object Analysis)으로 이루어지는데 구조 객체 분석에서는 클라스(Class), 객체(Object)와 이들의 유기적 결합체인 패키지(Package), 노드(Node) 및 콤포넌트(Component) 그리고 이들 사이의 관계를 분석하고 정의해낸다. 동작 객체 분석에서는 각 클라스(Class)의 동적 특성을 파악해낸다. 소프트웨어의 개발에서 분석단계는 매우 중요하다.

3.2 설계(Design)

설계는 구조적 설계(Architectural Design), 기계적 설계(Mechanistic Design) 그리고 상세 설계(Detailed Design)로 이루어진다. 구조적 설계는 소프트웨어 모듈의 배치, 개발 및 동시성(Concurrency)의 관점의 모델을 구성한다. 이 단계에서 정의된 구조의 각 요소들은 전체 소프트웨어 구성의 전략적 차원에서 정의되며 구조 설계 패턴(Architectural Design Pattern)이 널리 사용된다. 시스템의 특정기능을 위해 여러 개의 객체들이 모여 군집(Collaboration)으로 기능을 수행하며, 기계적 설계(Mechanistic Design)를 통해서 메카니즘(Mechanism)으로 구체화된다. 기계적 설계에서 기계적 설계 패턴(Mechanistic Design Pattern)이 널리 이용된다. 상세 설계는 각 클라스(Class)의 내부를 정의하고 구조를 상세화하여 선정된 프로그래밍 언어에 적합하게 상세화 된다.

설계에서는 다양한 패턴(Pattern)들이 사용되는데 이는 설계 작업을 효율적이고 신뢰성있게 하는 중요한 도구가 된다. 패턴(Pattern)은 주로 공통적이고 중요하게 발생하는 문제에 대한 일반적인 해결책으로 알려진 것들이다. 이미 여러 시스템의 개발에 패턴(Pattern)으로 제공되어 성공적으로 동작하고 있다면 이는 이미 어느 정도 검증된 구조와 동작방식이라고 인정할 수 있다. 이와 같은 다양한 객체로 구성된 검증된 패턴을 사용하여 시스템의 신뢰성을 높일 수 있다. 원자력 분야의 소프트웨어들의 공통적인 문제인 고 안전성 고 신뢰성을 해결하기 위한 다양한 검증된 설계 패턴(Design Pattern)들을 개발, 발굴하고 이를 적절히 사용할 수 있는 환경이 된다면 소프트웨어의 안전성과 신뢰성에 관련된 문제의 상당부분을 해결할 수 있으리라고 본다. 그림 1과 그림 2는 Safety Executive Pattern의 한 예인 Class Diagram과 Sequence Diagram을 각각 보여주고 있다.

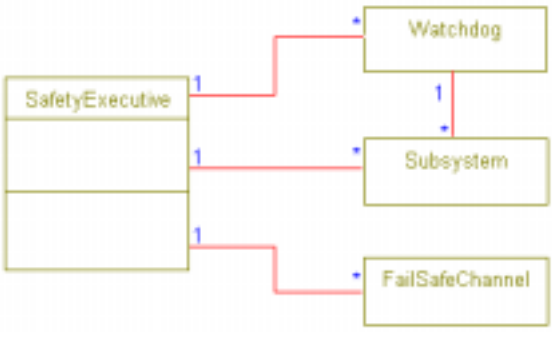

그림 1. Safety Executive Pattern의 Class Diagram

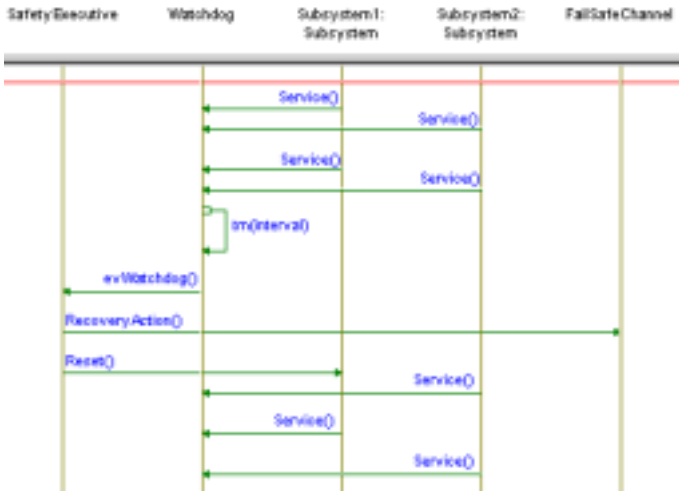

그림 2. Safety Executive Pattern의 Sequence Diagram

4. UML을 이용한 소프트웨어의 확인 및 검증 (Verification and Validation)방법

소프트웨어를 수학적, 체계적으로 확인 및 검증하는 것은 매우 어렵다. 소프트웨어의 각 모듈이 현재의 출력은 현재의 입력에 의해서만 정의되는 함수처럼 될 수 있다면 수학적으로 매우 쉽게 확인 및 검증을 해석적 방법으로 수행 할 수 있다. 하지만, 소프트웨어의 프로시듀어(Procedure)들은 현재의 입력뿐 아니라 주변의 여러 변수들에 의해 영향을 받기 때문에 복잡한 시스템을 수학적으로 쉽게 모델링하기는 힘들다. 실제로 Z와 같은 정형언어들은 그 사용을 위해서는 상당수준의 수학적 배경이 요구된다[2].

UML은 정형언어는 아니지만, Statechart, Sequence Diagram등과 같은 시뮬레이션이 가능한 형태의 모델링 언어를 제공하므로 반정형 언어(Semiformal Language)라고 볼 수 있고 그 표현방법만으로는 매우 간단하고 잘 정의된 표현방법을 제공한다. 또 UML은 소프트웨어 엔지니어링에서 요구하는 기본 개발개념을 담고 있기 때문에 확인 및 검증 활동을 보다 적절히 지원할 수 있는 표기법이다.

소프트웨어의 확인 및 검증은 분석(Analysis), 검토와 조사(Review & Inspection) 및 시험(Test)의 3가지 방법을 통해서 소프트웨어 확인 및 검증 계획에 따라 각 단계별 개발결과물들의 건전성, 적합성 및 완전성을 확인하고 검증한다[5].

UML을 통해서 분석모델을 작성하게 되므로 이 자체가 확인 및 검증의 분석활동의 상당부분 차지하게 되며 이의 검토와 필요하다면 UML CASE(Computer Aided Software Engineering) 도구 을 통해서 모의실험(Simulation)을 통해서 확인 및 검증 활동을 완성시킬 수 있다. 확인 및 검증의 요건으로 완전 정형언어(Full Formalism)를 요구하는 시스템 외에는 UML 자체만으로 분석모델의 역할을 충분히 행할 수 있으며 그래픽 표기(Graphical Notation)와 다양한 CASE 도구들의 지원으로 모의실험(Simulation)을 손쉽게 수행할 수 있다. 원자력분야에서도 완전정형을 요구하는 원자로 보호계통을 제외한 나머지 안전관련 계통에서도 검증도구로서 사용 가능하다.

3. 결론

지금까지 원자력 발전소의 디지털 계측제어시스템의 소프트웨어 개발을 위해 객체지향 개발기법 사용을 고려해 보고 객체 지향 소프트웨어 개발의 표준 모델링 언어인 UML(Unified Modeling Language)을 사용한 소프트웨어 개발 방법과 확인 및 검증에 대해서 살펴보았다.

위에서 언급한 바와 같이 UML을 이용한 객체지향방법은 구조적 설계방법의 여러 가지 단점을 극복하기 위해 개발되었고 특히 객체모델 및 코드의 재사용, 각종 설계 패턴 지원, 반정형성(Semi-formalism) 지원 등은 신뢰성 있는 소프트웨어의 개발 및 소프트웨어 확인 및 검증(Verification and Validation)을 위한 큰 장점을 제공할 수 있다. 원자력 발전소의 디지털 제어시스템과 같은 실시간 내장형 시스템을 위한 각종 설계 패턴, 구현 패턴을 개발하고 검증하여 향후 타 시스템 개발에 사용한다면 소프트웨어의 신뢰성을 상당부분 해결될 것으로 예상된다.

참고문헌

[1] B. P. Douglass, *Doing Hard Time*, Addison Wesley, 1999.

[2] S. Levy Incorporated, *Handbook for Verification and Validation of Digital Systems*, EPRI, Dec., 1994.

[3] N. Storey, *Safety-Critical Computer Systems*, Addison-Wesley, 1996

[4] D. Harel, *Statecharts: A Visual Formalism for Complex Systems*, Science of Computer Programming, vol. 8, pp. 231-274, 1987.

[5] R. S. Pressman, *Software Engineering*, McGraw-Hill, 1992]

[6] 한국원자로안전기술원, *경수로형 원전 안전 심사 지침 제7.0절, 1998. 5*