'99추계학술발표회 논문집
한국원자력학회

평가논리를 유지하면서도 크기가 최대화된 절차서 수행 단위의 설계와 이를 원소로 한 전산화 절차서의 개발

정연섭, 신영철, 정경훈, 성찬호

한국전력공사 전력연구원
대전광역시 유성구 문지동 103-16

요 약

정형화된 "action"과 "check"을 기본으로 전산화절차서 시스템을 개발하였다. "actions"과 "checks"은 접고, 이동이 가능하고, 상호 관련된 기능을 가진 2차원의 플로우챠트 속에 만들어져 있다. action과 check는 컴퓨터에 의해 자동으로 평가되어 지지만, 필요하다면 운전원에 의해 최종 결정이 가능하다. action과 check의 자세한 사항은 boolean instruction과 n out of m의 수학적 논리를 가지는 논리 tree속에 기술되어져 있다. 컴퓨터는 운전원이 boolean instruction 내에있는 관련된 공정변수나 제어기기를 조작할 수 있도록 지원한다. 이러한 컴퓨터의 지원에도 불구하고, 그 내부 logic은 운전원이 제어 루프내에서 그것들을 받아들이기가 쉽게, 또 투명하게 되어있다. boolean instruction은 3가지 상태를 가질 수 있고, n out of m 논리는 3가지 상태로 정의되어 있다. 그 정의는 절차서 수행중에 문맥의 전환 수를 최소화 할 뿐 아니라, action과 check의 평가를 가능하게 한다. 공정변수는 인지가 쉽도록 텍스트와 그래픽 심벌의 두가지 형태로 나타나있다. 또한 절차서 실행중에도 절차서의 어떤 부분으로도 접근할 수 있는 새로운 방법이 고안되어져 있다.

Abstract

A computerized procedure system was developed on the basis of formalized action and check. The actions and checks are rendered into two dimensional flowchart with foldable, dynamic, and interactive functions. The action and check are evaluated by computer automatically, and can be overridden by operator if necessary. Detail of each action and check is described in the logic tree with several boolean instruction set and n out of m mathematical operator. Computer supports operators to execute the boolean instruction providing relevant process parameters and control devices. In spite of all these computer support, its inner logic is simple and transparent to operators to keep them in the control loop. The boolean instruction can have 3 states. And n out m operator is defined on the 3 states. The definition minimizes not only number of context switching in procedure execution, but also enables evaluation of the action and check. The process parameters are presented in both text label and graphical symbol to improve readability. Moreover, new mechanism to access any part of a procedure is devised while keeping the main stream of procedure execution.

1. 서 론

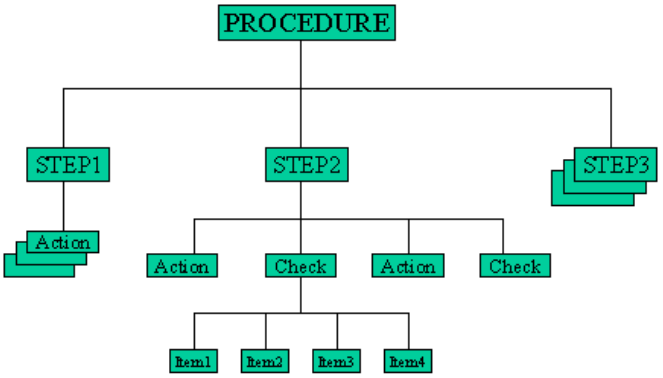
원자력 발전소에서는 다른 화학공장과 다르게 절차서를 통한 운전이 요구 되어왔다. 여기에는 여러 가지 이유가 있겠지만 절차서를 통하여 안전성이 확보될 수가 있기 때문이다. 화학공장에서는 다소 안정성이 문제가 되더라도 높은 수준의 I&C 자동화를 통하여 넓은 범위까지 운전이 가능하므로, 서너 개의 버튼 속에 절차서의 모든 순서를 입력하여 둠으로써, 덩치 큰 절차서 없이 몇 개의 버튼으로 운전이 가능한 것으로 여겨진다. 그러나 원자력 발전소인 경우 넓은 운전범위를 망라하고, 안정성 있고, 자동화된 I&C의 설계가 어려우므로, 운전원을 통하여 그 범위를 대신하도록 일이 할당되었다. 그리고 운전원의 경험이나 훈련에 의하여 운전이 가능하지만, 더 높은 안전성 확보를 위해 종이에 기록된 절차서를 참조하도록 요구되었다. 이와 같은 맥락에 따르면 원자력운전의 편의성과 안전성을 높이는 방법에는 2가지의 방법이 있어 보인다. 하나는 일반 공정에서와 같이 높은 수준의 I&C 자동화를 통한 방법이고 또 하나는 종이에 기록된 절차서의 개선이다. 80년, 90년대를 통하여 많은 전산화 절차서가 개발되고 평과 된 것이 사실이다. 그러나 불행하게도 높은 자동화 수준의 I&C 대신에 차선책으로 강구된 절차서 시스템이 오히려 설계가 힘들고, 운전원에게 사용하기 어려운 시스템이 되어버렸다.

본문에서 제시하고 있는 절차서는 운전원과 컴퓨터사이에 역할분담이 명확히 되어있고 운전원은 컴퓨터 판단을 뒤집을 수 있는 권한이 있다. 반면에 컴퓨터는 운전원이 저지르기 쉬운 사소한 실수를 방지하도록 사용자 인터페이스를 잠그는 기능도 제공한다. 비록 컴퓨터가 Boolean 논리에 따라 판단을 하지만 결코 운전원에게 숨겨진 상태에 있지 않다. 또 이 시스템은 절차의 빠른 이해를 돕기 위해 흐름도로 절차가 표시되고 현재 수행중인 단계는 매우 세밀하게 논리도로 묘사된다. 물론 수행중인 단계에서 참조할 발전소 상태나 제어 접근방법도 함께 표시된다. 이와 같은 설계에서 흔히 저지르기 쉬운 오류는 절차서 수행의 기본단위를 너무 상세히 쪼개어 절차서의 수행단계의 갯수를 많게 하든지, 반대로 기본단위를 너무 크게 나누어 운전원이나 컴퓨터가 내리는 기본단위 해석이 애매하여지는 오류이다. 이 절차서의 핵심은 기본단위를 정형화된 형태를 항상 유지하되 기본단위의 크기를 극대화시킨 점이다.

2. 절차서 시스템의 사용자 인터페이스

이 시스템에서 읽혀지는 절차서 파일은 정형화된 절차서 수행단위를 기본으로 피라미드식으로 쌓아 놓은 것이다. 제일 상층부에는 "PROCEDURE"가 있고 이것은 "STEP"이라는 하위 단위를 담을 수 있는 그릇이다. "STEP"도 계속하여 "ACTION-CHECK"라는 정형화된 수행단위를 담는 그릇이다. 그러므로 "PROCEDURE"이나 "STEP"은 절차의 수행 단위라기보다는, 절차에 의미 혹은 번호를 부여하여, 운전원이나 혹은 컴퓨터에서 절차서의 수행 및 관리를 수월하게 만드는 경계라고 여겨진다. 다시 말해 종이 절차서에서 절차서의 이름과 단계의 번호를 부여함과 같고 전산화를 통하여 이 개념에 변화를 주지 않았다. 아래 그림은 피라미드식의 자료구조를 보여준다.

"ACTION-CHECK" 하위에는, 발전소의 상태를 점검이나 발전소의 상태를 변화시키거나 또 다른 절차를 호출하기 위한, 매우 구체적인 원소적 명령들이 있다. 이 명령들은 n out of m 논리에 의하여 결합되어 있다. 이렇게 구축된 절차서 파일은 XML의 규격에(참조1) 의하여 쓰여졌으며 일반 문서 편집기나 웹 브라우져를 통하여도 파일의 내용을 볼 수가 있다. 그러나 위와 같은 자료구조에 알맞은 브라우져의 설계의 목적은 운전자에게 이해하기 쉽고, 일관되고, 발전소의 현재상태를 제대로 전달하자는 데 있다. 여기서 언급하고 싶은 것은 절차서의 계층적 구조가 설계되고 난 후에 브라우저의 설계를 나중에 언급하였지만 이 두개의 설계는 서로 영향을 주는 불가분의



Hierarchical Structure of Procedure

관계에 있다. 이것은 마치 HTML의 규격이 웹브라우져를 염두에 두고 설계되는 것과 유사하다. 그러므로 똑 같은 HTML규격에 따르는 웹브라우져는 조금씩의 차이는 있지만 대부분의 웹브라우져의 기능은 유사하다. 비슷한 논리로 이 시스템의 자료구조에서 운전원에게 제공할 수있는 최적의 사용자 인테페이스는 아래와 같은 창이 분할된 윈도우 환경과 유사하리라 여겨진다.

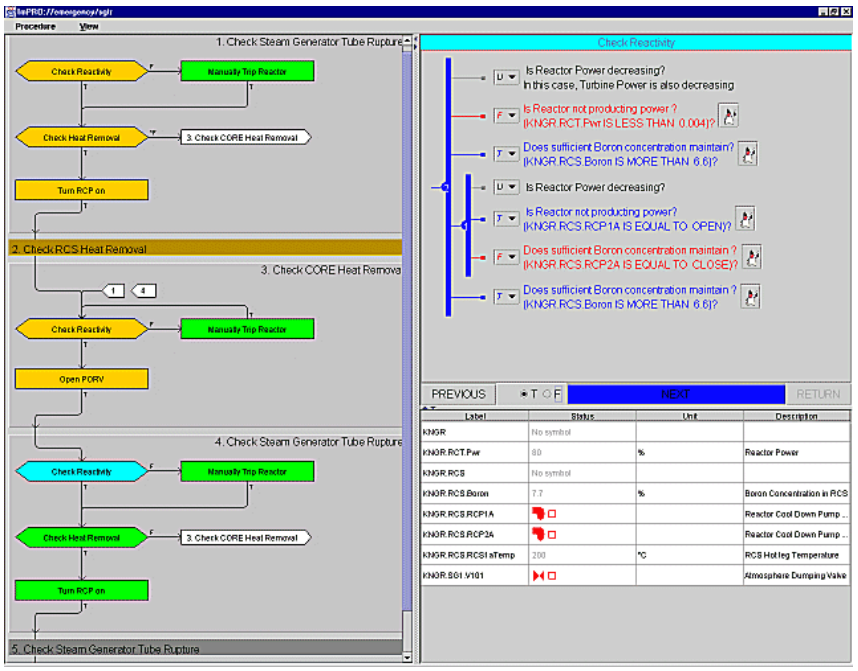


그림 1. 절차서 화면

[그림 1]에서 보듯이 좌측에 "STEP"층과 "ACTION-CHECK"층의 라벨이 흐름도 형식으로 표시되고 우측 상단에 "ACTION-CHECK"의 상세내용이 표시되며, 우측 하단에 발전소의 상태가 표시되도록 창을 분할하고 있다. 이 시스템을 사용중인 운전원에게는 3개간, 혹은 2개간의 사상이 일어난다. 그 사상이란 사용자 인터페이스의 뇌속 인식의 체계, 사용자 인터페이스 자체, 절차서 파일의 자료구조 사이의 관계이다. 3번째 항목에까지 의문을 품는 운전자는 이 시스템의 숙련자일 것이다. 이 관계가 정립되어 있는 운전원은 매우 쉽게 이 시스템을 사용할 수 있음은 물론이다.

그러므로 절차서 시스템의 설계자는 그 사상이 강해지도록 사용자 인터페이스를 일관되고 구별되게 설계해야 된다. 교육이나 훈련을 통한 인식체계 정립은 사용자 인터페이스의 갯수가 한정되어 있을 경우에는 가능하지만 절차서와 같이 수천 개가 넘는 다양한 화면에서는 거의 불가능하다. 이 시스템을 처음 대하는 운전원은 사용자 인터페이스 사이의 관계를 알아내려고 시도할 것이다. 이 시스템은 명쾌한 절차서 파일의 자료구조, 자료 구조와 사용자 인터페이스의 1대1 사상을 통하여 운전원의 인식체계가 자연히 구축되도록 설계되었다. 좌측 화면은 절차서 구조에서 "STEP"층, 더 정확하게는 계층적 자료구조에서 왼쪽에서 오른쪽으로 그려진 수평선을 따라 표현되어 있고, 사용자 인터페이스의 우측화면은 절차서 자료구조의 단면, 더 정확하게는 위쪽에서 아래쪽으로 내려오는 수직선을 따라 렌더링 되고 있음을 쉽게 인식할 수가 있다.

이 시스템에서 각 창의 배치는 고정되어 있다. 그렇지만 창 사이의 경계는 유동적이어서 한쪽 창의 정보가 많을 경우에는 창 크기의 조절이 가능하다. 고정된 창의 배치는 인지의 속도를 향상시켜주고, 의미의 변화가 없는 윈도우 제목을 생략할 수 있도록 돕는다. 한 창의 정보가 많을 경우에 창의 크기가 자동으로 조절되지 않으면 운전원에 의해서만 조절이 가능하다. 이런 보편적인 기능은 이 시스템만의 기능은 아니며 윈도우 스타일의 어떤 프로그램에서도 채택하는 방법이다. 사실 도스 프로그램에서 윈도우로 전환되면서 이런 상호작용이 인간에 미치는 효과의 장단점이 널리 인식되어 있으므로 이 시스템에서는 별도의 설계를 위하여 고민하지 않아도 된다. 간혹 이런 영향 평가가 힘든 경우는 구현중에 개발자가 여러 가지 설계의 사용성을 평가할 수 있다. 창이 정보의 양보다 작을 경우에 스크롤바가 생성된다. 각 창에 표시되는 심벌이나 테이블, 논리도는 동적이고, 발전소의 상태나 절차의 수행상태에 따라 색채가 변화거나 수축 팽창되면 자동 스크롤된다. 매우 복잡해 보이는 흐름도나 논리도 때문에 이미 그려진 그림을 올리는 것으로 생각할 수도 있지만 이 모든 렌더링(rendering)은 절차서 파일의 문자 자료에서 자동으로 그림으로 변환되어 처리된다. 자바로 프로그램된 정교한 알고리듬이 흐름도, 논리도, 테이블을 자동 생성시키고 있다. 많은 그래픽이 있지만 실제 파일의 크기는 매우 작다. 앞으로 계속 개발 중에 있지만 실행파일의 크기도 300Kbyte 이므로 매우 경제적인 알고리듬을 채택하고 있다.

[그림 1] 좌측의 흐름도는 수축된 "STEP"과 팽창된 "STEP"으로 구성된다. 둘 다 절차의 수행상태에 따라 배경색이 바뀐다. 이 "STEP"들은 위에서 아래로 배열되어 있으며 "STEP"를 이중클릭 하면 수축과 팽창사이에 전환이 일어난다. 디폴트 상태에서는 수행중이거나 수행된 "STEP"은 펼쳐진 상태로 존재한다. 특별히 한 "STEP"이 거대하지 않다면 펼쳐진 상태에서도 인접한 "STEP"을 볼 수 있어 전반적인 절차서의 윤곽을 알 수 있고, 모든 "STEP"이 수축된 상태에서는 절차서 전체가 한 윈도우 안에 보일 수도 있다. 각 "STEP"에는 대표하는 이름과 번호를 부여하여 운전원간 의사교환시 지칭할 수 있도록 했고 Proxy "STEP"의 표현시, 스크롤시에도 유용하게 사용이 가능하다. 또 수축된 흐름도 상태에서 절차서의 어느 부분도 쉽게 접근이 가능하다. 이것은 N4(참조2)나 COMPRO(참조3)의 절차서 한 부분의 접근성에 비하면 괄목할 만한 개선이며 종이 절차서에서 페이지를 넘겨 찾아가는 방법보다 용이하리라 기대된다.

팽창된 "STEP"은 내부에 화살표, 사각형, 양방향 사각형, 단방향 사각형으로 구성된다. 화살표는 절차의 흐름을 표시하며 참과 거짓에 따라 방향이 틀리다. 사각형은 "ACTION"의 심벌이고 양방향 사각형은 "CHECK"의 심벌이고 단방향 사각형은 Proxy "STEP"을 나타낸다. 이들은 내부에 문자열을 가지고 있거나 숫자를 혹은 둘을 동시에 가지고 있다. "ACTION"은 각각 하나의 입력 화살표와 출력 화살표를 가지고 있으며 출력 화살표는 항상 참으로 표시된다. "CHECK"는 하나의 입력선에 두개의 출력선을 가지고 있다. 출력선이 참과 거짓으로 모두 표시되는 것은 모름상태와 구별을 위한 것이며 모름상태에서는 다음 상태로 이동이 불가능하다. 그러므로 모름상태는 흐름도에서 표시되지 않는다. 수축된 "STEP"에서와 마찬가지로 절차의 진행상태는 색채로 구별된다. Proxy "STEP"은 화살표의 길이가 한 "STEP"이상 떨어져 있을 경우에 복잡성을 피하기 위해

도입한 개념이다. 이 흐름도에서 진출하는 Proxy "STEP"은 어떠한 심벌에서도 가능하지만 진입하는 Proxy "STEP"는 각 "STEP"의 초기에만 놓여 있다. 그러므로 진출하는 경우에는 한 개만 놓이므로 공간 확보에 문제가 없어 내부에 숫자와 명칭이 표시되지만, 진입하는 경우에는 여러 개가 동시에 존재할 수가 있다. 그래서 숫자만 표시된다. 이러한 설계는 대칭적 관점에서 보면 불균형이 있는 것처럼 보이지만 실제 절차서 사용에서 운전원의 인식형태를 분석하여 보면 불합리적인 것만은 아니다. 진행상태는 진행중, 진행됨, 진행안됨의 3가지 상태가 있으며 그 의미도 "NEXT" 버튼으로 지나갔느냐 안 갔느냐의 매우 단순한 논리에 근거한다. 실제 발전소 상태와는 경우에 따라서는 차이가 있을 수 있다. 완전히 자동화된 시스템이라면 매우 복잡한 논리를 코딩하여 색채변화를 주겠지만 사람이 관여하는 사용자 인터페이스에서는 운전원이 이해 못하는, 혹은 자신의 논리는 숨겨 놓고 결과만 표시하는 설계는 이 시스템에서 전적으로 배제하고 있기 때문이다. 이와 같은 설계원칙은 웹브라우져 에서도 발견되는데 가령 다른 웹페이지를 참조하였다면 글씨의 색채가 바뀌는 인터페이스와 유사하다.

당연해 보이는 "CHECK"과 "ACTION"도 더 자세히 음미하여 보면 경우에 따라서는 부적절하게 보일 수 있다. 사람의 행위나 컴퓨터의 작업은 점검하고 행동하는 반복적인 패턴을 가지고 일어나지만 사람인 경우에 "CHECK"와 "ACTION"이 한 문장 속에 들어 있을 경우에 이해하기 쉬울 수도 있으며 결과가 2개 이상인 "CHECK"도 있을 수가 있다. 컴퓨터의 프로그램언어를 예를 들면 if, then, else, while, do, switch, function call의 기본적인 명령으로 거의 어떠한 논리도 표현이 가능하다. 이 언어가 완벽하기 때문에 포트란에서 시작되어 자바로 이어지는 어떠한 프로그램 언어에서도 유사한 제어문장을 가진다. 한 언어의 우월성은 결코 제어문장의 우월성에 있지 않고 객체지향성에 표현되는 그런 특성에 있는 것이다. 절차서 시스템도 이런 제어 언어를 몰라 채용하지 않는 것이 아니라 사람을 염두에 두고 있기 때문이다. 예를 들어, 문자로 표현된 프로그램 코드를 완벽하게 흐름도로 나타내거나 프로그래머가 아닌 사람이 이해하도록 흐름도를 그리기는 어렵다. 물론 이것이 가능하다면 SW V&V는 어렵지도 않겠지만, 이 시스템에서는 운전원이 이해할 수 있는 범위 내에서, 가장 자동화 정도가 많이된, 그리고 수행의 기본 단위가 너무 잘지 않는 흐름도를 채택하였고 이것이 이 설계의 핵심이다. 또 NRC의 보고서(참조4)에 의하면 흐름도에서 "CHECK"과 "ACTION"을 하나로 묶어 무정형으로 만들지 못하도록 권고하고 있다.

팽창된 "STEP"을 이중클릭 하면 수축된다고 하였는데 이것도 내부에 있는 "CHECK"과 "ACTION"를 제외한 다른 나머지 영역을 클릭한 경우이다. 이것들이 클릭 될 때에는 그 "STEP"이 수축되기보다는 클릭된 "CHECK"과 "ACTION"에서 또 하나의 절차서 임시수행이 시작된다. 발전소의 동적인 특성이나 운전원의 수행한 기억은 휘발성이 있기 때문에 절차서 수행이 순차적으로 진행된다고 장담할 수가 없으며, 운전원의 인터뷰나 시뮬레이터를 통한 평가 또는 공정의 특성상 절차서 임시 수행은 필요 불가결하다. 이러한 개념은 HRP의 COPMA(참조5)에서 처음 도입되었으며 이 절차서에서 그 개념을 건설적으로 변경하였다. COPMA에서는 한 절차서에서 임시 절차서 수행이 무한개까지 가능하였다. 물론 이러한 설계가 전적으로 잘못되었다고 평가할 수 없지만, 운전자에게 어느 절차서 임시수행을 따라 가는 것이 좋을지 몰라 혼란만 야기시켰다. 운전원 평가에서 COPMA 개념에 따라 운전한 운전원이 아무도 없었으며, 굳이 이런 평가 없이도 충분히 예측 가능한 설계특성이다. 이 시스템에서는 절차서 주 수행을 통하여 절차서 수행의 궤적을 관리하고, 한 순간에 1개 이하의 절차서 임시 수행기능으로 돌출적 절차수행, 더 나아가 계속수행 단계 처리, 운전원과 협력이 가능하도록 설계하였다.

절차서 임시 수행의 개시는 위에서 언급한대로 이중클릭으로 개시되고 절차서 주 수행으로 복귀는 우측 상단 윈도우에 안에 있는 "RETURN" 버튼에 의한다. 물론 이 버튼은 임시 수행 상태에서만 활성화되어 있어 운전원의 주 수행에서 불필요한 오류를 최소화 시켰다. 이와 같은 설계특성에 의하면 두 상태 사이의 전이가 모두 운전원에 의하여 개시된다. 어느 것도 컴퓨터에 의해 자동으로 개시되지 않는다. 컴퓨터가 복잡한 논리를 통하여 운전원을 성가시게 하지 않는다. 운전

원은 절차서에 표시된 대로 읽고 해석하고 컴퓨터가 주는 신뢰성 있는 자료에 근거하여 주 수행과 임시 수행을 섞어 가면 된다. 운전원의 개입이 운전원의 부담으로 해석될 수도 있다. 그러나 이 해석은 적절하지 못하다. 오직 개시는 운전원으로 나타났지만 그 이후의 모든 후속조치, 흐름도를 다시 그리고, 스크롤 되고 하는 모든 것이 컴퓨터의 지원 하에, 운전원이 예측한대로 일관성 있게 일어나므로 운전원의 부담은 최소화된다. 오히려 운전원은 상위의 판단자로서 문제해결 능력이 향상된다.

여기서 이 시스템에서 컴퓨터에 의한 운전원의 지원 정도에 대하여 부연할 필요가 있어 보인다. 유사한 예를 들면 자동차의 운전자는 오직 핸들과 제동장치와 가속장치만으로 자신의 물리적 이동에 혜택을 입고 있다. 운전원이 자동차를 밀지 않고 헤드라이트를 켜기 위해서 전선을 잡고 있지 않다. 이 모든 지원은 엔진이나 기타 장비에 의해 지원되고 있다. 이 시스템도 현상파악에 필요한 주요 기능은 운전원의 판단에 의하고 다른 것은 컴퓨터의 도움을 받는다. 현학적인 컴퓨터가 인공지능이나 복잡한 논리를 동원하여 운전원의 핸들을 교체하려고 하지 않는다. 염두에 두어야 될 또 한가지 사실은 주 수행과 임시 수행의 개념이 사람을 위한 것이지 발전소 자체를 위한 것이 아님이다. 발전소 자체는 주 수행이든 임시 수행이든 명령이 들어오면 밸브를 열고 밸브를 닫지, 임시 수행이라 혹은 주 수행이라 다른 행동 특성을 보이지 않는다. 만약 절차서의 수행이 제대로 되었나를 평가하는 안전 기관이 있다면 그들의 기준 자체도 발전소의 입장에 가까울 것이다. 이와 같은 논리는 절차서의 수행 이력(로그)의 설계에 단서를 제공한다. 즉 그 이력은 운전자 인터페이스에서 보이는 궤적과 차이가 있을 것이다. 이 절차서에서는 이 기능이 아직은 구현되어 있지 않다.

흐름도에서 계속 수행단계는 종이 절차서에서와 마찬가지로 별표로 화살표 옆에 표시되어 있다. 일반적으로 계속 수행단계는 조건이 만족되는 경우와 불만족 되는 경우가 있으며 조건이 만족되는 방향으로 별표가 표시된다. 계속 수행단계는 그 단계가 지나가면 컴퓨터에 등록되어 계속 감시된다. 이 단계는 발사 개시 시간이라는 개념을 지니도록 설계되었다. 그 시간 이후에는 운전원에게 자신의 존재를 주지시켜준다. 이 시스템에서 늘 강조하지만 컴퓨터평가만을 위한 복잡한 논리의 입력은 근본적으로 설계에서 고려하지 않았으므로 계속 수행단계의 조건 평가가 컴퓨터에서 의해 완전히 자동화되지 않는다. 다른 일반 단계에서 느끼는 자동화 정도와 똑같이 운전원이 판단이 필요한 경우가 있다. 이 입력을 위해 발사 개시 시간이 도입되었으며 만약 이 시스템에서 지원하는 Boolean 논리식에 따라 컴퓨터만으로 판단이 가능하다면 발사 개시 시간은 연장된다. 운전중에 다음의 발사시간은 변경될 수가 있다.

우측상단 윈도우는 현재 수행중인 "CHECK"과 "ACTION"의 상세내용을 보여준다. "CHECK"인 경우에는 발전소 상태를 점검하는 항목들이 논리형 수목도에 따라 관계지어지고 "ACTION"인 경우 발전소 상태에 영향을 줄 수 있는 항목으로 논리형 수목도가 그려진다. 이 시스템의 특성인 사용자 인터페이스의 일관성은 여기서도 증명된다. 체크리스트란 표현이 많이 있지만 체크리스트의 논리형 수목도가 무엇이며 "ACTION"에 어떻게 참 거짓을 줄 수가 있는가? 그러나 이 시스템은 새로운 방법을 정의하여 사용자 인터페이스에 일관성을 유지하였다. 운전원은 항상 나타나는 매우 간단한, 아마 특별한 설명이나 훈련 없이도, 논리도만 이해한다면 어떤 복잡한 절차서 수행이 가능할 것이다. 이런 일관성이야말로 운전원에 주는 혜택 못지 않게 설계자, 검토자, 구현자 모두에게 놀라울 만큼 장점을 제공한다. 이들 항목들은 n out of m논리에 의하여 결합되어 있다. 만약 각각의 항목이 참과 거짓의 단순 Boolean논리라면 n out of m논리의 결과는 자명하다. 그러나 이 시스템에서는 참 거짓 모름의 3가지 상태에서 파생되는 논리식이기 때문에 조금의 사고가 요구된다. 이 시스템에서 고안한 정리체제는 다음과 같다. 만약 각 항목에서 참인 갯수가 x1, 모름의 갯수가 x2, 거짓의 갯수가 x3라 하면 x1이 n보다 크면 이 논리도는 참이고 x1+x2가 n보다 크면 이 논리도는 모름이며 그 외는 거짓이다. 이 정리체제는 3가지 상태와 n out of m논리 연산자를 원

소로 하는 집합에서 완벽한 지식체계이며 이 시스템에 그대로 구현되어 있다. 선을 하나 그어 이것이 "Goto"가 되기도 하고 부모, 자식의 관계가 되기도 하는 모호한 사용자 인터페이스를 제거하는 확실한 방법은 사용자 인터페이스 밑에 놓여있는 논리를 정확히 정립하는 것이다. 그러면 각 항목의 참 거짓 모름은 어떻게 정의할 것인가? 그 단계에 진입하여, "ACTION" 항목인 경우 수행했다면 참이고 수행전에는 거짓이다. "CHECK"항목인 경우 컴퓨터나 운전원에 의한 판단전에는 모름이고 판단 후에는 3가지 상태를 가질 수가 있다. 컴퓨터에 의해 판단 가능한 항목은 애니메이션 그림으로 구별되어 표시되며 애니메이션이 살아 있을 경우에는 컴퓨터에 의한 평가가 주기적으로 나타나므로 운전원에 의한 상위입력이 불가능하다. 운전원이 애니메이션 버튼을 클릭하면 애니메이션은 죽으며 그 이후 운전원의 선택이 가능하다. 애니메이션을 다시 살리면 컴퓨터에 의한 평가가 재활 된다.

운전원에 의한 선택에는 어떤 제약도 없다. 각각의 항목에는 그 항목을 조작하는데 필요한 사용자 인터페이스가 같이 제공된다. 가령 "CHECK"인 경우에는 3상태 콤보박스가 나타나 운전원의 선택을 도와주며 "INITIATE", "FINISH", "INPUT", "SET", "MESSAGE"등과 같은 원소적 단위의 명령도 컴퓨터가 충분히 지원을 한다. "INITIATE"나 "FINISH"시에 절차서를 적재하거나 삭제하는데 컴퓨터가 도움을 주며 "INPUT"명령시 운전원간 혹은 절차서간 정보의 공유가 가능하도록 변수의 값 설정이 가능하다. "SET"명령은 제어화면을 호출시켜준다. 마지막으로 "MESSAGE" 명령은 단지 토글버튼만 존재하는데 컴퓨터가 어떻게 도움을 줄지 모르기 때문이다. 이 명령에서는 운전원이 읽고, 해석하고, 조작한 후 토글 버튼을 누르면 된다. 계속적으로 언급한 바와 같이 이 시스템은 운전원을 대신할 인공지능적 능력을 가지지 않도록 설계되었다.

컴퓨터가 판단할 수 있는 항목은 기본적으로 Boolean 방정식이며 발전소 Component의 상태도 판단이 가능하다. 아래 예는 이것을 잘 보여 준다. 발전소 변수, 사칙연산자, 지수, 로그 등의 일반 함수가 포함된 일반 Boolean 방정식이다. 단순화보이는 이 방정식의 판단에는 동적인 파싱기술, 동적인 변수의 관리등 상당한 알고리듬적 처리를 요한다. 이 시스템에서는 JavaCC라는 파싱기를 통하여 동적으로 자바코드를 생성하고 여기에 변수관리 루틴을 첨가하여 오류가 거의 없는 Boolean 방정식 평가기를 만들었다. 앞에서 언급했듯이 이 평가기는 3가지 상태를 발생시킨다. 컴퓨터가 이 논리를 판단하고 있을 때 식이 같이 표시되어 운전원이 컴퓨터가 하는 일을 유리속을 보듯이 훤히 알 수 있다. Component 상태 판단은 더 세밀한 처리가 필요하다. 결과만 표현하자면 RCP.Activity == "OPEN"과 같은 컴퓨터 냄새가 풍기는 표현 대신에 RCP == "OPEN"이라는 인간적 표현이 가능하도록 자료구조, 발전소 정보의 구조화에 많은 노력을 기울였다. 물론 이러한 구조화가 브라우져나 편집기에서 적절히 처리되어 절차서 작성자나 운전원에게는 일상의 언어로 투명하게 나타난다.

```
RCT.Power > 0.004
RCT.Power - TBN.Power > 1.0
RCT.Power* sin(45) > 5.0
RCP == "OPEN"
```

이렇게 처리된 항목들은 최종(상위) 논리도에 의하여 참 거짓 모름으로 이어진다. 최종 논리도 뿐만 아니라 내부에 재귀호출식으로 포함된 모든 항목들은 주기적으로, 가령 0.2초 간격으로 계속적으로 갱신된다. 그에 따라 색채가 바뀜은 두말할 나위가 없다. 이 시스템에서 "CHECK"와 "ACTION"이 절차서의 수행단위이고 그 이하의 항목이 수행단위라고 표현하지 않는 이유는, 컴퓨터나 운전원이 "CHECK"과 "ACTION"를 한 시야속에서 인식할 수 있고 동시에 평가가 가능하기 때문이다. 시시각각으로 바뀌는 이 값은 논리도에도 표시되지만 "NEXT"버튼 옆의 라디오 버튼에 최종값이 자동으로 토글된다. 왜 논리도에서 나오는 값은 3상태인데 라디오 버튼은 2가지 상태인

가? 이유는 모름상태에서는 모든 라디오버튼이 선택되지 않은 상태에 있고 "NEXT"버튼을 비활성화 시키도록 설계했기 때문이다. 이런 설계의 이유는 운전을 하여보면 쉽게 느끼겠지만 "NEXT"의 남용을 막고 운전원에게 절차를 읽고 해석할 기회를 제공한다. 흐름도에서 참과 거짓의 화살표만 존재하는 이유도 여기에 있다. 그럼 모름상태에서는 다음 단계로 전이가 불가능한가? 운전원이 라디오버튼을 토글시켜 모름 상태를 해제하면 언제든지 다음 단계로 전이가 가능하다.

"ACTION"에서는 더 강력한 제약이 걸리는데 참이 아니면 전이가 불가능하다. "ACTION"의 진입시 규칙에 의하면 항상 거짓이 되도록 논리도는 쓰여져 있으므로 정상적인 방법에서는 몇개의 항목을 실행하여야 참이 된다. 이런 설계는 운전원에게 절차의 항목을 빠뜨리지 말고 절차를 수행하라는데 목적이 있다. 만약 운전원이 라디오 버튼에서 참을 선택하고 이동한다면 이것 또한 무방하다. 이런 길을 열어 둠은 경우에 따라 필요하기 때문이고 자동차 핸들을 조금만 잘못 조작해도 충돌이 일어나지만 그래도 사람에게 핸들을 맡기는 설계원칙과 같다. 그리고 절차서 수행의 이력에는 실행한 "ACTION"의 항목들이 기록되고 있음을 상기하여 주기 바란다.

"NEXT"버튼을 선택하면 참 거짓에 따라 다음 단계로 이동한다. 그것에 따르는 후속조치는 모두 컴퓨터가 지원하여 준다. NRC의 보고서에 의하면 종이절차서 조차도 절차의 전이의 단계를 줄일 것을 당부하고 있다. 이것은 전이가 일어날 때마다 운전원에게 가하여지는 인지적 부담이 크기 때문이다. 비록 컴퓨터의 지원을 받는다 하더라도 새로 그려지는 논리도, 새로 나타나는 변수리스트, 조금 스크롤되는 흐름도등 모든 것이 인지적 부담인 것은 사실이다. 어느 절차서 시스템도 전이가 없을 수는 없겠지만 이 시스템에서는 "CHECK"과 "ACTION"의 크기를 최대한 키워 설계하여 전이의 갯수를 줄이도록 노력하였다. 만약 이 시스템에서 운전원의 판단만으로 이동한다면 라디오 버튼과 "NEXT" 버튼의 2번의 클릭으로 이동이 가능하다.

"PREVIOUS"버튼은 이전단계가 있을 경우에만 활성화되고 눌려졌을 때 이전단계로 이동한다.

"CHECK"에는 두개의 종류가 있으며 하나는 현재의 순간에만 존재하다가 "NEXT"버튼을 누름과 같이 사라지는 보편적인 것이며 또 다른 하나는 소위 계속수행단계라는 것이다. 이것은 계속 컴퓨터에 남아 관리된다. 이 둘의 사용자 인터페이스에서 차이는 거의 없도록 설계하였다. 물론 일관성 유지를 위한 설계이다. 그래도 운전원에게 흐름도의 별표와 "NEXT"버튼 옆에 나타나는 상하 화살표로 현재가 계속 수행단계임을 인식시키고 있다. 운전원은 하나의 옵션이 더 생긴 것 외에 유사한 방법으로 운전이 가능하다. 상하 화살표는 발사 시간을 변화시키는 기능이다.

계속적 수행단계의 설계는 절차서 설계의 큰 골격을 결정하는 중요한 요소이다. 만약 계속적 수행단계를 전적으로 컴퓨터에 의한 프로그램 코드로 설계하라면 컴퓨터 과학에서 자주 인용하는 인터럽트처리 방식을 사용하면 된다. 인터럽트 처리 루틴을 작성하여 이벤트 핸들러에 등록시키면 쉬운 문제이다. 그러나 운전원에게 일이 할당되는 이 시스템에서는 인터럽트 처리방식은 사람의 특성을 고려하지 않은 설계이다. 만약 이 방식이 성공적이었다면 종이 절차서조차도 계속 수행단계를 주 절차서에서 분리하여 인터럽트처리 루틴을 만들듯이 작성되어야 했을 것이다. 그러나 종이 절차서를 보면 보통단계에 끼여 군데군데 존재한다. 그래야만 운전원의 이해도가 증가했기 때문이다. 하지만 보통단계에 끼워 있는 계속단계는 인터럽트에서 복귀를 위해 "RETURN"문이 있어야되는 이것의 처리가 문제이다. 만약 인터럽트를 통하지 않고(그 단계의 초기 실행시), 이 단계를 지나갈 때 "RETURN"은 아무 의미가 없고 혼란만 야기하기 때문이다. 물론 종이 절차서에서는 일반적으로 "STEP"단위로 계속수행단계가 정의되고 처음 실행시, 그 다음 반복 실행시 운전원이 적절히 판단하여 실행하기 때문에 그렇게 문제되지 않을 수도 있다. 이 시스템에서 계속수행단계의 처리도 종이 절차서에서 운전원이 하는 행동과 일치하게 설계되어 있다. 먼저 계속 수행단계를 위한 특별 절차서를 만들지 않고 한 절차서에 같이 존재한다. 그리고 논리상으로 있어야만 될 "RETURN"문 또한 존재하지 않는다. 그러면 계속 수행의 처음 수행에서는 아무 문제가 없다. 또 계속 수행의 조건이 깨어져 계속수행으로 진입할 때도 문제되지 않는다. 그런데 문제는 계속수

행을 언제 마칠 것이지는 "RETURN"이 절차서에 명시적 명령으로 존재하지 않기 때문에 운전원이 알지 못할 수가 있다. 물론 절차의 내용에서 운전원이 종이 절차서에서 했던 방식대로 알 수 있지만, 앞에서도 언급했지만 "RETURN"문은 인간공학적이지 못하고, 절차서의 분리 또한 절차서 해석을 떨어뜨린다. 이 시스템에서는 계속 수행단계는 처음에는 절차서 주 수행으로 나타나고 계속수행단계가 깨어질 경우에는 절차서 임시수행으로 처리된다. 이때에는 "NEXT"옆의 "RETURN" 버튼이 활성화되어 운전원에게 이제 계속수행단계를 그만하고 절차서 주 수행으로 전이할 것을 종용한다. 이 유사한 설계원칙이 HTML에서도 발견되는데 이 규격에 의하면 Goto의 관계는 있어도 Goto/Return의 관계는 없고 주 HTML, 서브루틴 HTML의 개념도 없다. 물론 문제가 되기는 하지만 오히려 그런 개념을 도입한 설계보다는 심각성이 덜하다.

절차서에 사용되는 버튼들을 한곳에 집중시키고 이 윈도우에 넣은 이유는 단순하다. 일단 이 버튼들의 기능범위가 "ACTION-CHECK" 수준에서 일어나기 때문에 이 윈도우 내부에 놓여 있으며, 포인팅 디바이스의 물리적 이동거리를 최소화하기 위하여 집중시켰다. 그렇지만 잘못 선택되는 것을 막기 위해 이차원 배열대신 일차원 배열을 채택하였다. 이런 설계요건은 기본적이라 언급의 필요성이 없으며 더 중요한 것은 운전원의 시야의 움직임과 뇌에서 일어나는 인지적인 처리를 빠르게 하는 설계이다. 일반적으로 사람은 자신의 워킹메모리에 사실(현상)을 쌓아두고 수많은 규칙을 시도하여 그 중에 가장 사실에 부합하는 규칙을 통하여 판단한다. 운전원은 훈련, 교육, 경험을 통하여 규칙이 축적된다. 사실(현상)은 발전소 운전중에 계속 바뀌는 것이고 사용자 인터페이스를 잘 설계하면 사실(현상)의 빠른 입력이 가능하다. 이 시스템에서 설계된 3개의 윈도우에 동적으로 나타나는 그 정보는 운전원이 워킹메모리에 투영될 바로 그것이다.

우측의 하단은 "CHECK"과 "ACTION"에 필요한 발전소의 정보가 테이블 형식으로 나타난다. 변수명, 변수의 상태, 단위, 변수 설명이 차례로 나타난다. 변수명은 발전소/시스템/단위기기의 따라 호칭되며 이를 위해 XQL(Extended marked Query Language)를 고안하고 운전원의 이해와 절차서 커널과 통신에서 빠른 응답이 가능토록 활용하였다. 동적인 변수의 생성기법을 이용하여 시스템의 메모리 사용을 최소화하여 시스템의 안정성을 획기적으로 향상시켰고 변수의 등록과 등록해제를 통해 통신의 오브헤드를 줄이고, 정적자료와 동적자료를 동일한 방법으로 갱신하기 때문에 온라인에서 컨피그레이션(configuration)이 부분적으로 가능하다.

변수의 상태는 "OPEN"이나 "CLOSE"같은 문자대신에 감시화면에서 사용되는 똑같은 심벌을 사용하여 인지 속도를 증가시켰다. 뿐만 아니라 글자에서는 하나의 특성만 표현 가능하나 심벌에서는 다중의 특성도 표시 가능하다. 뿐만 아니라 감시화면에 나타나는 심벌의 모양은 발전소마다 틀릴 수가 있으므로 발전소 자체에서 재작성이 가능하도록 하였다. 이것은 어렵지 않은 작업으로 발전소의 자료구조, 더 정확히는 절차서 커널의 자료구조에 의존하여 대표적인 발전소의 기기, 펌프, 밸브, 온도계 등에 대해서 대표적 심벌을 만들어 모든 발전소의 기기에 연결만 시켜주면 된다. 현재 이 심벌의 편집기는 구현되어 있지 않다. 심벌을 그리는 코드를 컴파일 하여 두고 절차서 커널 디렉토리에 놓으면 운전원의 절차서 화면은 이를 자동으로 받아 사용한다. 이것에 의한 통신의 부담은 거의 없도록 설계되어 있다. 이 기능은 자바언어의 클래스 로더기능을 많이 이용하였다. 또 변수 심벌은 그 변수의 상태를 변화시킬 때 참조될 수가 있다. 가령 그 심벌을 클릭하여 제어화면을 호출하거나 혹은 감시 화면을 호출할 수가 있다. 이 기능은 변경된 COPMA에서 도입되어 운전원의 수행 상태를 평가 한 적이 있는데 매우 유용한 기능으로 부각되었다. 그래서 이 시스템에서도 또한 참조기능이 존재한다.

단위가 제공됨은 당연한 기능이다. 오히려 사용자 측면에서 볼 때, 단위의 문제점은 Boolean방정식에서 일어난다. 가령 A > 0.005 라고 할 때 A의 단위가 Kg이고 0.005의 단위가 g인 경우 Boolean 방정식 평가자는 단위까지 고려하지 않을 것이다. 물론 절차서 편집기에 단위를 데이타베이스에 문의하여 단위의 일치를 시키겠지만 절차를 쓰고 난 후에 발전소 관리자가 단위를 바꿔버렸다면 절차서 시스템은 잘못된 판단을 할 가능성이 있다. 왜냐하면 절차서 파일에 기록된 0.005

는 그대로 남아 있을 것이기 때문이다. 현재의 이 시스템은 이런 기능의 보정까지 제공하지 않는다. 절차서에서 나타나는 수식은 단순 숫자의 비교가 아닌 물리적 단위를 지닌 비교이다.

변수에는 발전소의 파라미터와 절차서에서 운전원간 정보의 교환, 다른 단계와 정보의 교환을 위해 사용되는 CPS관련 변수가 있다. 이 변수는 운전원이 값을 입력할 수 있고 한 운전원이 입력한 값을 또 다른 운전원이 활용할 수 있다. 가령 SFSC 점검자가 SFSC의 결과를 주기적으로 입력시켜 놓으면 EOP 수행자는 이를 쉽게 활용할 수 있다. 그런 긍정적 기능이 있으면서도 잘못 설계되면 CPS관련 변수가 메모리에 남아 시스템의 불안전성을 초래할 수 있다. 이 시스템은 카운터 개념을 도입하여 절차서의 적재시 혹은 절차서 종료시에 메모리를 정리한다. 또 변수에는 일반단계의 짧은 순간에만 나타나는 변수와 계속 수행단계에서와 같이 계속 남아 있으나 표시되지 않는 변수도 있다.

[그림 2]는 절차서의 호출, 다중 절차서 수행, 운전원간 절차서 대출 현황을 보여 주는 다이얼로그이다. 일반적으로 다이얼로그화면은 운전원의 요청에 의하여 잠시 동안 기능을 하다가 운전원의 요청에 의하여 사라지는 화면을 말한다. 이런 다이얼로그는 평상시에는 운전원에 의해 계속 참조되지 않으나 어떤 순간 절대적으로 필요한 운전원의 판단이 필요한 경우에 사용한다. 그럼 다중 절차서의 수행이 계속 참조되지 않으며 타 운전원의 대출 상태가 계속 참조되지 않는단 말인가? 물론 절대적인 기준은 없지만 상대적으로 흐름도나 논리도나 정보표시 화면에 비하면 덜 계속 참조되기 때문이고, 한정된 화면에 모든 것을 수용할 수는 없다. 어떠한 설계에서도 이런 절충은 있게 마련이다.

좌측 윈도우는 발전소의 모든 절차서를 분류하여 이름별로 보여주고 대출 현황도 보여준다. 대출 현황은 약 10초 정도마다 갱신된다. 이것은 도서관의 서재와, 특히 우측의 책상이라는 개념과 대비하여, 유사한 행동 특성을 보인다. 그러나 또 상이한 개념도 있다. 가령 책을 RO가 빌려갔다면 그 책은 없어져야 하지만 이 윈도우에서는 그대로 남아 있으며 대출 표시만 된다. 또 한 사람이 빌려갔다면 다른 사람은 빌려 갈 수가 없다. 물론 운전원도 서재에서 느끼는 행동특성과 똑 같은 상호작용을 원하지는 않겠지만 이 윈도우를 보고 운전원의 인지체계속에 서재와 유사하더라는 느낌을 주었다면 다행이라 여겨진다. 좌측의 한 항목을 클릭하며 그 복사본이 대출되고 우측의 윈도우에 나타난다. 이미 자신이 대출한 책이면 아무 변화가 없다.

우측 절차서에서는 책상에서 사용중이 절차서의 항목이 나타난다. 절차서명과 절차서의 처음 시작날짜와 어떻게 절차서가 열렸는지를 보여준다. 원자력 발전소의 절차서는 며칠동안, 운전원이 교체되면서 실행되는 경우도 있으므로 시작시각은 이런 의미에서 도입되었다. 또 절차서를 호출하는 방법에는 3가지의 경우가 있으며 각각의 경우에 대해서 구별되게 표시하므로써 절차서간 전이시 운전원에게 절차서 선택의 오류를 줄이고자 노력하였다. 우측항목을 클릭하면 클릭된 절차서의 흐름도가 주 윈도우에 표시된다. 운전원은 이 윈도우에서 무제한의 절차서간 전이가 가능하고 어떠한 제한도 없다.

절차서의 전이는 일반적으로 운전원에게 큰 인지적 충격을 준다. 이것은 한 절차서 안의 "STEP" 전이에 비교할 때 말이다. 그러므로 절차서 작성은 될 수 있으면 절차서간 전이가 최소화되도록 해야된다. 그러나 종이 절차서를 보더라도 완전히 없앨 수는 없다.

또 절차서의 시스템 설계자는 절차서 전이시 발생하는 부차적인 인적행위를 컴퓨터가 도와주도록 하여 운전원이 빨리 전이된 절차서의 문맥에 적응할 수 있도록 해야된다. 이 문맥이란 과거부터 남아 있는 절차서 수행의 이력과, 이력에 따라 조절되어 온 발전소의 상태이다. 절차서 수행이력은 절차서 내부에서 기록되어 있으므로 그래도 다시 표현하여 주면 되나, 그 이력에 따라 공조되어온 발전소 상태는 타 절차서 수행으로 말미암아 흐트러진 상태로, 경우에 따라서는 더 잘 공명된 상태로 나타날 수가 있다. 이 시스템에서는 절차서의 수행이력은 이전 상태 그대로 재 시작되고 전이시 걸리는 시간은 무시할 정도이다.

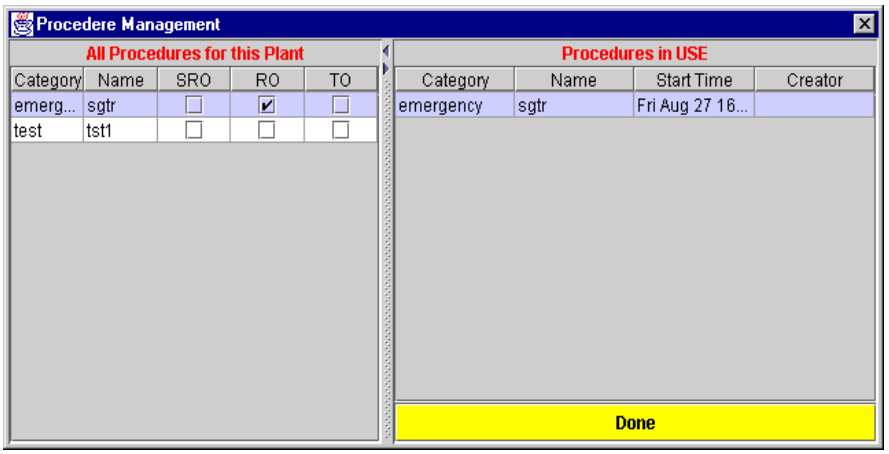


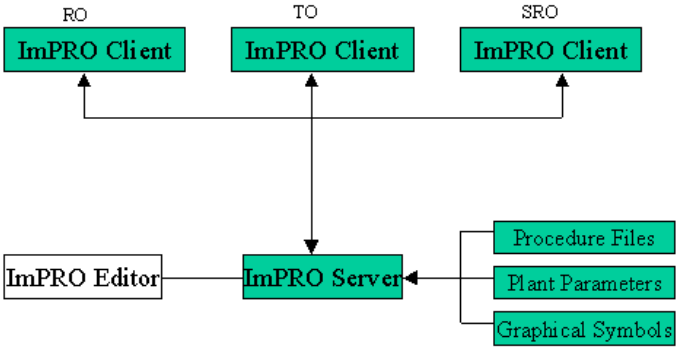
그림 2. 절차서 다이얼로그

절차서 전이와 불가분의 관계가 있는, 문서 내부에서 항해를 제약하는 특성을 크게 2개로 분류가 가능하다. 더 이해하기 쉽게 표현하자면 HTML가 같이 한 문서가 다른 문서를 어떤 제약도 없이 호출하여도 되는 경우와 컴퓨터 프로그램에서와 같이 주 절차서와 서브루틴 절차서가 있어 호출함과 호출됨의 관계에 있는 경우이다. 컴퓨터 프로그램에서는 호출된 서브루틴이 완결되기 전에 주루틴이 실행되었다가는 안된다. 이 두개의 특성이 어느 것이 좋으냐는 의미가 없다. 현재에 두개의 특성은 각각의 분야에서 괄목할 만한 성공을 던져주기 있기 때문이다. 이 차이는 항해에 결국 인간이 개입되었느냐, CPU만에 의해 일어나느냐에 따라 판가름된다. 절차서에서도 이 둘 중에 하나를 선택해야된다. 간혹 어떤 절차서 시스템에 절차서간의 관계를 주종의 개념을 도입하고 있으나 이 시스템의 절차서간에는 HTML 문서에서와 같이 주종관계가 없다. 한 절차서가 다른 절차서를 호출 할 수 있고, 역으로 호출된 절차서가 호출한 절차서를 역으로 호출하여도, 호출된 절차서를 다 수행하지 않고 되돌아와도 시스템은 어떠한 제약도 가하지 않는다. 운전원이 종이에서 했던 그것이다. 자동화는 이것을 제약하지 않는다.

절차서 수행이 끝났을 때 절차서 반납은 어떻게 일어나는가? 절차서 반납은 이 다이얼로그에서 제공되지 않으며 주윈도우의 메뉴를 통하여 일어난다. 종료하기 전에 정보가 풍부한 곳에서 충분히 사태를 인식하고 반납하라는 의도이다. 물론 가능한 설계이다. 비대칭성의 설계이다. 이 시스템에서는 3 곳에서 비대칭성을 발견 할 수가 있다. 사람이 대칭의 개념으로 얻는 하나의 이점은 반만 기억하여도 전체를 유추할 수 있다는 것이다. 이 시스템에서 발견되는 3곳의 비대칭은 다음과 같다. 왜 절차를 대출하는 그 화면에서 절차를 반납하지 않을까? 절차서 임시 수행의 개시와 임시 수행의 종료에 다른 방법으로 일어나는가? 진출하는 Proxy "STEP"의 모양과 진입하는 Proxy "STEP"의 모양이 틀린가?

이 시스템은 서브와 클라이언트의 개념으로 구성되어 있다. 적어도 3사람의 사용자가 있으며 절차서에 필요한 자원, 절차서 파일, 그래픽 심벌, 발전소자료를 중앙에서 관리하면 자료의 일관성, 보수유지성, 절차서의 편집에서 유리하고 운전원간 협력이 유리하기 때문이다. 이 시스템 또한 다른 시스템과 통합되어 같은 네트웍 환경에 놓이게 될 텐에 서브 클라이언트 구조가 통신망에 교통량 폭주를 예방할 수가 있다. 또 시스템을 분리함으로써 경량화 시킬 수 있으며 설계, 구현, V&V과정에서 해석이 가능하도록 도와준다. 설계나 구현을 하면서 느끼는 것이지만 이해의 난이도는 이해대상의 갯수의 제곱에 가깝다는 것이다. 결국 이해라는 것은 이해 대상의 각각을 이해하고 대상 사이 관계를 규명하는 것인데 가장 단순한 관계가 대상사이의 행렬 관계로 표현되기 때문이다. 이런 구조 속에 부정적으로 작용하는 하나의 요소는 통신망의 안정성이다. 그러나 이것은 수많은 모델의 개선과 수많은 V&V과정을 통과하여 입증된 기술이 많으므로 절차서 시스템에서

이것 때문에 깊이 우려할 항목은 아니다. 각각의 클라이언트는 자신이 RO혹은 TO임을 표시하는 자료와 색채의 선호도 폰트의 선호도를 표시한 컨피그레이션(configuration) 파일 하나와 실행파일 하나만을 가지고 있다. 필요한 자료가 있으면 서브에게 문의하고 자료를 전달받는다. 절차서 편집기는 아직 구현되지 않았다. 프로젝트의 성공 비결에는 가장 핵심이 되는 모듈부터 개발해야 성공 여부를 알 수 있다. 편집기가 매우 중요한 역할을 하나 그것도 이 시스템의 사용성 보다 중요할 수는 없다. 서브는 개발되어 있으며 자신의 사용자 화면을 가지고 있다. 이 사용자 화면은 자원을 관리하거나 시스템의 건전성을 검사하는데 사용된다.



Architecture of ImPRO

3. 결 론

규격화 된 절차서의 기술언어로부터 생성되는 일관된 사용자 인터페이스를 주는 전산화 절차서를 개발하였다. 흔히 전산화 과정에서 저지르기 쉬운 과잉의 자동화를 조심하여 어떤 경우에도 운전원의 판단이 우선 되도록 사용자 환경을 설계했으며, 또 컴퓨터가 지원할 수 있는 역할도 명확히 하고 컴퓨터의 판단 근거가 운전원에게 결코 숨겨지지 않도록 하였다.

흐름도 형식으로 절차서가 표시되고 상세 내용은 수목형 논리도로 표시되면 관련 정보는 테이블 형식으로 나타난다. 단계간 전이를 최소화하기 위해 최소 실행단위의 크기를 최대한 키웠으며 또 큰 크기로 인하여 상태의 논리가 애매하지 않도록 정교하게 설계하였다.

참고문헌

[1] Extensible Markup Language(XML), W3C Working Draft, 1997. 8

[2] Implementation procedure Writing Guideline for I.A.H Emergency Operating Procedure, EDF (Rev.A 94.5)

[3] M.H. Lipner and S.P. Kerch, Operational Benefits of an Advanced Computerized Procedure System, IEEE, 1995

[4] Learned From Special Inspection Program for Emergency Operating Procedures, NREG-1358, 1988. 10

[5] COPMA-II User's Manual, 1996.8