□ 기술해설 □

I-CASE의 도구 통합과 방법

강원대학교 양 해 술*

목 차

1. 서 론

최근, 소프트웨어 개발환경이 급변함에 따라 사용자의 요구사항이 다양해졌으며, 기존에 개발된 소프트웨어를 빈번히 수정해야 하는 번거로움과 이에 수반되는 시간과 비용이 점점 증가하고 있다. 따라서, 고품질/고신뢰성의 소프트웨어를 개발하면서도 상대적으로 비용을 절감할 수 있는 방법 즉, 비용 대 효과가 높은 방법을 개발하고자 하는 노력이 시도되고 있다[9,10].

이와 같은 시대적인 조류에 부응하여 1980년대 초에 개발된 CASE는 노동 집약적인 수준에 머물러 왔던 소프트웨어 개발 방식을 부분적으로 자동화하여 소프트웨어의 개발을 효율적으로 지원하여 왔다. 이러한 CASE의 기본 개념은 하드웨어나 시스템 인터페이스를 모델링하여 적용하기 때문에 CASE는 Computer Aided Systems Engineering의 의미로 사용되기도 한다.

그리고, CASE는 퍼스널 컴퓨터와 웍스테이션의 대량 보급에 힘입어 CASE 도구 사용을 통한 소프트웨어 개발 과정의 자동화와 생산성을 향상시킬 수 있다는 확신하에 소프트웨어 개발 업체들로부터 많은 호응을 얻고 있다. 한 예로 CASE 도구의 효시라고 할 수 있는 Excelerator의 사용자들은 요구 분석 및 설계 단계에서 30~40%의 생산성 향상을 얻었고, 품질 또한 크게 개선되었다고 보고하고 있으며, CASE 시장의 규모가 매년 70%씩 성장하고 있는 점을 감안하면 CASE에 대한 연구의 중요성이 점차 증가하고 있음을 알 수 있다[8].

그러나, 대부분의 CASE 도구들은 생명주기의 일부분만 지원하고 있으므로 생명주기의 전단계를 자동화하여 소프트웨어 개발을 지원하는 자동화 도구의 개발이 절실히 필요해지고 있다. 현재, CASE는 광범위하게 연구되고 있으며 가장 주목을 받고 있는 연구분야로는 첫번째로 도구의 통합화이고, 두번째는 도구의 통합화를 위한 프레임워크, 그리고 세번째로는 CASE 도구의 분산개발 환경이 있다. 본 연구에서는 첫번째와 두번째의 연구분야를 고려한 I-CASE의 도구 통합과 방법에 대한 4가지 관점에 대하여 살펴보

* 종신회원

고자 한다.

본 연구의 구성은 현재의 CASE 도구에 대한 문제점을 고찰하고, 이 문제점을 개선하기 위한 방법으로 I-CASE의 필요성과 특징을 기술한다. 그리고, 본 연구의 핵심인 I-CASE 도구 통합과 방법에 대하여 고찰해 보고, 마지막으로 결론 및 향후 연구방향에 대하여 기술한다.

2. CASE의 정의와 문제점

2.1 CASE의 정의

CASE는 소프트웨어 개발 과정을 지원하기 위한 자동화된 도구들을 의미하는 것으로 "소프트웨어 개발의 자동화"라고도 한다. CASE가 등장하기 이전의 전통적인 소프트웨어 개발 기술은 "도구(Tool)"와 "방법론(Methodology)"의 두가지 범주로 분류된다. "도구"의 범주에는 3세대, 4세대 언어들이 포함되는데 대부분의 도구들은 소프트웨어 개발 단계중에서 프로그래밍 단계에 보다 많은 중점을 두고 있다. 이러한 도구들의 특징은 소프트웨어 개발 집단의 전체적인 지원보다는 개별적인 개발자들을 지원하는데 중점을 두고 있으며 주로 소형 시스템에서 많이 이용되었다. 그리고 "방법론"의 범주에는 구조적 분석과 설계, 구조적 프로그래밍 등의 수작업에 의한 소프트웨어 개발을 위한 실무적인 관점에서의 일관성 있는 기법들의 체계적인 묶음이라고 할 수 있다[10].

현재의 CASE 도구에서는 지원 단계를 소프트웨어 생명주기 전과정으로 확대하여 각각의 부분적인 단계를 지원하고 있다. 이러한 CASE 도구는 기존의 구조적 방법론들이 수작업으로 개발함으로써 발생하는 시간적/물적 노력을 절감하기 위해 구조적 분석을 지원하는 다이아그램의 작성, 문서화 등을 자동화하여 구조적 방법론을 실용적으로 이용할 수 있도록 한다[7].

그러나, 이와 같은 CASE 도구는 현재 통합형 도구로서 지원되지 않아 소프트웨어 개발의 완전 자동화가 가능하지 않으므로 많은 연구와 개발을 통해 그림 1과 같이 소프트웨어 생명주기 전단계를 자동화할 수 있는 도구의 개발이 필요하다

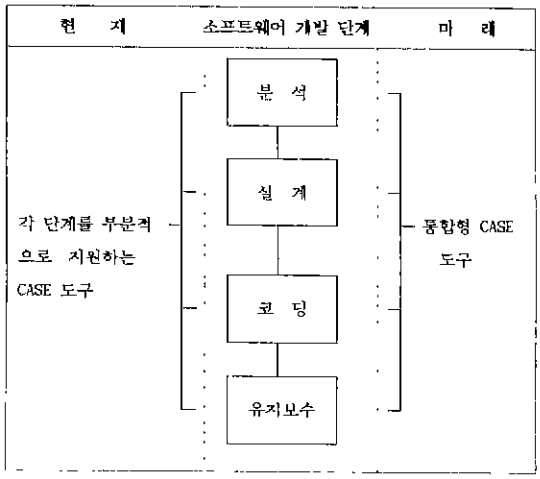

그림 1 현재의 CASE와 미래의 통합형 CASE에 대한 비교

고 할 수 있다.

2.2 CASE의 구성요소

CASE의 구성요소는 프로젝트 관리를 포함하여 광범위한 소프트웨어 개발과 유지보수 작업을 지원하고 있다. 이러한 CASE의 구성요소는 시스템 모델, 중앙정보저장소 및 개발 기법 등의 유기적인 결합을 통해 정보 시스템 개발의 신뢰도 제고에 큰 역할을 담당하고 있다[5,6].

2.2.1 다이아그램 작성 기능

사용자나 시스템 설계자가 시스템의 범위와 내용을 다이아그램 형태로 표현함으로써 흐름을 이해하고 개선할 수 있도록 한다.

2.2.2 정보관리 기능

모든 시스템 정보 및 결과물에 관한 정보 즉, 데이타 요소, 데이타의 구조, 프로세스의 논리 구조, 요구사항, 설계 내용 등과 같은 정보를 보관 및 검색하고 수정할 수 있도록 한다.

2.2.3 프로토타입 기능

사용자에게 시스템의 구체적인 모델을 신속하고 편리하게 제시하여 사용자가 자신의 요구사항을 확인할 수 있도록 한다.

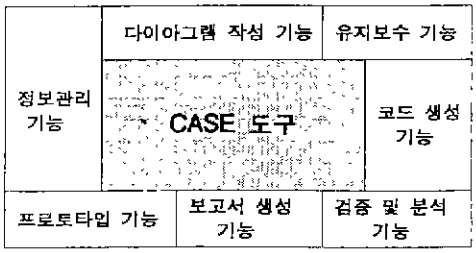

그림 2 CASE의 구성요소

2.2.4 보고서 생성 기능

요구되는 기술적인 문서 또는 사용자 문서 등을 생성한다.

2.2.5 검증 및 분석 기능

시스템 명세서의 일관성, 완전성, 정확성 등을 분석/검증한다.

2.2.6 코드 생성 기능

시스템 명세서로부터 실행 가능한 코드를 생성한다.

2.2.7 유지보수 기능

기존 시스템을 새로이 문서화하거나 분석, 재구성 또는 역공학을 가능하게 한다.

2.3 문제점

현재 개발된 대부분의 CASE 도구들은 소프트웨어의 생명주기 각 단계의 일부분만을 지원하고 있기 때문에 소프트웨어 개발의 완전 자동화를 이룰 수 없다. 이와 같은 도구는 일관성 있는 작업을 할 수 없으며, 개발작업의 중복성과 불필요한 시간과 비용이 소요된다. 또한, 이러한 문제점으로 인해 인터페이스의 다양화, 데이타의 다양화, 제어의 다양화, 개발 도구와 관리 도구의 불일치 등의 문제점이 발생한다. 이러한 문제점의 관점에서 통합형 CASE 도구에서 요구되는 특성들에 대하여 살펴보면 다음과 같이 정의할 수 있다[1,2,3].

2.3.1 사용자 인터페이스 통합(User Interface Integration)

통합형 도구를 개발하기 위해서는 각각의 도구에서 지원되는 사용자 인터페이스를 통일시켜야 한다. 즉, 통합형 CASE 도구의 각 단계를 지원하는 도구들에 대하여 명령어, 메뉴 등의 통일과 입출력 양식이나 윈도우 등의 사용이 통일되어야 한다.

2.3.2 데이타 통합(Data Integration)

현재의 CASE 도구들을 통합형 CASE 도구로 구성함에 있어 데이타 공유는 가장 핵심적인 문제점 중 하나이다. 이것은 생명주기 각 단계를 지원하는 도구들간의 데이타를 중앙정보저장소에 일괄적으로 관리 유지함으로써 데이타의 무모순성과 일관성을 유지할 수 있다.

2.3.3 제어 통합(Control Integration)

통합형 CASE 도구를 제어하기 위해서는 각각의 도구들에 대한 제어를 통합해야 한다. 즉, 프로세스의 동작, 실행, 종료의 관리, 그리고 도구들간의 상호동작에 대한 기능을 관리하기 위해서는 반드시 제어의 통합화가 이루어져야 한다.

2.3.4 개발 및 관리 도구 통합(Development and Management Tools Integration)

기존의 CASE 도구들은 소프트웨어 개발 도구와 관리 도구가 분리되어 있기 때문에 효율적인 프로젝트의 관리가 이루어질 수 없다. 개발 도구와 관리 도구가 하나의 환경으로 통합된다면 소프트웨어 개발에 있어서 프로젝트를 효율적으로 관리할 수 있게 된다. 즉, 개발 도구로 생성된 중간 생산물을 프로젝트 관리 도구인 품질관리 도구를 이용하여 생성된 중간 생산물의 품질을 체크할 수 있도록 한다.

3. I-CASE의 특징

이상적인 CASE 도구는 2절에서 설명한 모든 구성요소를 포함해야 한다. 그러나 현재 대부분의 CASE 도구들은 일반적으로 구성요소 중 몇 가지만을 제공하고 있으며 구성요소간의 연결도 강력하지 못하다. 이러한 약한 연결은 시스템

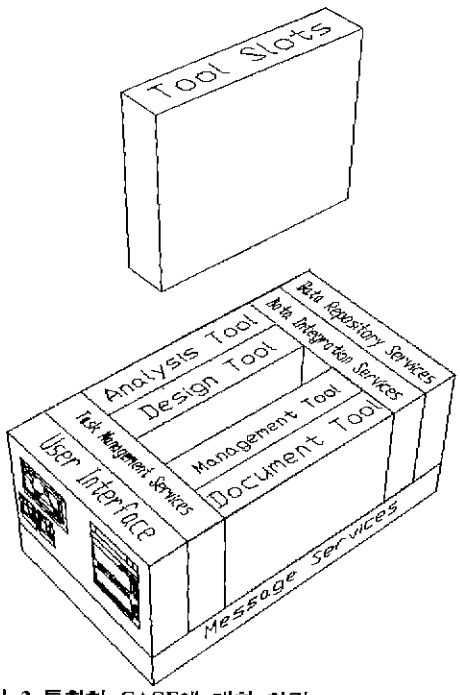

그림 3 통합형 CASE에 대한 환경

개발 활동에서 생명주기의 한 단계로부터 다음 단계로의 진행이 자연스럽게 처리되지 못한다. 특히, CASE 제품을 개발하는 많은 연구자들이 의욕적으로 도전하고 있는 분야가 바로 CASE 도구와 코드 생성기간의 강력한 연결을 제공하는 통합된 구조를 구현하는 것이다. 따라서 가장 이상적인 통합형 CASE에 대한 환경은 그림 3과 같이 구성할 수 있으며, 또한 통합형 CASE의 플랫폼을 그림 4와 같이 구성할 수 있다. 이러한 통합형 CASE에 대한 특징을 살펴보면 다음과 같다.

- 계획, 분석, 설계, 조작의 각 작업에 대하여 복수개의 다이어그램을 제공하는 소프트웨어 워크벤치(Workbench)가 있다. 이 워크벤치는 완전히 통합화되어야 하며, 어떤 워크벤치로부터도 직접 액세스할 수 있어야 한다.
- 복수개의 워크벤치로 작성된 정보가 중앙정보저장소에 저장되어 통합화된 형태로 축적되어야 한다.
- 코드 생성기는 대상 시스템의 운영체제나 데이타 사전을 포함한 데이타베이스 관리시스템(DBMS)의 기능을 제공하여야 한다.

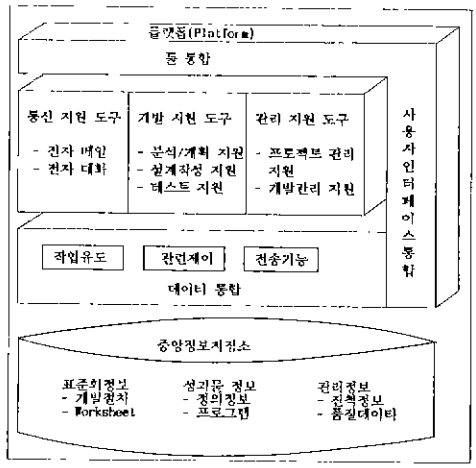

그림 4 통합형 CASE의 플랫폼

- 생성기로부터 생성되는 출력정보는 최적화 도구에 전달되어 최적의 실행 효율을 얻도록 변환된다.
- 통합형 CASE 도구는 소프트웨어 생명주기 전단계의 개발작업을 통합화된 형태로 지원한다.
- 통합형 CASE 도구는 단순히 소프트웨어 공학뿐만 아니라 정보공학도 지원할 수 있도록 설계되어야 한다.
- 충분히 정규화된 엔터티 관련 모델이나 데이타 모델을 사용하여 시스템 설계를 할 수 있어야 한다.
- 충분히 설계용 워크벤치는 화면 레이아웃, 대화절차, 장표 레이아웃 등의 설계용구로부터 구성되지만 이 정보는 중앙정보저장소내에 통합되어야 한다.
- 문서화는 모두 자동적으로 생성되어야 한다.
- 통합형 CASE 도구는 매우 복잡한 시스템을 소규모팀이 분담하여 개발할 수 있는 것처럼 복잡한 시스템을 분할하여 작업할 수 있도록 지원되어야 한다. 그리고, 분할된 각 시스템간의 인터페이스는 중앙정보저장소에서 정확히 관리되어야 한다.

4. I-CASE의 통합과 방법

CASE 도구의 궁극적인 주목적은 소프트웨어의 개발을 효율적으로 지원하기 위한 환경과 자

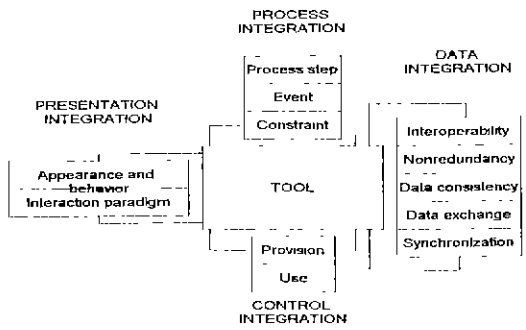

그림 5 4가지 관점으로 통합된 CASE 도구

동화된 도구를 구성하는 것이다. 현재 개발되어 부분적인 단계를 지원하는 CASE 도구들을 통합하여 생명주기 전과정을 자동화할 수 있는 통합형 CASE를 구성한다면 보다 소프트웨어의 개발이 용이하고 고품질의 소프트웨어를 개발할 수 있다고 본다[2,4].

여기에서 통합(Integration)이란 현재 소프트웨어 개발 각 단계를 지원하는 CASE 도구들을 결합시켜 하나의 통합된 도구로 개발하는 것이다. 그러나 이러한 도구들은 그들만의 고유한 특성을 가지고 있기 때문에 통합에 많은 어려움이 따른다. 따라서, 각 도구들에 대한 구성요소와 특성들을 면밀히 조사하고 관찰하여 통합에 필요한 기능들과 방법들을 정리하여야 한다.

본 연구에서는 통합형 CASE 도구를 개발하기 위해 생명주기 각 단계를 부분적으로 지원하는 현재의 CASE 도구들에 대한 통합을 사용자 인터페이스 통합, 데이타 통합, 제어 통합, 프로세스 통합의 4가지 관점으로 분류하여 정의하였다. 그림 5는 통합형 CASE를 4종류의 관점에서 통합한 도구의 관계 다이어그램을 나타낸 것이다.

4가지 관점 중 첫번째인 사용자 인터페이스 통합은 사용자의 노력을 최소화하고 풍부한 개발환경을 제공하므로써 사용자의 능률을 높이기 위한 것이다. 두번째의 데이타 통합은 개발환경에서 발생하는 모든 정보를 일괄적으로 관리하기 위한 것이다. 그리고, 세번째의 제어 통합은 유연성 있는 통합 개발도구와 통합 환경을 제공하기 위한 것이다. 마지막 프로세스 통합은 각 도구들에 대한 상호동작을 효율적으로 지원하기 위한 것이다. 이와 같이 본 연구에서는 통합형 CASE 도구를 개발하기 위해 현재 개발되어 있는 다양한 CASE 도구들을 통합할 것인지와 동일 기능에 대한 도구의 밀접된 특성이나 각 도구들에 대한 통합을 어떻게 할 것인지에 대하여 다음과 같은 4종류의 관점에서 고찰한다.

4.1 사용자 인터페이스 통합(User Interface Integration)

통합형 도구에서 지원되는 각 도구들의 사용자 인터페이스는 사용자의 개발환경을 최대한으로 지원하기 위해서 통일된 환경과 통합된 패러다임을 지원하여야 한다. 이러한 통합형 CASE 도구의 사용자 인터페이스에 대한 통합은 다음과 같은 관점으로 분류하여 살펴 볼 수 있다.

4.1.1 Appearance and Behavior

현재 개발되어 있는 CASE 도구들은 각기 다른 사용자 인터페이스 개발환경을 제공하고 있기 때문에, 현재의 CASE 도구들을 이용하여 통합형 CASE 도구를 개발할 때에는 각 도구들의 특징을 최대한 활용할 수 있는 방법을 이용하여 통일된 조작방법과 표현방법으로 사용자 인터페이스를 제공할 수 있도록 하여야 한다.

Appearance는 입출력 양식이나, 명령어 메뉴 등과 같이 스크린에 표현되는 메뉴나 양식들이 통합형 CASE 도구를 지원하는 각 도구에서 동일하게 제공되어야 한다. 그리고, Behavior도 마우스의 클릭, 윈도우의 조작방법 등과 같이 동작처리가 통합형 CASE 도구의 각 도구에서 동일하게 제공되어야 한다. 이와 같이 Appearance와 Behavior가 동일하게 통합됨으로써 사용자의 인식력을 줄이고 개발환경의 효율화를 기대할 수 있다.

4.1.2 Interaction Paradigm

통합형 CASE 도구에서 제공되는 사용자 인터페이스가 상호작용(Interaction) 패러다임을 제공하기 위해서는 현재의 CASE 도구들이 어느 정도의 은유적(Metaphor), 관념적 모델링을 제공하는지를 알아야 한다. 만약 현재의 CASE 도구들이 동일한 형태의 은유적, 관념적 모델을

사용하고 있다면 통합형 CASE에서의 Interaction 패러다임 통합은 잘 이루어질 수 있다.

그리고, 은유적 모델링은 단일 은유와 다중 은유를 적절히 균형있게 이용하는 것이 매우 중요하다. 왜냐하면 단일 은유로 모델링할 수 없을 때 다중 은유를 이용하여 모델링하여야 하기 때문이다. 그러나 다중 은유가 많을수록 은유에 대한 모호성이 커짐으로 단일 은유와 다중 은유의 비율은 적절하게 배분되어야 한다.

Interaction 패러다임에 대한 예를 들면, 데이타베이스를 액세스하고 브라우징하는 도구는 화일 캐비넷, 서랍, 서류 등과 같은 관리 시스템의 은유를 이용할 수 있다. 이러한 은유는 캐비넷과 서랍, 서랍과 서류간의 관계에서 Interaction 패러다임의 제한 관계를 살펴볼 수 있다.

4.2 데이타 통합(Data Integration)

데이타 통합의 목적은 통합형 CASE 도구에서 각 도구들의 동작과 처리를 위해 일관성있는 정보를 제공하기 위한 것이다. 각 기능들을 지원하는 도구들이 처리하는 데이타가 일관성 없이 서로 다르게 저장되는 경우는 많은 오버헤드가 소요되고, 또한 데이타의 일관성 그리고 통합된 도구로서의 기능을 올바르게 제공할 수 없다. 데이타 통합을 데이타의 관리와 표현의 측면에서 다음과 같은 다섯가지의 특징으로 살펴볼 수 있다.

4.2.1 Interoperability

이 개발환경은 사용자에게 제공되는 각각의 도구들이 실질적으로 서로 다른 데이타 모델로 처리될지라도 사용자 인터페이스로 공통의 스키마(Schema)를 제공하기 때문에 사용자는 데이타의 일관성을 고려하지 않고도 자유롭게 데이타를 처리할 수 있다. 그러나 통합된 개발환경을 제공하는 입장에서는 도구가 잘 통합되지 못한 것처럼 보인다. 왜냐하면 다른 데이타 모델이나 작업을 처리하기 위해 하나의 도구에서 생성된 데이타를 다른 도구에서 처리할 수 있도록 변환하여야 하기 때문이다. 예를 들면, 설계 도구에서 생성된 정형화된 데이타는 프로그래밍 도구에 처리되기 위해서는 다시 정형화된 데이타로 변환(Conversion) 되어야 하기 때문이다.

따라서, 통합형 도구는 데이타에 대하여 공통된 관점(Viewpoint)을 가질때 각각의 CASE 도구의 통합은 용이해진다. 이와 같이 공통된 관점으로 데이타를 처리하기 위해서는 개발환경의 구조를 공통으로 내부구조(Internal structure)로 구성함으로써 각 도구들에서의 관점들을 좁힐 수 있고 Interoperability의 통합이 용이해진다.

4.2.2 Nonredundancy

통합형 CASE 도구의 각 도구에서 생성되는 데이타가 독립적으로 저장되거나 관리되지 않기 때문에 데이타의 중복이나 많은 저장장치의 낭비가 발생하지 않도록 처리하는 특성이다. 정보저장소에 저장되는 대부분의 정보는 일관성에 대한 유지보수가 어렵기 때문에 Nonredundancy의 통합이 매우 어렵다. Nonredundancy는 통합된 각 도구들에서 생성되는 데이타의 일관성을 유지할 수 없기 때문에 데이타의 중복을 최소화한 정보만을 유지하기 위한 특성이다.

4.2.3 Data Consistency

데이타의 일관성은 통합형 CASE의 각 개발단계를 지원하는 도구들에서 생성된 데이타를 정보저장소내에 일관성있게 저장하여 일관성있는 데이타를 제공할 수 있도록 하는 특성이다. 즉, 동일한 의미의 데이타가 각 도구마다 다르게 적용되면 도구들간의 통합이 불완전하기 때문에 저장되는 데이타를 빈번히 관찰하고 검색하여 데이타의 일관성을 최대한 유지시키는 특성이다.

4.2.4 Data Exchange

데이타 교환(Data Exchange)은 Interoperability의 특성과 유사하다. 그러나 데이타 교환 특성은 지속성없는(Nonpersistent) 데이타도 이용한다는 점에서 Interoperability와 차이가 있다. 그리고, Data exchange와 Interoperability를 다른 관점에서 구별하면 Data exchange에 대한 메카니즘은 Interoperability를 지원하는 메카니즘과 다르나, 이러한 Data Exchange는 동일한 정보저장소에 데이타를 저장함으로써 Data Ex-

change의 특성을 용이하게 처리할 수 있다.

4.2.5 Synchronization

일반적으로 도구를 실행하기 위해서는 일관성있는 데이타와 일관성없는 데이타 모두를 처리하게 된다. 그러나 통합형 CASE 도구에서는 형식적으로는 일관성있는 데이타만을 처리한다. 즉, 통합형 CASE 도구는 각 개발 단계를 지원하는 도구들에서 생성되는 일관성없는 데이타를 일관성있는 데이타로 요구한다.

이와 같이 통합형 CASE 도구에서는 일관성있는 데이타를 요구하기 때문에 통합형 도구중 어떤 하나의 도구에서 데이타 교환이나 처리에 의해 생성되는 데이타를 다른 도구에서 동시에 처리하기 위해 변환되는 기능을 Synchronization이라 한다. 예를 들면, 어떤 소스코드를 브라우저와 디버거를 이용하여 처리하는 경우 각 도구는 소스코드에 대하여 동일한 관점을 가지나 각 도구들에서 소스코드는 처리될 수 있는 정보를 변환되어 처리되는 것이다.

이 Synchronization의 특성은 데이타 일관성의 특성과 매우 유사하나 데이타의 일관성을 지원하는 메카니즘과 Synchronization을 지원하는 메카니즘이 다르기 때문에 구별된다.

4.3 제어 통합(Control Integration)

유연성있는 기능 결합을 지원하기 위해 통합형 CASE 도구는 기능들이 분할되어야 한다. 그리고, 통합형 도구에서 제공되는 모든 기능들은 서로 다른 도구에서도 액세스될 수 있어야 한다. 그러나 통합형 도구 자체를 어떤 도구에서 어떤 기능을 제공하는지 알 수 없기 때문에 기능 분할된 도구들을 상호 동작할 수 있도록 제어해야 한다. 예를 들면 각 도구들의 처리를 위해 데이타를 요구하거나, 각 도구에서 데이타나 데이타의 레퍼런스를 전달하는 기능을 처리하기 위해서 제어(Control)가 통합되어야 한다. 이와 같은 것을 고려한 제어 통합은 데이타 통합을 보완하는 작용을 한다. 제어 통합에 대한 2가지 특성에 대하여 살펴보면 다음과 같다.

4.3.1 Provision

통합형 도구에서 하나의 도구에서 제공되는 기능을 다른 도구에서 사용할 수 있도록 기능을 제공하는 특성이다. 즉, 통합형 도구의 A 도구에서 제공하는 기능들을 다른 도구들에서 A 도구의 기능을 요구하거나 이용할 때 A 도구의 기능을 제공할 수 있도록 하는 것이다. 예를 들면, 프로젝트 관리 도구에서 텍스트 작업을 할 수 있는 에디팅 서비스를 요구할 경우 에디팅 도구의 기능을 참조 기능한 한도까지 제공하는 것이다.

4.3.2 Use

통합형 도구내에서 모든 도구들에서 제공되는 기능을 어떤 하나의 도구에서 이용할 수 있도록 제공되는 특성이다. 어떤 하나의 도구가 동일 환경내에서 다른 도구들에 의해 제공되는 기능을 이용할 수 있다면 그 도구는 Use Integration만으로도 용이하게 통합될 수 있다. 유용한 Use Integration을 이루기 위해서는 모듈화된 도구가 필요하지만, 비슷한 서비스를 제공하거나 대체 서비스에 대해 고려되지 않은 도구는 유용한 Use Integration을 이루지 못한다. 그리고, 모듈화된 도구로 통합된다면 유사한 도구가 만들어질 수 있으므로 실행 환경내에서 어떤 서비스를 제공하는지를 예상하여 그 도구에 대한 기능을 통합할 수 있는 수단을 제공하거나 통합형 도구내에서 다른 도구에서 제공되는 호환성 있는 기능을 제공할 수 있는 도구 제공 서비스(Tool-Provided service)로 대체할 수 있는 방법을 제공해야 한다.

4.4 프로세스 통합(process Integration)

통합형 CASE 도구를 개발하기 위해서 예상되는 각 도구들의 프로세스 집합들을 통합하여 구현하는 데에는 많은 어려움이 있다. 이러한 통합형 CASE 도구를 개발할 때 고려해야 할 문제점으로 각 도구들에 관련된 동일의 Process step의 통합을 고려하여야 한다. 예를 들면, 설계를 지원하기 위한 관련 도구가 그래픽 설계 도구와 설계 분석 도구로 구성된다면, 이중 설계

분석 도구의 Process step이 어셈블러와 디버거를 필요로하기 때문에 도구간의 상호 관련성이 없다. 그러나, 이 두개의 도구는 설계를 지원하는 도구를 지원하기 때문에 서로 통합된 하나의 Process로 구성되어야 한다. 이러한 프로세스 통합을 위해 다음과 같은 3가지 관점으로 분류하여 설명한다.

4.4.1 Process step

통합형 CASE 도구에서 Process step의 성능을 지원하기 위해 각각의 도구들을 통합하여 제공하기 위한 특성이다. 통합형 CASE 도구에서의 Process step 기능은 여러 도구에서 처리될 수 있도록 분할되어야 한다. 그리고 분할된 각 도구들이 그 기능들을 수행할 때에 반드시 처리되어야 할 전제조건(Precondition)들이 할당된다. 이러한 전제조건은 다른 도구들에 의해서 처리된 결과에 의해서 만족되어진다. 예를 들면, Compile-and-debugger라는 Process step이 C++ 전처리기, C 컴파일러, 디버거를 이용한다고 가정하면, C++ 전처리기는 C++ 프로그램을 C 프로그램으로 생성하고, 생성된 C 프로그램은 C 컴파일러에 의해서 컴파일된 C를 생성한다. 그리고 소스레벨을 지원하는 디버거는 C 컴파일러에 의해 생성된 정보를 이용하여 처리한다. 그러나 Debugger는 C 프로그램이 C++ 전처리기에 의해 생성된 것임을 알지 못하기 때문에 C++ 컴파일 체인과 디버거는 Process step 관점에서의 통합은 어렵다. 왜냐하면, C++ 컴파일 체인과 디버거는 C++ 프로그램의 소스레벨 디버깅을 처리할 수 있는 응집력있는 Compile-and-debugger step으로 분해하지 못하기 때문이다.

4.4.2 Event

특별한 프로세스를 지원하기 위해 필요로하는 Event들이 통합된 CASE 도구에서 제공할 수 있도록 지원하는 특성이다. 이러한 Event에는 두가지 관점이 있다. 첫번째는 어떤 도구의 전제조건은 다른 도구들에 의해 생성된 Event들을 반영해야 하고, 두번째로 어떤 도구들의 전제조건을 만족시킬 수 있는 Event들을 생성하는 것이다. 만일 도구들간에 지속적으로 Event notification들을 생성하고 처리한다면 도구들간의 Event 통합은 용이하게 이루어진다고 할 수 있다.

예를 들면, 모듈 단위 테스트를 계획하고 스케줄하려고 한다면 프로세스는 다음과 같은 것을 요구한다.

- 완전하게 개발된 모듈(Unix Lint tool에서 동작)
- 형상관리 시스템에서 체크된 모듈
- 테스트 담당자가 이용

즉, 세가지 Event들 모두에 대한 통지(Notification)는 단위 테스트 스케쥴링에서의 전제조건에 해당된다. 이 경우 Lint tool은 모듈이 Lint를 완벽하게 통과하였다는 것을 나타내는 Event를 생성할 수 있으며, 형상관리 도구는 모듈이 시스템에서 점검되었다는 것을 나타내는 Event를 생성할 수 있고, 자원 효율성(Resource Availability) 도구는 테스트 담당자가 이용할 수 있다는 것을 알리는 Event를 생성한다. 이러한 도구들이 단위 테스트 스케쥴링 도구의 전제조건을 만족시키기 위해 필요한 Event들을 생성해내기 때문에 이 도구들은 단위 테스트 스케쥴링 도구로 잘 통합될 수 있다.

Event는 지속성있는 객체 관리 시스템의 일부분인 데이타 트리거(Data Trigger) 메카니즘과 몇가지 유사점을 가지고 있으나, Event들이 지속성 또는 비지속성 데이타에 대해 관계없이 알려 준다는 것이 차이점이다.

4.4.3 Constraint

통합형 CASE 도구에서 관련 도구들이 Constraint를 실행하기 위해 협력체계가 용이하도록 제공하는 특성이다. 이러한 Constraint를 실행하기 위한 2가지의 측면이 고려된다. 첫번째는 하나의 도구의 수락(Permitted) 기능이 다른 도구의 기능에 의해 Constraint되는 것이다. 두번째는 도구의 기능이 다른 도구의 수락 기능에 의해서 Constraint되는 경우다. Constraint integration은 Data consistency integration과 유사하게 보이나, Data consistency integration은 데이타 값에 대한 Constraint에 중점을 두고, Constraints integration은 프로세스 상태에 대한 Constraint에 중점을 두고 있는 것이 차이점이다. 이 Const-

raint에 대한 예를 살펴보면, Process constraint 관점에서는 동일한 사람이 모듈의 코드나 테스트 양쪽 모두를 처리할 수 없다고 가정할 때, 자원 할당 도구(Resource Allocation Tool)가 코딩 타스크 작업을 한 사람에게 할당한다면 형상관리 도구는 이 사람을 체크함으로써 동일한 모듈을 테스팅할 수 없도록 하는 것이다.

5. 결론 및 향후 연구과제

CASE 도구의 개발은 소프트웨어의 생산성 문제를 해결할 수 있는 방법으로써 오래전부터 연구되어 왔고 CASE 도구가 상용화되어 많은 종류의 CASE 도구가 판매되고 있다. 현재의 대부분의 CASE 도구들은 소프트웨어 생명주기 부분들만을 지원하는 도구가 대부분이므로 소프트웨어의 생산성을 높이기 위해서는 생명주기 전단계를 지원할 수 있는 통합형 CASE의 개발이 절실히 필요하다고 볼 수 있다.

본 연구에서는 통합형 CASE의 도구 통합과 방법에 대하여 간략히 기술하였으나 이것은 현재 통합형 CASE의 방법론으로서 연구되고 있는 부분이다. 이 방법론은 통합형 CASE 개발시 요구되는 부분들이므로 중점적으로 설명하였다. 아직 통합형 CASE에 대한 연구는 획기적인 단계에 이르지 못했지만 국내외적으로 통합형 CASE에 대한 관심은 점점 증가하고 있기 때문에 빠른 시일내에 통합형 CASE의 개발이 이루어지리라 예상한다.

앞으로 통합형 CASE에 대한 연구과제로서 통합형 CASE의 프레임워크, 데이타의 일관성을 유지하기 위한 정보저장소, 각 도구들간의 통합을 위한 방법론 등의 연구가 필요하다고 본다.

참고문헌

[1] V. R. Basili and D. M. Weiss, "A Methodology for Collection Valid Software Engineering Data," IEEE Trans. Software Eng., Nov. pp. 728~738, 1984.

[2] A. I. Wasserman, "Tool Integration in Software Engineering Environment, in Software Engineering Enviroments. Int'l Workshop on Environment, F. Long, ed., Springer-Verlag, Berlin, pp. 137~149, 1990.

[3] V. Stenning, "On the Role of an Environment, "Proc. Int'l Conf. Software Eng., IEEE CS Press, Los ALamitos, Calif., pp. 30~34, 1987.

[4] M. I. Thomas, "Tools Integration in the pact Environment, "Proc. Int'l Conf. Software Eng., IEEE CS Press, Los ALamitos, Calif, pp. 13~22, 1989.

[5] V. J. Mercurio *et al.*, "AD/Cycle Strategy and Architecture," IBM System F., Vol. 29, No. 2, pp. 170~288, 1990.

[6] M. Dowson, B. Nejmeh, W. Riddle, "Fundamental Software Process Concepts," Ref. No 7-7, Software Design and Analysis, Boulder, Colo, 1990.

[7] B. Nejmeh, "Characteristics of Integrable Software Tools, "Tech Report 89036-N, Software Productivity Consortiu, Herndon, Va., 1989.

[8] 日本情報處理學會, "CASEの問題點と今後の 方向", 1993.

[9] 양해술, "객체지향 CASE 환경의 기초기술", 정보과학회특집, 9권 2호, 1991.

[10] 양해술, "CASE 개발기술과 고도이용", 정보산업, 1993.

양 해 술

1975 홍익대학교 전기공학과 학사
1978 성균관대학교 정보처리학과 석사
1991 日本 오사카대학 기초공학부 정보공학과 소프트웨어공학 공학박사
1975 ~1979 육군중앙경리단 전자계산실 근무
1984 ~1992 성균관대, 명지대 경영대학원 및 정보산업대학원 강사
1986 ~1987 日本 오사카대학 객원연구원
1994 ~현재 한국산업표준원 이사
1994 ~현재 한국정보과학회 학회지 편집부위원장
1994 ~현재 한국정보처리응용학회 논문지편집위원장
1980 ~현재 강원대학교 전자계산학과 교수
관심 분야 : 소프트웨어 공학(특히, S/W 품질보증과 평가, SA/SD, OOA/OOD/OOP, CASE), 전문가 시스템.