자치적 분산처리 시스템을 위한 객체지향 소프트웨어 개발 프레임워크에 대한 연구

염근혁*

An Object-Oriented Software Development Framework for Autonomous Decentralized Systems

Keun-hyuk Yeom

〈요 약〉

분산처리 시스템을 위한 소프트웨어 개발 방법 및 이를 지원하는 프레임워크의 개발은 분산처리 시스템을 위한 좋은 소프트웨어의 개발을 위해 매우 필요한 일이라 할 수 있다. 즉, 분산처리 시스템을 위한 신뢰성 있고 적합성과 확장성을 가진 소프트웨어의 개발은 매우 중요한 일이다. 자치적 분산처리 시스템(Autonomous Decentralized Systems)은 온라인 확장성과 온라인 유지보수성 및 fault tolerance기능을 가진 분산처리 시스템이다. 이 논문에서는 자치적 분산처리 시스템을 위한 객체지향 소프트웨어 개발을 지원하는 프레임워크에 대하여 논한다. 이 프레임워크는 객체지향 요구 분석과 객체지향 설계, 구현, 프로세서에 타스크(task)의 할당, 검증 및 유지보수로 구성되어 있으며, 온라인 확장성과 온라인 변경성(modifiability)을 지원한다. 자치적 분산처리 애플리케이션 소프트웨어 개발을 지원하는 프레임워크는 객체지향 computation 모델을 근간으로 하고 있다. 이 논문에서는 프레임워크외에도 자치적 분산처리 소프트웨어 개발을 위한 CASE(Computer Aided Software Engineering) 환경에 대하여서도 논한다.

* 부산대학교 공과대학 컴퓨터공학과

1. 서 론

컴퓨터와 통신 기술의 빠른 발전은 분산처리 시스템이 여러 애플리케이션 영역에서 널리 사용될 수 있는 계기를 제공하여 주었다 [Yau et al. 1990]. 몇몇 애플리케이션 영역에서는 시스템이 부분적인 오류나 시스템의 확장 또는 유지보수 시에도 멈추지 않고 동작을 계속하기를 요구한다. 그러므로, 이러한 시스템들은 fault-tolerance 기능뿐만 아니라, 온라인 확장성 및 온라인 유지보수 특성을 가지고 있어야 한다. 이러한 특성들을 지니기 위해 온라인 확장성, fault-tolerance 및 온라인 유지보수 기능들을 가진 자치적이고 분산된 구성요소들로 이루어진 자치적 분산처리 시스템(Autonomous Decentralized Systems)이 개발되었다 [Ihara & Mori 1984, Kawano et al. 1993].

이 자치적 분산처리 시스템은 철강 생산 과정 제어 시스템과 고속전철 교통 제어시스템 등과 같은 여러 영역에 사용되고 있다. 이러한 시스템을 효과적으로 이용하기 위하여 자치적 분산처리 시스템을 위한 효과적인 애플리케이션 소프트웨어 개발 방법론이 필요하다.

분산처리 시스템을 위한 소프트웨어의 개발은 프로세서간의 통신, 동기성(synchronization) 등과 같은 복잡성 때문에 중앙처리(centralized computing) 시스템을 위한 소프트웨어 개발보다도 더욱 더 복잡하며 어려운 일이다 [Yau et al. 1990, Yau et al. 1992]. 그러나, 객체지향 패러다임이 실세계 문제들의 분산구조를 잘 반영하고 상속적으로 concurrent 행위를 표현하는데 적절하기 때문에 자치적 분산처리 시스템과 같은 대규모의 분산처리 시스템을 위한 소프트웨어 개발을 지원하는데 적당하다. 근본적으로, 어느 프로그래밍 언어를 가지고도 분산 애플리케이션을 프로그래밍하는 것이 가능하다. 그러나, 소스 언어가 분산 환경에 적절한 constructs나 상위 레벨의 추상화(abstraction) 기능 등을 가지고 있지 않다면, 프로그래머의 작업은 상당히 복잡하고 어려운 일이 될 것이다. 이러한 문제들을 해결하기 위해 자치적 분산처리 시스템을 위한 소프트웨어 개발을 지원하는 객체지향 computation 모델 [Yau & Oh 1993]을 개발했다.

이 논문에서는 위에서 언급한 computation 모델을 근간으로 자치적 분산처리 시스템을 지원하는 객체지향 소프트웨어 개발 프레임워크에 대하여 논한다. 우리가 개발한 프레임워크는 객체지향 요구사양 분석, 객체지향 설계, 구현, 할당(allocation), 검증 및 유지보수 등의 단계들로 구성되어 있다. 또한 이러한 단계들로 구성된 소프트웨어 개발 프레임워크를 지원하는 CASE 개발 환경에 대한 연구에 대하여서도 논한다.

2. 객체지향 Computation 모델

우리가 개발한 객체지향 소프트웨어 개발 프레임워크 [Yau et al. 1995]에 대하여 논하기 전에 프레임워크의 근간이 되는 computation 모델 [Agha 1986, Briot & Ratuld 1989, Levy & Tempero 1991, Yau & Oh 1993]에 대하여 간략히 논하고자 한다.

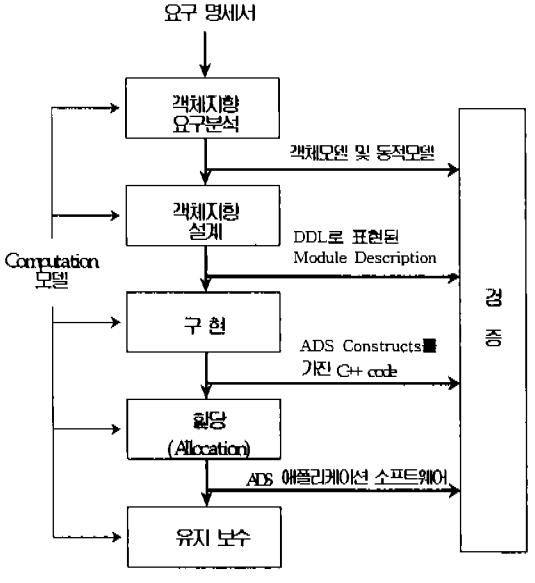
computation 모델은 객체지향 개념을 가지고 으며, 응용 소프트웨어 레벨에서 온라인 확장성과 온라인 변경성(modifiability) 등을 지원한다. 이 computation 모델에서 자치적 분산처리 시스템을 위한 소프트웨어는 모듈의 집합으로 표현되며, 각 모듈은 그 자체의 제어 쓰래드(control thread), 객체베이스, 인터페이스베이스와 베이스 유지관리 시스템 등을 가지고 있다.

객체베이스는 모듈 안에 정의된 객체 클래스의 instance들을 포함한다. 한 모듈의 객체베이스 안에 있는 객체들은 그 모듈의 지역 객체(local object)라고 불려지며, 다른 모듈의 객체베이스 안에 있는 객체들은 이 모듈의 리모트 객체라고 한다. 인터페이스베이스는 객체 메소드들과 이 메소드들이 속하는 객체들의 이름을 포함한다. 인터페이스에는 import 인터페이스와

export 인터페이스 두 가지 종류가 있다. import 인터페이스는 그 스스로 불려지는 다른 객체에 존재하는 메소드들의 리스트이며, export 인터페이스는 다른 객체에 의해서 불려지는 메소드들의 리스트이다. 베이스 유지관리 시스템은 온라인 특성들을 지원하기 위해 동적으로 객체베이스와 인터페이스베이스를 변화시키는 기능들을 제공한다.

다음은 computation 모델이 가진 중요 특성들이다.

- 클래스와 객체가 소프트웨어 개발의 근간으로서 사용된다.
- 이중 레벨의 encapsulation이 존재한다: 모듈레벨과 객체레벨
- 비동기(asynchronous) 리모트 객체 메소드 호출(invocation)과 동기적(synchronous) 지역객체 메소드 호출(invocation)이 사용된다.
- full logical location transparency가 지원된다.
- 온라인 확장성과 온라인 변경성 등을 지원한다.



<그림 1> 자치적 분산처리 시스템 지원을 위한 객체지향 소프트웨어 개발 프레임워크

3. 프레임워크

프레임워크는 다음의 단계들로 구성된다: 객체지향 요구 분석, 시스템 설계 및 객체 설계, 구현, 할당, 검증 및 유지보수의 단계들로 구성된다. 자치적 분산처리 시스템을 위한 소프트웨어 개발 프레임워크가 <그림 1>에 나타나 있다.

자치적 분산처리 시스템을 위한 소프트웨어 개발은 요구 명세로부터 시작되며, 이 요구 명세는 후에 객체지향 요구 분석 기술에 의하여 객체모델과 동적모델로 변환된다. 여기서 객체지향 요구 분석 방법은 Rumbaugh의 OMT(Object Modeling Technique) [Rumbaugh et al. 1991]를 기반으로 확장된 기술이다. 객체모델은 클래스들과 애플리케이션 정의영역에 대한 지식 및 요구명세서로부터 도출된 클래스들 사이의 구조와 관계를 나타내며, 동적모델은 객체모델에 있는 각 클래스의 행위를 상태전이 다이어그램을 사용하여 표현한다. 이 객체모델과 동적모델은 설계단계에서 모듈에 있는 내용을 표현하기 위하여 설계 묘사 언어(Design Description Language(DDL))로 전환된다. 그 후 DDL로 표현된 각각의 모듈 설계는 C++ 프로그래밍 언어로 구현되며, 이 구현된 모듈들은 프로세서에 할당된다.

각 단계의 작업이 끝난 후에 작업의 결과가 정확한가를 검증하며, 유지보수 기술이 자치적 분산처리 애플리케이션 소프트웨어를 관리하기 위하여 적용되어 질 수 있다. 본 논문에서는 자치적 분산처리 시스템을 지원하기 위한 소프트웨어 개발 프레임워크를 논하면서, 특히 요구 분석 및 검증, 시스템 설계와 이 단계들을 지원하는 CASE 툴의 개발에 대하여 중점적으로 논한다.

4. 객체지향 요구분석과 검증

객체지향 요구분석은 더욱 이해하기 쉬운 요구사양을 제공하고, 분석 후에 객체지향 설계 및 구현을 적절히 지원하기 때문에 그 사용이 급속

도로 확산되고 있다. 여기서 우리는 객체지향 분석 방법과 그 산출물인 요구사양의 검증에 대하여 논한다.

실세계 응용 문제들에 대한 정확하고 간결하며 이해하기 쉽고 올바른 모델의 생성을 위하여, 이 문제들의 요구명세들을 조사하고 그것이 내포하고 있는 의미들을 분석하여 형식적이고 엄격한 방법으로 이것들을 다시 묘사하여야 한다 [Rumbaugh et al. 1991]. 이 분석 모델들은 객체모델과 동적모델들로 구성되어 있다. 일반적으로 객체모델은 다음과 같은 방법으로 생성된다 [Rumbaugh et al. 1991]:

1. 요구명세의 객체 클래스들을 인지한다.
2. 클래스들 사이의 관련성을 인지한다.
3. 객체의 속성들을 인지한다.
4. 공통된 구조를 공유하기 위하여 상속성을 사용하여 클래스들을 구조화한다.
5. 객체모델이 올바르게 생성될 때까지 위의 단계들이 반복한다.
6. 마지막으로 계층적인 객체모델을 만들기 위하여 top-down과 bottom-up 방법을 적용한다.

기존에 만들어진 대부분의 객체지향 요구분석 방법들에서, 요구분석의 결과는 클래스 정의, 클래스사이의 계층 구조 및 클래스들 사이의 관련성과 같이 클래스들 사이의 정적 정보에 주로 초점을 맞추고 있다. 하지만, 분산처리 시스템에서는 클래스들 사이의 정보 교환 행위, 객체의 persistency 및 객체의 역할 등과 같이 더 많은 정보들이 요구분석 단계에서 필요로 하며 설계단계에서 이용되어 진다. OMT [Rumbaugh et al. 1991]에서 상태전이 다이어그램을 사용한 동적모델은 통신 및 일치성의 검사를 하기에는 적절치 않다. 그러한 이유로 기존의 상태전이 다이어그램을 확장한 새로운 동적모델을 만든다. 그 생성과정은 다음과 같다.

1. 전형적인 상호 작용 순서에 따른 시나리오를 작성한다.
   이 시나리오들은 주요 상호작용, 외부와의 표현 방식, 정보의 교환방식 등에 관한 내용을 보여 준다.
2. 각 시나리오에 대한 사상 추적도(event trace)를 작성한다.
   각 시나리오에 존재하는 외부 사상들을 인지하여 순차적 리스트를 만든다.
3. 사상 추적을 근간으로 상태전이 다이어그램을 생성한다.
   각 시나리오 와 사상 추적들은 상태전이 다이어그램의 한 패스(path)에 대응되어 진다.
4. 시스템 레벨에서 일치성과 완결성을 체크한다.
   각 사상은 sender와 receiver가 존재해야 하며, 한 상태전이 다이어그램에 나타난 사상이 다른 상태전이 다이어그램에 대응되는 사상들과 일치하는가를 확인한다.

실세계에서 소프트웨어 시스템은 수많은 객체 클래스와 그들 사이의 연관성으로 구성되어 있다.

실세계에 존재하는 이러한 시스템을 다루기 위하여 객체모델을 layer의 개념을 사용하여 계층적 객체모델을 개발한다. 또한, 이를 실용화하기 위하여 객체클래스와 그 연관성을 인지한 후, top-down과 bottom-up 방법을 조화시켜 클래스들을 집단화한다. 다음은 클래스의 집단화를 위한 가이드라인이다:

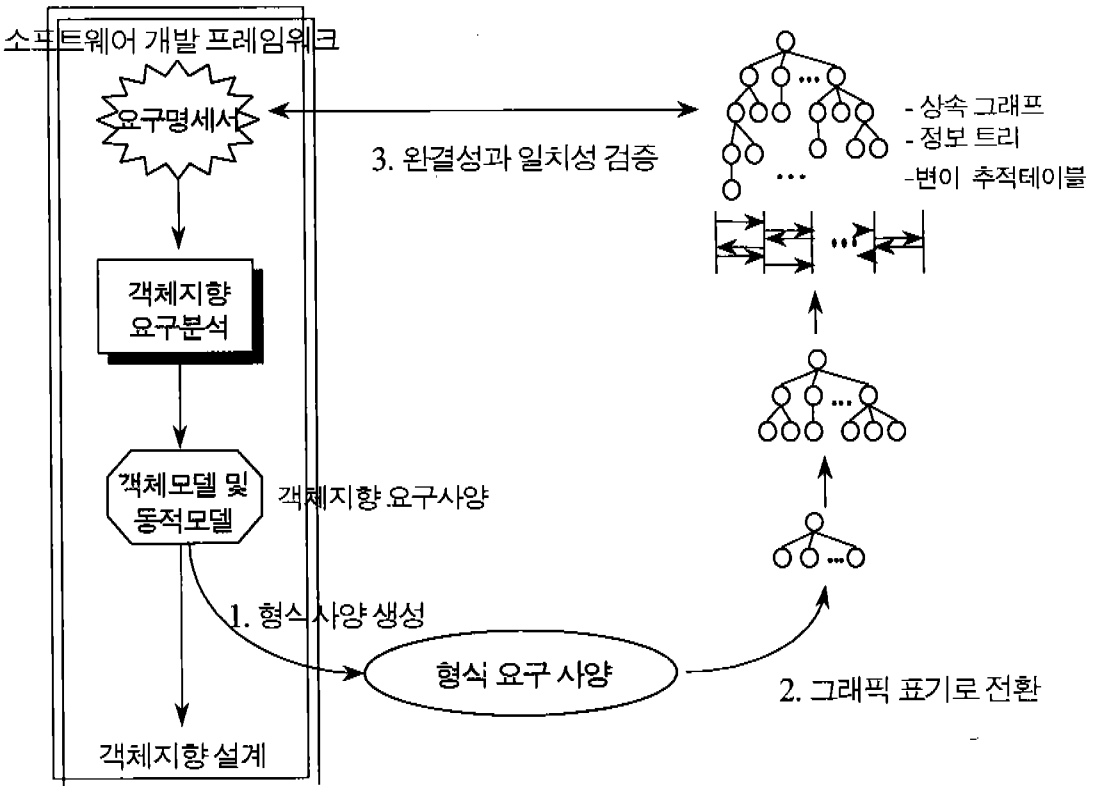
- 자주 통신을 하는 클래스들은 같은 그룹으로 집단화한다.
- 두 개의 스크린사이의 다리역할을 하는 클래스의 수를 최소화하도록 한다.
- 시스템의 주요 영역을 인지하고, 이를 일의 분할과 병렬 개발을 위한 기초로 활용한다.
- 각 분할된 영역에 객체지향 요구분석 방법을 적용한다.

요구명세는 자연어로 쓰여진 소프트웨어 시스템의 상위-레벨 묘사이며 객체지향 요구분석 시

정의영역 지식의 사용을 필요로 하며 객체지향 요구사양을 생성하기 위하여 주어진 요구명세로부터 중복되고 불필요한 정보는 무시되기 때문에 요구명세와 객체지향 요구사양 사이에는 약간의 차이점이 존재할 수 있다. 이러한 차이들은 인지되고 시스템에서 그것들의 영향은 요구사양 검증 시 평가되어 져야 한다. 우리의 검증 방법은 불필요한 정보로 인식되어 요구사양에서 빠진 정보와 요구명세에는 존재하지 않지만 요구사양에는 명시되어 있는 정보들을 인지한다 [염근혁 1996, Yau et al. 1994].

객체지향 요구사양을 검증하기 위하여 자연어로 쓰여진 요구명세와 객체지향 분석의 결과인 객체지향 요구사양 사이의 완결성과 일치성을 체크한다 [염근혁 1996, Yau et al. 1994].

이것을 수행하기 위해 객체지향 요구분석에 의해 생성된 객체모델과 동적모델로 표현된 객체지향 요구사양을 형식 사양 언어로 쓰여진 형식 사양으로 변환하며, 이렇게 변환된 형식사양은 그래픽형식의 표기로 변환된다. 이 후 그래픽 표기에 나타난 정보들을 자연어로 주어진 요구명세와 비교함으로써 요구사양의 일관성과 완결성을 검증한다. 그래픽 표기는 문제 그 자체의 복잡성을 증가시키지 않고 체계적으로 검증을 수행하며 요구사양의 정보를 조직적으로 체계화하기 위하여 사용된다. 요구분석의 결과인 요구사양을 검증하는 전체 과정이 <그림 2>에 나타나 있다.



<그림 4> 요구분석 검증 과정

우리의 검증방법의 입력은 객체모델과 동적모델로 표현되는 객체지향 요구사양과 고객에 의해 주어진 자연어로 쓰여진 요구명세이다. 검증과정은 다음과 같이 요약될 수 있다:

1. 생성된 객체지향 요구사양을 형식 사양언어로 묘사되는 형식 요구사양으로 변환한다.
2. 단계1에서 얻어진 형식사양으로부터 상속 그래프, 정보 트리 및 변이 추적 테이블로 묘사되는 그래픽 표기로 변환한다.
3. 주어진 요구사양의 각 요구명세 및 정의영역 지식과 그래픽 표기로 표현되고 있는 객체지향 요구사양의 정보들을 비교함으로써 요구사양의 일치성과 완결성을 체크한다.

5. 객체지향 설계

시스템 설계 시, 후에 상세 설계에서 결정되어 질 많은 자세한 내용들을 제공하는 시스템의 구조가 결정되어야 한다. 전체 시스템을 위한 상위 레벨 설계를 할 때, 설계자는 후에 일을 여러 개발자가 동시에 독립적으로 여러 모듈에서 일을 할 수 있도록 문제를 모듈들로 나누어야 한다.

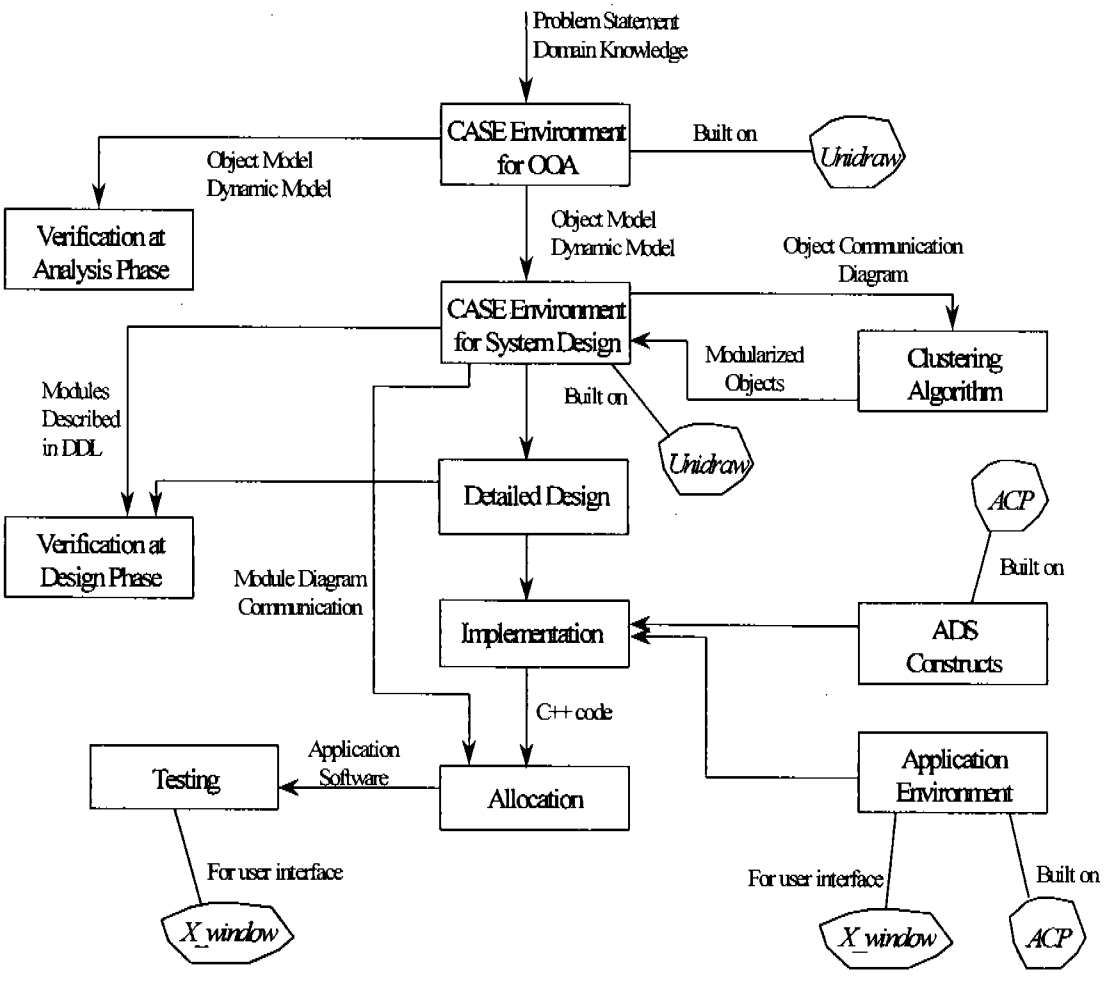
소프트웨어 개발 프레임워크를 지원하기 위하여 개발된 computation 모델에서 자치적 분산처리 시스템은 여러 모듈들의 집합으로 볼 수 있으며, 각각의 모듈은 그 자체의 제어 thread와 지역 객체 및 import, export를 포함하는 인터페이스로 구성된다. 이러한 구조를 기반으로 다음과 같은 시스템 설계 과정을 만들 수 있다:

1. 요구명세, 객체모델과 동적모델들로부터 모든 객체와 객체사이의 상호 통신관계를 인지한다.
   요구분석 시 객체모델링을 수행하면서 객체보다는 클래스에 중점을 두고 모델링을 하였다. 그러나, 시스템에서 서로 다른 역할을 담당하고 있는 객체들과 그들 사이의 통신 교환행위는 시스템 설계 단계에서 인지되어져야 한다.
2. 객체들을 모듈로 묶는다.
   우리의 computation 모델을 기본으로 생각할 때, 모듈은 객체도 function도 아닌, 서로 상호 연관되고 다른 모듈에 대한 잘 정립된 인터페이스를 가진 객체들과 그들 사이의 연관성, 동작(operation)들, 사상물과 제약조건의 패키지(package)이다. 모듈의 형태 또는 객체들과 상호 연관된 요소들의 집단화(clustering)는 개발되어지고 있는 시스템의 성능에 직접적인 영향을 미친다. 이러한 이유로 모듈간의 통신비용을 줄이고 concurrency를 증가시키며, fault-tolerance 및 신뢰성을 성취하기 위하여 좋은 clustering 알고리즘이 필요하다. 이 목적을 성취하기 위해 통신을 최소화하고 concurrency를 극대화하며 기능성과 사용자의 제약조건을 수용하는 기준을 바탕으로 객체들을 체계적으로 모듈로 집단화하는 clustering 알고리즘을 개발하여 사용한다. 우리가 개발하고 있는 clustering 알고리즘은 제약조건으로 기능성과 concurrency를 가지고 통신을 최적화 하는 bottom-up heuristic 알고리즘이다.
3. 각 모듈의 제어 쓰래드(control thread)를 인지한다. 제어 쓰래드는 어느 순간에 단지 하나의 객체만이 행위를 하는 상태 전이 다이어그램의 집합에서 제어가 흘러가는 모습을 보여주는 패스(path)이다. computation모델에서 각 모듈은 객체들을 초기화하고 모듈의 프로세스 행위를 정의하는 모듈의 바디(body)를 가지며, 이 바디는 하나의 제어 쓰래드를 갖는다.
4. computation 모델에서 정의된 설계 묘사 언어(DDL)를 가지고 각 모듈을 묘사한다.
   우리가 개발한 DDL은 자치적 분산처리 시스템을 지원하는 소프트웨어 설계를 묘사하기 위하여 사용된다. 소프트웨어 설계에서 DDL의 사용은 시스템 설계와 구현 사이의 차이점들을 감소시킨다. 또한 자치적 분산처리 시스템을 위한 소프트웨어에서는 시스템이 지

원하는 통신 방법인 브로드케스팅 방법을 사용하기 위하여 content code [Mori et al. 1986]가 각 모듈에 할당되어야 한다.

6. 구현 및 할당(Allocation)

우리의 프레임워크에서 코딩은 computation 모델이 지원하는 메소드 호출 (invocation), guard 문장 등과 같은 자치적 분산처리 constructs를 가진 C++프로그래밍 언어를 사용하여 완성된다. 자치적 분산처리 constructs를 사용하였을 때의 이점은 프로그래머가 동기성, 통신방법 및 프로세서의 위치 등에 구애받지 않고 코딩을 할 수 있다는 것이다. 자치적 분산처리 construct들은 우리의 computation 모델에서 지원하는 제어 메카니즘에 의해 자동적으로 C++코드로 변환된다.



<그림 6> 자치적 분산처리 시스템의 애플리케이션 소프트웨어 개발을 지원하는 프레임워크를 위한 CASE 개발 환경

구현이 완성된 후, 각 모듈들은 자치적 분산처리 시스템의 프로세서들에 할당되어 져야 한다.

모듈 할당을 위하여 우리는 분산처리 시스템을 구성하는 서브 시스템들 사이의 통신을 최소화하고 로드(load)를 균형 있게 분산시킨다는 기준을 사용한다 [Yau & Satish 1993]. 이러한 기준을 바탕으로 서브 시스템들이 LAN으로 연결되었고 자치적 분산처리 시스템과 같이 브로드케스팅(broadcasting)을 통신 수단으로 사용하는 분산처리 시스템에 모듈들을 할당시키기 위한 heuristic 할당 알고리즘을 개발하였다.

이 알고리즘에서는 로드의 균형과 통신을 최소화함으로써 시스템의 concurrency를 증가시키며, 두 개의 복사된 모듈들은 같은 서브 시스템에 할당하지 못하게 하는 제약조건을 사용하여 fault-tolerance의 기능을 구현하였다. 그뿐만 아니라, 사용자가 어떤 특정 모듈을 어느 특정된 서브 시스템에 할당하는 것을 허용하도록 구현되었다.

7. CASE 개발환경

시스템 분석가로 하여금 우리가 개발한 객체지향 요구분석 방법에 따라 주어진 요구명세로부터 객체모델과 동적모델을 생성하는 것을 돕고, 시스템 설계자가 시스템 설계 방법에 의해 시스템 구조를 손쉽게 설계할 수 있도록 하기 위하여 CASE 환경을 개발하였다. 여기서 개발된 CASE 환경은 일반적인 그래픽 편집기능, 제약조건 유지보수 기능, 오류 검사와 CASE 툴로부터 자동적인 출력을 생성하는 기능 등으로 구성되어 있다. 그래픽 유저 인터페이스를 사용한 편집기는 일반적인 유저 인터페이스와 개발 환경 하에 있는 툴들 사이의 데이터의 전송을 향상시키기 위해 이중 레벨의 integration을 사용하며, 제약조건 유지보수를 위해 구성요소들 사이의 연결성을 검사한다. 또한 요구분석이나 시스템 설계를 돕기 위해 CASE 툴로부터 생성된 출력을 자동적으로 연결시켜 주는 기능을 포함하고 있다.

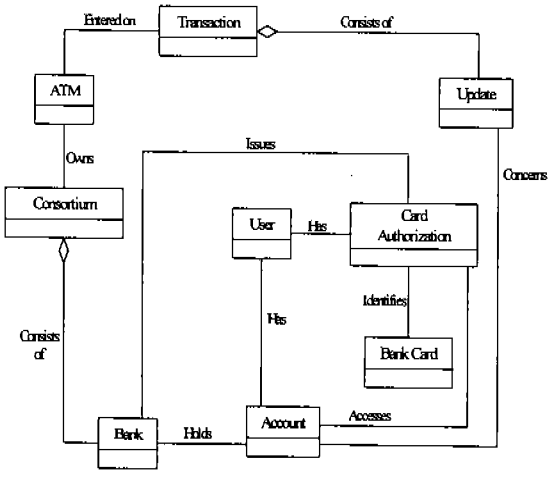
<그림 3>에 나타낸 것처럼 자치적 분산처리 시스템을 위한 응용 소프트웨어 개발을 지원하는 CASE 환경을 제공하는 것이 우리의 목적이다.

그림에서 보듯이 이중레벨의 integration이 지원되고 있다. 첫 번째 레벨은 일반적인 유저 인터페이스이다. 명령과 툴은 객체모델 툴, 동적모델 툴, 사상 추적 편집기 등과 같은 객체지향 요구분석과 객체 통신 모델 툴, 모듈모델 툴 등과 같은 시스템 설계 둘 다를 지원하는 툴들에서 동일하게 접근할 수 있다. 통합(integration)의 다음 레벨은 툴들 사이의 또 다른 연결 역할을 하는 툴 사이의 데이터 전송을 위한 것이다. CASE 툴들은 서로서로 정보를 가져오고, 일치성을 검사하기 위하여 통신할 필요성이 있다. 예를 들어, 시스템 설계를 위한 CASE 툴은 클래스 정의, 통신의 동적행위 등의 정보를 객체지향 요구분석을 위한 툴에서부터 가져 올 수 있다.

우리의 CASE 툴의 설계는 Unidraw [Vlissides 1990]의 프레임워크에 근간을 두고 있다. 이 툴은 객체지향 프로그래밍 언어인 C++를 사용하여 구현되었으며, X 윈도우 환경에서 동작한다. 또한 대화 상자, 텍스트 편집기, 구조적 그래픽 기능 등과 같은 객체지향 그래픽 툴킷인 InterView [Linton et al. 1992]로부터 몇몇 객체들을 사용하였다.

객체모델 툴은 뷰어, 풀-다운(pull down) 메뉴 등으로 구성되어 있으며, 풀-다운 메뉴는 지정된 명령을 수행하는 제어, 이 툴을 이용하는 제어, 뷰어를 확대/축소하는 패너(panner), 객체모델 툴에 의해 관리되는 상태변수의 값을 표시하는 상태변수 뷰 등을 포함하고 있다. 상태변수는 객체모델의 이름과 변화 상태를 나타낸다. 사용자는 툴의 왼쪽 끝에 있는 적절한 제어 명령을 선택함으로써 이 툴을 사용할 수 있다. 동적모델 툴은 객체모델 툴과 매우 비슷하며, 상태변수 뷰, 풀-다운 메뉴, 뷰어, 패너 등으로 구성되어 있다. 이 툴을 이용하여 move, select 등과 같은 동작요소와 상태, 사상, 클래스, 변이 등과 같은

동적모델의 구성 요소들을 만들 수 있다. 구현단계에서 computation 모델은 통신, 동기성, 다른 자치적 분산처리 시스템의 특성 등을 제어하는 자치적 분산처리 시스템 소프트웨어를 위한 언어의 construct들을 제공한다. 이러한 자치적 분산 처리 시스템의 construct들은 선행 프로세서(preprocessor)에 의해 C++ 프로그래밍 언어로 변환된다. 선행 프로세서는 자치적 분산처리 시스템 애플리케이션 모듈을 읽어서, computation 모델에 의해 정의된 문법과 같은 가를 검사하고 소스코드를 생성한다. 할당 알고리즘 또한 X윈도우 환경 하에서 구현되었으며, 이 알고리즘은 cluster, spillover, update-load 등의 3단계로 크게 나눌 수 있다. cluster 프로시져에서는 두 개의 노드를 피봇(pivot) 노드와 비피봇 노드로 나누어 하나의 싱글 노드로 병합함으로써 연합(compound) 노드를 형성한다. spillover 단계에서는 로드(load)의 균형적인 분석을 성취하며, 한 모듈이 서브 시스템에 안전하게 할당될 수 있는 가를 검사한다. 매번 할당이 일어날 때마다 할당이 수행된 서브 시스템의 로드가 update-load라는 프로시져를 통해 변경될 수 있다. 앞으로, 분산처리 시스템을 위한 소프트웨어 개발을 지원하는 CASE 개발환경과 통합된 기능을 가진 툴을 완성할 것이다.



<그림 7> ATM 시스템의 객체모델

8. 예 제

이 섹션에서는 단순화된 현금 자동 인출 시스템(ATM 시스템)을 사용하여 우리의 방법을 설명하고자 한다. 이 시스템을 위한 요구 명세는 다음과 같다:

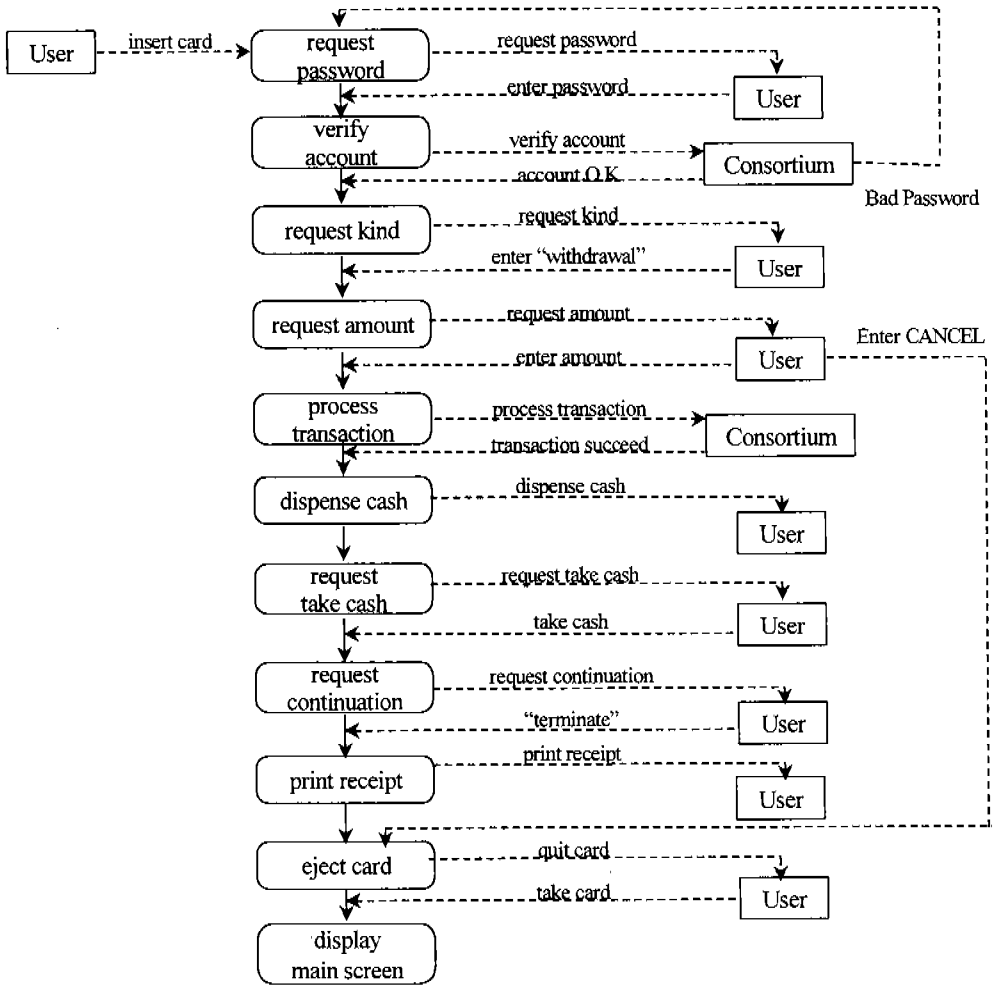
Develop the software to support a computerized banking system with automatic teller machines (ATMs) to be shared by a consortium of banks. Each bank has its own computer to maintain its account and make updates to accounts. ATMs communicate with a central computer of the consortium. An ATM accepts a bank card, interacts with the user, communicates with the central computer to process transactions and/or dispense cash.

단순화를 목적으로 시스템은 2개의 현금 자동 인출기와 2개의 은행, 하나의 은행 컨소시엄으로 구성되어 있다고 가정하자.

4.1절에서 언급한 객체지향 요구분석 과정에 따라 ATM 시스템의 객체모델과 동적모델을 얻을 수 있다. <그림 4>는 ATM 시스템의 객체모델을 나타내며, <그림 5>는 ATM 시스템의 한 클래스인 ATM 클래스의 상태변이 다이어그램을 보여주고 있다. 객체지향 요구 검증에서는 우선 객체지향 요구사양을 형식사양으로 변환하고 이를 다시 그래픽 표기로 변환한다. < 그림 6>은 그래픽 표기의 하나인 정보 트리를 사용하여 ATM 시스템을 설명하고 있다. 그래픽 표기로 변환된 요구사양은 요구명세와 비교하여 일치성을 검사한다.

예를 들어, 요구명세 문장 "ATM dispense cash."를 생각해 보자. 정보 트리에서 이 문장에 대응되는 패스(path)는 <그림 6>에서 화살표로 나타내어진 것같이 ATM, dispense_cash, cash, ≤$200과 같은 노드들을 선택함으로써 인지될 수 있다.

그래픽 표기의 정보와 주어진 요구 명세가 일치성을 검증하기 위하여 비교된다.



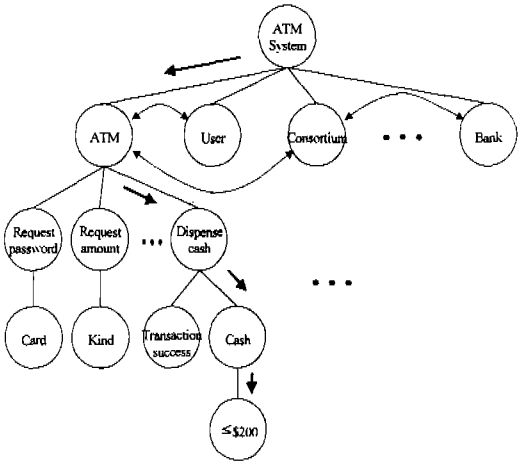
<그림 8> ATM 클래스의 상태변이 다이어그램

위의 예에서, 우리는 "less than or equal $200"라는 정보가 객체지향 요구사양에서 불일치됨을 알 수 있다. 다음으로 이 정보가 특정 영역 지식으로부터 온 것인지 아니면 요구분석을 하는 동안에 발생된 에러인지를 결정한다. 결과적으로, "less than or equal to $200"라는 정보는 영역 지식으로부터 온 것임을 알 수 있다.

시스템 설계에서는 먼저 객체들과 객체 상호간의 통신을 인지한다. 이 예제에서 객체들은 ATM1, ATM2, Bank1, Bank2, Consortium, Bank_Card, Card_Authorization, Account, Transaction, Update, User 이며, 객체간의 통신은 "Consortium communicates with ATM1, ATM2, Bank1 and Bank2.", "Updates communicate with corresponding Bank.", "Updates communicate with Accounts and Transactions." 그리고 "Users communicates with ATMs.". 다음 단계는 객체의 집단화(clustering)이다. 집단화를 실행한 후의 모듈 통신 다이어그램이 그림7에 보여 지고 있다. 이 후, 각 모듈의 제어 쓰래드를 인지하고, 각 모듈을 설계 묘사

언어(DDL)로 표현한다.

9. 결 론

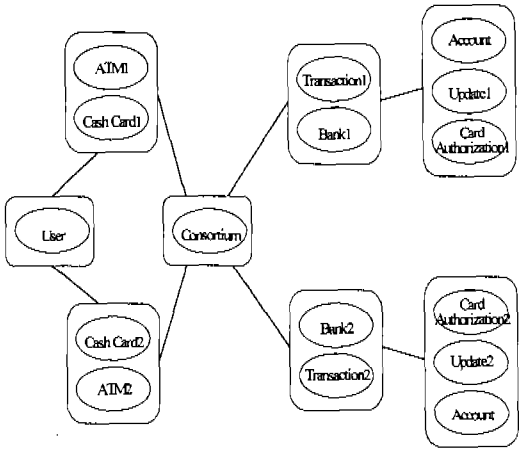


<그림 6> ATM 시스템을 위한 정보 트리

본 논문에서 우리는 자치적 분산처리 시스템을 위한 애플리케이션 소프트웨어 개발을 지원하는 객체지향 프레임워크에 대하여 논하였다. 우리의 프레임워크는 객체지향 요구분석, 객체지향 설계, 구현, 할당, 검증 및 유지보수로 구성되어 있다.

또한 복잡하고 대규모인 시스템을 관리하기 위하여 계층적 객체모델링 방법을 소개하였으며, 통신을 최소화하고 로드를 균형 있게 할당하는 할당 알고리즘을 개발하였다. CASE 개발 환경에서는 CASE 툴을 사용하여 사상 추적으로부터 자동적으로 동적모델을 만들었다.

자연어로 쓰여진 요구명세는 가끔 같은 요구사항에 대하여 상이한 분석을 이끌어 내고 요구분석을 하는 동안에 심각한 문제를 야기시키는 모호성을 포함하고 있기 때문에 요구분석 시에 요구명세에 존재하는 모호성의 문제는 앞으로 풀어야 할 중대한 문제이다. 객체지향 요구 분석 방법은 Rumbaugh의 OMT(Object Modeling Technique) [Rumbaugh et al. 1991]을 기반으로 Unified Modeling Language [Booch et al. 1997]을 접목시켜 좀 더 효율적인 요구분석을 하기 위하여 수정·보완될 계획이다. 설계 검증 시에는 자치적 분산처리 시스템을 위한 소프트웨어 개발에서 동시성(concurrency), 통신, deadlock과 같은 설계 결과와 밀접한 관계를 가지고 있는 특성들을 다룬다. 또한 자치적 분산처리 시스템에서 동작되는 응용 소프트웨어에 대한 테스트 역시 고려되어야 한다.



<그림 7> Clustering 후 ATM 시스템의 모듈 통신 다이어그램

현존하는 분산처리 소프트웨어에서의 온라인 변경성은 시간과 비용을 들여 반복적으로 할당을 수행하지 않는다면 시스템의 성능을 저하시킨다. 그러므로, 분산처리 시스템에서 온라인 변경을 지원하는 효율적인 재할당(reallocation) 알고리즘의 개발이 필요하다. 분산처리 애플리케이션 소프트웨어 개발 프레임워크의 표준화를 위하여 우리가 개발한 computation 모델을 개방형 시스템 환경에 적합하도록 확장할 것이며, CORBA [Siegel 1996]를 이용하여 우리의 객체지향 소프트웨어 개발 프레임워크를 보편화할 것이다. 또한 자치적 분산처리 시스템은 싱글 루프 환경뿐만 아니라 다중 루프 환경에서도 동작되기 때문에 computation 모델을 다중 루프 환경에 적합하도록 확장할 계획이다.

〈참고문헌〉

- 염근혁, "분산 시스템을 위한 객체지향 소프트웨어 개발에서 요구사양에 대한 검증 방법", 「한국정보과학회 봄학술발표논문집」, 1996, 4월, pp. 643-646.
- Agha, G., *Actors: A model of Concurrent Computing in Distributed Systems*, MIT Press, Cambridge, Mass., 1986.
- Booch, G., I. Jacobson and J. Rumbaugh, *Unified Modeling Language Version 1.0*, Rational Software Corporation, Santa Clara, CA, 1997.
- Briot, J-P and J. de Ratuld, "Design of a Distributed Implementation of ABCL/1," *SIGPLAN Notes*, Vol. 24, No. 4, 1989, pp. 15-21.
- Ihara, H. and K. Mori, "Autonomous Decentralized Computer Control Systems," *IEEE Computer*, Vol. 17, No. 8, August, 1984, pp. 57-66.
- Kawano, K., M. Orimo and K. Mori, "Autonomous Decentralized Systems: Concept, Data Field Architecture and Future Trend," *Proc. Int'l Symposium on Autonomous Decentralized Systems*, 1993, pp. 28-34.
- Levy, H.M. and E.D. Tempero, "Modules, Objects and Distributed Programming: Issues in RPC and Remote Object Invocation," *Software Practice and Experience*, Vol. 21, No. 1, 1991, pp. 77-90.
- Linton, M.A., P.R. Calder, J.A. Interrante, S. Tang and J.M. Vlissides, *InterViews Reference Manual, Version 3.1*, Stanford University, Stanford, CA, 1992.
- Mori, K., H. Ihara, Y. Suzuki, M. Koizumi, M. Orimo, K. Nakai and H. Nakanish, "Autonomous Decentralized Software Structure and Its Application," *Proc. ACM-IEEECS Fall Joint Computer Conference*, 1986, pp. 1056-1063.
- Rumbaugh, J., M Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice-Hall, Englewood Cliffs, New Jersey, 1991.
- Siegel, J., *CORBA Fundamentals and Programming*, John Wiley & Sons, New York, 1996.
- Vlissides, J., "Generalized Graphical Object Editing," Technical Report: CSL-TR-90-427, Computer Systems Laboratory, Stanford University, Stanford, CA, June 1990.
- Yau, S.S., X. Jia, and D.-H. Bae, "Trends in Software Design for Distributed Computing Systems," *Proc. Second IEEEWorkshop on Future Trends of Distributed Computing Systems*, 1990, pp. 154-160.
- Yau, S.S., X. Jia and D.-H. Bae, "Software Design Methods for Distributed Computing Systems," *Journal of Computer Communications*, Vol. 15, No. 5, May, 1992, pp. 213-223.
- Yau, S.S. and G.-H. Oh, "An Object-Oriented Approach to Software Development for Autonomous Decentralized Systems," *Proc. Int'l Symposium on Autonomous Decentralized Systems*, 1993, pp. 37-43.
- Yau, S.S. and V.R. Satish, "A Task Allocation Algorithm for Distributed Computing Systems," *Proc. 17th Int'l Computer Software & Applications Conference*, September, 1993, pp. 336-342.
- Yau, S.S., D.-H. Bae, and K. Yeom, "An Approach to Object-Oriented Requirements Verification in Software Development for Distributed Computing Systems," *Proc. 18th Int'l Computer Software and Application Conference*, November 1994, pp. 96-102.
- Yau, S.S., K. Yeom, B. Gao, L. Li and D.-H. Bae, "An Object-Oriented Software Development Framework for Autonomous Decentralized Systems," *Proc. Int'l Symposium on Autonomous Decentralized Systems*, April, 1995, pp. 405-411.