MRP 시스템의 신뢰성을 위한 객체지향 컴포넌트 개발 사례

- A Case Study on OOP Component Build-up for Reliability of MRP System -

서 장 훈 *
Seo Jang Hoon

Abstract

Component based design is perceived as a key technology for developing advanced real-time systems in a both cost- and time effective manner. Already today, component based design is seen to increase software productivity, by reducing the amount of effort needed to update and maintain systems, by packaging solutions for re-use, and easing distribution.

Nowdays, a thousand and one companies in IT(Information Technology) industry such as SI(System Integration) and software development companies, regardless of scale of their projects, has spent their time and endeavor on developing reusable business logic. The component software is the outcome of software developers effort on overcoming this problem; the component software is the way propositioned for quick and easy implementation of software. In addition, there has been lots of investment on researching and developing the software development methodology and leading IT companies has released new standard technologies to help with component development. For instance, COM(Component Object Model) and DCOM(Distribute COM) technology of Microsoft and EJB(Enterprise Java Beans) technology of Sun Microsystems has turned up. Component-Based Development (CBD) has not redeemed its promises of reuse and flexibility. Reuse is inhibited due to problems such as component retrieval, architectural mismatch, and application specificness. Component-based systems are flexible in the sense that components can be replaced and fine-tuned, but only under the assumption that the software architecture remains stable during the system's lifetime.

* 명지대학교 산업공학과 박사과정

In this paper, It suggesst that systems composed of components should be generated from functional and nonfunctional requirements rather than being composed out of existing or newly developed components. about implements and accomplishes the modeling for the Product Control component development by applying CCD(Contract-Collaboration Diagram), one of component development methodology, to MRP(Material Requirement Planning) System

Keyword : Component Based Design(CBD), Contract-Collaboration Diagram(CCD), MRP, CBD, Sofrtware architecture

I. 서 론

오늘날 수많은 시스템 통합업체나 소프트웨어 개발 업체들은 프로젝트의 크고 작음에 관계없이 반복되는 비즈니스 로직의 개발에 많은 비용과 노력을 소비하고 있다. 컴포넌트 소프트웨어의 출현은 이 문제를 극복하려는 개발자들의 오랜 고민의 결과였다. 소프트웨어 개발을 더욱 빠르고 손쉽게 할 수 있는 방안으로 제시됐던 것이 컴포넌트 소프트웨어였던 것이다.

컴포넌트 기반 소프트웨어 시스템 개발은 소프트웨어 업계에서 나타난 표준적인 컴포넌트 기반 구조 기술의 특성과 소프트웨어 생산성 모두를 주목할만하게 향상시키기 위한 솔루션으로 EJB, CORBA, ActiveX, JavaBeans와 같은 컴포넌트 기술의 공통적인 목적과 함께 오늘날 이미 유용하게 이용하고 있다.(김운용, 최영근, 2000, pp547-550)

컴포넌트를 개발하기 위한 방법론에도 많은 연구와 발전이 있어왔으며 컴포넌트 개발을 돕기 위한 기술 표준들이 선두 IT업계를 중심으로 계속 발표되고 있다. 대표적으로 마이크로소프트의 COM(Component Object Model)이나 DCOM(Distribute COM), 그리고 Sun MicroSystems의 EJB(Enterprise Java Beans)기술 등이 등장하였다. 본 논문에서는 컴포넌트 개발 방법론인 계약-협동 다이어그램(Contract-Collaboration Diagram)을 이용하여 MRP(Material Requirement Process)시스템에 적용하므로써 생산 관리 컴포넌트 개발을 위한 모델링을 수행하고 구현하고 있다. 현재 많은 제조 및 금융 회사들이 그들의 운영관리 정보시스템을 데이터 중심 시스템으로부터 프로세스 중심 시스템(워크플로우 시스템)으로 전환하려는 시도를 하고 있다. 이와 같은 변화는 급변하는 소비자의 요구를 신속하게 반영하여 서비스의 질을 높임으로서 치열한 경쟁이 지속하는 국제 시장에서 생존하기 위한 노력의 일환이다. 데이터 검색 기능 자동화 뿐만 아니라 회사내의 업무 프로세스를 자동화하면 기업 전체의 업무 생산성이 향상되는 것은 자명하다.

이러한 업무 프로세스를 자동화하기 위한 방법론에도 많은 변화가 있었다. 프로세스

중심의 구조적 방법론에서 데이터를 중심으로 하는 정보공학, 그리고 최근에 등장한 객체지향 방법론과 컴포넌트 방법론까지 다양한 방법론이 소개되었다. 컴포넌트 기반 개발은 인터넷 시대의 무한 경쟁으로 살아 남기 위한 수단이라고 할 수 있다. 남보다 더 빠르게, 더 질이 좋은 소프트웨어를 만들어 시장을 선점하고 새로운 비즈니스모델에 신속히 대처할 수 있는 소프트웨어를 만들기 위한 노력의 일환으로 컴포넌트 기반 개발에 대한 관심이 높아져 가고 있다. 이러한 컴포넌트를 잘 만들고 또 컴포넌트를 이용한 애플리케이션을 만들기 위해서는 적절한 방법론의 선택이 중요하다.(배두환, 2000, pp25-30) 결과적으로, 이러한 문제점에 대한 논의 및 보완을 위해서 본 논문에서는 생산관리 프로세스 관점에서 계약-협동 다이어그램을 통한 자재소요계획(Material Requirement Planning) 컴포넌트를 구현함으로써 컴포넌트 개발을 통해 얻을 수 있는 기대효과와 앞으로의 발전방향을 제시하였다.

II. 본 론

2.1 계약-협동 다이어그램 소개

본 논문에서 인용하고 있는 새로운 비즈니스 프로세스 분석 모형인 계약-협동 다이어그램(contract-collaboration diagram)은 객체지향 프로세스 정의 모형이다. 이 모형은 함수지향(function-oriented) 프로세스 정의 모형인 IDEF0와 유사하게 기본 모델링 블록(primitive modeling block)을 이용하여 프로세스를 정의한다. 그러나 IDEF0에서는 함수의 입출력 데이터 관계에 의해서 함수간의 프로세스가 정의되나, 계약-협동 다이어그램은 객체간의 계약 관계에 의해서 프로세스가 정의된다. 이 모형은 기존의 절차적(Procedural) 프로세스 분석 방법의 한계인 객체 표현의 한계, 프로세스 단위의 불명확성, 사건의 연관성 문제, 및 협동 프로세스의 표현의 한계를 극복하고자 제시된 것이며 다음과 같은 특징을 갖는다.

- 프로세스 수행 주체인 객체를 프로세스 분석 단계부터 명확히 표시하는 모형이다.
- 객체가 협동해야할 대상(객체)을 명확히 표현함으로서 프로세스의 기본 단위를 정의할 수 있다.
- 제약식을 이용하여 사건에 의해서 서로 관련된 프로세스들을 하나의 선언적(declarative) 메타(meta) 프로세스로 표현하는 모형이다. 선언적 방법에서는 객체의 상태변수(state variable or attribute)들간에 유지되어야할 일관성(consistency) 조건을 제약식으로 표현하고 제약식이 파괴되는 사건이 발생할 때 다시 제약식을 만족하는 상태로 복귀하기 위한 과정을 비즈니스 프로세스로 정의한다. 메타 프로세스란 동일 수행 객체 군에서 발생하는 다수의 프로세스를 제약식을 이용하여 하나의 프로세스로 표현한 것을 의미한다.

· 메타(meta) 프로세스로부터 사건-조건-활동 규칙(event-condition-action rule)을 이용하여 프로세스들을 추출할 수 있기 때문에 능동(active) 데이터베이스 시스템을 이용하여 구현할 수 있는 모형이다.

본 논문에서는 계약-협동 다이어그램을 설명하기 위해서 온톨로지(ontology) 방식을 따르기로 한다. 온톨로지 개념은 원래 고대 그리스 철학에서 출발하였으며, 현재 인공지능의 핵심 연구 분야인 지식 기반 시스템(knowledge management system)을 구축하는데 필수적인 요소로 사용된다. 온톨로지는 다음과 같이 정의된다.(김창욱외 2명, 2000, pp25-30)

"온톨로지는 모델링 도메인의 모든 사물(thing)을 정의하고 이들간의 관계를 명시하는 학문 분야이다"

온툴로지에 의거하여 계약-협동 다이어그램에 필요한 요소를 정의하면 다음과 같다.

[정의 1] 도메인 객체(domain object)

도메인 객체는 모델링하고자 하는 시스템 내에 존재하는 객체를 의미한다. 예를 들어 고객, 완제품, 부품, 재고, 기계 등이 이에 속한다.

[정의 2] 능동 도메인 객체(active domain object)

능동 도메인 객체는 도메인 객체의 한 종류로서 스스로 사건을 발생시킬 수 있는(self event generating) 객체를 의미한다. 예를 들어 고객, 기계 등이 이에 속한다.

[정의 3] 수동 도메인 객체(passive domain object)

수동 도메인 객체는 도메인 객체의 한 종류로서 스스로 사건을 발생시킬 수 없는 객체를 의미한다. 예를 들어 완제품, 부품, 재고 등이 이에 속한다.

[정의 4] 계약 관계(contract relationships)

계약 관계는 계약-협동 다이어그램의 핵심 부분으로 모든 비즈니스 프로세스는 객체간의 계약 관계로 정의된다는 가정 하에 제안된 개념이다. 다시 말해서 모든 비즈니스 프로세스는 두 개 이상의 객체의 계약에 의해서 시작되고, 연속된 객체간의 계약에 의해서 프로세스가 진행되며, 모든 계약이 완료되는 시점이 프로세스의 종료 조건이 된다.

[정의 5] 클라이언트 객체(client object)

능동 도메인 객체의 한 종류로 계약 관계를 시작시키는 객체를 의미하며 고객 객체가 이에 속한다.

[정의 6] 서버 객체(server object)

능동 도메인 객체의 한 종류로 클라이언트 객체로부터 시작된 계약 요청을 성사시켜 주는 객체를 의미하며 고객 객체가 이에 속한다.

[정의 7] 계약 객체(contract object)

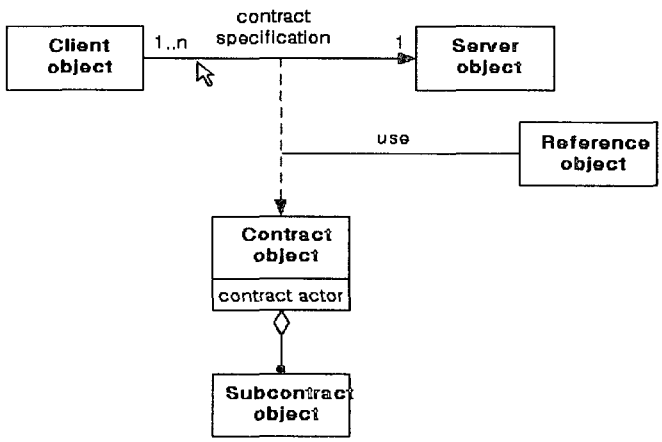
클라이언트 객체의 요청에 의해서 서버 객체와의 계약이 성사되면 생성되는 객체를 의미한다. 계약 객체는 스스로 사건을 발생시킬 수 없는 수동 개체로서, 일반적으로 계약 객체는 클라이언트 객체와 서버 객체와의 계약 사항을 명시하는 객체 종류이다.

생산관리 시스템에서의 객체 객체는 주로 스케줄링(scheduling)에 의해서 생성되는 잡(job)을 명시한다.

[정의 8] 참조 객체(reference object)

참조 객체는 계약을 맺을 때 참조되는 객체이며, 주로 수동 도메인 객체를 의미한다. 예를 들어 완제품, 부품 등은 스스로 사건을 발생시킬 수 없으나 계약 관계를 형성할 때 필수적으로 필요한 참조 객체이다.

위와 같은 정의를 가지고 계약-협동 다이어그램의 기본 모델링 블록을 설명하면 < 그림 2.1 >과 같다.

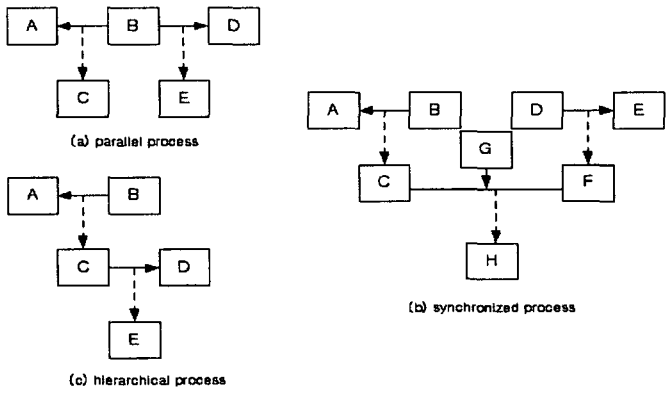


< 그림 2.1.1 > 계약-협동 다이어그램에서의 기본 모델링 블록

< 그림 2.1.1 >에서 "contract specification"은 계약 이름을 정의하는 부분이며, 구조적 제약식(structural constraint)을 포함한다 (이 그림에서는 1..n을 의미함). 주의할 점은 구조적 제약식은 계약 시점에 참여하는 객체의 수를 의미한다는 것이다. 다시 말해서 [그림 1]에서 계약 시점에 "client object"는 하나이상이 계약에 참여하고 "server object"는 하나만 참여한다는 뜻이다.

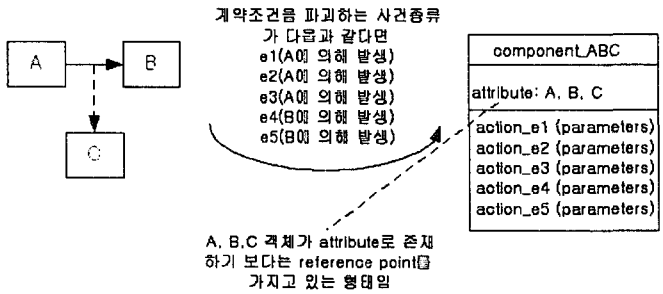
< 그림 2.1.1 >에서 "contract object"는 다시 여러 개의 "subcontract object"를 포함할 수 있는데 (선택 사항), 이는 "contract object"가 다시 계약을 맺을 때 "contract object" 자체가 계약을 맺을 수도 있으나 이 객체의 일부분(객체 지향에서는 애트리뷰트로 표기함)이 다른 서버 객체와 계약을 맺는 경우도 있기 때문에 필요하다. "contract object"는 다시 새로운 계약 관계를 맺을 수 있다. 그러나 "contract object"는 스스로 사건을 발생시킬 수 없기 때문에 계약의 클라이언트가 될 수 없다. 이를 보완하기 위해서 "contract object" 표기에서 "contract actor"를 명시한다. "contract actor"는 "contract object"의 계약 주체가 된다.

< 그림 2.1.1 >에서 "reference object"는 계약 시점에 참조되는 수동 도메인 객체이며 선택 사항이다. 계약-협동 다이어그램은 기본 모델링 블록을 연결하여 도메인의 프로세스를 기술하는 방식이며, 이는 곧 프로세스가 계약과 협동의 연속으로 묘사된다는 것을 기본 전제로 한다. 계약-협동 다이어그램으로 묘사할 수 있는 프로세스의 종류는 < 그림 2.1.2 >와 같이 병렬 프로세스, 동기화 프로세스, 및 계층적 프로세스가 있으며, 이들 프로세스 모형을 적절히 혼합함으로써 복잡한 비즈니스 프로세스가 표현된다.



< 그림 2.1.2 > 표현 가능한 프로세스 종류

객체지향 관점에서 볼 때 비즈니스 컴포넌트(component)는 "high cohesion" 즉 연관성이 많은 객체들의 집합과 이들에 의해서 서비스되는 함수들의 인터페이스 집합으로 정의될 수 있다. 기존의 객체지향 방법들은, 예를 들어 UML (unified modeling language)에서 제공하는 use-case 방법론, 연관성이 많은 객체들의 집합을 명확하게 추출하는 방법을 제시하지 못하였다. 반면에 본 연구에서 제시하는 계약이라는 관계에 의해서 연관성이 많은 객체들의 집합을 제시한다. 계약-협동 다이어그램에서의 컴포넌트는 < 그림 2.1.3 >에서 보는 바와 같이 기본 모델링 블록을 구성하는 "클라이언트 객체", "서버 객체", 및 "계약 객체"를 의미한다. 또한 인터페이스는 이들 객체들의 계약 조건을 파괴하는 각 사건에 대응되는 계약 조건 복구 함수가 된다. 두 개의 기본 모델링 블록(컴포넌트)이 연결되어 있다면 이는 공유 객체가 존재한다는 의미가 된다. 따라서 공유 객체의 상태 변화를 감지하는 조정자(mediator) 객체를 둔다면 해당 컴포넌트끼리 직접적인 인터페이스 호출을 하지 않아도 된다는 장점이 있다.



< 그림 2.1.3 > 모델링 블록의 컴포넌트 변환

2.2 컴포넌트 인터페이스 정의

계약-협동 다이어그램은 비즈니스 프로세스를 생성하는데 사용되는 모형이다. 이는 계약이라는 단어에서도 쉽게 알 수 있다. 그러나 일반적으로 프로세스 진행은 계약-협동 다이어그램의 순서대로 진행되지는 않는다. 진행 도중에 사건이 발생하면 프로세스가 중단되거나 역방향으로 흘러갈 수 있다. 따라서 컴포넌트를 정의, 즉 인터페이스를 정의하려면 다음과 같은 단계를 거쳐야 한다.[12][13]

① 각 기본 모델링 블록마다 계약 조건을 명시한다.
② 기본 모델링 블록 내에서 발생할 수 있는 사건을 명시한다. 사건 종류는 크게 객체 삽입, 삭제, 갱신으로 구분할 수 있다.
③ 계약 조건을 파괴하는 사건을 추출한 후 각 사건에 대응하여 파괴된 계약 조건을 복구하는 함수를 정의하고 그 함수에 대한 인터페이스를 정의한다.

계약 조건은 다시 다음과 같은 세 가지로 구분할 수 있다.

① 제 1 제약 조건 (C_{1} constraint) - 계약 성사 시점에 만족해야 하는 조건이다.
② 제 2 제약 조건 (C_{2} constraint) - 협동 과정에 만족해야 하는 조건이다.
③ 제 3 제약 조건 (C_{3} constraint) - 계약 완료 시점에 만족해야 하는 조건이다.

제 1 제약 조건은 클라이언트 객체와 서버 객체의 상태 변수(state variable or attribute)가 계약 성사 시점에 만족해야 하는 조건과 이들의 계약 조건인 계약 객체의 상태 변수의 조건을 명시하는 것이다. 클라이언트 객체, 서버 객체, 그리고 계약 객체 간의 만족 조건은 (A->B)->C의 형태로 나타난다. 예를 들어 "Product Job"(클라이언트 객체)와 "Factory"(서버 객체)간의 계약에 의하여 "MPS"(계약 객체)가 생성되는 계

약관계의 경우, 모든 "Product Job"은 "Factory"에서 생산 가능하여야 하며 이들의 계약관계는 "MPS"로 명시된다는 계약 조건을 다음과 같이 기술할 수 있다.

(for all Product Job -> exist Factory : 생산 가능) -> exist MPS

진리표(truth table)를 이용하여 이와 같은 제 1 제약 조건 중 클라이언트 객체와 서버 객체간의 관계에서 계약 조건을 파괴하는 사건과 이에 대한 함수를 보면 아래 <표 2.2.1>과 같다.

< 표 2.2.1 > 제 1 제약 조건 진리표

A의 상태	B의 상태	계약조건 (A->B)	함수	계약조건
T	T	T	해당사항 없음	
T	F	F	A delete	T
			new B insert	T
			B update	T
F	T	T	해당사항 없음	
F	F	T	해당사항 없음	

위의 진리표에서 볼 수 있듯이 제 1 제약 조건을 파괴하는 사건은 클라이언트 객체의 삽입이며 계약 조건을 만족시키려면 세 가지(A delete, new B insert, B update)의 함수가 존재한다. 주의할 점은 "new B insert"의 경우는 거의 발생하지 않으며, 따라서 "B update"를 하여 제 1 제약 조건이 만족했는가를 판단하고 그렇지 못하면 "A delete"를 실행하여야 한다. 따라서 제 1 제약 조건 파괴에 대한 인터페이스는 하나로 표현한 후 이 인터페이스 내부에서 세 가지 함수의 호출을 결정할 필요가 있다.

클라이언트 객체와 서버 객체간의 계약 조건이 만족되면 그에 해당하는 계약 객체가 생성되어야 한다. 즉 클라이언트 객체의 삽입에 대한 서버 객체의 갱신 작업의 결과, 클라이언트 객체와 서버 객체간의 계약 조건이 참이 된다면 이에 대한 계약 객체가 생성되어야 한다.

계약-협동 다이어그램에서 기본 모델링 블록은 두 단계를 거친다. 첫째 단계가 계약 단계이고 둘째 단계가 협동 단계이다. 계약 단계를 통과하려면 제 1 제약 조건이 만족해야 한다. 한편 계약 단계가 성사되면 협동 단계에 들어서게 되는데 이때 만족해야할 제약 조건이 제 2 제약 조건이 된다. 이 제약 조건은 계약이 성사된 후 계약 당사자(클라이언트 객체와 서버 객체)의 상태 변수 갱신(사건)이 계약을 파괴할 수도 있기 때문에 명시하는 조건이다. < 표 2.2.2 >의 진리표를 이용하여 설명하면 다음과 같다.

< 표 2.2.2 > 제 1 제약조건의 진리표

A의 상태	B의 상태	계약조건(A->B)	함수	계약조건
T	T	T	해당사항 없음	
T	F	F	B update	T
F	T	F	A update	T
F	F	F	A,B update	T

제 3 제약 조건은 협동 단계가 끝나서 계약이 해지될 때 만족해야 하는 제약 조건이다. 따라서 제 3 제약 조건이 False이면 아직 협동 단계에 있다는 뜻이 된다.

이들 제약식간의 관계를 설명하면 다음과 같다.

Step 1. 계약 단계 - 독립 사건이 발생하면 사건을 일으킨 클라이언트 객체와 한 개 이상의 서버 객체(도메인 객체)는 계약 의존 관계를 맺어야하며 이때 계약 시점 제약식(C_1 제약식)을 만족해야만 계약이 성사된다. 이 경우

① C_1 제약식이 만족하면 계약이 성사된다.

② C_1 제약식이 만족하지 않는다면 서버 객체가 사건을 발생하여 제약식이 만족할 때까지 기다린 후 계약을 성사시킨다. 또는 클라이언트 객체가 독립 사건을 취소하는 사건을 발생하여 C_1 제약식을 만족시킨다.

Step 2. 협동 단계 - 계약이 성사되면 계약 사항을 명시하는 계약 객체가 계약 함수에 의해서 생성되며, 클라이언트 및 서버 객체들은 협동 단계에 들어선다.

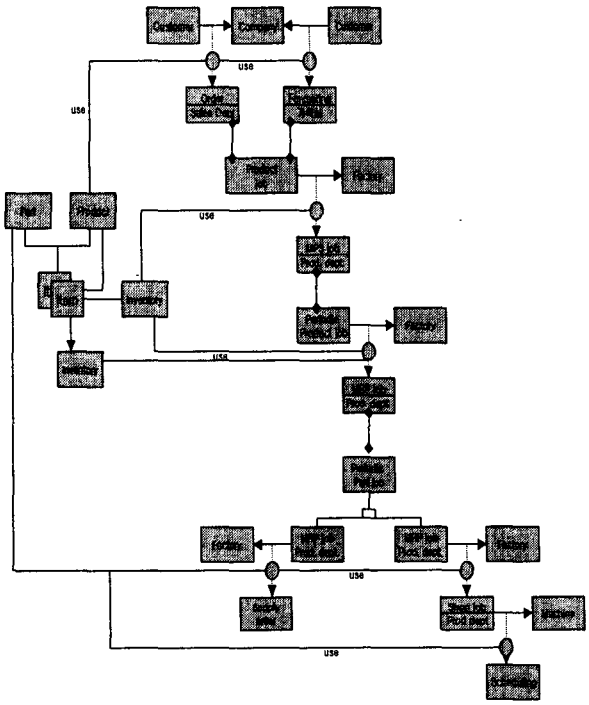
Step 2-1. 협동 단계 중에 C_2 제약식이 파괴되는 사건이 발생하면

① C_2 제약식이 만족되도록 계약 의존 관계에 있는 객체들간에 상태변수를 조정(갱신 사건)하거나, 또는

② Step 1로 돌아가서 다시 계약을 하거나 계약 취소를 한다.

Step 2-2. 협동 단계 중에 C_3 제약식이 만족되는 사건이 발생하면 계약 과 협동은 성공적으로 끝난다.

2.3 컴포넌트 기술 방법



< 그림 2.3 > MRP시스템 상위도메인을 포함한 계약-협동 다이어그램

< 그림 2.3 >는 MRP시스템 상위도메인을 포함한 계약-협동 다이어그램 규칙을 적용한 것이다. 생산 관리 도메인은 주문, 재고, 생산 계획 등 다수의 객체가 존재하며 특히 이들 객체간의 다양한 형태의 연관이 존재하며 복잡한 모델을 이루고 있다. 또한 각 객체의 상태는 다른 객체의 상태와 밀접한 관계를 맺고 있으며, 이러한 관계의 제약 조건을 만족시키는 것이 생산 관리 시스템의 역할이다.
< 그림 2.3 >은 생산 관리 도메인에서 나올수 있는 객체들을 계약-협동 다이어그램의 요소들로 분류한 것이다.

- 클라이언트 객체 리스트 : Customer
- Contract actor : sales dept., prod. dept., shop dept.
- 서버 객체 리스트 : Company, Factory, Shop, Machine
- 참조 객체 리스트 : Item, BOM-unit, Inventory
- 계약 객체 : Order, MPS job, MRP job, Machine job

위 < 그림 2.3 >에서 보듯이 한 개의 클라이언트 객체와 서버 객체 그리고 계약 객체가 하나의 블록을 형성하고 있다. 이러한 블록을 재사용 가능한 컴포넌트 단위로 규정하고 제약식을 표현하기 위한 기호들을 사용하여 각각의 컴포넌트에서 제약식을 서술하도록 한다.

2.3.1 컴포넌트 제약조건

제약식을 표현하기 위하여 사용되는 기호는 다음과 같다.

- for all, use, exist : 조건식을 표현하기 위한 예약어이며 각각 "모든 ~에 대하여", "~를 참조하여", "~가 존재해야 한다"의 의미를 가진다.
- a:"A" : "A"라는 객체형의 한 객체 인스턴스를 의미한다.
- a.어트리뷰트 : "A" 객체형의 한 어트리뷰트 값을 의미한다.
- &&, || : 각각 논리곱, 논리합을 의미한다

각각의 컴포넌트 제약식은 아래 표에서 제시하였다.

< 표 2.3.1 > 주문 콤포넌트

주문 컴포넌트	Client Comp	Customer	Contract Comp	Order
	Server Comp	Company	Reference Comp	Product
제 1제약식	등록된 고객이 주문 -> 주문을 접수함		(**for all** cust:'Customer' -> **exist** co:'Company') **use** 'Product' -> **exist** 'Order'	
제 2제약식	접수된 주문내역이 변경되지 않고 유지된다.		**For all** cust:'Customer', co:'Company', o:'Order' : 계약관계(cust, co, o) **&&** (co.생산능력 >= sum(or.주문량))	
제 3제약식	주문 완성 -> 주문 상태변경		**For all** cust:'Customer', co:'Company, o:'Order' : 계약관계(cust, co, o) **&&** (o.주문상태 = "주문완료")	

< 표 2.3.2 > 수요예측 컴포넌트

수요예측 컴포넌트	Client Comp	Market	Contract Comp	Forcasting
	Server Comp	Company	Reference Comp	Product(Item)
제 1제약식	회사가 수요예측 -> 시장에 제품에 대한 수요가 있슴 -> 수요예측 데이터를 얻음		(**for all** co:'Company -> **exist** m:'Market') **use** 'Product' -> **exist** 'Forcasting'	
제 2제약식	회사의 수요예측 데이터가 변경없이 유지되어야 한다.		**For all** co:'Company , m:'Market', f:'Forcasting' : 계약관계(co, m, f) **&&** (co.생산능력 >= sum(f.수요예측량))	
제 3제약식	수요예측 발생 -> 수요예측 데이터의 상태가 변경된다.		**For all** co:'Company , m:'Market', f:'Forcasting' : 계약관계(co, m, f) **&&** (f.수요예측입력상태 = '수요예측입력완료")	

< 표 2.3.3 > 대일정계획 컴포넌트

대일정계획 컴포넌트	Client Comp	Product-job	Contract Comp	Mps-job
	Server Comp	Factory	Reference Comp	Inventory
제 1제약식	주문과 Forcasting에 의해서 얻어진 Product job을 공장에 할당 -> Mps job데이터를 얻는다.		(for all p:'Product job' -> exist f:'Factory', I:'Inventory' : (f.생산능력 >= p.현재부하 + f.현재부하) || (I.현재고 >= p.수량)) -> exist 'Mps-job'	
제 2제약식	공장에 할당된 Mps-job 데이터가 변하지 말아야 한다.		for all p:'Product job', f:'Factory', i:'Inventory', mps:'MPS job' : 계약관계(p, f, mps) && (p.납기 >= mps.예상완료시간) && (f.생산능력 >= sum(mps.생산량))	
제 3제약식	할당 완료 -> MPS-job의 상태가 변경된다.		for all p:'Product job', f:'Factory', i:'Inventory', mps:'MPS job' : 계약관계(p, f, I, mps) && (mps.생산상태 = "생산완료")	

< 표 2.3.4 > 자재소요계획 컴포넌트

자재소요계획 컴포넌트	Client Comp	Part-Purchase-job	Contract Comp	SupplyOrder
	Server Comp	Supplier	Reference Comp	Part(Item)
제 1제약식	계산된 부품구매 job을 공급자에게 요청 -> 요청받은 수량이 공급자의 부하를 넘지 말아야 함 -> 공급오더가 생성됨		(for all pppj:'Period Part Purchasing job' -> exist su:'Supplier', i:'Inventory' : (i.현재고 < pppj.수량)) use 'Part' -> exist 'Supply order'	
제 2제약식	구매요청계약요건이 변경되어서는 안된다.		for all pppj:'Period Part Purchasing job', su:'Supplier', so:'Supply order' : 계약관계 (pppj, su, so) && (pppj.납기 >= so.예상도착시간)	
제 3제약식	주문 완성 -> 주문 삭제		for all pppj:'Period Part Purchasing job', su:'Supplier', so:'Supply order' : 계약관계 (pppj, su, so) && (so.공급오더상태 = "공급오더완료")	

< 표 2.3.5 > 자재구매오더 컴포넌트

자재구매오더 컴포넌트	Client Comp	Purchasing	Contract Comp	Supply order
	Server Comp	Supplier	Reference Comp	Part(Item)
제 1제약식	구매부서가 납품업체에 납품요구 -> 공급오더 발생		(for all pj:'Purch job' -> exist su:'Supplier' : (su.생산능력 + su.현재고 >= pj.주문량 + su.현재부하)) -> exist 'Supply order'	
제 2제약식	납품업체와 이루어진 공급주문이 변하지 말아야 한다.		for all pj:Purch job, su:Supplier, so:Supply order: 계약관계 (pj,su,so) && (so.납기 >= pj.예상완료시간) && (su.생산능력 >= sum(pj.요구량))	
제 3제약식	공급오더의 상태변경		(so.주문상태 = "주문완료")	

< 표 2.3.6 > ShopSchedule 컴포넌트

ShopSchedule 컴포넌트	Client Comp	Part-mfg-job	Contract Comp	Shop-job
	Server Comp	Shop	Reference Comp	Part(Item)
제 1제약식	계산된 기간별 부품생산 job을 Shop에 할당 -> Shop job 데이터 생성		(for all ppmj:'Periodic Part Mfg job' -> exist s:'Shop', i:'inventory' (s.생산능력 >= s.현재부하 + ppmj.부하) || (i.현재고 >= ppmj.수량)) -> exist 'Shop job'	
제 2제약식	Shop에 할당된 기간별 부품생산 데이터가 변하지 말아야 한다.		for all ppmj:'Periodic Part Mfg job', s:'Shop', sj:'Shop job' : 계약관계(ppmj, s, sj) && (ppmj.납기 >= sj.예상완료시간) && (s.생산능력 >= sum(sj.생산량))	
제 3제약식	할당 완료 -> Shop job 데이터 삭제		for all ppmj:'Periodic Part Mfg job', s:'Shop', sj:'Shop job' : 계약관계(ppmj, s, sj) && (sj.생산상태 = "생산완료")	

< 표 2.3.7 > Machine Schedule 컴포넌트

Machine Schedule 컴포넌트	Client Comp	Shop-job	Contract Comp	Scheduling
	Server Comp	Machine Reference	Reference Comp	Part(Item)
제 1제약식	할당된 Shop job이 기계에 할당 -> Machine job 데이터 생성		(for all sj:'Shop job' -> exist m:'Machine') -> exist 'Scheduling'	
제 2제약식	Machine에 할당된 Scheduling이 변하지 말아야 한다.		for all sj:'Shop job', m:'Machine', sch:'Scheduling' : 계약관계(sj, m, sch)	
제 3제약식	할당 완료 -> Machine job 데이터 삭제		for all sj:'Shop job', m:'Machine', sch:'Scheduling' : 계약관계(sj, m, sch) && (sch.생산상태 = "생산완료")	

2.4 컴포넌트 이벤트

다음은 각각의 컴포넌트에서 발생하는 이벤트들을 서술한다. 이벤트란 제약식을 파괴하는 객체 a의 전이(Transition)을 의미한다. 상태변이의 전이를 사건으로 보았을 때 발생할 수 있는 사건 종류는 객체 삽입, 삭제, 상태변수 변경이 있다. 한편 도메인 객체 중에서 비즈니스 시스템 내부에 있는 객체에 의해서 발생하는 사건을 내부(internal)사건, 시스템 외부에 있는 객체에 의해서 발생하는 사건을 외부(external)사건으로 정의한다. 독립 사건은 외부 사건 중에서 삽입 사건으로 정의하며, 그 밖의 사건은 모두 종속 사건으로 정의한다.

- Entity Event : 객체 스스로 발생시키는 이벤트, 주로 도메인 객체(domain object)에서 발생
- Contract Event : 두 개의 도메인객체, 또는 계약객체와 도메인객체 사이의 계약상태에서 발생하는 이벤트

본 논문에서는 각각의 컴포넌트 이벤트에대해서 아래<표 2.4.1~2.4.7>에서 정리하여 나타내었다.

< 표 2.4.1 > 주문 컴포넌트 이벤트

	Event	조건	활동
제 1제약 조건	'Customer' 생성		'Product'을 참조하여 'Order'을 생성한다.
제 2제약 조건	'Order' 변경	주문내역(주문.duedate이나 주문.수량)의 변경을 원할 경우	'Order'의 주문내역을 바꾼다.
제 3제약 조건	'Order'의 주문 상태 변경	주문이 완료되었을경우	주문상태 -> "주문완료"

< 표 2.4.2 > 수용예측 컴포넌트 이벤트

	Event	조건	활동
제 1제약 조건	'Market' 생성		'Product'을 참조하여 'Forcasting'을 생성한다.
제 2제약 조건	'Forcasting' 변경	예측데이터의 변경을 원할 경우	'Forcasting'의 데이터을 변경시킨다.
제 3제약 조건	'Forcasting'의 수요예측입력상태 변경	수요예측입력이 완료되었을경우	수요예측입력상태 -> "입력완료"

< 표 2.4.3 > 대일정계획 컴포넌트 이벤트

	Event	조건	활동
제 1제약 조건	'Periodic Product job'생성 (계약시점)	생산능력이 충분한 'Factory'가 있거나 'Inventory'의 현재량이 수량보다 큰 경우 (두 조건의 조합문제가 MPS의 전개 알고리즘)	'BOM-unit'을 참조하여 'MRP job'을 생성한다.
		Otherwise, 재고이용 및 생산 불가	'Periodic Product job' 생성을 취소하고 전 블록인 MPS 생성프로세스에게 재계약을 지시한다
	'Factory'의 생산능력 변화(공장내부의 예상치못한 사고) (협동시점)	'Factory'의 생산능력이 계약관계에 있는 'Product job'의 수요를 처리할수 없는 경우	'MPS job'의 조정 또는 재생성 ('Inventory'를 할당 또는 다른 'Factory'에 할당)
제 2제약 조건	'MPS job의 예상완료시간 변경	'MPS job'의 예상완료시간이 'Product job'의 납기보다 늦어지는 경우	1. 'MPS job'의 조정 2. 불가능한 경우는 'Factory'의 생산능력을 조정하여 제 1제약조건의 파괴사건을 구동하여 'MPS job'의 재생성을 요구(역전파 프로세스)
제 3제약 조건	'MPS job의 생산상태 변경	계약관계에 있는 모든 'MPS job'의 생산상태가 "생산완료"로 변경된 경우	'MPS job'를 삭제하여 계약관계 해소 (또는 historical' 정보로 변경)

< 표 2.4.4 > 자소요계획 컴포넌트 이벤트

	Event	조건	활동
제 1제약 조건	'Product job' 생성 (계약시점)	생산능력이 충분한 'Factory'가 있거나 'Inventory'의 현재량이 수량보다 큰 경우 (두 조건의 조합문제가 MPS의 전개 알고리즘)	'BOM-unit'을 참조하여 'MRP job'을 생성한다.
		Otherwise, 재고이용 및 생산 불가	'Periodic Product job' 생성을 취소하고 전 블록인 MPS 생성프로세스에게 재계약을 지시한다.
	'Factory'의 생산능력 변화(공장내부의 예상치못한 사고) (협동시점)	'Factory'의 생산능력이 계약관계에 있는 'Periodic Product job'의 수요를 처리할 수 없는 경우	'MRP job'의 조정 또는 재생성 ('Inventory'를 할당 또는 다른 'Factory'에 할당)
제 2제약 조건	'MRP job'의 예상완료시간 변경	'MRP job'의 예상완료시간이 'Periodic Product job'의 납기보다 늦어지는 경우	1. 'MRP job'의 조정 2. 불가능한 경우는 'Factory'의 생산능력을 조정하여 제 1제약조건의 파괴사건을 구동하여 'MRP job'의 재생성을 요구(역전파 프로세스)
제 3제약 조건	'MRP job'의 생산상태 변경	계약관계에 있는 모든 'MRP job'의 생산상태가 "생산완료"로 변경된 경우	'MRP job'를 삭제하여 계약관계 해소 (또는 historical' 정보로 변경)

< 표 2.4.5 > 자재구매오더 컴포넌트 이벤트

	Event	조건	활동
제 1제약 조건	'Periodic Part Purchasing job' 생성	* supplier의 부하는 고려하지 않기로 한다. 'Inventory'의 사용가능한 수량이 필요부품 수량보다 적을 경우	'Part'을 참조하여 'Supply Order'을 생성한다.
제 2제약 조건		* Supplier는 모든 'Supply Order'에 대해서 책임을 진다.	
제 3제약 조건	'Supply Order'의 자재구매오더입력상태 변경	자재구매오더입력이 완료되었을경우	자재구매오더입력상태 -> "입력완료"

< 표 2.4.6 > ShopSchedule 컴포넌트 이벤트

	Event	조건	활동
제 1제약 조건	**'Periodic Part Mfg job' 생성** (계약시점)	생산능력이 충분한 'Shop'이 있거나 'Inventory'의 현재량이 수량보다 큰 경우	'Part'을 참조하여 'Shop job'을 생성한다.
		Otherwise, 재고이용 및 생산 불가	'Periodic Part Mfg job' 생성을 취소하고 전 블록인 MRP 생성프로세스에게 재계약을 지시한다.
	'Shop'의 생산능력 변화(기계 고장 등의 원인) (협동시점)	'Shop'의 생산능력이 계약관계에 있는 'Periodic Part Mfg job'의 수요를 처리할 수 없는 경우	'Shop job'의 조정 또는 재생성 ('Inventory'를 할당 또는 다른 'Shop'에 할당)
제 2제약 조건	**'Shop job'의 예상완료시간변경**	'Shop job'의 예상완료시간이 'Periodic Part Mfg job'의 납기보다 늦어지는 경우	1. 'Shop job'의 조정 2. 불가능한 경우는 'Shop'의 생산능력을 조정하여 제 1제약 조건의 파괴사건을 구동하여 'Shop job'의 재생성을 요구(역전파 프로세스)
제 3제약 조건	**'Shop job'의 생산상태 변경**	계약관계에 있는 모든 'Shop job'의 생산상태가 "생산완료"로 변경된 경우	'Shop job'를 삭제하여 계약관계 해소 (또는 historical' 정보로 변경)

< 표 2.4.7 > MachineSchedule 컴포넌트

	Event	조건	활동
제 1제약 조건	**'Shop job'생성** (계약시점)	생산능력이 충분한 'Machine'이 있는 경우	'Part'을 참조하여 'Scheduling'을 생성한다.
		otherwise, 생산불가	'Shop job'생성을 취소한다. (이 경우 전 블록인 ShopSchedule 생성 프로세스에게 재계약(재계획)을 지시하는 경우이다.
	'Machine'의 생산능력 변화(기계 고장등의 원인) (협동시점)	'Machine'의 생산능력이 계약관계에 있는 'Shop job'의 수요를 처리할 수 없는 경우	'Scheduling'의 조정 또는 재생성 (다른 'Machine'에 할당)
제 2제약 조건	**'Machine'의 예상완료시간 변경**	'Machine'의 예상완료시간이 'Shop job'의 납기보다 늦어지는 경우	1. 'Scheduling'을 조정 2. 불가능한 경우는 'Machine'의 생산능력을 조정하여 제 1제약 조건의 파괴사건을 구동하여 'Scheduling'의 재 생성을 요구(역전파 프로세스)
제 3제약 조건	**'Scheduling'의 생산상태 변경**	계약관계에 있는 모든 'Scheduling'의 생산상태가 "생산완료"로 변경된 경우	'Scheduling'를 삭제하여 계약관계 해소 (또는 historical' 정보로 변경)

2.5 구현 컴포넌트 정의

이제까지 컴포넌트 제약조건과 이를 파괴하는 이벤트들, 그리고 이러한 이벤트들을 복구하는 복구함수들을 정의하였다. 이러한 복구함수들이 여기서 서술하고자 하는 컴포넌트들의 인터페이스로 정의된다. 즉, 하나의 컴포넌트는 도메인객체들와 계약객체 그리고 인터페이스들로 구성된다. 예를 들어서, 첫 번째 컴포넌트인 주문 컴포넌트는 다음과 같이 정의할수 있다.

```
component MakeOrder
{
  class Customer *cust;
  class Order *order;
  interface customer_Created(class Customer);
  interface order_Updated(class Order);
  interface order_orderStatus_Updated
                        (class Order);
                                    }
```

2.6 객체지향과 컴포넌트 기술

분산 컴퓨팅 기술은 최근에 새롭게 떠오른 패러다임은 아니다. 이미 오래 전부터 유닉스 계열에서는 분산 컴퓨팅을 위한 프로토콜(대표적으로 RPC)이 등장 했으며 후에는 DCE (Distributed Computing Environment), TINA와 같은 프레임웍으로 발전하였다.

이러한 기술과 CORBA/DCOM과 같은 기술의 근본적인 차이점은 기존의 프로토콜들이 객체지향 플랫폼이 아니라는 것이다. 이번 장에서는 객체(컴포넌트)라는 요소가 가미된 분산객체 기술만을 다루고자 한다. 분산객체 기술은 분산 컴퓨팅 기술과 객체지향 기술의 접목으로 한걸음 발전하게 되었고 객체지향 프로그래밍 기법은 컴포넌트를 기반으로 한 형태로 발전하게 되었다.

객체지향 프로그래망 기법과 컴포넌트 기법을 동일한 것으로 오해할 수 있으나 근본적으로 몇가지 차이가 있다. C++ 혹은 Java와 같은 객체지향 프로그래밍 기법은 주로 소스 레벨에서 적용이 된다. 따라서 컴파일 된 C++객체와 Java객체는 서로 다른 바이너리 구조를 가지고 있기 때문에 일반적으로 상호호출을 할 수 없다. 설령 소스 레벨이라고 해도, 서로 다른 언어간에 혼용해서 사용하기란 쉽지 않은 일이다. 그러나 컴포넌트는 일반적으로 바이너리 레벨에서 상호 호환이 가능하다. 예를 들면, 마이크

로소프트의 Visual C++와 Visial Basic은 소스 레벨에서는 상호 호환이 안되지만, 컴파일을 한 COM 객체는 서로 호환이 된다는 것이다. COBRA 객체의 경우도 마찬가지다. 따라서 인터넷/인트라넷과 같은 이기종 환경에 더욱 잘 들어맞는다고 할 수 있다. 혹자는 컴포넌트 기술을 어린이 장난감인 레고에 비유한다. 마치 자동차 생산 라인과 같이 정형화되고 인터페이스가 통일된 부품들을 조립하듯이 컴포넌트를 사용하는 것이다. 레고와 같은 독립된 객체를 지원하기 위해 ActiveX 컨트롤고 JavaBeans 같은 컴포넌트 모델이 있으며, 이들 독립된 객체들이 서로 유기적으로 상호 작용할 수 있는 프레임웍을 CORBA와 DCOM 같은 모델이 제공하는 것이다. CORBA와 DCOM의 컴포넌트 모델에 대해서는 뒤에서 다시 알아보도록 하고 일단 객체지향 기술은 소스레벨, 즉 내부 구조가 들여다 보이는 White-Box이고 컴포넌트는 내부 구조가 보이지 않으며 규격화되고 통일된 인터페이스를 통해서만 외부 세계와 정보를 주고 받는 Black-Box라고 알아두면 좋을 것 같다.

이러한 컴포넌트 기술을 바탕으로 위치 투명성을 제공하는 통신 인프라만 가지게 된다면 바로 분산객체 기술의 모습을 갖추게 되는 것이다. 현재 가장 각광받고 있는 대표적인 분산 객체 기술은 마이크로소프트의 윈도우 플랫폼을 기반으로 한 DCOM(Distributed Component Object Model)과 800여 업체가 컨소시엄을 결성해서 사양을 확정한 CORBA(Common Object Request Broker Architecture)가 있다. 이 두 가지 기술은 현재 지속적으로 시장 선점을 위해 각축을 벌이고 있는 상황이어서 개발자 입장에서는 어느 진영의 기술을 수용해야 할지 혼돈을 줄 수도 있다. 그러나 이 두 가지 기술은 장단점이 있으므로 적용할 프로젝트의 성격에 따라 각 기술을 평가해 적용한다면 좋은 결과를 얻을 수 있을 것이다.(Markku, L. and Ari. J. 1998, pp24-35)

III. MRP 컴포넌트 구현

3.1 시스템 구현 환경

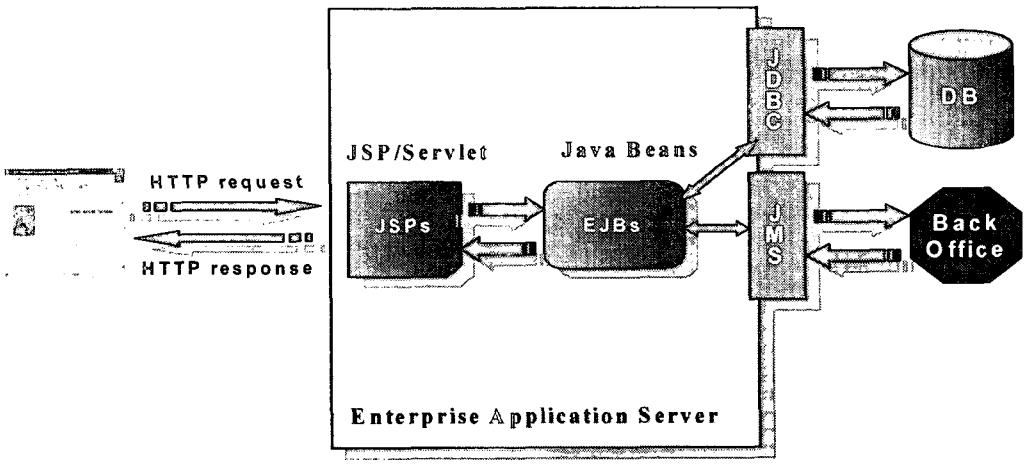
본 연구에서는 Windows 2000 server을 바탕으로 Web환경이므로 클라이언트는 브라우저(MS Explorer)만을 요한다.

MRP 컴포넌트 프로그램의 추가적인 구현 환경은 다음과 같다.

- 운영체제 : Window 2000 server
- Web Application Server : PowerTier for EJB(version 6.0)
- 웹 개발 도구: EJB (Enterpriser Java Beans), JSP(Java Server Page)

- 웹서버 : Apache Web server 2.0
- 데이터베이스 시스템 : Oracle 8i
- 웹브라우저 : Internet Explorer 6.0

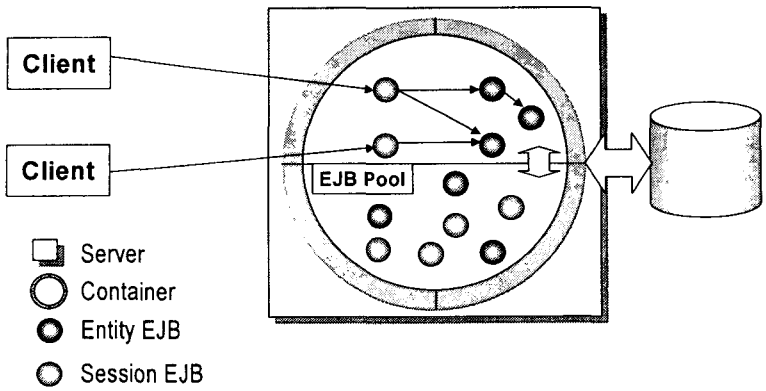
3.2 시스템 아키텍처



< 그림 3.2.1 > 시스템 아키텍처

< 그림 3.2.1 >에서 보는 바와 같이 본 논문에서의 시스템 구성은 인터넷을 통한 어플리케이션 사용자, 어플리케이션 프로그램을 지원해주는 서버, 그리고 데이터 베이스 프로그램의 3계층으로 이루어져 있다. 일반적으로 어플리케이션 사용자는 웹 브라우저를 사용하여 인터넷에 접속하게되며 모든 업무는 웹 어플리케이션 서버에서 관리된다. 본 논문에서 사용하고 있는 PowerTier for EJB 웹 어플리케이션 서버(Web Application Server)는 다음과 같은 특징이 있다.[8]

· 기존에 프로세스 중심의 CGI(Common Gateway Interface)의 문제점을 보안한 JAVA Thread기술을 채택하여 과도한 네트웍의 부하를 해결
· Sun의 표준 컴포넌트 기술(Enterprise Java Beans)을 사용하므로써 재사용 가능한 업무 프로세스 구현 가능
· Load Balancing이나 Clustering기능을 사용하여 대용량의 트랜잭션 처리 가능
· 각종 개발 툴(Tool)을 제공하므로써 개발의 용이성 부여

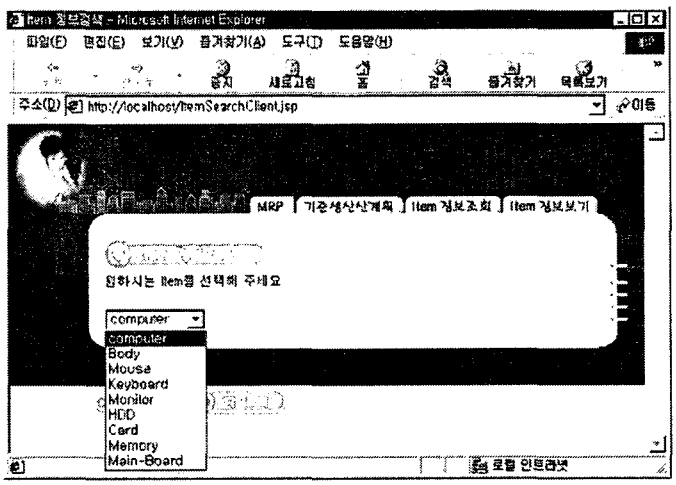


< 그림 3.2.2 > EJB 아키텍처

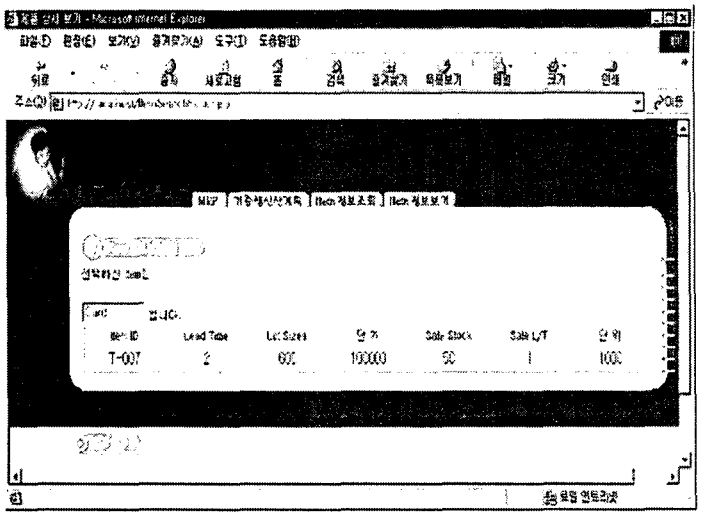
< 그림 3.2.2 >는 EJB 아키텍처를 나타낸 것이다. 일반적으로 엔티티 빈(Entity Bean)은 데이터베이스내의 공유 데이터를 표현하며 세션 빈(Session Bean)은 하나의 비즈니스 로직으로 표현될 수 있다. 생산 관리 시스템에서 일반적인 객체들은 데이터베이스에 저장되어야 할 것들이므로 엔티티 빈으로 표현을 하며 이들 객체들이 수행하는 업무 프로세스는 세션빈으로 표현한다. 이러한 엔티티 빈과 세션 빈은 재사용 가능한 컴포넌트이다.[3]

3.3 시스템 구현화면

아래 < 그림 3.3.1 >은 생산관리 시스템 사용자가 인터넷을 통하여 원하는제품 또는 부품을 ComboBox를 이용하여 검색하는 윈도우이다.

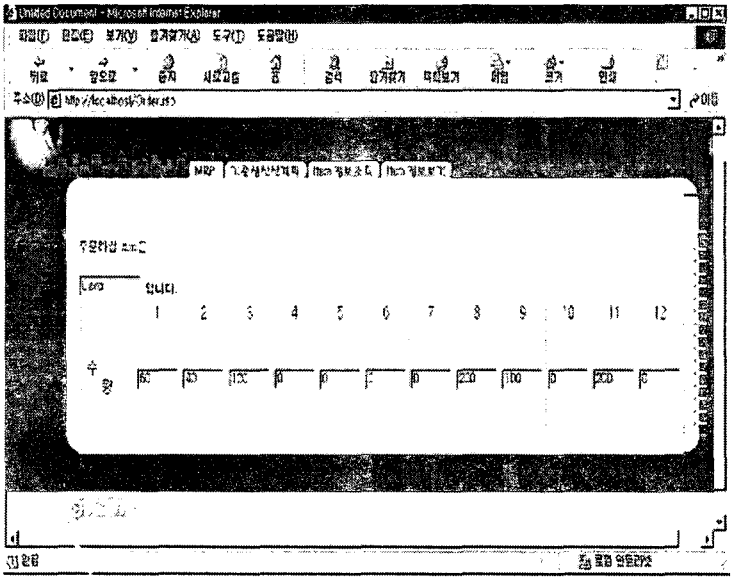


<그림 3.3.1> Item 검색

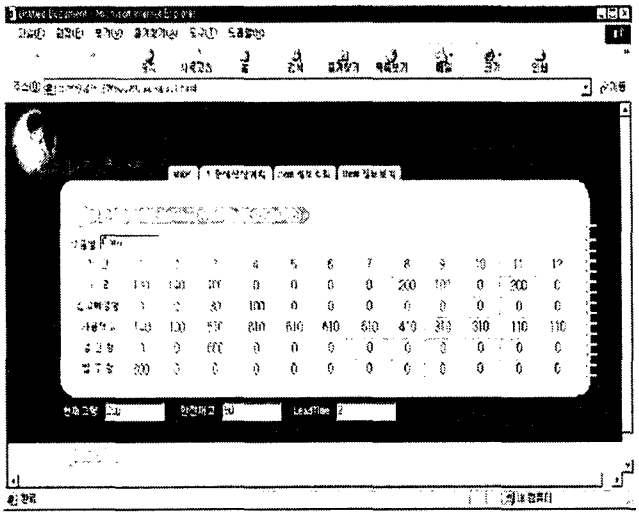


< 그림 3.3.2 > Item 상세정보

< 그림 3.3.2 >에서는 검색한 제품의 제품ID, Lead Time, Lot size, 가격정보, 안전재고, 안전 lead time 그리고 재고량 등 상세정보를 검색 할수 있다.



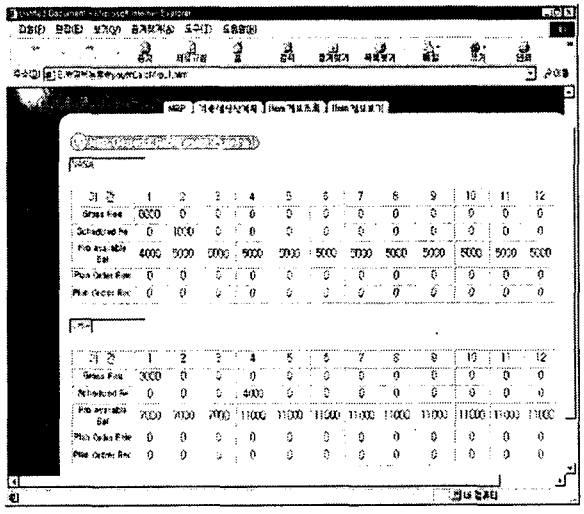
< 그림 3.3.4 > MPS 계산



< 그림 3.3.3 > 제품 주문 윈도우

< 그림 3.3.3 >에서는 위에서 검색한 제품의 주문 과정을 보여주는 윈도우다. 각각의 기간동안 주문량을 입력함으로써 다음에 수행될 MPS계산 프로세스의 입력값으로 계산된다. 주문량 입력은 생산능력이 상당히 유동적이고 제품시장 변동이 작다는 가정을 갖는다면 주문주기에는 영향을 받지 않게 되기 때문에 실시간으로 주문정보를 입력할 수 있게 된다.

< 그림 3.3.3 >에서는 MPS계산 프로세스를 보여주는 윈도우이다. 주문량을 입력받아 Lot size, lead time 그리고 안전재고를 고려하여 계산되는 프로세스이다. 이러한 모든 입력 데이터는 생산전략이 수립되고, 총괄생산계획을 세운다음 생산능력계획을 고려한 결과이다.



< 그림 3.3.5 > MRP 계산

< 그림 3.3.5 >은 MRP 프로세스를 보여준다. BOM 데이터를 참고하여 각각의 구성 Item의 소요량을 계산한다. 본 논문에서 구현된 MRP 계산 어플리케이션(Application)은 주문정책에 따른 생능력을 고려하기 때문에 Pull-System 관점에서 구현하였다.

IV. 결론 및 제언

하루가 다르게 발전하고 있는 IT기술의 흐름은 개발자들의 고민을 해결해주지 못하고 더욱더 어려운 환경으로 이끌어가고 있다. 컴포넌트가 21세기 정보기술(IT) 산업의 지각변동을 몰고 올 새로운 패러다임으로 급부상하고 있다.

컴포넌트는 SW를 이제까지처럼 계획, 설계, 분석, 구현, 테스트하는 순차적인 개발과정으로 보는 것이 아니라 레고블록이나 자동차 부품처럼 조립하는 개념으로 간주하기 때문에 현재 IT산업을 이루고 있는 기존 틀과 구조를 깨뜨리고 새로운 시장 질서를 창조할 것으로 전망된다.

컴포넌트가 각광받는 이유는 개발 및 유지보수 생산성과 SW 품질 향상을 획기적으로 향상시킬 수 있는 개념이기 때문. 컴포넌트 SW는 레고블록을 쌓는 것처럼 일단 표준화된 프레임워크와 규칙을 기반으로 만들어진 기능별 SW를 필요한대로 조립하면 원하는 SW를 만들 수 있다. 개발이나 유지보수에도 특별한 노력이 요구되지 않으며 표준화, 규격화를 지향하기 때문에 SW 품질도 눈에 띄게 좋아진다. 이러한 컴포넌트의 특장점은 본 논문의 구현 결과에서 나타났듯이 인터넷이라는 웹환경의 부상으로 더욱 강력한 위치를 차지하게 됐다. 인터넷은 모든 업무를 빠르게 변화시키고 있으며 표준화할 것을 요구하고 있으며 이러한 업무를 지원하는 SW 및 정보시스템도 표준에 기반해 빠른 변화를 수용해야만 하는데 이러한 요구에 가장 부합하는 것이 바로 컴포넌트 개념이다. 특히 앞으로 IT 라이프사이클이 점점 짧아지고 기업 인수합병(M&A)이 보편화함에 따라 정보시스템 교체 및 신규수요가 급증할 것으로 예견되는 상황에서 이 같은 컴포넌트형 SW의 중요성은 점점 더 커질 것으로 예상된다.

본 논문은 이러한 환경적 배경과 발전 방향에서 계약-협동 다이어그램을 이용하여 MRP 관리 프로세스에 적용하였다. 본 연구의 산출물은 다음과 같다.

- MRP 관리 도메인을 적용한 계약-협동 다이어그램
- 계약-협동 다이어그램으로부터 재사용 가능한 컴포넌트 추출
- 추출한 각각의 컴포넌트를 EJB기반의 Web Application Server인 PowerTier로 구현

본 연구에서는 생산 관리 도메인을 모델로 하였다. 더 나아가 전략적 자원 관리(ERP) 시스템에 적용하여 누구나 필요시 재사용 가능한 형태의 컴포넌트를 사용 가능하도록 설계 및 개발되어야 할 것이다. 그리고 앞으로 해결해야할 대전제는 이러한 분

산객체 기술을 적용함으로써 정보시스템 개발자는 고객의 요구사항에 따라 비즈니스 로 로직만을 컴포넌트 형태로 구현하여 이들이 상호 연계되어 실행될 수 있도록 룰을 정해주는 것이고, 그 외의 것은 모두 분산객체 기술에 의존하자는 것이다. 이러한 방법은 전체 시스템의 안정성을 획기적으로 높이는 코드 재활용을 극대화하게 된다. 결과적으로클라이언트는 어느 서버와 통신이 이루어지고 있는지 전혀 알 필요가 없게 된다. 이렇게 클라이언트 개발자로 하여금 서버의 세부사항을 숨겨줌으로써 각 모듈의 독립성이 높아지며, 결과적으로 향상된 개발 생산성고 효율적인 유지/보수가 가능하게 될 것으로 판단된다.

V. 참 고 문 헌

[1] 김운용, 최영근, "패턴적용을 통한 기능분할 GUI 컴포넌트 구조 모델", 한국정보처리학회 추계학술발표논문집, 제7권 2호, pp.547-550, 2000.

[2] 배두환, "컴포넌트 기반 방법론", 프로그램세계, pp06-07, 2000.

[3] 김창욱, 전진, 김성식, "객체지향 비즈니스프로세스 모델링 : 계약-협동 네트 모형" , pp25-30, 2000.

[4] Clemens, S. "Import is Not Inheritance Why We Need Both : Modules and Classes," , ECCOP' 92 , pp19-32 , 1992.

[5] Cuno, P., and Clemens, S. "Why Objects Are Not Enough", CUC'96, pp15-19, 1996.

[6] Herman, L., and Stanley, Y. "Component Interoperability in a Virtual Enterprise Using Events/Triggers/Rules", *http://smart.npo.org*.

[7] Markku, L., and Ari, J. "Extending the Object-Oriented Software Process with Component-Oriendted Design", Journal JOOP, JULY, pp24-35, 1998.

[8] Richard, M. *Enterprise JAVABEANS* , O'reilly, pp123-127, 1998.

[9] Won, K., and Ki-joon, C. "Component-Based Knowledge Engineering Architecture", Journal JOOP, OCTOBER, pp59-60, 1999.

[10] Schmidt, Douglas, "Pattern-Oriented Software Architecture Vol.2 H/C", John Wiley & Sons, Inc., 2000.

[11] Cooper, James William, "Java Design Patterns : A Tutorial", Addison Wesley Longman, Inc., pp17-18, 2000.

[12] Ed Roman, "Mastering Enterprise JavaBeans™ and the Java™ 2 Platform, Enterprise Edition", John Wiley & Sons, Inc., pp314-321, 1999.

[13] "Business Process Modeling Language (BPML) Working Draft v0.4," BPMI.org, *http://www.bpmi.org/bpmi-downloads/WD-BPML-20010308.pdf*, Mar. 2001.

[14] D. A. Chappell, V. Chopra, et al. *Professional ebXML Foundations*, Wrox, 2001.

[15] H. E. Eriksson and M. Penker, *Business Modeling with UML: Business Patterns at Work*, Wiley&Sons, 2000.

[16] van der Aalst and K. Hee., "Business Process Redesign: A Petri-net-based approach," Journal of Computers in industry, Vol. 29, No. 1, pp. 15-26. 1996.

[17] A. Tsalgatidou, et al, "Multilevel Petri nets for Modeling and Simulating Organizational Dyn amic Behavior," *Simulation & Gaming*, Vol. 27, No.4, pp.484-506, Dec. 1996.
[18] D. Moldt and R.Valk, " Object oriented Petri nets in Business Process Modeling," *Lecture Notes in Computer Science*, Vol. 1806, pp.254-273, 2000.
[19] H. Gou, B. Huang, S. Ren, "A UML and Petri net integrated modeling method for business processes in virtual enterprises," *Proc. 2000 AAAI Symp. - Brint Knowledge to Business Process*, pp. 142-144, March 2000.
[20] van der Aalst , A Class of Petri nets for Modeling and Analyzing Business Process, *Computi ng Science Report*, No. 95/26, Eindhoven University of Technology, 1995.
[21] B. Bohem, COCOMO home page, http://sunset.usc.edu/research/COCOMOII/index.html.

저 자 소 개

서 장 훈 : 명지대학교 산업공학과를 졸업, 동 대학원 산업공학과 석·박사 취득, 아주대 경영학석사(MBA), 현재는 한국능률협회 컨설턴트로 활동중이다. 주요 관심분야는 e-Business 조직 및 전략경영, 품질공학, Data-Mining. 6sigma. IT-관리프로세스 평가이다.