2000-06

CBD 이해와 활용 (Component Modeling 편)

이찬경과장(10Dr.com 포트, 기술사)
chanlee@samsung.co.kr

구조적 방법론, 객체지향 방법론이 정립되는 과정을 보면 개발 기술이 어느 정도 보편화되고 나서야 방법론이 정립되곤 했다. CBD 의 경우도 마찬가지인 것 같다. CBD 개발기술에 대한 참고서적과 자료는 많이 나와 있는 반면에 상대적으로 CBD 분석, 설계 즉 component Modeling 에 대한 참고서적은 찾아보기가 쉽지 않으며, component modeling 을 제대로 이해하고 component 를 개발하는 개발자들도 그리 많은 것 같지 않다.

Component modeling 에 대한 방법론은 아직 없는 것일까?

그렇지는 않다. Microsoft, Rational, Anderson consulting 등 소프트웨어 메이저 벤더들은 나름대로의 component modeling 에 대한 방법론을 가지고 있지만 이런 것들이 회사 내부용이거나 고가의 교육 또는 컨설팅 비용을 지불하고 얻을 수 있는 것이어서 일반 개발자들의 접근이 어려운 것이다. 또한 그 회사가 가지고 있는 상품이나 환경에 의존적이서 범용적으로 적용하기에는 무리가 있는 것도 사실이다.

본 보고서에서는 필자가 습득한 여러 component modeling 에 대한 여러 지식 중에서 Microsoft 사의 MSF DCS 와 Rational 사의 UML 에 대한 장단점을 분석하여 이를 보완, 통합하여 component modeling 에 대한 이해와 수행을 보다 쉽고 간결하게 하는 component modeling 방법론을 제시하고 한다.

1. MS 의 DCS 와 Rational 의 UML

Component Based Development 라는 개념이 IT 업계의 관심을 끌기 시작한 것도 얼마 되지 않았고, component 제작 방법에 대한 참고 자료도 충분하지 않다. 객체지향 방법론을 적용하면 되지 않을까 하는 생각이 들지만 CBD 개념 편에서 언급한 바와 같이 객체지향과 CBD 에는 차이가 존재한다.

현재 까지 일반적이거나 표준화된 component modeling 방법은 찾아 보기 어렵고, 다만 각자 기존의 방법을 수정하여 적용하고 있는 실정이다. 따라서 본 보고서에서는 Object Objected Modeling 의 defacto standard 가 되어가고 있는 UML(Unified Modeling Language)에서 제공하는 분석 설계의 도구들과 Microsoft 의 COM(Component Object Model) component modeling 방법인 DCS(Design Component Solution)에서 제공하는 component 제작 절차를 적절히 혼합하여 일반적이고 범용적으로 적용할 수 있는 component modeling 방법을 제시하고자 한다.

이에 앞서 먼저 DCS 와 UML 의 내용과 장단점을 을 간단히 살펴보기로 하자.

1.1 MS 의 Design Component Solution

실제 MS 의 개발자들이 COM component 을 개발하는 방식을 보면 scenario 와 specification 기술에 중점을 두고 있다. 분석 설계 tool 과 method 는 개발자의 취향에 따라 여러 가지를 선택하여 사용하고 있으며, 어떠한 도구를 사용하던지 component 설계의 밑 바탕에 흐르는 개념은 MSF(Microsoft Solutions Framework) DCS 에 기본을 두고 있다.

MSF DCS(Design Component Solution)는 MS 의 개발자들의 다년간 쌓아온 개발 경험으로부터 탄생된 COM component based application 구현을 위한 framework 이다. MSF 자체가 methodology 가 아니라 framework 을 표방하고 있어 분석 설계과정에서 목표와 많은 절차를 제시하고 있으나, 특정한 도구나 methods 는 제시하지 않고 있다. 설계자가 임의로 선택하도록 권고하고 있는 것이다. 따라서 구체적인 실행 방법에는 언급이 부족하다. (특히 Physical Design 에 대한 부분은 상당히 개념적인 수준이다.)

MS 의 방법론 guideline 인 MSF DCS 에 따르면 다음과 같은 절차를 권고 하고 있다.

- Conceptual Design
- Logical Design
- Physical Design

각 단계마다 달성하여야 할 목표를 정의하고 있으나 목표를 달성하는 방법은 분석 설계자의 재량에 맡기고 있다. MS 의 이러한 방법은 분석 설계자의 능력과 취향을 최대한 존중한다는 점과 CBD 에 대한 개발 know-how 가 자체적으로 많이 쌓여있고 표준화되어 있어, 분석 설계 결과물을 통한 의사소통에 별 문제가 없다는 뜻일 것이다. 하지만 이러한 방법은 application 전체 기능면에서 component 를 바라보는 것이 아니라 component 자체 기능에 초점을 맞추게 할 우려가 있다. 즉 어떠한 component 에 기능을 먼저 정의하고 그 component 로 application 을 구성하는 식이 되는 것이다.

우리가 MS 처럼 이러한 방식을 사용한다면, component specification 의 일관성과 정확성에 대한 보증을 할 수 없으며 산출물의 유지 관리 및 의사 소통에 문제가 발생될 가능성이 크다. 그리고 자칫 component 간의 상호 연관 관계가 고려되지 않을 수 있으므로 component 간 coupling 이 증가할 가능성이 있다. 따라서 아직까지 충분한 분석 설계 능력이 뒷받침되지 않은 우리에게는 Conceptual, Logical, Physical Design 단계마다 표준화된 방법과 절차가 요구된다 하겠다.

1.2 Rational UML

객체지향 분석, 설계 방법론 중의 하나로 UML 이 표준화되어 가고 있다. 이는 순수 object modeling 방법으로 주요 절차는 다음과 같다.

- Use Case 정의
- Use Case 별 scenario 작성
- Scenario 별 Sequence Diagram 또는 Collaboration Diagram 작성
- Logical Classes 추출 및 Logical Package 작성
- Class Diagram 작성
- 주요 Classes 에 대한 State-Transient Diagram 작성
- Physical Package 작성
- Logical Package, class 를 physical package 로 mapping
- Component Diagram 작성

물론 각 단계 별로 위 활동들이 반복되고 각 활동에 대한 정확한 목표와 도구를 제시하고 있다. 그러나 component 설계와는 내용적으로 interface 정의 부분과 implementation inheritance 에서 차이를 보인다. 또한 component 를 분리하는 부분이 상당 부분 경험에 의존적인 것으로 남아 있다. 그리고 Class Diagram 또는 State-Transient Diagram 등 Component Modeling 에는 불필요하거나 부담되는 부분이 존재하기도 한다.

결론적으로 DCS 에 제시하는 각 절차를 UML 에서 제시하는 methods 를 이용하여 수행하는 것이 component 를 설계하는 방법으로 가장 적합한 것으로 판단된다. Component 특성인 interface 등을 고려하면서 UML 의 도구를 선택적으로 추출하여 Conceptual 과 Logical Design 의 수행 방법을 구축하고, MS 의 COM Component 기반에서의 Physical Design 방법을 구현하고자 한다.

2. Conceptual Design

2.1 단계 목표

Conceptual design 단계에서는 단순히 application 에 대한 사용자의 요구 사항을 수집하는 일 뿐만 아니라 system 이 풀어야 할 문제점을 파악하고 이를 개발자나 사용자가 명확히 이해할 수 있는 방법으로 정의하는 과정이다.

- User 가 application 에 대해서 원하는 것이 무엇인가
- application 제공하여야 할 기능과 문제점은 무엇인가

Conceptual Design 의 목적을 달성하는 도구로는 Use Case 와 scenario 를 이용한다. Use Case 는 application 외부에서 application 의 서비스를 정의한 것이고 scenario 는 각 서비스 별 사용 줄거리이다.

특히 scenario 를 통해 business 측면에서의 요구 사항을 잡아낼 수 있으며, 특정 프로세스나 기능이 어떻게 사용되어야 하는가에 대한 요구사항을 쉽게 나타낼 수 있다. 또한 프로젝트 팀원이 scenario 을 통하여 application domain 과 문제를 정확히 인식하게 되고 공통된 시각을 가지게 된다. 또한 사용자와 의사 소통을 원활히 하는 도구이기도 하다.

따라서 분석자는 application 의 Use Case 와 scenario 를 개발하고 찾음으로써

Conceptual Design 단계를 진행한다. 특히 scenarios 는 business problem 을 해결하려는 사용자의 관점을 표현하고 business problem 과 사용자의 요구 사항을 만족하는 solution 을 찾는 데 중점을 두어야 한다.

2.2 절차

단계의 목표를 달성하기 위한 도구로 UML 에서 제공하는 Actor 와 Use Case 를 정의하고 Use Case diagram 과 시나리오를 작성한다.

1) Business Goals 설정

먼저 사용자와 interview 또는 자료 조사 분석을 통하여 application 이 달성해야 할 목표와 scope 을 설정한다.

2) Definition of Actor

Application 에 특정 기능을 요청하거나 event 를 발생시키는 외부 객체를 Actor 로 정의한다. Actor 는 application 사용자일 수도 있고 외부 시스템 또는 분석 domain 밖에 또 다른 객체가 될 수도 있다.

3) Definition of Use Cases

하나의 Actor 에 의해 발생되는 event 에 대해서 application 이 반응하는 일련의 transaction 을 하나의 Use Case 로 정의한다. 즉 Actor 가 특정 목적을 가지고 application 과 interaction 하는 것을 정의한 것이다. 즉 application 이 외부 Actor 에게 제공해야 되는 기능, Actor 가 application 에 바라는 기능을 포괄적으로 명명한다.

하나의 Use Case 안에는 여러 packages, classes 그리고 하위 Use Cases 가 포함될 수 있다.

4) Scenario Description

하나의 Use Case 에는 여러 개의 scenario 가 존재할 수 있으며, scenario 안에는 application 이 풀어야 하는 business problem 의 해결책과 사용자의 요구 사항이 반영되어야 한다.

이 scenario 는 Logical Design 단계에서 어떤 이벤트와 logical class 가 추출되어야 하는지에 대한 근거를 제공하게 된다.

5) Use Case Diagram 작성

Conceptual Design 단계의 산출물로 그림 2-1)와 같은 Use Case Diagram 을

작성한다. Use Case Diagram 으로 Actor 와 User Case 들로 구성되어 있으며 이 들간의 dependency 가 화살표로 나타나 있다. 이 diagram 은 application 구축의 시발점이 된다.

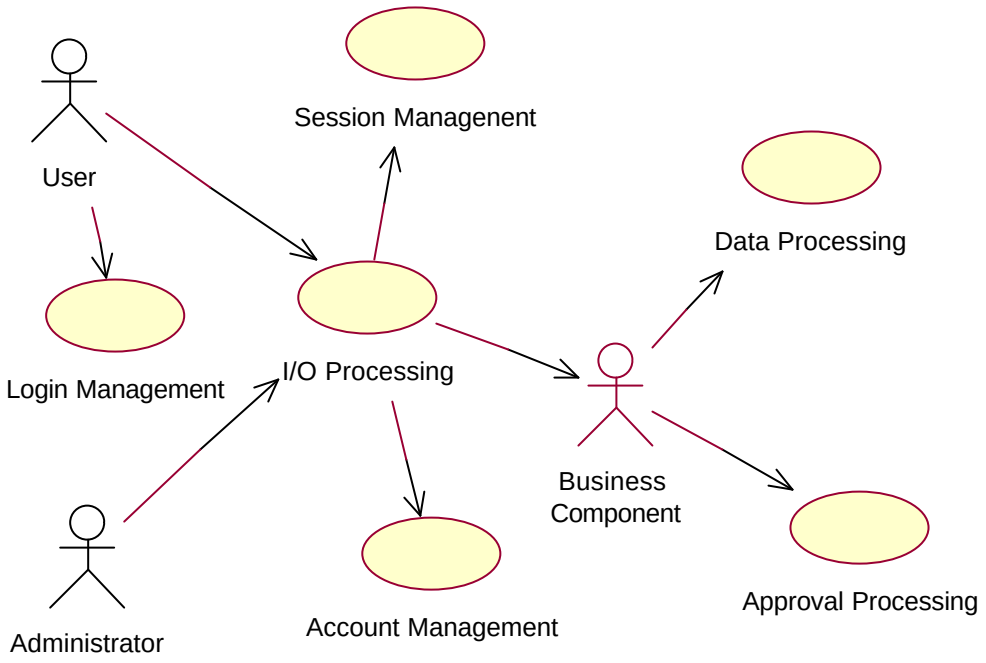

그림 2-1) Use Case Diagram 의 예

3. Logical Design

3.1 단계 목표

이 단계에서는 conceptual 단계에서 도출한 Use Case 와 시나리오를 토대로 logical class 를 추출하고, class 가 제공해야 할 service 를 정의하는 일, 또한 class 간 연관관계를 정의하는 것이 주요한 task 이다.

Logical Design 단계에서는 Sequence Diagram 과 Class Diagram 을 도구로 사용한다. 이 두 가지 도구는 병렬 작업으로 수행된다. Class Diagram 에서는 추출된 class 의 내용을 정의하고 class 간의 dependency 가 기술된다. Class Diagram 을 작성할 때 처음에는 class 의 상위 집합인 logical package 단위로 표현하고 단계가 진행되면서 package 안에 local class 로 상세화하도록 한다.

Sequence Diagram 은 앞에서 도출된 scenario 를 각 class 간의 상호 작용으로 표현한 것이다. Scenario 를 통해서 class 간에 전달되는 message 와 data 가 명확해지고, 새로운 class 가 발견되기도 한다. 발견된 새로운 class 는 Class Diagram 에 다시 추가시키고 다른 class 와의 dependency 를 정의한 후 추가된 class 를 포함하여 Sequence Diagram 을 다시 작성한다.

Sequence Diagram 대신 Collaboration Diagram 을 이용할 수 있다. 이 diagram 은 data 흐름을 Sequence Diagram 보다 명확히 파악할 수 있다. 그러나 scenario 관점에서 볼 때, 하나의 프로세스 흐름과 class 간의 상호작용을 파악하기엔 Sequence Diagram 이 더 효과적이다.
그림 3-1)과 같이 두 가지 도구는 상호 interactive 하게 작용하며 보완적 관계에 있다.

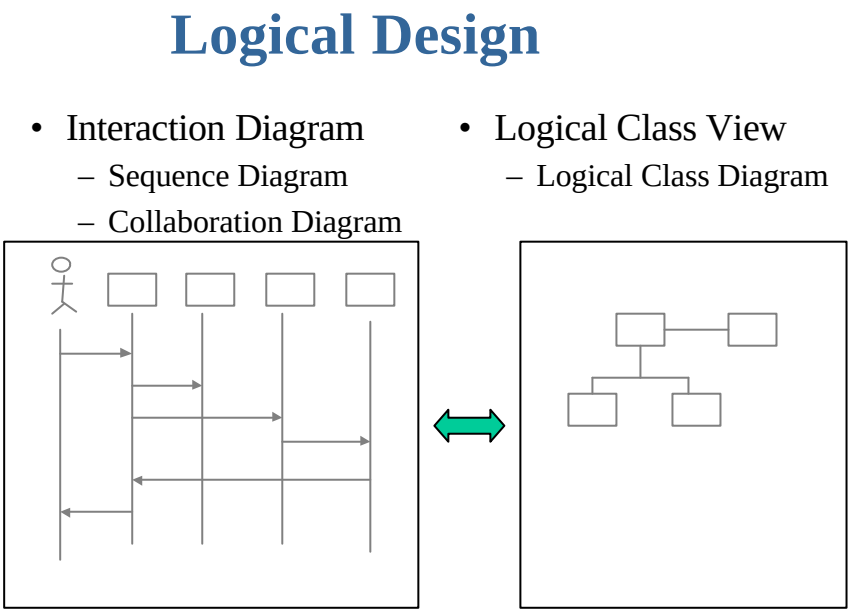

그림 3-1) Sequence Diagram 과 Class Diagram

3.2 절차

앞에서 언급되었듯이 아래의 각 절차는 class 가 발견될 때 마다 반복되어야 하며, 반복되면서 상세화되어야 한다.

1) Logical Class 도출

Conceptual Design 에서 도출한 Use Case scenario 를 통해 application 에서

사용되는 logical class 를 추출한다. Logical class 는 일종의 business object 나 service 로서 scenario 상에서 반복적으로 사용되는 명사나, 주요 기능을 class 로 추출한다.
Class Diagram 을 이용하여 class 간의 연관관계를 표시하고 그룹화한다. 이 때 주의 할 것은 inheritance 를 위한 class 의 일반화(generalization) 또는 상세화(specialization)는 하지 않으며 서비스 layer 와는 관계없이 비슷한 기능이나 같은 category 의 서비스는 logical package 로 그룹화 한다.

2) Sequence Diagram 작성.

Sequence diagram 은 scenario 하나 하나를 diagram 으로 표현한 것으로 앞 단계에서 도출된 logical class 간의 상호 작용을 중점적으로 표현하며 논리적으로 application 에 추가되어야 class 를 찾아 내는 것이 주요 목적이다.
Sequence Diagram 이 상세화 되는 과정에서도 scenario 가 담고 있는 사용자 요구 사항은 그대로 유지되고 있는 지 계속적으로 검사해야 한다.

3) Class 상세화

Sequence Diagram 을 통하여 하나의 class 가 전달 받은 message 를 method 로 전환하고 method 안의 parameters 를 정의한다.
위의 과정 중에서 새로운 logical class 가 발생될 수 있고, 발생될 경우 새로운 class 를 추가하여 위 과정을 다시 반복한다.

4) Logical Package 작성

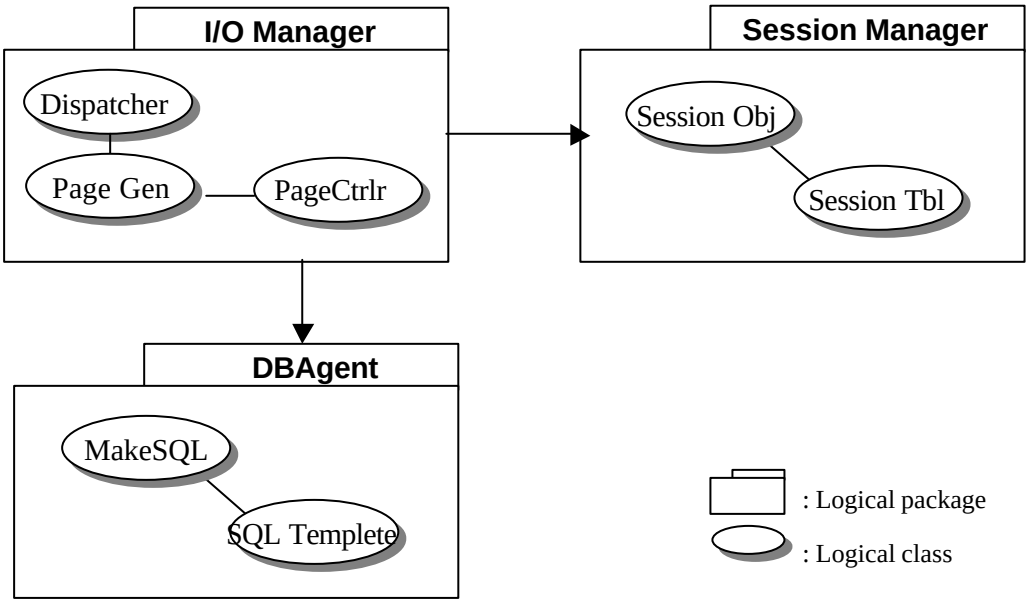

그림 5-2) Logical Class Diagram

Logical design 의 산출물은 logical package 에 대한 class diagram 의 상세화이다. Logical package 라는 것은 application 이 포함하고 있어야 할 기능들을 그룹화한 단위로, 하나의 package 는 여러 logical classes 로 구성되어 있고 package 간에는 dependency 를 가지고 있다. Logical design 단계에서의 classes 는 특정한 기술 또는 제품과는 연관되지 않은 상태이다.

4. Physical Design

4.1 단계 목표

Physical Design 의 주요 목표는 logical 단계에서 정의된 논리적 solution 이나 service 들을 물리적인 component 단위로 packaging 하는 것이다. 이 때 performance, administration, development process requirements 등이 고려되어야 한다. 또한 기존의 infrastructure 와 technology 을 도입하여 위험요소를 최소화하고 개발 기간을 단축하도록 하는 것이 Physical Design 의 주요 목적이다.

Logical Design 에서는 구현 기술과는 독립적인 관점에서 진행되었다면, Physical Design 에서는 구현 기술 운영 환경을 선택하고 이를 고려하여 component 가 설계되어야 한다. 따라서 본 보고서에서는 Microsoft 의 플랫폼을 개발환경으로 가정하여 설명하겠다.

4.2 절차

Physical Design 부분은 DCS 나 UML 에서도 상세하게 언급되지 않은 것으로, 여기서 제시되는 절차와 방법은 필자의 경험으로 토대 구성된 것이다. Physical Design 을 위해 다음과 같은 8 개의 절차를 제시한다.

1. Split Classes into service layers from logical packages
2. Take Logical DB design to the Physical Design
3. Leverage System Services
4. Combine/Separate services as need
5. Define Interface and parameters
6. Determine COM class
7. Determine component
8. Package 화

위의 8 가지 절차는 Physical Design 단계에서 계속적으로 반복되며 수행하여야 한다.

1) Split into Service layer

Logical Package 또는 class 내의 methods 들을 User Service, Business Service, Database Service 3 개의 영역으로 분리해 낸다.
즉 하나의 Logical Package 는 기능 단위로 묶여져 있고, 하나의 기능을 수행하기 위하여 사용되는 methods 들을 세 개의 layers 로 분리하여 표현한다.

2) Take Logical DB design to the Physical DB Design

Logical Design 에서 storage 가 필요한 class 에 대해서 storage type 을 결정하고 3rd normalization 형태로 table 을 디자인한다.
이 step 에서는 DBA 의 도움을 받는 것이 적절할 것이다.

3) Leverage system service

Logical Design 단계에서 정의된 기능들 중에 시스템이나 상용화된 component 로 대치될 수 있는 부분을 추출하여 낸다.
예를 들어, login 처리는 NT login process 를 이용할 수 있으며, MTS, MSMQ, SQL-Server 등과 같은 Back Office 제품 군이 시스템의 일부 기능을 담당할 수도 있다. 또한 ADO, ASP object , MS object 등 이미 존재하는 범용 또는 전용 component 들을 시스템 안으로 도입하는 단계이다.
이러한 활동을 위해서는 domain 의 기술에 대한 충분한 지식을 가지고 있어야 하며, 향후 제품의 upgrade 가능성도 고려되어야 한다.

4) Combine/Separate services as need

System provided service 를 제외한 나머지 methods 를 토대로, 필요하다면 class 를 합치거나 분리하여 낸다. 즉 system provided service 부분과 설계 대상이 되는 부분을 구분하는 작업이다.

5) Define Interfaces

앞 단계에서 분리 통합된 class 에 대해서 methods 를 검증한다. Class 내부에는 여러 개의 methods 가 존재할 수 있다. 이 중 외부에서 사용되는 methods 를 찾아 내어 interface 화한다. Interface 를 정의할 때는 다음과 같은 사항을 고려하여야 한다.

- 비슷한 서비스는 통합한다.
- 재사용성을 고려한다.

- 될 수 있는 대로 일반화 시킨다.
- MTS Declarative security 를 고려
- MTS transaction 고려

위의 고려 사항 중에 마지막 두 가지에 대한 보충 설명이 필요할 것이다. Interface 를 정의하는 단계에서 "웬 MTS 가 튀어 나오는가?" 하고 의문을 가질 수 있다. 먼저 앞서 언급하였듯이 Physical Design 단계에서는 개발환경과 밀접하게 연관되어 설계되어야 한다고 하였으며 본 보고서에는 MS 의 플랫폼을 가정하여 설명함을 다시 한번 상기하자. 그런 경우 대부분의 application 에서는 MTS 을 사용하게 될 것임으로 MTS 의 특징을 고려하여 component 가 설계되어야 할 것이다. MTS 의 특징 중 설계 단계에서 고려되어야 할 항목이 security 와 transaction 인 것이다.

▷ MTS 의 Security

- Declarative Security MTS role define
- Programmatic Security MTS role 을 이용하는 programming

MTS 를 이용하기 위해서 먼저 MTS package 를 만들고 component 를 등록한다. Component 를 등록하게 되면 그 component 가 가지고 있는 interfaces 와 methods 를 MTS 에서 관리하게 된다. 이 때 package level 에서 interface level 까지 role 을 할당할 수 있다. Role 속에는 user 또는 user group 를 할당할 수 있다. 즉 할당된 user 나 group 만이 MTS package 안에 interface 를 이용할 수 있는 것이다. 이렇게 user 나 user group 을 role 에 할당하는 것이 MTS declarative security 이다.
Interface 를 추출하는 단계에서 MTS declarative security 를 고려하라는 말은 추출된 interface 가 사용자 권한에 따라 나뉘어 질 수 있도록 interface 를 추출하라는 것이다.

▷MTS Transaction

MTS 의 package 단위는 transaction 단위를 고려하여야 한다. MTS package 별로 transaction 속성을 주도록 되어있다. 이 는 transaction 을 효과적으로 처리하기 위함인데 하나의 transaction 에 관여하는 class 들은 같은 component 로 모듈화하는 것이 좋으며 상호 transaction 을 일으키는 component 는 같은 MTS package 에 등록하도록 한다.

6) Determine Parameters

Methods 의 parameter 를 정의하는 단계로서 사용될 language 에 따라 정의되어야 한다.

예를 들어 ADO 를 사용하려면 VB construct(variant) 를, OLE DB 사용하려면 C++ construct 를 사용해야 할 것이다. 물론 C++에서 ADO 를 사용할 수 있지만 variants 를 다루어야 하는 부담이 생긴다. (다음 ADO version 은 좀더 다루기 쉬운 data type 을 지원할 것이다.)

이 때쯤부터 구현 language 로 어떤 것을 쓸 것인가를 생각할 때이다. 다음 표가 language 를 결정하는데 참고가 될 것이다.

표 1) VB vs. VC++ for implementation

항목	VB 5.0	VC++
Easiness	Easy(3 배)	Difficult
Runtime Speed	Slow	Fast(1.5 배)
in/out	[in] [out] only	can create [out] param. support
Datatype	variants type only	MILD set, any data type
Custom Interface	not support	Proxy/Stub MIDL
COM Support	Excellent(DLL,EXE) COM control, Document	OK(ATL/MFC)

7) Determine Parameter Passing

Parameter 를 정의한 후 type 을 정의하는 순서로 network utilization 을 고려하여 결정하여야 한다.

▷Pass all parameters at once

가능하면 한 번에 필요한 모든 parameter 를 넘긴다. Parameter 를 넘길 때 사용될 수 있는 data type 에는 다음과 같은 것들이 있다.

- Safearray
- Long parameter lists
- Structure
- ADO recordsets
- Custom marshal objects

8) Determine Component class

이 단계에서는 추출된 methods 를 그룹화하여 component class 를 만든다. class 는 component 안에 포함되게 되며 component 를 분할하는데 중요한 영향을 끼친다.class 를 분리하는 데는 다음과 같은 사항이 고려되어져야 한다.

- 같은 functionality 를 그룹화한다.
- Public VC++ class
- Interaction more overhead than C++ class
- Not too large or small
- Consider MTS security
- Consider MTS transaction

9) Determine Component

Component class 을 물리적으로 그룹핑하는 작업이다. 이 때의 고려 사항은 다음과 같다.

- DLL 또는 EXE 로 만들 것인가를 결정한다.
- Highly cohesion loosely coupling

10) Packaging

MTS 의 package 에 어떤 component 를 그룹을 등록시킬 것인가에 대한 것으로 다음과 같은 고려사항이 있다.

- Security boundary server package (package 내부에서는 security 를 check 하지 않음)
- Speed 하나의 component 는 memory 상에 같은 process 이므로 빠르게 작동된다.
- Fault isolation MTS

지금까지 Physical Design 에 대한 과정 흐름을 그림 4-1)에 요약하여 나타내 보았다.

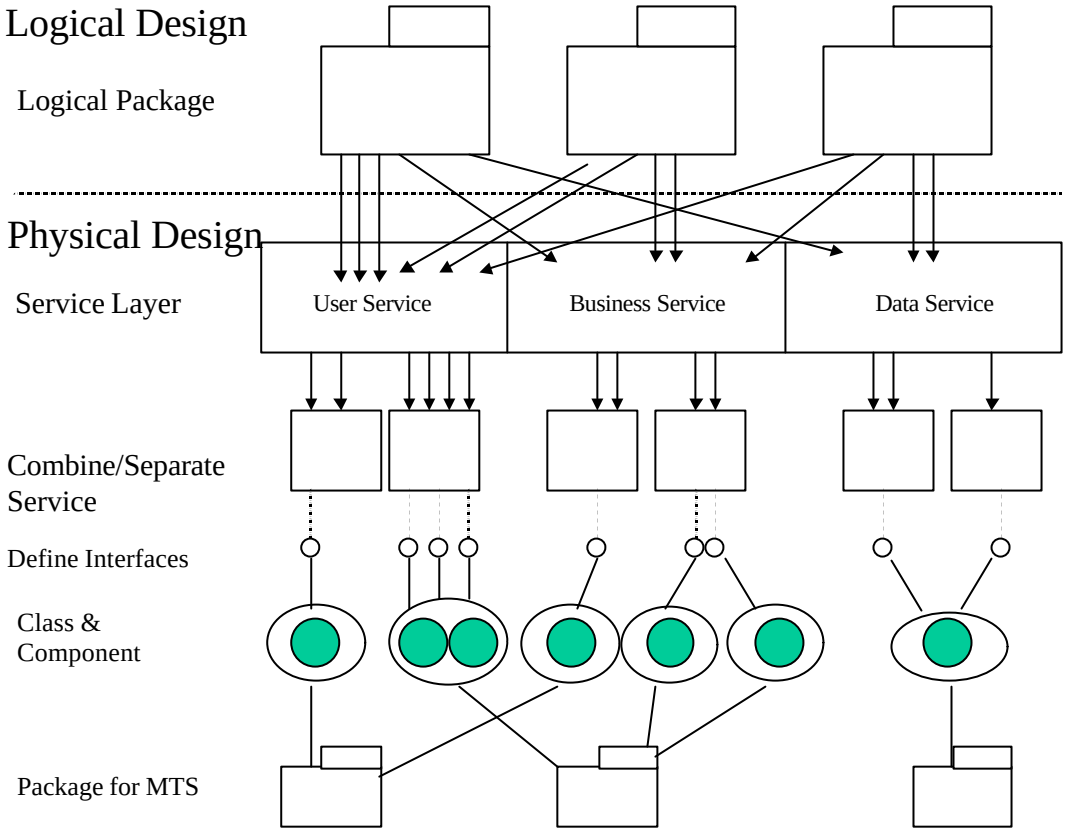

그림 4-1) Physical Design 흐름

5. 맺음말

일반적으로 방법론이라 하면 상당히 개념적인 것이 보통이다. 분석 설계 단계에서 제시된 분석 설계 방법을 그대로 적용한다고 하더라도 그 산출물을 그대로 구현 단계에서 적용하기에는 많은 부분에서 거리감이 발생하는 것을 많이 느껴 왔다.

Application type 이라든지 domain 환경 또는 개발 환경이 설계 단계에서 반영되지 않는다면 구현 할 때 이를 커버하기 위해 많은 공수를 투자하게 되는 것이다.

이런 관점에서 볼 때 보고서에서는 DSC의 절차와 UML 도구를 적절히 조합하여 실용적인 component modeling 방법을 제시하고자 하였다. Physical Design 단계에서는 MS 라는 특정한 개발 환경하에서 component 설계를 상세하고 실질적으로

접근하였지만 다른 개발 플랫폼일지라도 개념과 절차는 그리 다른지 않을 것이라 판단된다.

이러한 접근 방법이 비록 정답이라고 확신하지 않지만, component modeling 의 중요성에 대한 인식을 정립하고 , 여러 개발자들의 component 개발과정에 조금이라도 도움이 되는 참고자료가 되었으면 한다.