모델 구조 가이드라인
Rational Software Modeler, Rational Systems Developer 및 Rational Software Architect 용 ("일반 RUP" 오리엔테이션)

백서

Bill Smith, Model Driven Development, IBM Rational Software

V7.0
2006 년 3 월 24 일

목차

1. 소개

문서의 대상 정의

이 문서는 Rational Software Architect(RSA), Rational Systems Developer(RSD) 및 Rational Software Modeler(RSM) 제품(통틀어 "RS*x*")의 사용자를 지원합니다. 특히 Rational Unified Process(RUP)에 있는 모델링 안내를 RS*x*의 사용에 적용하고자 하는 사용자를 위해 작성되었습니다. Rational Software Modeler(RSM) 사용자인 경우에도 유용한 문서를 찾을 수 있습니다. 그러나 일부 섹션은 RSA 및 RSD 에만 유용한 기능을 반영함에 주의하십시오.

이 문서는 RS*x*에서 새로운 모델 세트를 작성하는 데 초점을 맞추고 있습니다. RS*x*는 처음 사용하지만 이전에 Rational Rose 또는 Rational XDE 를 사용한 경험이 있으며 이들 제품으로부터 모델을 가져오려는 경우, 이 문서를 통해 사용자가 가져온 모델을 다시 구조화할 수 있습니다.

이 문서에서는 사용자에게 RUP 및 UML 에 관한 일부 지식이 있다고 가정합니다. 또한 RS*x*의 기본적인 개념과 "오퍼레이션 이론에 "익숙하다고 전제합니다. 이에 대한 정보는 "The New IBM Rational Design and Construction Products for Rose and XDE Users를 참조하십시오.

목적

RUP 는 시스템의 솔루션 도메인과 문제점에 대해 제대로 정의된 관점을 표시하는 모델 세트(예: 유스 케이스 모델, 분석 모델 및 디자인 모델)를 설명합니다. 이 모델 세트의 유용성은 여러 실제 프로젝트에서 입증되었습니다. 따라서 RUP 를 따르지 않더라도 이러한 모델 구조를 고려할 가치가 있습니다. 이 문서에서는 RS*x*를 사용하여 이를 실현하는 방법을 설명합니다.

기타 모델링 접근 방식도 고려될 수 있겠지만, 이를 지원하는 모델 구조화 안내는 이 문서의 범위 밖입니다. 예를 들어, 서비스 지향 아키텍처의 모델링을 위한 "비즈니스 기반 개발" 접근 방식을 사용할 수 있습니다. 이 방식을 사용하면 비즈니스 프로세스 모델(UML 2 활동 모델로 표시될 수 있음)로 시작한 후 바로 비즈니스 프로세스에서 타스크를 자동화하는 서비스 세트의 계약 지정으로 진행합니다. 그러나 이러한 접근 방식에서 제안되는 모델 구조는 이 문서에 설명된 내용과 상당히 다를 수 있습니다.

이 문서에서 설명된 프로젝트 및 모델 구조는 가이드라인일 뿐, 의무사항이 아님을 기억하십시오. RS*x*에서 특정 RUP 아티팩트를 모델링할지 여부는 사용자 고유 개발 프로세스의 고려사항입니다. 이는 주로 프로젝트에 특정한 결정입니다. 또한 RUP 는 엄격한 프로세스 규칙 세트가 아님을 기억하십시오. 이는 프로세스 정의를 공식화하는 프로세스 *프레임워크*입니다. 이러한 프로세스 정의는 매우 공식적인 것부터 매우 간단한 것까지 범위가 다를 수 있습니다.

UML 모델링을 사용하는 방법 또한 매우 공식적일 수도 있고 매우 비공식적일 수도 있습니다. 모델을 구축 중 엄격하게 따라야 하는 정규 아키텍처 도면처럼 다루도록 선택하거나, 또는 모델을 디자인의 광범위한 개요를 제안하지만 프로젝트가 구현으로 이동하면 처분 가능한 것으로 간주되는 스케치처럼 다루도록 선택할 수도 있습니다. RS*x*는 모델링이나 프로세스 범위의 어디에서든 사용자를 지원할 수 있습니다. 이러한 관점에서 이 문서에서 제공되는 가이드라인의 목적은 사용자의 사고를 구속하기 위함이 아니라 사용자가 RS*x*의 기능을 사용하여 최적의 프로세스를 촉진하는 방법을 이해하는 데 도움을 주기 위함입니다.

RS*x*를 통해 모델을 단지 청사진이 아닌 자동으로 구현의 중요 부분을 생성할 수 있는 스펙으로서 사용할 수 있게 됨에 주의하십시오. 이는 모델에서 모델 및 모델에서 코드로의 변형으로 달성됩니다. 모델 기반

개발(MDD)을 수행하는 데 변환을 사용하면 모델 구조와 관련된 특수 관심사항이 생깁니다. 모델과 변환을 사용하여 모델 기반 개발(MDD)을 수행하려면 developerWorks의 Rational 디자인 및 생성 영역에서 RS*x* MDD 특정 자원도 참조해야 합니다.

범위

이 문서에서는 RS*x* 에서 RUP 모델 아티팩트를 표시하는 방법을 설명하고 이러한 아티팩트의 내부 조직 구조에 대한 가이드라인을 제공합니다. 다음은 설명되지 *않습니다.*

- RUP 모델 아티팩트의 개념적 기반을 다시 언급하거나 확장 설명을 제공합니다.
- 연관된 RUP 아티팩트의 시맨틱 및 다이어그램 컨텐츠를 지정하는 프로세스 또는 기법을 설명합니다.

RUP 아티팩트의 컨텐츠를 정의, 개발 및 모델화하는 방법에 대한 도구 중립적 정보는 RUP 를 참조하십시오.

RS*x* 모델의 컨텐츠를 개발하는 도구 특정 기법에 관한 정보는 다음을 참조하십시오.

- 제품 문서(학습서, 샘플, 온라인 도움말)
- RUP 구성의 도구 사용 도움말(이 백서는 그 일부임)
- developerWorks 에서 RS*x* 관련 자원

인쇄상의 규정

IBM Rational Rose 또는 XDE 에서 이주 중인 RS*x* 사용자의 관심사항에 대한 토론은 음영이 있는 배경의 테두리가 있는 텍스트 상자에 사이드바 방식으로 제공됩니다.

> *XDE/Rose*
> 이전 XDE 또는 Rose 사용자의 관심사항 토론

문서의 구조

다음에 오는 기본 개념 및 용어 섹션이 작업 용어를 설정하고 RS*x* 제품에서 모델이 구현되는 방법에 대한 일부 일반 정보를 제공합니다.

다음으로, RUP 모델을 RS*x* 모델로 맵핑 섹션에서는 RUP 에 의해 정의되는 모델 유형을 RS*x* 가 지원하는 방법에 대해 설명합니다.

그 다음으로 여러 유형의 모델을 구성하는 가이드를 제공하는 여러 섹션이 있습니다. 이러한 섹션의 일부에서는 프로세스, 모델링 접근 방식 및 아키텍처 제어의 관점에서 사용자가 선호하는 정도에 따라 모델을 사용하는 여러 방법에 대해 설명합니다.

마지막으로 팀에서 모델을 사용하는 것과 연관된 문제점에 대한 논의가 있습니다. 이는 파일 경합 및 병합을 최소화하도록 팀 구성원 간 모델의 공유를 가능하게 하고 규모를 관리하는 전략을 다룹니다.

2. 기본 개념 및 용어

모델

RUP 에서 모델은 "특정 관점에서의 솔루션 모델 또는 문제점의 완전한 스펙"으로 정의됩니다. 문제점 도메인 또는 시스템은 도메인 또는 시스템의 여러 관점을 표시하는 다수의 모델로 지정될 수 있습니다. RUP 는 다음과 같은 특정 모델 세트를 제안합니다.

- 비즈니스 유스 케이스 모델
- 비즈니스 분석 모델
- 유스 케이스 모델
- 분석 모델(디자인 모델에 포함될 수 있음)
- 디자인 모델
- 구현 모델
- 배치 모델
- 데이터 모델

RUP 자체는 도구와 무관합니다(tool-agnostic). RUP 에서 볼 때, 모델은 냅킨 또는 화이트보드의 그림, 모델링 도구의 일부 사항 또는 마음 속의 이미지일 수도 있습니다. 따라서 RUP 관점에서 모델은 논리적 개념입니다.

RS*x* 컨텍스트에서는 모델을 논리적 관점에서 설명할 수 있지만 실제적 관점에서도 설명할 수 있습니다. 두 응용프로그램에 대해 Timesheet 관리 응용프로그램에 대해 작업하는 세 명의 분석가로 이루어진 한 팀과 Call Center 응용프로그램에 대해 작업하는 다섯 명의 분석가로 이루어진 두 번째 팀이 있다고 가정하십시오. 두 팀 모두 요구사항을 캡처하기 위해 현재 작업 중이며 유스케이스 모델링에 RS*x*를 사용 중입니다. RUP 관점에서 보면 한 팀은 "Timesheet 응용프로그램의 유스 케이스 모델"을 빌드하고 다른 팀은 "Call Center 응용프로그램의 유스 케이스 모델"을 빌드 중이라고 할 수 있습니다. 그러나 팀이 RS*x*를 사용 중인 경우 모델에 실제 Manifestastion 이 있음을 인식해야 합니다. 이것이 다음 섹션의 주제입니다.

모델링 파일

RS*x* 모델은 파일로서 지속됩니다. (Eclipse 용어로 파일은 '자원'으로 간주됩니다.[1] 따라서 이 문서 또는 기타 소스에서 '모델링 자원' 용어가 나타나면 이는 '모델링 파일'과 동일한 의미입니다.) 광범위한 의미로 RS*x*는 두 가지 종류의 모델링 파일을 지원합니다.

- **사전 구현 모델링 파일.** 호스트 OS 파일 시스템에서 개별 파일로 저장됩니다. 이들 파일은 ".emx"라는 파일 이름 확장자를 가지며 다음을 포함합니다.
    - UML 시맨틱 요소(클래스, 활동, 관계 등...)
    - UML 시맨틱 요소를 보여주는 UML 다이어그램(또한 Java, C++ 또는 DDL 과 같은 기타 시맨틱 도멘인에 대한 대한 참조를 보여줄 *수도* 있음).

[1] Eclipse 에서 자원은 파일이지만 Eclipse 환경 내의 추가 특성 및 동작 또한 가집니다. 여기에 설명된 모델링 파일은 Eclipse 의 '자원'으로 취급됩니다.

- **구현** 모델 파일은 Eclipse *프로젝트로* Eclipse 작업공간에 저장됩니다. 프로젝트에는 다음 사항이 포함됩니다.
    - 구현 아티팩트(Java 소스 코드, 웹 페이지, XML 기반 메타데이터 파일, ...)
    - 구현 아티팩트을 직접 보여주고 반영하는 다이어그램 파일

  모델 시맨틱은 구현 아티팩트 자체에 있습니다. 예를 들어, Java 구현 시맨틱 모델은 직렬화되어 Java 소스 코드 파일의 콜렉션으로 저장됩니다. 각 다이어그램은 프로젝트의 자체 실제 파일에 있습니다. 다이어그램 파일은 다양한 확장자를 가질 수 있습니다. 가장 일반적인 확장자는 ".dnx"입니다. 구현 모델링 다이어그램은 코통 UML 표기법을 사용하지만 기타 표기법(예: IDEF1X 또는 데이터 시각화의 정보 엔지니어링 표기법 또는 웹 층을 디자인하는 데 사용하는 IBM 소유 표기법)을 사용할 수도 있습니다.

이 문서는 "사전 구현" 모델의 내부 구조를 조직하는 방법에 중점을 둡니다. **문서 내에서 "모델링 파일"이라는 용어는 "사전 구현" 모델링 파일**(.emx 확장이 있는 파일)을 의미합니다. 구현 프로젝트의 컨텐츠를 조직하는데 대한 안내는 기타 소스, 즉, Rational Software Architect 의 온라인 도움말, Rational Application Developer 및 Rational Web Developer Community Edition 에서 찾을 수 있습니다.

모델링 파일이 하나의 (논리적) 모델에 대한 모든 정보를 포함하지는 않습니다. 사실 모델링 파일에는 보통 모델의 서브세트만 포함됩니다. 예를 들어 위에 주어진 Timesheet 응용프로그램의 유스 케이스 모델에 대해 작업하는 세 팀으로 구성된 예제에서, 팀은 유스 케이스 모델을 실제로 세 가지 모델링 파일로 파티션되도록 선택하여 팀의 각 구성원이 동일한 파일을 사용하기 위해 경쟁하지 않고 유스 케이스의 다른 서브세트에서 작업하도록 할 수 있습니다. 이 문서의 마지막 섹션에서는 모델 파티션과 모델링 파일의 관리와 연관된 문제에 대해 설명합니다.

모델 유형

RUP 에서 모델은 유스 케이스 모델, 분석 모델 또는 데이터 모델과 같은 특정 유형입니다. RS*x* 에서 모델링 파일은 "엄격한 유형"이 아니라 "느슨한 유형"의 다양한 모델링 파일을 사용하는 규칙을 따를 수 있습니다. 이러한 규칙을 따르려면 다음 두 가지 방법 중 하나로 느슨한 유형을 생성할 수 있습니다.

- "공백" 모델링 파일(아래 참조)로 시작한 다음 해당 파일의 이름 지정 방법 및 파일에 넣을 컨텐츠(적용한 UML 프로파일 포함) 종류로 유형을 설정하십시오.
- 특정 모델 유형을 표시하는 사전 정의된 "템플리트 모델"을 기반으로 모델링 파일을 작성하십시오. RSx 제품은 이 문서에 설명된 모델 유형에 대해 기본 템플리트 설정을 제공합니다. 또한 사용자 고유 템플리트 모델을 작성할 수 있습니다. (developerWorks에서 제품 도움말과 포럼 및 기타 자원을 참조하십시오.)

두 경우 모두, RSx 에서 모델링 파일의 "유형"은 파일의 이름 지정 및 컨텐츠에 관련된 규칙입니다. 예를 들어, 이 도구는 유스 케이스 모델을 포함하는 파일이 해당 유스 케이스를 실현하는 클래스를 포함하지 않도록 해주지는 못합니다(논리적으로 말하면, RUP 는 분석 또는 디자인 모델의 일부로 고려됨).

이 안내서에서는 RSx 모델링 파일을 유형화하여 다룰 것을 권장합니다.

작업공간, 프로젝트 및 프로젝트 유형

Eclipse, WebSphere Studio 제품 또는 Rational Application Developer 에 익숙한 사용자는 파일이 프로젝트 내에 있으며, 해당 프로젝트는 여러 유형일 수 있고 그룹화되어 작업공간에서 관리될 수 있음을 이미 알고 있습니다. RS*x* 모델링 파일은 기타 다른 파일과 마찬가지로 프로젝트에 있습니다.

이 토론을 위해, RSx 및 Rational Application Developer 에 사용 가능한 모든 프로젝트 유형을 자세히 설명할 필요는 없습니다. 크게 보면 프로젝트의 다음 두 카테고리에 관심이 있습니다.

- UML **프로젝트**
- **구현 프로젝트**(엔터프라이즈 프로젝트, EJB 프로젝트, 웹 프로젝트 및 C++ 프로젝트와 같은 특수 프로젝트 유형을 포함).

앞서 RS*x* 는 다음과 같은 두 종류의 모델링 파일을 지원한다고 설명했습니다.

- 구현(예: 요구사항, 분석 및 디자인) 위의 추상 레벨에 있는 모델링에 사용하는 UML 모델을 포함하며 .emx 확장자를 가지는 파일.
- 구현 시맨틱을 포함하는 Eclipse 프로젝트(일반적으로 소스 코드 파일 아티팩트로 직렬화됨), 및 이 시맨틱을 반영하는 다이어그램.

프로젝트에 모델을 할당하는 규칙은 간단합니다.

a) "사전 구현" 모델링 파일을 UML 프로젝트에 두고

b) 구현 모델은 실제로 다음과 같으므로 스스로 처리할 수 있습니다.

[구현 모델] = [구현 프로젝트]

이 규칙에는 몇 가지 예외가 있습니다. 다음 UML 모델링 파일은 언어 특정 구현 프로젝트에 배치될 후보입니다.

- 디자인 '스케치 모델'(나중 섹션에서 설명됨)
- 테스트를 설명하는 시퀀스 다이어그램이 있는 모델이 프로젝트의 코드에 대해 실행됩니다.

> *XDE/Rose*
> Rose 및 XDE 오퍼레이션 이론에는 사용자가 코드와 동급의 추상 레벨에 도달할 때까지 반복적으로 디자인 모델을 정제한 다음 코드 모델 동기화 기법을 사용하여 코드 자체와 일치하는 해당 모델의 시맨틱을 유지하는 사례가 포함됩니다. 예를 들어, XDE 에서 구현 모델은 프로젝트의 코드 및 다이어그램뿐 아니라 구현 아티팩트에 독립적으로 지속하며 실제로 여분의 시맨틱 사본을 표시하는 '코드 모델' 파일로도 존재합니다.
>
> RSx 오퍼레이션 이론은 코드보다 상위 추상 레벨의 플랫폼 중립 모델(즉, 엔터프라이즈 IT 디자인 모델과 같은 디자인 모델)의 사용과 해당 모델에서 코드를 생성하는 변환의 사용을 장려합니다. 추상 레벨의 코드에서 RSA, RSD 및 RAD 는 추상 레벨의 구현에서 별도의 지속된 시맨틱 모델을 사용하지 않고도 UML 표기법에서 표현된 시맨틱 코드의 다이어그램을 그릴 수 있도록 합니다.
>
> RSx 가 추상 코드 레벨에서 UML 모델을 정의하지 않거나 해당 모델에서 코드를 생성하지 않도록 *막아주지* 않음에 주의하십시오. 사실, 이러한 유형의 사용법이 예상됩니다. 그러나 RSx 는 이러한 모델이 코드와 지속적으로 자동 동기화되는 기술을 제공하지 않습니다.

검토의 개념

다음 그림에서 위 설명 내용을 요약합니다. 위에서 설명한 대로, 그림은 한 팀은 Call Center 응용프로그램에 대해 작업하고 다른 팀은 Timesheet 관리 응용프로그램에 대해 작업하는 두 팀이 있는 시나리오를 반영합니다.

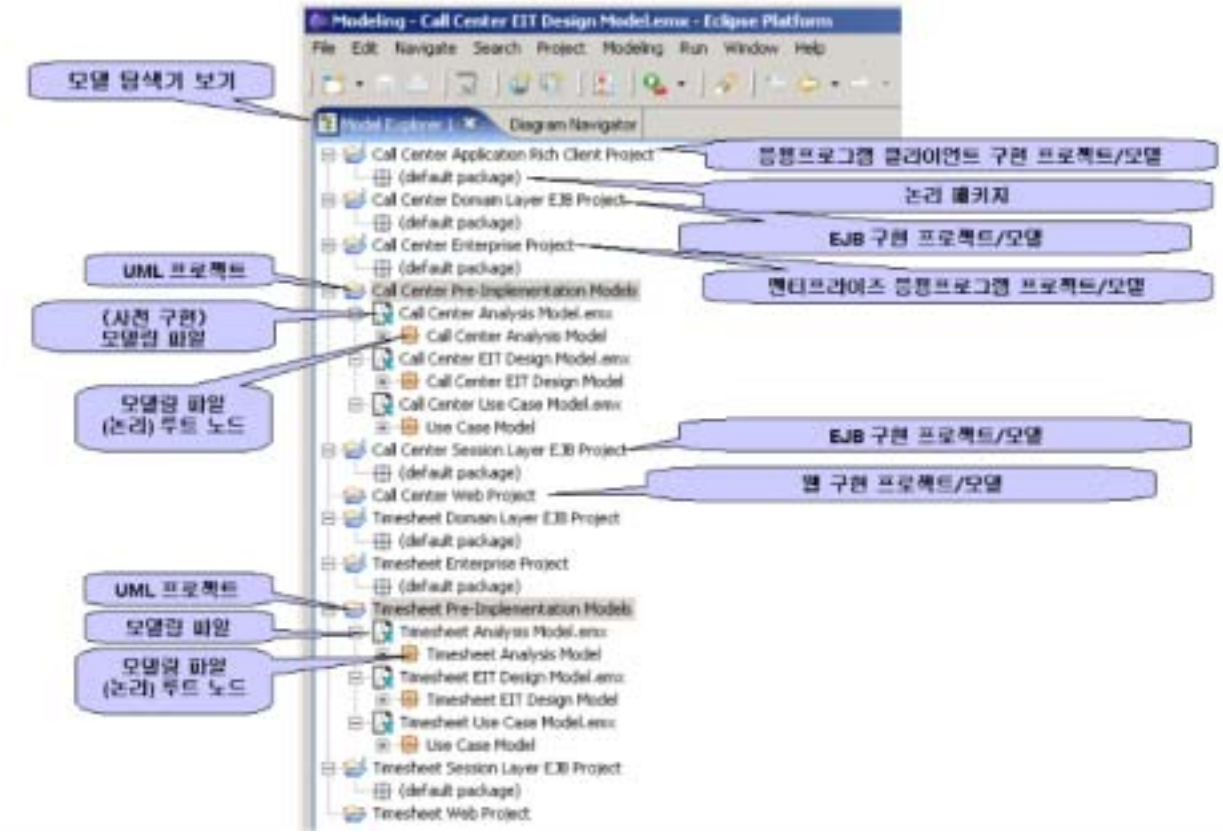


그림 2-1

*RSX*는 모델의 결합된 실제 및 논리 보기를 제공하는 탐색기 보기[2]를 제공합니다. 탐색기에 최상위 레벨 노드로 설명된 작업공간의 프로젝트가 표시되며 각 프로젝트 내에 해당 프로젝트에 속한 자원이 표시됩니다. **그림 2-1**은 모델 탐색기에 설명된 시나리오의 두 응용프로그램에 해당하는 프로젝트의 콜렉션을 설명합니다. 여기서 UML 프로젝트는 사전 구현 모델에 사용되었고 솔루션 적합 유형의 프로젝트 콜렉션은 구현 모델에 사용되었음을 알 수 있습니다.

[2] 6.x 버전에서는 "모델 탐색기"였습니다. 7.0 부터 모델은 범용 탐색기에 표시됩니다. 이 문서의 그림은 6.0 버전을 기준으로 하며 "모델 탐색기"를 보여주고 있습니다.

> *XDE/Rose*
> RSA 모델 탐색기와 반대로 Rose 및 XDE 의 모델 탐색기가 모델의 논리 보기만을 제공합니다. RSA 모델 탐색기에서 제공한 자원의 보기는 Eclipse 탐색기 보기가 제공하는 '순수' 실제 보기가 아님을 참고하십시오. 일부 실제 자원이 모델 탐색기에서 가시적인 반면 대부분 자원의 *논리* 보기를 표시하는 아이콘으로 표시됩니다.

그림 2-2 는 Timesheet 유스 케이스 모델이 일부 기능적으로 관련된 문제점 도메인의 서브세트를 표시하는 패키지로 내부적으로 조직될 수 있는 방법을 표시합니다.

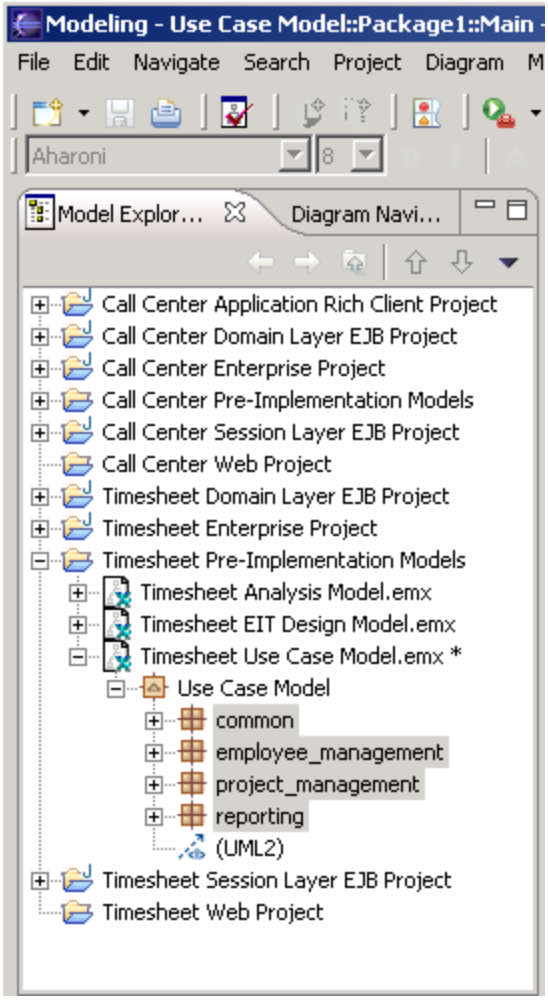


그림 2-2

그림 2-1 및 2-2 에서 각 사전 구현 모델은 단일 모델링 파일에 있습니다. **그림 2-2** 에서, Timesheet

유스케이스 모델은 문제점 도메인의 서브세트에 해당하는 여러 모델링 파일로 리팩토링될 수 있습니다. 각 모델링 파일의 루트는 전체 유스 케이스 모델을 구성하는 모든 모델링 파일에 걸쳐 일치하는 이름 공간을 유지하도록 이름 지정됩니다.

3. RUP 모델을 RS*x* 모델에 맵핑

다음 표는 가장 일반적으로 사용되는 RUP 모델을 RS*x* 모델링 파일 "유형"으로 맵핑하는 방법을 표시합니다. 맵핑은 보통 간단하지만 RSx 와 함께 RUP 를 실행하기 위한 가이드로 이 문서를 사용하는 핵심이 됩니다. 표에서 언급된 RSx 모델링 파일 유형은 표 바로 다음에 설명됩니다. 여러 모델링 파일 유형의 내부 조직 및 해당 모델 유형을 보유할 프로젝트 종류에 대한 가이드라인은 나중 섹션에서 제공됩니다. 이러한 나중 설명은 여기에 나열된 RSx 모델링 파일 유형 관점에서 제공됩니다.

RUP 모델	*RSx* 모델링 파일 유형
유스 케이스 모델	"유스 케이스 모델" 템플리트에 기반한 모델링 파일 (대체: 공백 모델링 파일로 시작, RUP 유스 케이스 모델 안내에 따라 컨텐츠 제한)
분석 모델	분석 모델 (대체: 공백 모델링 파일로 시작, RUP 분석 모델 안내에 따라 컨텐츠 제한) (대체: 디자인 모델의 <<analysis>> 패키지 사용)
디자인 모델	n 층 비즈니스 응용프로그램의 경우: "엔터프라이즈 IT 디자인 모델" 템플리트 기반 모델링 파일 (대체: 공백 모델링 파일로 시작, RUP 디자인 모델 안내에 따라에 따라 컨텐츠 제한) 기타 응용프로그램 유형의 경우: 공백 모델링 파일로 시작, RUP 디자인 모델 안내에 따라 컨텐츠 제한 디자인'스케치'의 경우: 공백 모델링 파일 선택적 보충사항: 구현 개요 모델로 사용하는 추가 공백 모델링 파일
구현 모델	구현 아티팩트 및 다이어그램 파일을 포함하는 Eclipse 프로젝트
배치 모델	공백 모델링 파일로 시작, RUP 배치 모델 안내에 따라 컨텐츠 제한

RS*x* 모델링 파일 유형

공백 모델링 파일

RSx 에서는 "공백 모델"을 작성하는 옵션을 제공합니다(파일→새로 작성→UML 모델→공백 모델). "공백 모델"은 모델 템플리트를 기반으로 하지 않는 모델링 파일입니다. 적용된 특수 프로파일이 없으며 단일 "기본" (자유 양식) 다이어그램을 제외한 기본 컨텐츠가 없습니다. **공백 모델링 파일을 임의의 유형의**

모델에 대한 시작점으로 사용할 수 있습니다. 파일 이름 지정 방법, 정의할 컨텐츠 및 적용할 프로파일을 선택함으로써 공백 모델링 파일을 사용하여 유스 케이스 모델, 분석 모델, 디자인 모델, 배치 모델 또는 기타 유형의 RUP 모델을 빌드할 수도 있습니다.

유스 케이스 모델링 파일

RSx 는 모델 템플리트를 기반으로 "유스 케이스 모델" 파일을 작성하는 옵션을 제공합니다. 템플리트는 *그림 3-1* 에 표시된 대로 기본 컨텐츠에 컨트리뷰션합니다. ("빌딩 블록" 컨텐츠와 검색 문자열이 사용되는 방법에 대한 설명은 이 문서의 영역 밖입니다. 템플리트에는 충분한 설명이 있는 지시사항이 포함됩니다.)

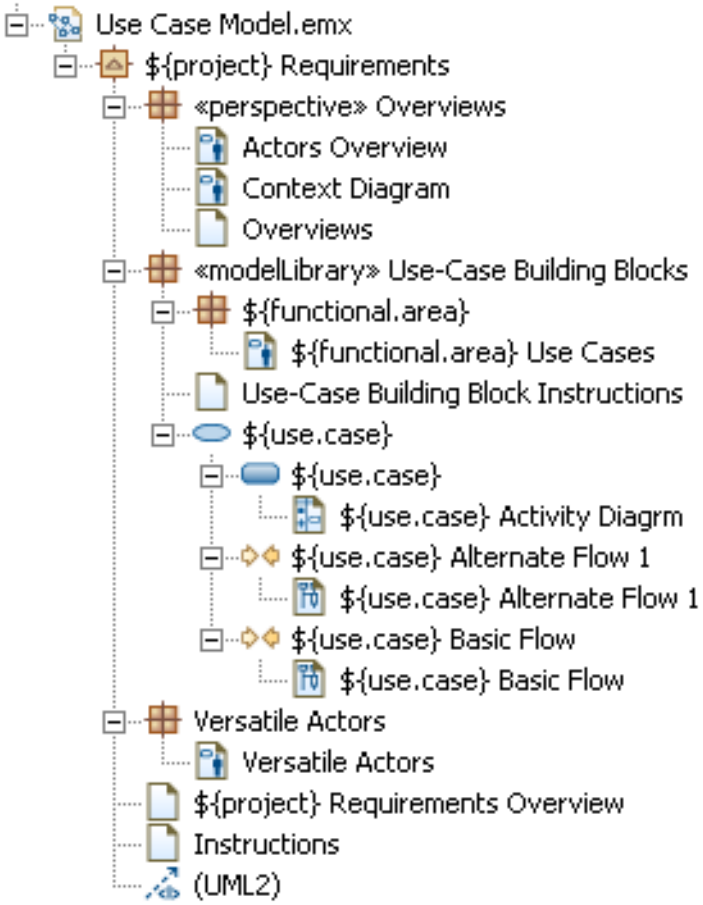


그림 3-1

분석 모델 파일

RSx 는 모델 템플리트를 기반으로 "분석 모델" 파일을 작성하는 옵션을 제공합니다. 템플리트는 *그림 3-2*에 표시된 대로 기본 컨텐츠에 컨트리뷰션합니다. 또한 "분석" 프로파일이 이 템플리트에서 작성된 모델 파일에 적용됩니다.

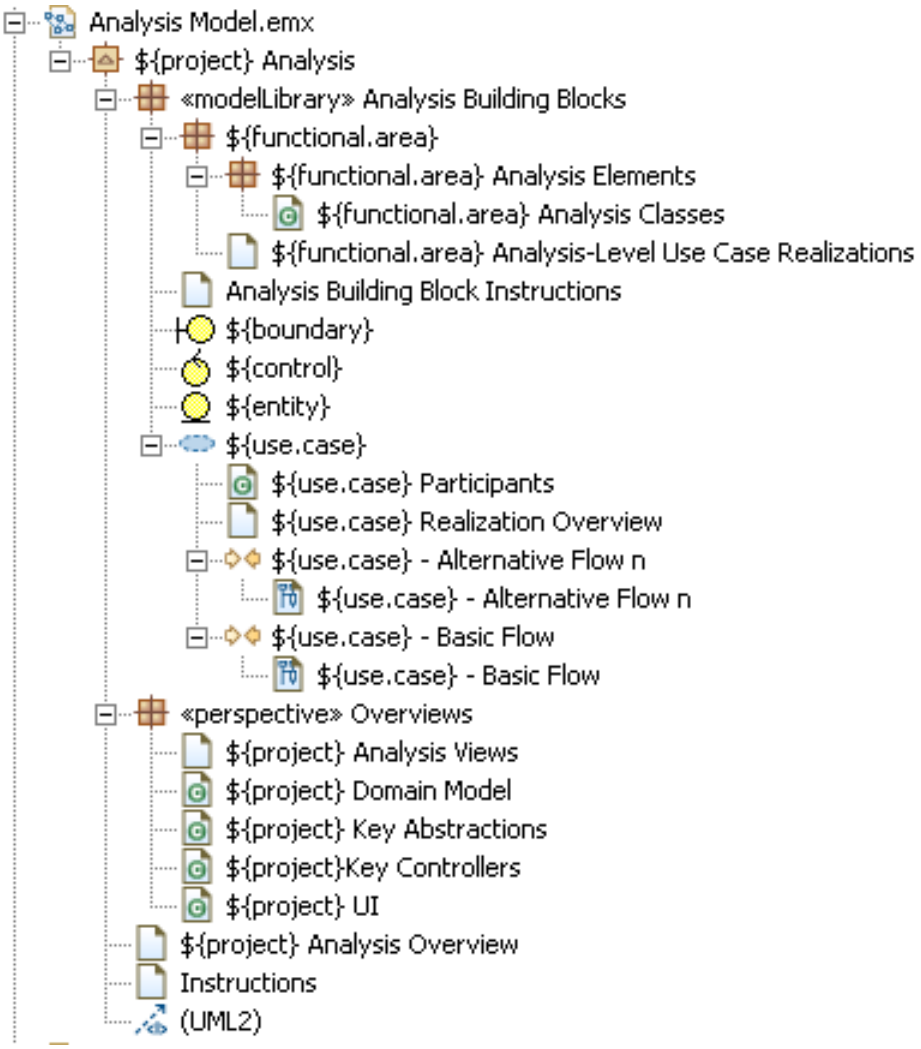


그림 3-2

엔터프라이즈 IT 디자인 모델링 파일

RSx 는 모델 템플리트를 기반으로 "엔터프라이즈 IT 디자인 모델(EITDM)" 파일을 작성하는 옵션을 제공합니다. 템플리트는 **그림 3-3**에 표시된 대로 기본 컨텐츠에 컨트리뷰션합니다. 더불어 "EJB 변형" 프로파일[3]이 템플리트에서 작성한 모델 파일에 적용됩니다. 이는 비즈니스 응용프로그램을 대상으로 하고 *RSX* 코드 생성 변형을 사용하여 해당 응용프로그램의 작성을 지원하는 경우 디자인(및 선택적으로 분석)에 사용하기에 적합한 템플리트입니다.

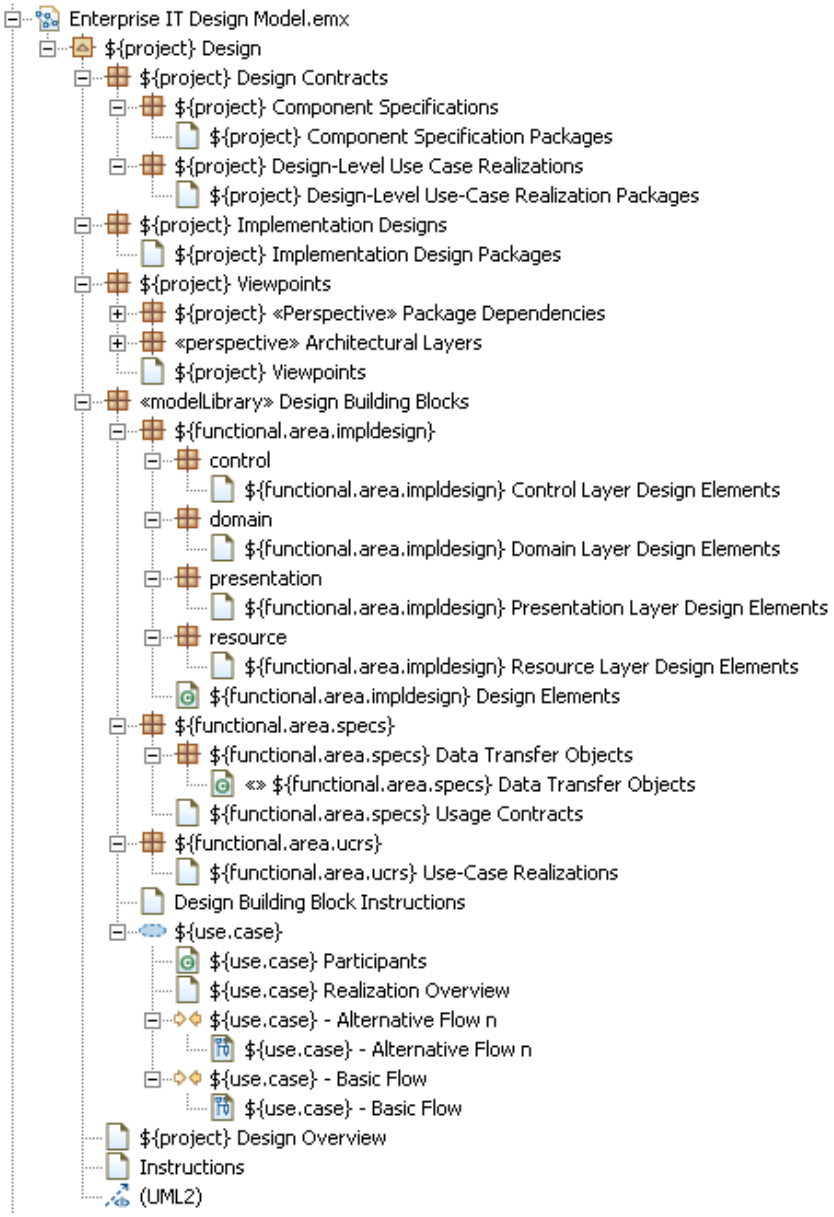


그림 3-3

[3] EIT 디자인 모델 템플리트의 일부로 제공되는 변형 가능 프로파일 세트는 제품의 갱신사항이 릴리스됨에 따라 발전하게 됩니다.

구현 개요 모델링 파일

더불어, 디자인 모델의 일부로서 구현이 조직되어야 하는 방법의 상위 레벨 보기를 캡처하는 "구현 개요 모델"을 정의하는 것이 유용할 수도 있습니다. "구현 개요 모델"은 디자인 단계의 초기에(코드가 생성 또는 작성되기 전에) 코드 및 관련 파일(메타데이터, 배치 설명자 등)이 있을 것으로 예상되는 실제 RSx 또는 Rational Application Developer 프로젝트 및 폴더/패키지를 표시하도록 사용됩니다. 이 모델을 사용하여 이러한 프로젝트 및 패키지 간 예상 종속성을 표시할 수도 있으며, 이는 시스템 빌드 요구사항을 식별하는데 유용할 수 있습니다. 구현 개요 모델은 솔루션 아키텍처의 비정규 개념 다이어그램을 유지하는 위치가 될 수도 있습니다.

구현 모델

앞에서 언급한 바와 같이, RS*x*에서 구현 모델은 구현 아티팩트와 해당 아티팩트를 설명하는 다이어그램(선택적)[4]을 포함하는 프로젝트로 구성되어 있습니다.

"스케치" 모델

"기본 개념 및 용어" 섹션에서 참고된 바와 같이, 디자인 모델을 시스템 수명동안 유지보수되고 아키텍처 제어를 지원/시행하는데 사용하는 정규 아키텍처 도면과 같이 다루도록 선택할 수도 있습니다. 또는 디자인을 제안하는 역할을 하고 이를 명백히하며 커뮤니케이션하는데 도움이 되지만 구현이 시작되면 버릴 수 있는 스케치처럼 다룰 수도 있습니다. RSx 는 두 접근 방식을 모두 지원합니다. 이 기능은 일반적으로 특정 접근 방식 대상으로 하지는 않지만 디자인 모델 사용 방법에 대한 사용자의 선택은 사용할 RS*x* 기능 및 해당 기능의 사용 방법을 판별하는데 확실히 도움이 됩니다. 이 문서에 표시된 가이드라인의 컨텍스트에서 구분이 필요한 경우 "스케치 모델"이라는 용어를 사용하여 모델이 보다 '처분 가능한' 방식으로 사용 중임을 나타낼 수 있습니다.

[4] 이 다이어그램을 작성하려면 파일→새로 작성→UML 모델을 사용하여 모델을 작성하는 대신, 파일→새로 작성→클래스 다이어그램을 사용하여 UML(또는 기타) 주석에 코드의 '보기'를 작성할 수 있습니다. 각 개별 다이어그램은 .dnx 확장명을 가진 별개의 파일로 유지되며 코드 파일처럼 버전 관리됩니다. 이 다이어그램에는 시맨틱 정보가 없고 주석만 있습니다. 모든 상관 시맨틱 정보는 코드 자체에 있습니다. 이러한 다이어그램 중 하나에서 클래스 이름 또는 오퍼레이션 서명과 같은 사항을 변경하는 경우, 실제로 기반 코드 자체가 변경됩니다. 코드에서 이러한 변경을 작성하는 경우(텍스트 편집기 사용), 변경된 코드가 표시되는 다이어그램이 자동으로 갱신됩니다.

4. 모델의 내부 구조를 조직하는 기법 및 일반 가이드라인

UML 모델의 컨텐츠를 조직하는 기본 도구는 패키지입니다. UML 패키지는 다음 두 가지 기본 목적으로 사용됩니다.

- 모델 정보의 레이블링, 조직 및 파티션
  - 문제점 또는 솔루션 도메인의 특정 주제 문제에 해당하는 요소 그룹화
  - 모델 정보의 다른 유형(예: 인터페이스, 구현, 다이어그램 등)을 구분
  - 기타 요소에 대한 종속성을 정의하고 제어하기 위해 요소 그룹화
  - 동일한 모델의 대체 보기를 제공하는 다이어그램 그룹화
- 이름 공간 설정
  - 모델 요소용
  - 모델 요소에서 생성된 구현 아티팩트용(모델 및 구현 언어 이름 공간 사이의 맵핑과 관련될 수도 있음)
  - 재사용 단위용

전통적으로 RUP 는 여러 모델 유형에 특정 패키징 전략을 제안합니다. 이러한 전략은 이 문서의 유형 특정 섹션에 반영되어 있습니다. 또한 RSx 는 여기서 설명하는 일부 추가 조직 도구를 도입합니다.

<<Perspective>> 패키지를 사용하여 시점 표시

두 가지 방법 이상으로 조직된 요소를 표시해야 하는 경우에 대체 조직 체계를 설명하는 다이어그램으로 추가 패키지를 작성할 수 있습니다. 이 동일한 기법은 모델의 패키지 체계를 포함하는 모델 컨텐츠의 특정 보기를 표시하는데 필요한 모든 경우에 유용할 수 있습니다. RSx 는<<Perspective>> 패키지 스테레오타입을 UML '기본 프로파일'의 일부로 제공하여 이 기법을 지원합니다. <<Perspective>> 패키지는 일반적으로 IEEE 471- 2000 "시점" 또는 시스템 엔지니어링을 위한 RUP 와 동등하다고 생각할 수 있습니다.

<<Perspective>> 패키지에 시맨틱 요소(클래스, 패키지, 연관 등)를 두지 마십시오. 대체 조직 관심사항 또는 응용프로그램 시점을 기반으로 하는 보기를 설명하는 다이어그램만을 두십시오. <<Perspective>> 스테레오타입을 패키지에 적용하면 여러 가지가 수행됩니다. 특정 시점을 표시하는 해당 패키지를 가시적으로 식별합니다. 시맨틱 요소가 <<Perspective>>패키지에 놓이는 경우 사용자에게 경고하는 모델 유효성 검증 규칙도 지원합니다. RSx 변형이 생략해야 하는 패키지의 지정자로서의 역할도 합니다.

토픽 다이어그램을 사용하여 특정 관심사항의 자체 갱신 표현 작성

설명하려는 요소를 수동으로 두는 '보통' 다이어그램과 대조적으로 토픽 다이어그램의 컨텐츠는 기존 모델 컨텐츠에 대해 실행되는 조회로 판별합니다.토픽 다이어그램을 작성하려면 '주제' 모델 요소를 선택한 다음 주제 요소와의 관계 유형을 기반으로 다이어그램에 표시하려는 기타 요소를 정의하십시오. 모델의 시맨틱 컨텐츠가 변함에 따라 토픽 다이어그램도 이에 맞춰 조정됩니다.

브라우즈 다이어그램을 통한 모델 점검

브라우즈 다이어그램은 *명확히* 모델 조직의 도구는 아닙니다. 이는 수동으로 다이어그램을 작성할 필요없이 모델 컨텐츠의 발견 및 이해를 돕는 것을 목적으로 합니다. 모델 조직의 컨텍스트에서 지속적인 다이어그램을 작성해야 하는 필요를 줄여주므로 이를 인지하고 있는 것이 유용합니다. 또한 모델의 크기 및 복잡도를 줄여 조직하기에 쉽게 할 수 있습니다.

브라우즈 다이어그램은 일종의 토픽 다이어그램과 같지만, 결코 지속적이지 않으며 언제나 작동 중에 생성된다는 차이점이 있습니다. 브라우즈 다이어그램을 작성하려면 모델 요소를 선택하고(다이어그램 또는 모델 탐색기) 컨텍스트 메뉴를 사용하여 "브라우즈 다이어그램에서 탐색"합니다. 이렇게 하면, 초점 주변에 방사형 레이아웃으로 표시되는 관련 요소와 함께 선택한 요소가 초점으로 표시되는 다이어그램이 작성됩니다. 그런 다음, 해당 브라우즈 다이어그램에서 연관된 요소 중 하나를 선택하여 다른 브라우즈 다이어그램의 초점으로 만드는 등, 얼마든지 이러한 방식을 계속할 수 있습니다.

다이어그램 내 탐색

RSx 에는 다이어그램 내 탐색을 위한 두 가지 메커니즘이 있습니다.

- 모델 탐색기에서 일부 기타 '호스트' 다이어그램으로 다이어그램 노드를 끌어 놓을 수 있습니다. 그런 다음, 호스트 다이어그램에 결과로 생긴 아이콘을 두 번 클릭하여 참조된 다이어그램을 열 수 있습니다.
- 모델에서 새 UML 패키지를 작성할 때마다 "기본" 다이어그램 (자유 양식 다이어그램)이 자동으로 작성됩니다. 기본값으로, 이 "기본" 다이어그램이 패키지의 '기본' 다이어그램으로서 작성됩니다. 다이어그램을 "기본" 외의 다른 이름으로 다시 이름 지정할 수 있으며 해당 다이어그램은 여전히 '기본'으로 취급됩니다. 패키지에서 다른 다이어그램을 선택하여 해당 패키지의 '기본' 다이어그램으로 작성할 수도 있습니다. '기본' 다이어그램의 용도는 다음과 같습니다. 패키지 자체를 일부 기타 '호스트' 다이어그램에 두는 경우 그런 다음 패키지를 두 번 클릭하여 기본 다이어그램을 열 수 있습니다.

이러한 메커니즘은 다음과 같은 조직 **가이드라인**을 지원합니다. 이는 어떤 유형의 모델에도 적용될 수 있습니다.

1. 설명할 각 모델링 파일의 기본 다이어그램(또는 기타 기본 다이어그램)을 작성하십시오.
    a. 모델링 파일의 각 최상위 레벨 패키지
    b. 모델링 파일의 루트 패키지에 있는 모든 기타 다이어그램의 다이어그램 아이콘 (즉, 기본 다이어그램 자체의 아이콘을 설명하지 마십시오.)
2. 설명할 각 최상위 레벨 패키지의 기본 다이어그램(또는 기타 기본 다이어그램)을 작성하십시오.
    a. 직접 포함하는 패키지
    b. 직접 포함하는 모든 기타 다이어그램의 다이어그램 아이콘
3. 각각의 이어지는 낮은 레벨의 패키지에 이 패턴을 반복하십시오.

5. 유스 케이스 모델의 내부 조직용 가이드라인

유스 케이스 모델 상위 레벨 조직

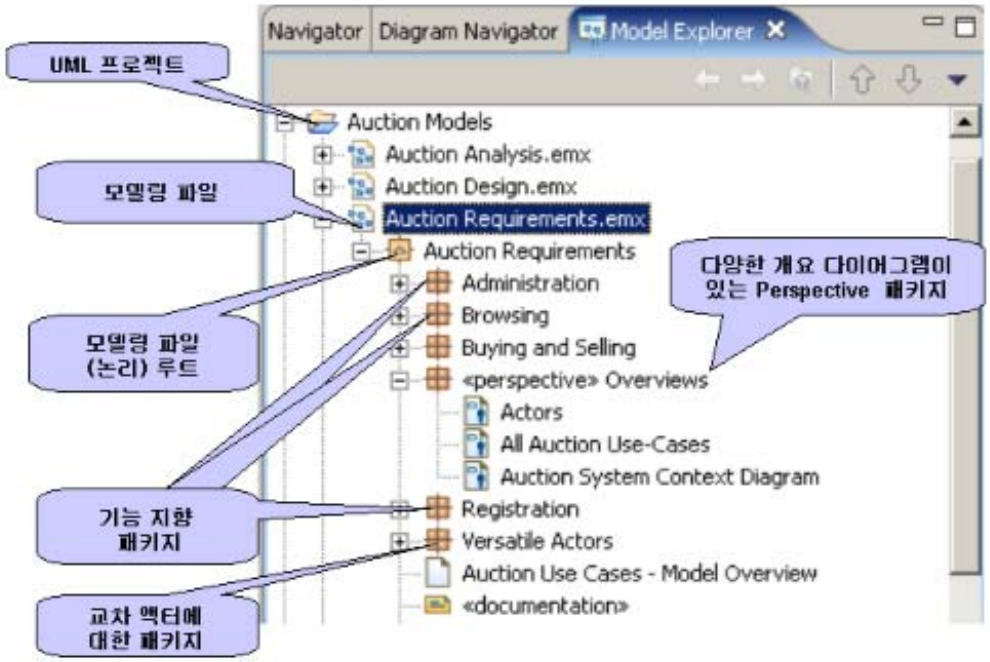


그림 5-1

그림 5-1은 유스 케이스 모델 구성을 위한 다음 가이드라인을 설명합니다.:

1. 최상위 레벨 패키지를 사용하여 기능 지향 그룹화를 설정하십시오. 근거:

   - 팀 구성원이 유스 케이스 모델에 대해 작업하는 경우 작업 부서 관심사항에 일반적으로 잘 맵핑합니다. 나중에 파일 경합이 문제가 되어 유스 케이스 모델을 여러 모델링 파일로 나누려는 경우에도 제대로 지원합니다(최상위 레벨 패키지당 별도의 모델링 파일을 작성하면 됨).

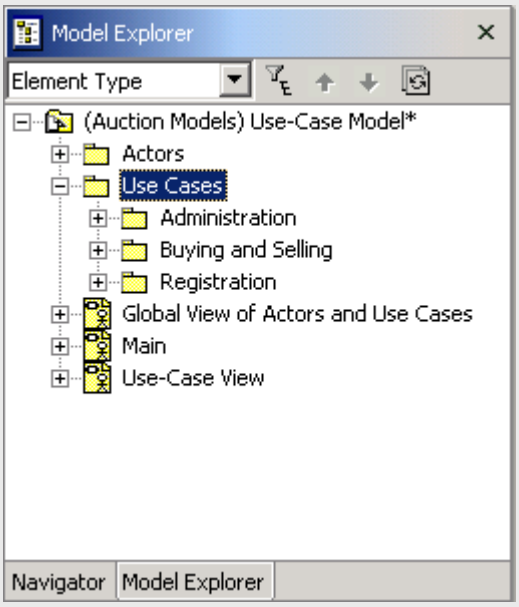
   - 기타 접근 방식과 비교하여 이는 최종 구현 조직에 일반적으로 보다 잘 맵핑됩니다. 각 연속된 하위 레벨의 추상을 시드하는데 변형을 사용하는 경우 중요합니다. 특히, 시드 컨텐츠를 유스 케이스 모델을 기반으로 하는 분석 모델로 생성하려는 경우, 유스 케이스 모델의 패키징 구조가 대상 분석 모델의 원하는 패키징 구조에 제대로 맵핑하기를 바랍니다. 또한, 분석 모델의 패키징 구조가 디자인 모델에 제대로 맵핑되고 디자인 모델의 패키징 구조가 구현을 포함하는 프로젝트 세트에 보다 잘 맵핑되기를 바랍니다. 이러한 맵핑이 보다 간단할 수록 한 추상 레벨에서 다음 레벨로의 변형을 구성하는데 보다 적은 작업이 필요하게 됩니다.

2. 다른 최상위 레벨 패키지를 사용하여 '광범위한 권한이 있는' 또는 '다양한 권한을 가지는' 액터를

캡처하십시오.

3. <<Perspective>> 패키지의 다이어그램을 사용하여 유스 케이스의 상위 레벨, 교차 보기를 캡처하십시오. 근거:

   - 기능 지향 그룹화로 조직된 모델의 시맨틱 요소를 유지하는 동안 교차 보기 및 '구조적으로 중요한' 유스 케이스의 보기를 제공하십시오.

> *XDE/Rose*
> *RSX* 가이드는 유스 케이스의 액터 및 기타를 위한 패키지를 작성하기 위한 용도의 유스 케이스 모델의 상위 레벨 조직용 기존 가이드를 약간 개정합니다. 그런 다음- 모델이 요구하는 크기 및 복잡도에 따라 보다 낮은 레벨의 패키지를 사용하여 다음 XDE 기반 예제에서 표시된 대로 기능 지향 그룹화를 설정합니다.
>
> 
>

유스 케이스 모델 컨텐츠

좋은 유스 케이스를 작성하는 방법 또는 좋은 유스 케이스 모델링의 할 일과 하지 말아야 할 일에 대한 자세한 자습서로서의 역할은 이 문서의 영역 밖입니다. 그러나 여기에는 액터 및 유스 케이스에 더불어 유스 케이스 모델에 포함될 수 있는 사항에 대한 간단한 설명이 있습니다.

- **권장사항:** 모델의 기타 패키지를 설명하고 이러한 패키지 및 각 '기본' 다이어그램으로의 드릴 다운을 지원하는 모델 루트에 '기본' 다이어그램을 작성하십시오.

- **권장사항:** 각 유스 케이스 패키지에서 패키지의 유스 케이스, 해당 유스 케이스의 관계 및 참여하는 액터를 설명하는 다이어그램을 포함하십시오. (경우의 수가 큰 경우 둘 이상의 다이어그램이 적합할 수 있음)

- **권장사항:** 문서 필드에 각 유스 케이스의 기본 및 대체 플로우를 설명하십시오[5](**그림 5-2** 참조).

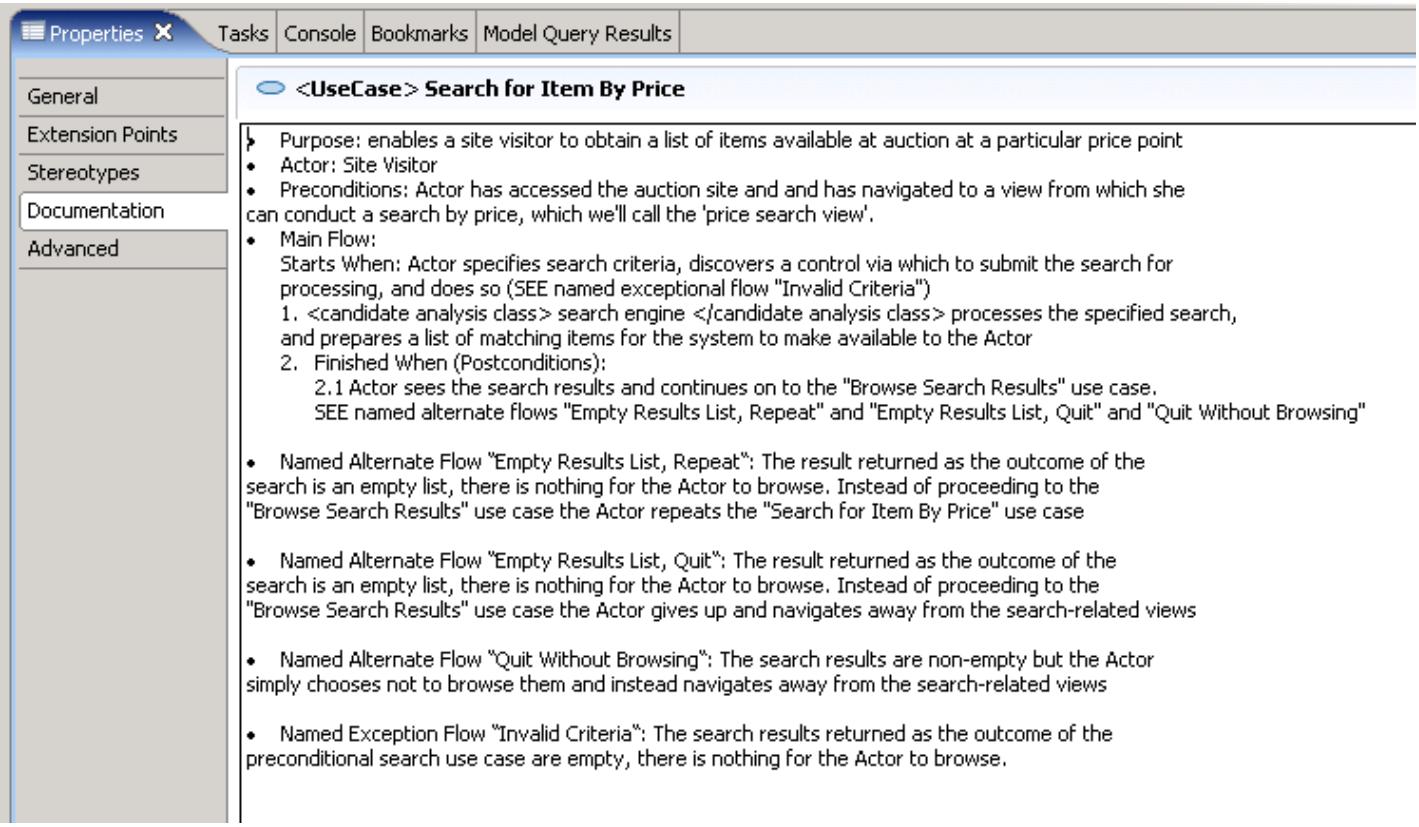


그림 5-2

- **선택사항:** 유스 케이스의 복잡도로 보장되는 경우, 활동 다이어그램을 추가하고 유스 케이스의 전체 활동 플로우를 반영하도록 작성하십시오(**그림 5-3** 참조). **근거:** 각(기본 및 대체/예외) 플로우에 해당하는 조건을 표시하는데 도움이 되며 모든 다양한 플로우가 결국 한데 모이도록 하는데 도움이 됩니다. (RS*x*에 활동 다이어그램을 추가하면 활동이 활동 아래의 다이어그램과 함께 유스 케이스에 자동으로 추가됩니다.)

- **선택사항:** 유스 케이스의 이름 지정된(기본, 대체 및 예외) 각 플로우의 '블랙 박스' 구현을 모델화하십시오. 유스 케이스에 협업 발생을 추가하십시오. 유스 케이스의 기본 플로우에 해당하는 상호작용 인스턴스와 더불어 각 대체 및 예외라고 이름 지정된 플로우의 상호작용 인스턴스를 추가하십시오.각 상호 작용 인스턴스의 시퀀스 다이어그램(또는 대체적으로 커뮤니케이션

[5] 유스 케이스 설명 예제에 설명된 형식화는 RTF 가능 편집기를 사용하여 텍스트로 된 '템플리트'를 작성한 다음 해당 템플리트를 복사하여 유스 케이스 설명 필드에 붙여넣으면 됩니다.

다이어그램)을 작성하십시오. 이러한 유스 케이스 협업 인스턴스는 분석 레벨 유스 케이스 구현(분석 모델에서 설명된 바와 같이) 또는 디자인 레벨 유스 케이스 구현(디자인 모델에 설명된 바와 같이)과 혼동되어서는 안됩니다. 이는 유스 케이스의 "화이트 박스" 구현이며 솔루션의 내부 요소간 상호작용을 설명합니다. 여기서 유스 케이스 모델에 대해 제안된 협업 발생은 엄격히 액터와 시스템 간 "블랙 박스" 상호작용입니다(**그림 5-3** 참조). **근거:** 이는 기술과 관련 없는 이해 당사자(stakeholder)에게 시스템의 사용자가 시스템과 상호작용하는 방법에 대한 상위 레벨 그림을 제공합니다. 구현의 일부로 필수인 여러 보기(화면 또는 페이지)를 식별하는데 도움이 될 수도 있습니다. 유스 케이스의 다양한 플로우의 이름을 시맨틱 모델 요소(즉, 협업 발생)에 지정하여 해당 플로우(시나리오)의 이름 지정 규칙 또한 공식적으로 설정합니다.

> *XDE/Rose*
> UML 1.x 에서는 이런 용도로 "협업 발생" 대신 "협업 인스턴스"를 사용했습니다.

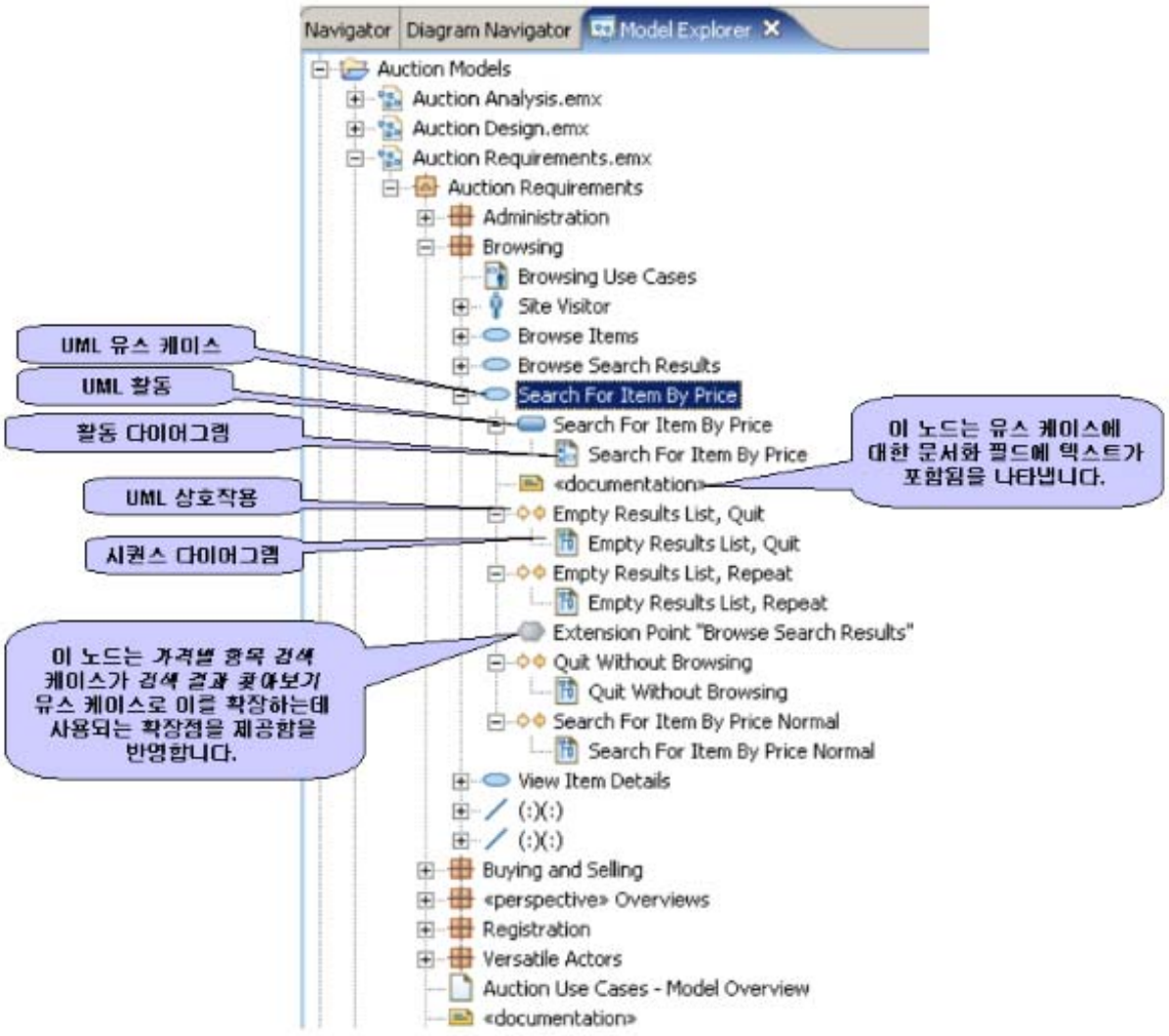


그림 5-3

- **선택사항:** RUP 가이드에 따라 사용자 아키텍처의 "구조적으로 중요한" 보기를 식별하고 특히 소프트웨어 아키텍처 문서를 유지보수하려는 경우, 최상위 레벨 <<Perspective>> 패키지를 추가하여 구조적으로 중요한 유스 케이스를 설명하는 유스 케이스 다이어그램을 포함하십시오. "아키텍처의 유스 케이스 보기" 패키지에 이름을 지정하고자 할 수도 있습니다.

6. 분석 모델의 내부 조직용 가이드라인

분석 모델은 솔루션의 '퍼스트 컷(first cut)'을 표시합니다. 이는 요구사항에서 최종 디자인을 얻는 디딤돌로서 비즈니스 도메인에 대한 정보를 캡처하는데 중점을 두고 비즈니스와 밀접한 추상의 상위

레벨에 있는 후보 솔루션 요소를 표시합니다. 분석 클래스 및 분석 레벨 유스 케이스 구현이 있는 위치입니다. 솔루션에 필요한 클래스(특히 시퀀스 다이어그램에 필요하다고 보이는 라이프라인에 해당하는 클래스)를 *찾기* 시작하는 모델링 유스 케이스 구현(주로 시퀀스 다이어그램을 사용)의 프로세스를 통해 이루어집니다. 또한, 유스 케이스 모델의 컨텐츠를 기반으로 하는 분석 모델 컨텐츠를 제안하는데 적용할 수 있는 일부 경험에 의한 규칙이 있습니다. 이는 나중에 이 섹션에서 다룹니다.

RUP 에서 분석 모델이 디자인 모델에 개별적으로 유지보수되어야 하는지 여부는 프로젝트 특정 결정이며, 별도의 분석 모델을 유지보수하는 가치가 있는지 여부를 기반으로 작성된 결정은 투자된 시간을 보증합니다. 별도 분석 모델을 작성했지만 유지보수하지 않으면 분석 클래스가 디자인 모델로 이동하여 정제됩니다. 또는 분석 모델이 점진적으로 전개되어 디자인 모델이 될 수도 있습니다[6]. 제품 특정 관점에서 고려할 수 있는 일부 옵션이 있습니다.

1. 모델링 파일(또는 파일 세트)에 있는 분석 모델을 분석모델을 기반으로 작성하십시오. 그런 다음, 수동 프로세스 또는 자동 변형을 사용하여 정제된 버전의 분석 요소를 엔터프라이즈 IT 디자인 모델 템플리트를 기반으로 두 번째 모델 파일에 작성한 다음 분석 모델링 파일을 제거하십시오. 이렇게 하면 별개의 분석 모델을 지속시키거나 버리도록 선택할 수 있습니다.
2. 분석 프로파일을 적용하는 엔터프라이즈 IT 디자인 모델 템플리트를 기반으로 모델링 파일(또는 파일 세트)에 분석 레벨 모델링을 수행하십시오. 이로써, 분석 클래스를 사용하여 유스 케이스 구현 모델링을 시작한 다음 디자인 인터페이스가 동작에서 역할을 하도록 반복하여 해당 구현을 정제할 수 있습니다.
3. 두 번째 및 세 번째 옵션의 혼합은 디자인 모델과 동일한 모델링 파일에 분석 모델 유형을 유지보수하기 위함입니다. 이를 수행하려면 분석 컨텐츠를 <<analysis>>키워드를 적용하는 패키지로 분리합니다. 이는 동일한 모델링 파일에서 분석 레벨 아티팩트를 보다 정제된 디자인 레벨 등가물로 보유할 기회를 제공합니다.

RSx 변형을 사용하여 구현 생성 시 이러한 변형이 여러 경우에 분석 레벨 요소를 입력으로 허용하며, 해당 요소를 디자인 요소로 수동으로 정제하는 일부 단계를 줄여줌을 인식해야 합니다. RSx 를 이와 같은 방식으로 사용할 경우 위의 옵션 중 2 또는 3 이 선호됩니다. RSx 의 일부로 패키지된 표준 코드 생성 변형에서 <<analysis>> 키워드가 있는 모델 패키지가 생략됩니다.

[6] 실제로는 RUP 가 디자인 모델에서 분석 클래스 및 분석 레벨 유스 케이스 실현(realization)을 디자인 모델에 작성한 후 해당 디자인 양식으로 직접 전개하는 옵션을 호출합니다. 해당 접근 방식 하에서, 디자인 모델을 "발견"함에 따라 일부 "순수 분석" 관점을 유지하는 방식으로 패키지를 작성할 수 있습니다.

분석 모델 상위 레벨 조직

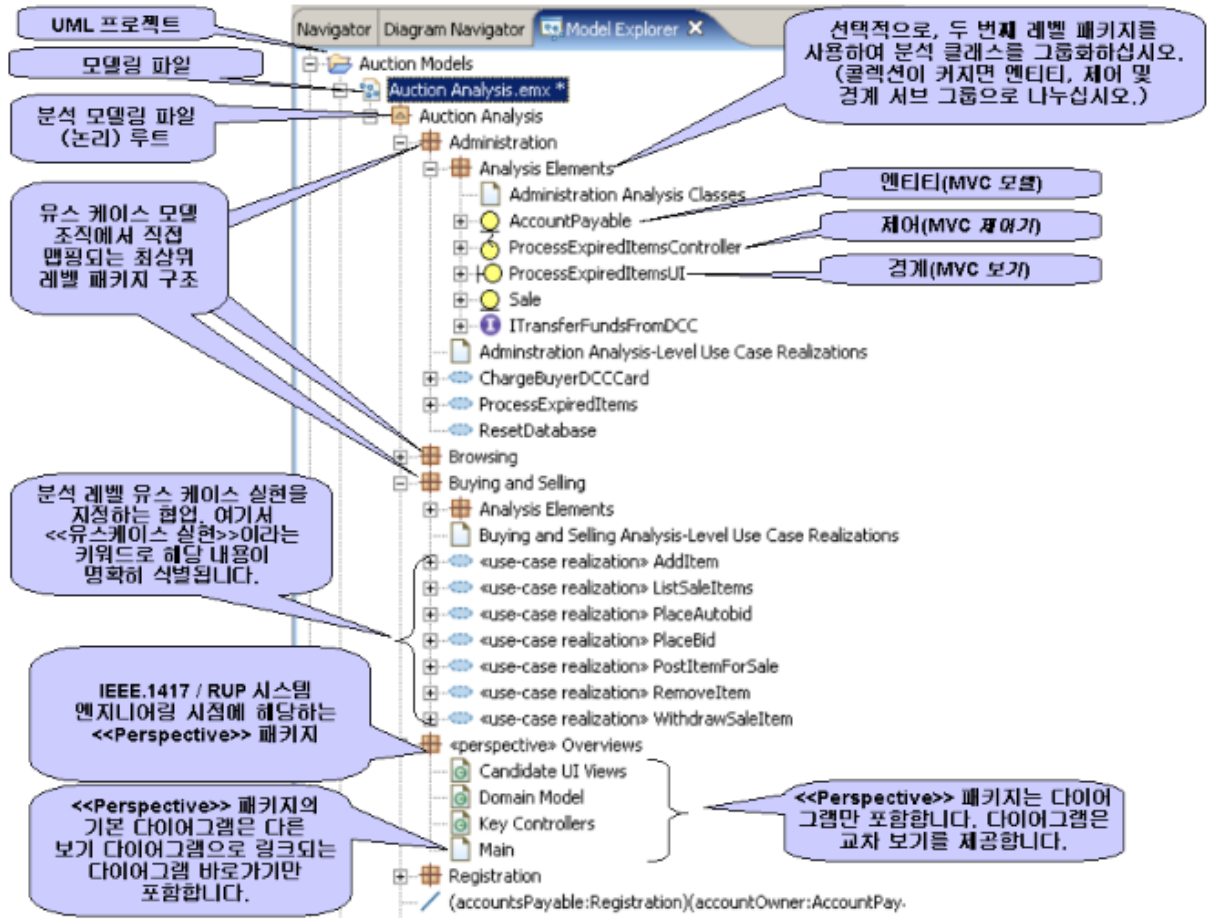


그림 6-1

*그림 6-1*은 분석 모델 구조화에 대한 다음 가이드라인을 설명합니다.

1. 최상위 레벨 패키지를 사용하여 분석 클래스에 대한 기능 지향 그룹화를 설정하십시오. 근거: 유스 케이스 모델과 동일한 근거.

2. 선택적으로, 최상위 레벨 패키지에서 서브패키지를 사용하여 분석 클래스를 수집하고 조직하십시오.

3. <<Perspective>> 패키지의 다이어그램을 사용하여 분석 요소의 대체, 상위 레벨 또는 교차 보기를 캡처하십시오. 근거: 기능 지향 그룹화로 조직된 모델의 시맨틱 요소를 유지하면서 다른 이해 당사자(stakeholder)에게 다른 관점을 제공하십시오.

이 접근 방식의 약간 다른 변형은 아래의 **그림 6-2**에 설명되어 있습니다. 여기서 분석 클래스에서 유스 케이스 구현을 분리하기 위해 최상위 레벨 패키지가 사용되는 것을 볼 수 있습니다. 해당 최상위 레벨 패키지 내에 최상위 레벨 패키지와 일치하는 기능 지향 서브패키지 세트가 있습니다.이런 식으로 유스 케이스 구현을 분리함으로써 *반드시* 유스 케이스 구현의 조직에 영향을 미치지 않고도 분석 클래스를 포함하는 패키지 구조를 리팩토링할 수 있습니다. (특히, 분석 모델이 디자인 *본래 위치에* 전개되는 경우, 클래스의 패키지 조직이 전개되어 유스 케이스에 원래 사용된 모델과 더 이상 일치하지 않을 가능성이 큼).

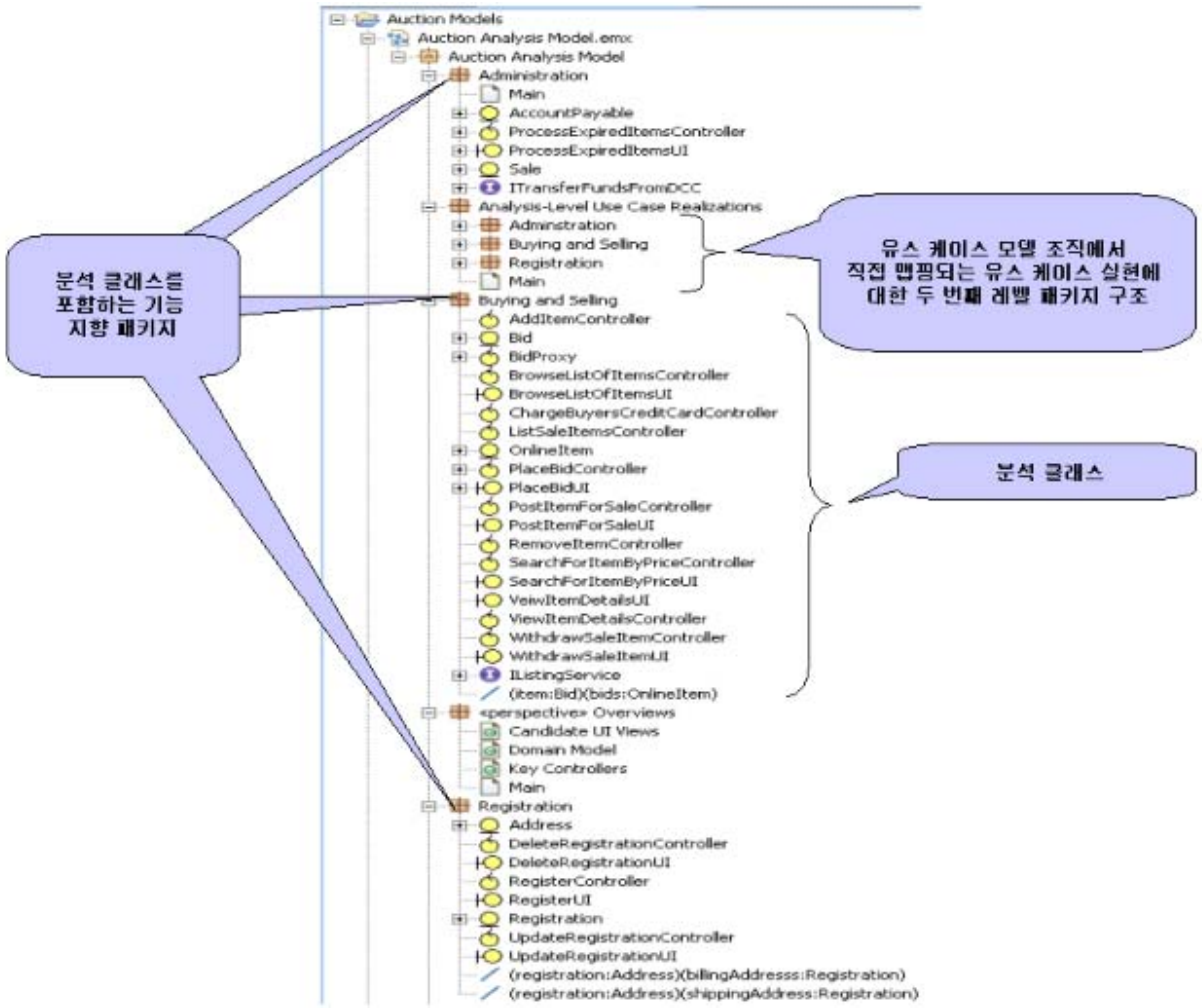


그림 6-2

사용자의 상황에 따라 여러 개별 그룹이 작성한 모델 컨텐츠를 병합 및 다시 사용해야 하는 이름 지정 규칙의 사용을 도입하게 되는 이유가 있을 수 있습니다(다른 (파트너) 비즈니스 그룹 포함). 이런 문제가 있는 경우 다음 **그림 6-3**에 표시된 대로 역으로 전환된 인터넷 도메인 이름 공간을 사용하는 것을 고려하십시오. 이는 *본질적으로* 분석 모델의 주요 관심사항이 아닐 수 있지만 분석 모델이 디자인 모델로 전개되도록 남겨두는 접근 방식을 *채택하고,* 디자인 레벨에서 재사용하거나 비즈니스 통합을 기대하는 경우 미리 계획하고자 할 수도 있음에 주의하십시오. 이 접근 방식을 채택할 때의

또다른 이점은 분석/디자인에서 생성된 코드 조직에 제대로 맵핑될 수 있어서 코드 생성 변형의 후속 구성을 단순화할 수 있다는 점입니다.

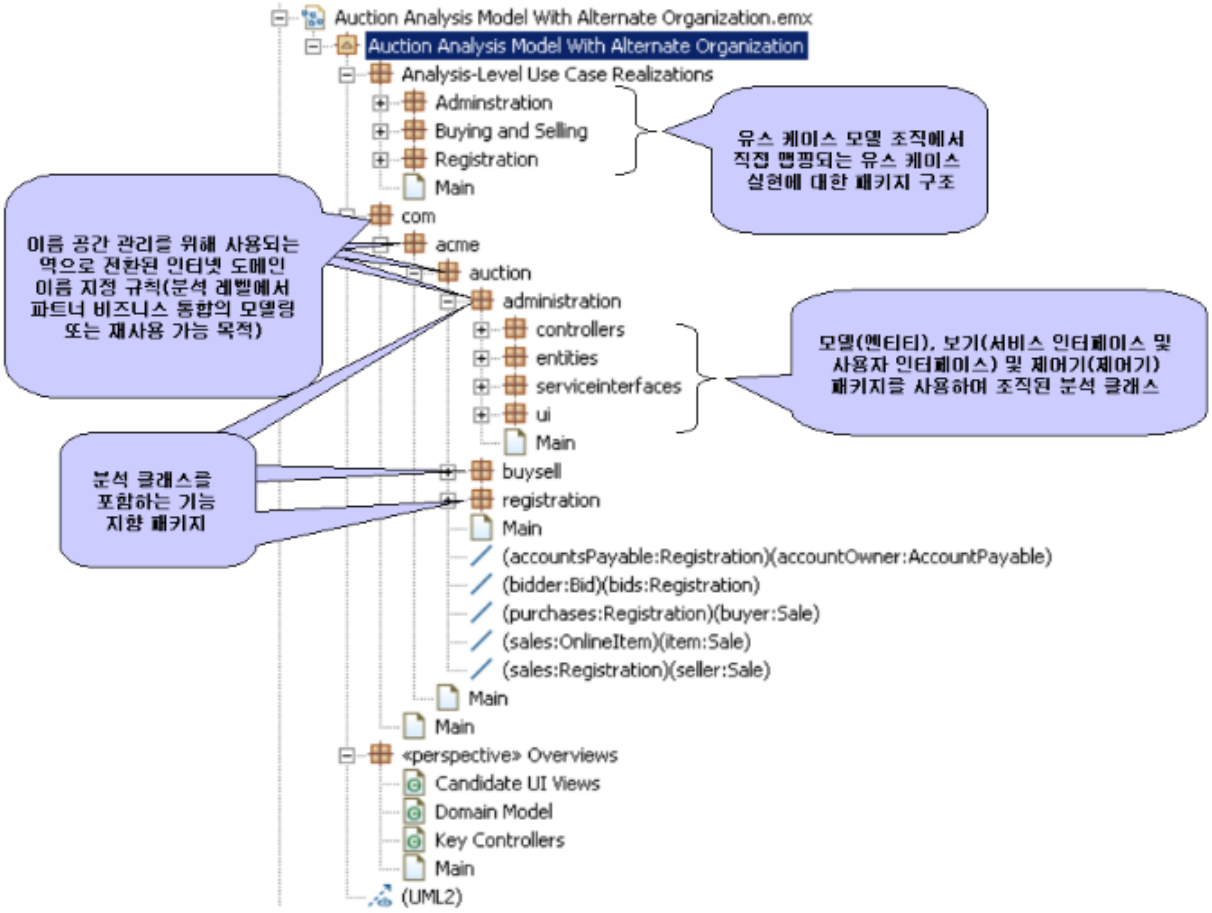


그림 6-3

분석 모델 컨텐츠

분석 클래스는 여러 방법으로 발견할 수 있습니다. 한 가지 방법은 유스 케이스 구현을 제안하는 시퀀스 다이어그램을 그리기 시작하는 것입니다. 이를 수행함에 따라 필요한 라이프라인을 발견하게 되는데, 일반적으로 각 라이프라인이 분석 클래스에 해당하게 됩니다. 이런 방식으로 클래스를 발견하는 경우, 해당 클래스를 분석 모델의 유스 케이스 구현 패키지에 작성할 수 있지만 해당 패키지에 그대로 두어서는 안됩니다. 앞서 분석 모델의 상위 레벨 조직용 가이드라인에서 설명한 바와 같이 분석 클래스를 기능 지향 패키지에 이동하려면 모델을 '리팩토링'해야 합니다(*그림 6-1*).

분석 클래스를 발견하는 다른 유용한 접근 방식: 경험에 의한 규칙을 기반으로 하는 클래스를 가지는 '시드' 분석 모델

- 각 유스 케이스에 대해(유스 케이스 모델 내), <<control>> 클래스를 분석 모델에 추가하십시오. <<control>> 클래스는 유스 케이스와 연관된 비즈니스 로직을 표시합니다. (나중에, 디자인 시세션 관리와 같은 관심사항에도 맵핑됨).
- 각 액터/유스 케이스 관계에 대해(유스 케이스 모델 내) <<boundary>> 클래스를 분석 모델에 추가하십시오. <<boundary>> 클래스는 솔루션 및 휴먼 액터 간 또는 솔루션 및 기타 외부 시스템 간 인터페이스를 표시합니다. 휴먼 액터에 해당하는 <<boundary>> 클래스는 결국 디자인 및 구현의 하나 이상의 사용자 인터페이스 아티팩트에 맵핑될 것입니다. 외부 시스템에 해당하는 <<boundary>> 클래스는 결국 디자인 및 구현의 특정 종류의 어댑터 계층에 맵핑될 것입니다.
- CRC 카드 분석 또는 유스 케이스 설명의 단어 분석과 같은 프로세스를 통해 추가 <<control>> 클래스(동사)와 <<entity>> 클래스(명사)를 식별하십시오.

이 시딩 접근 방식을 사용하여 분석 클래스를 식별하는 경우, 앞서 분석 모델의 상위 레벨 조직용 가이드라인에서 설명한 바와 같이 클래스를 기능 지향 패키지에 직접 둘 수 있습니다(*그림 6-1*).

그러나 분석 클래스를 찾으려고 할 때 원래 기능 패키지 조직의 변경사항이 필요함을 인지하고 있어야 함을 확신하게 됩니다.

선택사항: 분석 클래스 패키지의 두 번째 레벨 패키지를 사용하여 이러한 패키지의 컨텐츠를 더욱 체계화하십시오(**그림 6-4**).

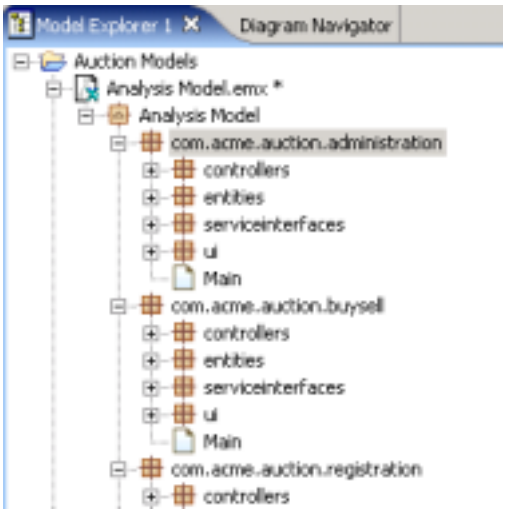


그림 6-4

권장사항: 분석 모델은 분석 레벨 유스 케이스 실현(realization)**을 포함해야 하며** 해당 실현은 유스 케이스가 분석 클래스 관점에서 수행되는 방식을 설명합니다. 각 분석 유스 케이스 실현(UML 협업으로 표시)은 유스 케이스 모델의 유스 케이스를 실현하며 해당 유스 케이스와 동일한 이름을 갖습니다. 각 분석 유스 케이스 실현(UML 협업으로 표시)은 유스 케이스 모델의 유스 케이스를 실현하며 해당 유스 케이스와 동일한 이름을 갖습니다(**그림 6-5** 참조). 각 이름 지정된 유스케이스 플로우[7]는 분석 레벨 구현으로 모델화되어야 하며 시퀀스 다이어그램(고유 상호작용을 자동으로 추가함)을 추가하십시오. **그림 6-6**은 시퀀스 다이어그램에서 작성한 대로 모델에 추가할 시맨틱 컨텐츠의 유형을 표시합니다.

[7] 유스 케이스 모델에서 이미 설정된 대로

(모델 탐색기 보기에서 임의의 UML 요소 유형을 필터링하여 **그림 6-6**에 표시된 '어지러움'을 많이 줄일 수 있음에 주의하십시오.)

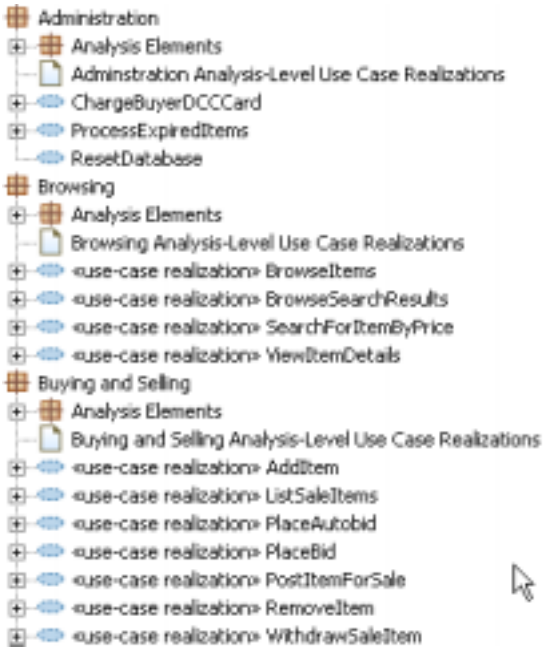


그림 6-5

선택사항: 유스 케이스 플로우의 시퀀스 다이어그램을 작성한 경우, 모델 탐색기에 소유하는 UML 상호작용을 선택할 수 있으며 커뮤니케이션 다이어그램을 추가할 수 있습니다. 새 커뮤니케이션 다이어그램이 시퀀스 다이어그램에 참여한 분석 클래스 인스턴스로 자동으로 채워집니다.

권장사항: 각 유스 케이스 구현(UML 협업)에서 구현 종속성 관계 및 유스 케이스 모델에서 해당하는 유스 케이스를 작성하십시오(**그림 6-6**). 토픽 다이어그램 및 추적성 분석과 같은 기능을 사용하여 모델의 추적성 관계를 이해할 수 있으므로 추적성 관계를 설명하기 위해 영구적인 다이어그램을 꼭 가질 필요가 없습니다. 따라서 예를 들어 다음과 같은 특정 종류의 '일회성' 다이어그램을 사용하여 관계를 작성할 것을 권장합니다.

- 협업에 자유 양식 다이어그램을 추가하십시오.
- 협업을 여기에 끌어 놓으십시오.
- 유스 케이스를 여기에 끌어 놓으십시오.
- 종속성 관계를 그리십시오.
- 마지막으로, (모델 탐색기의) 협업에서 다이어그램을 삭제하십시오.

권장사항: 각 유스 케이스 구현의 "참가자" 다이어그램을 포함하여 구현에 참여하는 분석 클래스(즉, 인스턴스가 유스 케이스의 구현을 설명하는 상호작용 다이어그램에 표시되는 분석 클래스) 및 상호작용 다이어그램에 설명된 협업을 지원하는 클래스 간 관계를 표시하십시오(**그림 6-6** 참조).

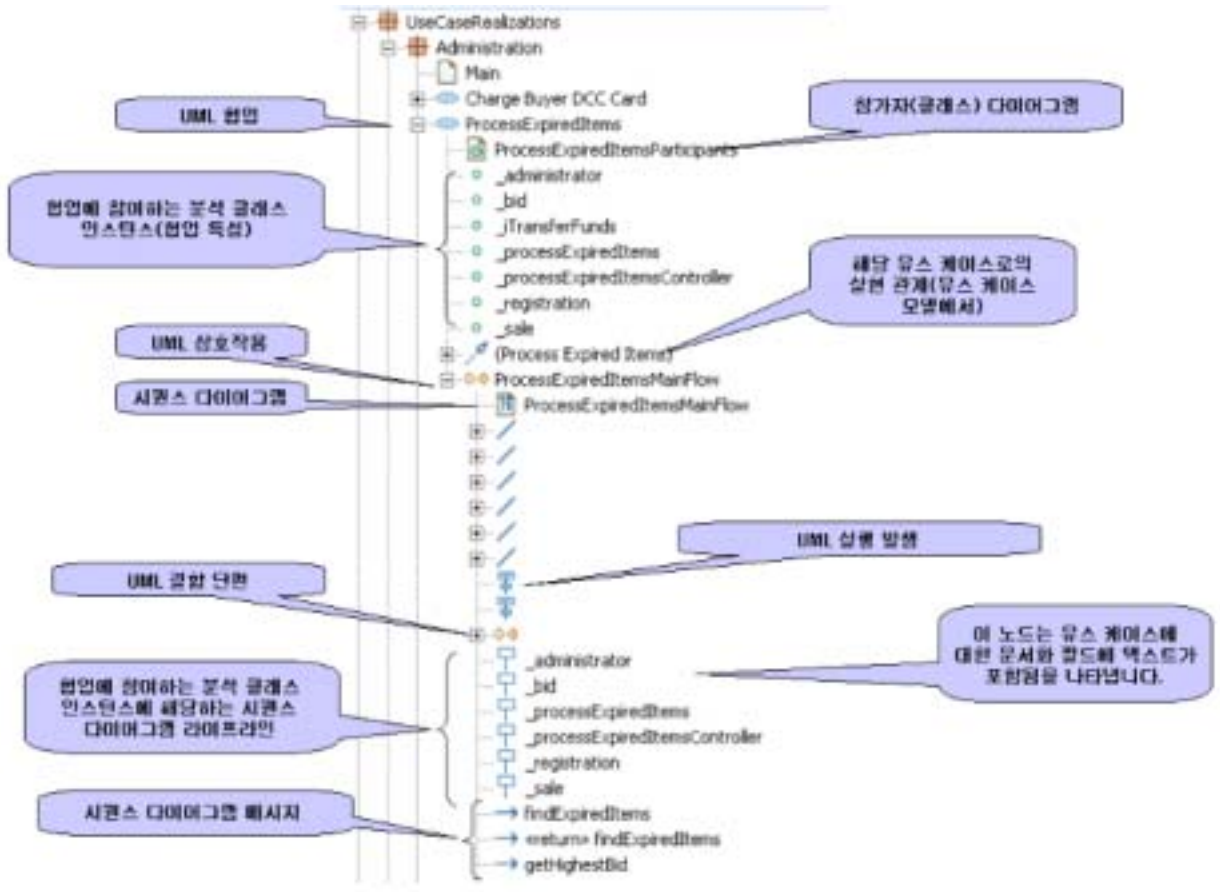
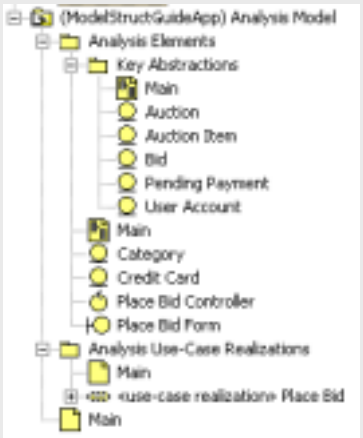


그림 6-6

XDE/Rose

아래에 표시된 대로 **분석 모델에 대해** 전통적으로 권장되는 구조가 RSx 에서는 분석 클래스의 기능 지향 패키지 조직을 강조하도록 수정됩니다. 또한 핵심 추상 패키지의 사용(그렇지 않은 경우 기능 지향 패키징 접근 방식을 의미)이 <<Perspective>> 패키지의 핵심 추상 다이어그램의 사용으로 대체되었음에 주의하십시오.



7. 디자인 모델의 내부 구조 용 가이드라인

디자인 모델 상위 레벨 조직

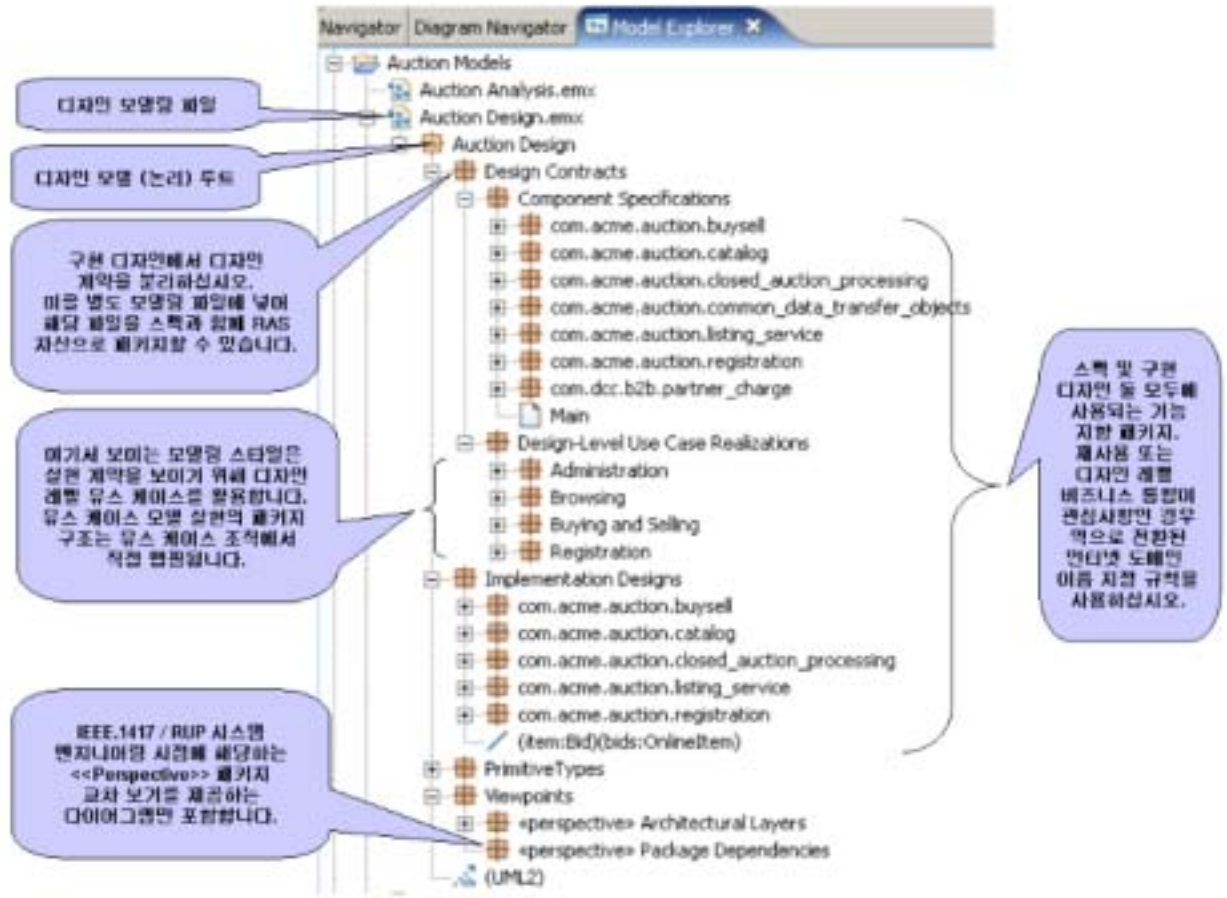


그림 7-1

그림 7-1은 디자인 모델 구조화에 대한 다음 가이드라인을 설명합니다.:

1. 구현 디자인에서 스펙을 분리하십시오. 이 그림은 이를 달성하는 최상위 레벨의 "디자인 계약" 및 "구현 디자인" 패키지를 표시합니다.

2. 보다 낮은 레벨 패키지를 사용하여 기능 지향 그룹화를 설정하십시오. 예를 들어, 분석 중에 사용한 조직부터 시작하여 분석 클래스가 실제 디자인 클래스, 컴포넌트 및 서비스에 맵핑되는 방법을 결정함에 따라 발전하게 할 수 있습니다. (초기 조직 체계는 디자인 중에 발전할 가능성이 큼 - 자세한 설명은 아래 참조).

이 시점에서 서브시스템이라는 단어를 설명하고자 합니다. 버전 2 보다 이전 버전의 UML 에서 서브시스템은 패키지의 특수 유형이었습니다. UML 2 에서 서브시스템은 컴포넌트의 특수 유형이며 컴포넌트는 패키지를 포함할 수 있습니다. UML 2 에서, <<subsystem>> 컴포넌트는 패키지의 실행 가능한 조직적/이름 공간 대안이지만 서브시스템 대 패키지의 적절한 사용은 명확히 설명되지 않습니다.제안사항: 특정 응용프로그램의 디자인 서브시스템과 같이 세분화 레벨에는 패키지를 사용하고, 서브시스템은 엔터프라이즈 전반에 걸친 아키텍처 보기의 전체 응용프로그램(예: CRM 또는 SCM)을 표시하도록 남겨두십시오.

> *XDE/Rose*
> *이 문서의 작성 시에는 Rose 및 XDE 모델 가져오기 도구에서 UML 1.x 서브시스템을 UML2 서브시스템이나 «subsystem» 키워드가 적용된 패키지에 맵핑하는 옵션을 제공할 것으로 예상되었습니다.*

3. 디자인 요소의 조직은 시스템의 유스 케이스가 조직되는 방법에서 발전할 가능성이 큽니다(별도의 분석 모델이 유지보수되는 경우, 유스 케이스 모델에서 또는 분석 모델에서도 가능). 패키지를 사용하여 디자인 계약을 디자인 요소 스펙(사용법 계약) 및 디자인 레벨 유스 케이스 구현(구현 계약)으로 더 세분화하고 유스 케이스 자체의 조직을 계속 반영하는 유스 케이스 구현의 패키지 하위 구조를 유지보수하십시오.

4. 기능 영역의 구현 디자인 및 스펙을 구성하는 요소의 두 번째 레벨 조직 체계의 기본으로서 아키텍처 계층의 사용을 고려하십시오(자세한 설명은 아래 참조).

5. 시맨틱 모델 요소를 그룹화하는 UML 컴포넌트 및 패키지 내에 해당 그룹에 특정한 보기를 제공하는 다이어그램을 두십시오. 이 가이드라인은 해당 그룹화가 비즈니스 도메인의 기능 지향 서브세트를 기반으로 하는지, 아키텍처 계층을 기반으로 하는지 또는 사용자 수행 사항을 기반으로 하는지와 관련됩니다. '기본' 다이어그램이 패키지 또는 컴포넌트 자체와 동일한 이름을 갖도록 하고 패키지의 컨텐츠 개요를 표시하도록 작성하십시오. 이렇게 하면, 일부 다이어그램이 설명하는 사항과 밀접하게 유지되어 모델의 탐색 및 이해가 쉬워집니다.

6. 디자인 모델에 역으로 전환된 인터넷 도메인 이름 공간의 사용을 도입하고자 할 수도 있습니다. 근거:
   - 기본적으로 이를 수행하는 동일한 이유는 다음과 같은 언어 특정 구현에서도 중요합니다.
     a. 여러 모델 기반 응용프로그램(특히 파트너 회사와)이 포함된 통합 작업과 관련된 시나리오
     b. 재사용 시나리오
   - 변형을 구현으로 바꾸는 후속 구성을 단순화할 가능성이 큽니다(소스 대 대상 위치 및 이름 맵핑).

7. *이름 공간 맵핑의 부담 및 잠재적 혼동을 없애기 위해* 대상 구현 플랫폼에서 올바른 패키지 이름을 사용하는 것을 고려하십시오. (대부분의 경우, 이는 단순히 "이름에 밑줄 외에 공백이나 구두점을 사용하지 않는 것"을 의미합니다.)

8. 패키지 이름에 소문자를 사용하여 패키지의 클래스 이름과 보다 구분하기 쉽게 하십시오.

9. 인터페이스 및 컴포넌트 또는 이를 구현하는 클래스에 다른 이름의 사용을 고려하십시오. 인터페이스 및 구현 이름에 ILoan 과 Loan 또는 Loan 과 LoanImpl 을 사용하십시오. 이는 실제로 모델에

필요하지는 않지만 생성된 모델에는 유용한 아이디어입니다. 이를 통해 일부 후속 변형 구성 작업이 줄어들게 됩니다.

10. 다음 시나리오에서 코드가 생성되지 않는 모든 분석 레벨 컨텐츠는 "analysis"로 스테레오타입화된 패키지에 분리되어야 합니다[8].
    - A) 별도의 분석 모델 사용을 생략하고 디자인 모델을 분석 레벨 컨텐츠로 채우며 동일한 모델의 디자인 레벨 컨텐츠 또한 작성하는 동안 추상의 분석 레벨의 컨텐츠를 유지보수하도록 선택해야 합니다.
    - B) EIT 디자인 모델로부터 모델에서 코드로 변형을 구동합니다.

11. <<Perspective>> 패키지의 다이어그램을 사용하여 디자인 요소의 상위 레벨, 교차 보기를 캡처하십시오. 근거: 기능 지향 그룹으로 조직된 모델의 시맨틱 요소를 유지하면서 다른 유형의 이해 당사자(stakeholder)에게 적합해 보이는 보기, '구조적으로 중요한' 컨텐츠 및 교차 보기를 제공하십시오.

디자인 모델의 패키징 구조가 시간이 흐름에 따라 발전함을 인식하는 것이 중요합니다. 결국, 조직은 사용자 구조를 컴포넌트 및 서비스로 구성하는 방법에 해당해야 합니다. *디자인*의 마지막 조직에 대한 접근 방식은 일반적으로 재사용가능 자산을 패키징하고 디자인으로부터 디자인에서 생성된 구현 아티팩트(코드, 메타데이터, 문서)를 보유할 프로젝트 및 폴더 세트로 가장 간단하게 맵핑하는 최적의 잠재성을 제공합니다.

그러나 *초기* 조직은 유스 케이스 모델에 사용한 다음 분석 중에 개정된 조직적 접근 방식에 다소 직접적으로 대응해야 합니다[9]. 사실상(이전 섹션 "분석 모델의 내부 조직용 가이드라인"에 설명된 대로), 분석 모델이 적절히 디자인으로 전개되도록 선택할 수 있습니다. 즉, 디자인의 초기 조직이 함께 결합하고 느슨한 결합으로 이루어진 비즈니스 관심사항 세트로 그룹화하고 교차 또는 재사용가능한 요소로 분리하는 경향이 있습니다. 초기 조직의 이 접근 방식은 다음과 같은 이유로 효율적임이 증명됩니다.

- 분석 또는 유스 케이스 모델 컨텐츠에서 디자인 모델 컨텐츠를 생성하는 변형을 사용하려는 경우 소스 패키지에서 대상 패키지로 맵핑이 단순하고 간단합니다.
- 기능 결합 및 패키지의 느슨한 결합을 기반으로 하는 초기 조직 접근 방식이 확실히 최종 컴포넌트 지향 조직에 맵핑하는 최적의 기회이며 디자인 프로세스의 일부로서 수행해야 하는 리팩토링의 양을 줄여야 함을 의미합니다.
- 패키지의 느슨한 결합에는 팀 워크플로우를 향상하고 디자인이 여러 모델링 파일로 분해되는 경우 재사용이 가능하도록 돕는 잠재력이 있습니다.

대체 접근 방식 또한 가능하며 다음과 같은 일부 경우에는 *마지막* 조직으로서 권고됩니다.

- EJB 를 포함한 J2EE 기반 웹 응용프로그램을 대상으로 하는 경우, 디자인 조직이 J2EE 프로젝트

[8] 이러한 패키지는 변환에 의해 생략됩니다..

[9] 분석 클래스의 패키징은 재사용 및 예상치 않은 기능적 요구사항을 보다 잘 지원하기 위해 발견되는 경우 보통 많은 부분 리팩토링됩니다.

관련 Rational Application Developer 및 RSA 의 규칙을 기대할 수 있습니다.[10] 특히, 아키텍처 계층에 해당하는 최상위 레벨 디자인 패키지를 정의하도록 선택할 수도 있습니다(세션 및 도메인으로 하위 계층화된 비즈니스 및 프리젠테이션). 이는 확실히 플랫폼 중립적 접근 방식이 아니므로 디자인 중인 솔루션이 J2EE 외의 플랫폼에서 구현되지 않을 것임을 아는 경우에만 권장됩니다.

- 보다 일반적으로, 개발자 전문가 및 작업 부서에서 프리젠테이션 및 비즈니스 계층에 해당하는 n 층 응용프로그램을 빌드하는 경우가 많으므로 이러한 아키텍처 계층에 해당하는 최상위 레벨 패키지를 사용하도록 선택할 수도 있습니다. 특정 아키텍처를 지원하기 위해 *특정* 비즈니스 기능을 지원하는 클래스를 조직할 때는 *주의를 기울이십시오*. 이렇게 하면 변경하기가 어려워집니다.
- 비컴포넌트/서비스/서브시스템 지향 조직 접근 방식을 사용해야 하는 이유가 있는 경우, 코드 생성 변형을 구성하는데 추가 노력을 투자하여 디자인 조직을 대상 프로젝트 및 폴더 세트에 여전히 맵핑할 수 있도록 해야 합니다. '맵핑 모델'로 표시되는 동반 모델의 특수 유형을 특히 복잡한 맵핑을 정의하는데 사용할 수 있습니다.

디자인 모델 컨텐츠

디자인 모델에 필요한 사항에 대한 엄격한 규칙은 없지만 다음 제안이 유용할 수 있습니다.

[10] 간략하게 말하면, 시스템 또는 응용프로그램 또는 대형 서브시스템당 엔터프라이즈 프로젝트 및 각 엔터프라이즈 프로젝트용, 프리젠테이션 층용 웹 프로젝트 및 EJB 프로젝트가 일반적으로 컴포넌트 또는 소수의 서브시스템에 해당하는 여러 EJB 프로젝트 및 일반적으로 분리된 EJB 프로젝트가 컴포넌트 또는 서브시스템당 세션(세션 EJB) 및 도메인(엔티티 EJB) 계층에 사용되는 경우. 자세한 정보는 이 문서의 섹션 9 를 참조하십시오.

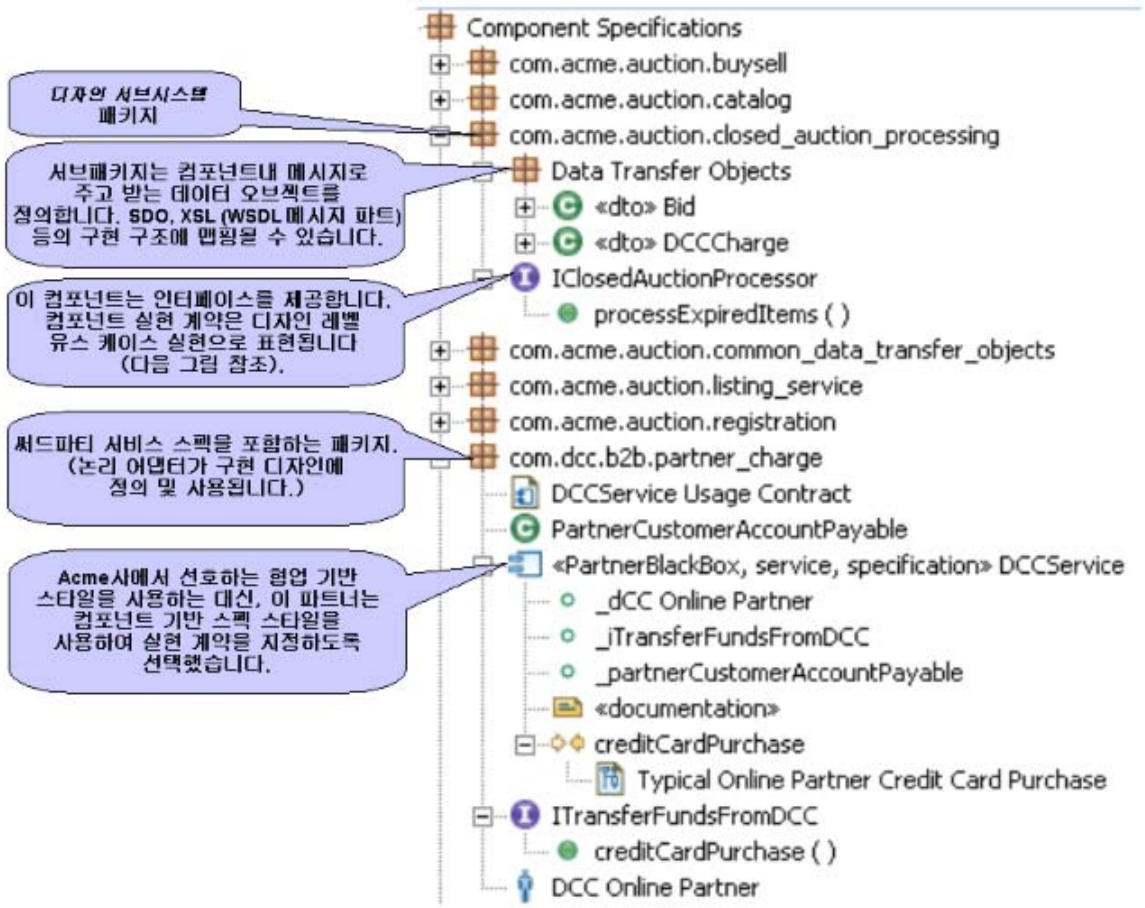

디자인 서브시스템 패키지
서브패키지는 컴포넌트내 메시지로 주고 받는 데이터 오브젝트를 정의합니다. SDO, XSL (WSDL 메시지 파트) 등의 구현 구조에 맵핑될 수 있습니다.
이 컴포넌트는 인터페이스를 제공합니다. 컴포넌트 실현 계약은 디자인 레벨 유스 케이스 실현으로 표현됩니다 (다음 그림 참조).
써드파티 서비스 스펙을 포함하는 패키지. (논리 어댑터가 구현 디자인에 정의 및 사용됩니다.)
Acme사에서 선호하는 협업 기반 스타일을 사용하는 대신, 이 파트너는 컴포넌트 기반 스펙 스타일을 사용하여 실현 계약을 지정하도록 선택했습니다.
Component Specifications
com.acme.auction.buysell
com.acme.auction.catalog
com.acme.auction.closed_auction_processing
Data Transfer Objects
«dto» Bid
«dto» DCCCharge
IClosedAuctionProcessor
processExpiredItems ( )
com.acme.auction.common_data_transfer_objects
com.acme.auction.listing_service
com.acme.auction.registration
com.dcc.b2b.partner_charge
DCCService Usage Contract
PartnerCustomerAccountPayable
«PartnerBlackBox, service, specification» DCCService
_dCC Online Partner
_iTransferFundsFromDCC
_partnerCustomerAccountPayable
«documentation»
creditCardPurchase
Typical Online Partner Credit Card Purchase
ITransferFundsFromDCC
creditCardPurchase ( )
DCC Online Partner

그림 7-2

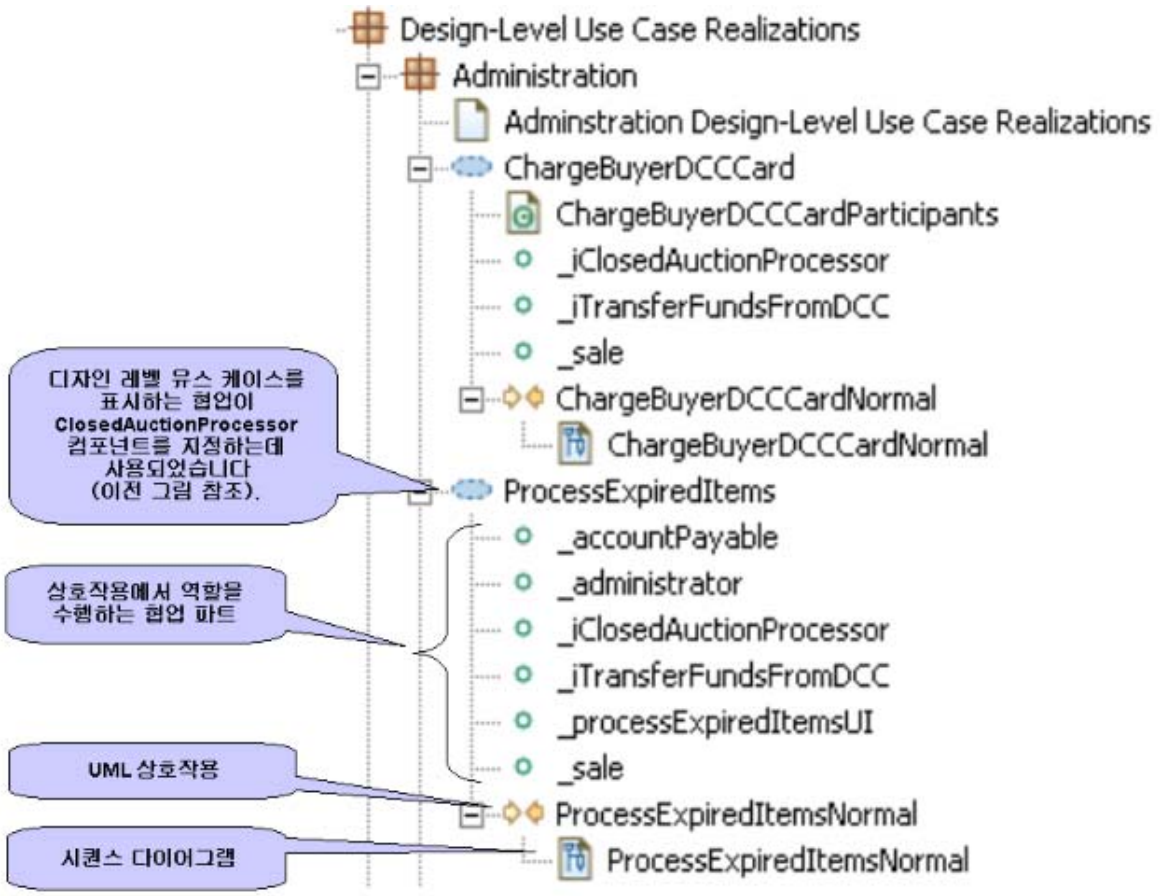


그림 7-3

그림 7-2 및 **그림 7-3** 은 **그림 7-1**에 설명된 조직 구조를 고수하고 디자인 계약이 지정될 수 있는 방법을 설명합니다.

- "ClosedAuctionProcessor" 컴포넌트의 사용법 계약이 단일 인터페이스로 표현됩니다[11] (**그림 7-2**). 해당하는 실현(realization) 계약이 협업으로 표현되는 단일 디자인 레벨 유스 케이스 실현으로 지정됩니다(**그림 7-3**).[12] 분석 레벨 유스 케이스 실현(realization)은 분석 클래스 간의 협업을 표시하는 반면 디자인 레벨 실현은 보다 구체적인 디자인 요소 간의 협업을 표시합니다[13]. 디자인 모델의 스펙 서브세트가 구현 디자인 서브세트와 개별적으로 패키지되어야 하는 경우, 디자인 레벨 유스 케이스 구현이 분석 또는 디자인 스펙 요소만을 사용할 뿐 해당 역할에 구현

[11] 물론 컴포넌트에는 다수의 제공된 인터페이스가 있을 수 있지만, 이 예에서는 한 개만 있는 것으로 합니다.

[12] 기타 컴포넌트가 여러 시스템 유스 케이스에 참여할 수도 있으므로 해당 실현 계약이 여러 유스 케이스 실현에 있을 수도 있습니다. 이러한 경우에 컴포넌트의 인터페이스와 동일한 패키지에 해당 유스 케이스의 유스 케이스 구현을 구성하는 여러 다이어그램의 링크를 두는 "{컴포넌트 이름} 사용 위치"라는 다이어그램을 포함할 수도 있습니다.

[13] 가능한 또 다른 차이점: 디자인 레벨 실현의 일부 "참가자" 다이어그램이 분석 레벨 유스 케이스 실현에 제안된 참가자 클래스 다이어그램 대신(또는 이에 덧붙여) 컴포넌트 연결(wiring)을 설명하는 컴포넌트 다이어그램일 수도 있습니다.

디자인 요소는 결코 사용하지 않는 것이 중요합니다.

- 써드파티 "DCCService"의 사용법 및 구현 계약은 모두 한 패키지에 있습니다[14]. 다시 한 번, 사용법 계약이 단일 인터페이스로 구성되었음을 확인하지만 이런 경우, 구현 계약은 <<specification>> 컴포넌트를 사용하여 표현됩니다(**그림 7-2**). (그렇지 않으면, 구현 계약의 스펙이 거의 비슷하며 동작을 사용하여 표현됩니다(이 경우, "creditCardPurchase"라는 상호작용). 협업 대신 컴포넌트를 사용하는 기타 예제가 **그림 7-4**에 표시됩니다.

- 오퍼레이션은 <<specification>> 컴포넌트(사용하는 경우) 또는 인터페이스를 구현하는 구현 디자인의 클래스류로 구현될 수 있는 인터페이스로 정의합니다.

- 데이터 전송 오브젝트의 스펙(제공된 오퍼레이션의 매개변수 유형 역할을 하며 XML 스키마 또는 SDO와 같은 구현 구조에 맵핑될 수 있음)은 사용법 계약의 일부로서 포함될 수도 있습니다. 분배 가능하게 디자인되지 않은 컴포넌트의 경우, 데이터 전송 오브젝트를 오퍼레이션 매개변수로 사용하는 유형의 스펙으로 지정하도록 선택하거나 선택하지 않을 수 있습니다. 분배 가능한 서비스의 경우(예: 웹 서비스), 서비스 오퍼레이션이 로컬 주소 공간의 오브젝트를 참조하지 않는 것이 기본이므로 DTO를 사용해야 합니다.

> *XDE/Rose*
> 이전 버전의 UML 에서 유스 케이스의 가이드는 유스 케이스당 협업 인스턴스 및 구현의 중요한 각 플로우의 상호작용 및 시퀀스 다이어그램을 사용하기 위함이었습니다.
>
> RSx 에서는 이제 UML2 시퀀스 다이어그램이 대체 실행 경로의 표기법을 지원하므로 한 상호작용 및 다이어그램만을 자주 사용 가능할 수 있어야 합니다.
>
> 또한, UML 2 에는 더 이상 '협업 인스턴스'가 없습니다. 대신, '협업 사용'이 있으며, 이는 해당 유형으로 협업이 필수입니다. 따라서 RSx 는 유스 케이스 실현(realization)을 표시하기 위해 협업을 사용합니다.)

- [14] 여기서는 DCC 회사가 Acme 회사에 UML 스펙을 제공하면 Acme가 이를 디자인 모델로 통합하는 상황을 가정하고 있음에 주의하십시오. 이는 역으로 전환된 인터넷 도메인 공백이 유용하게 되는 시나리오의 유형입니다.
-

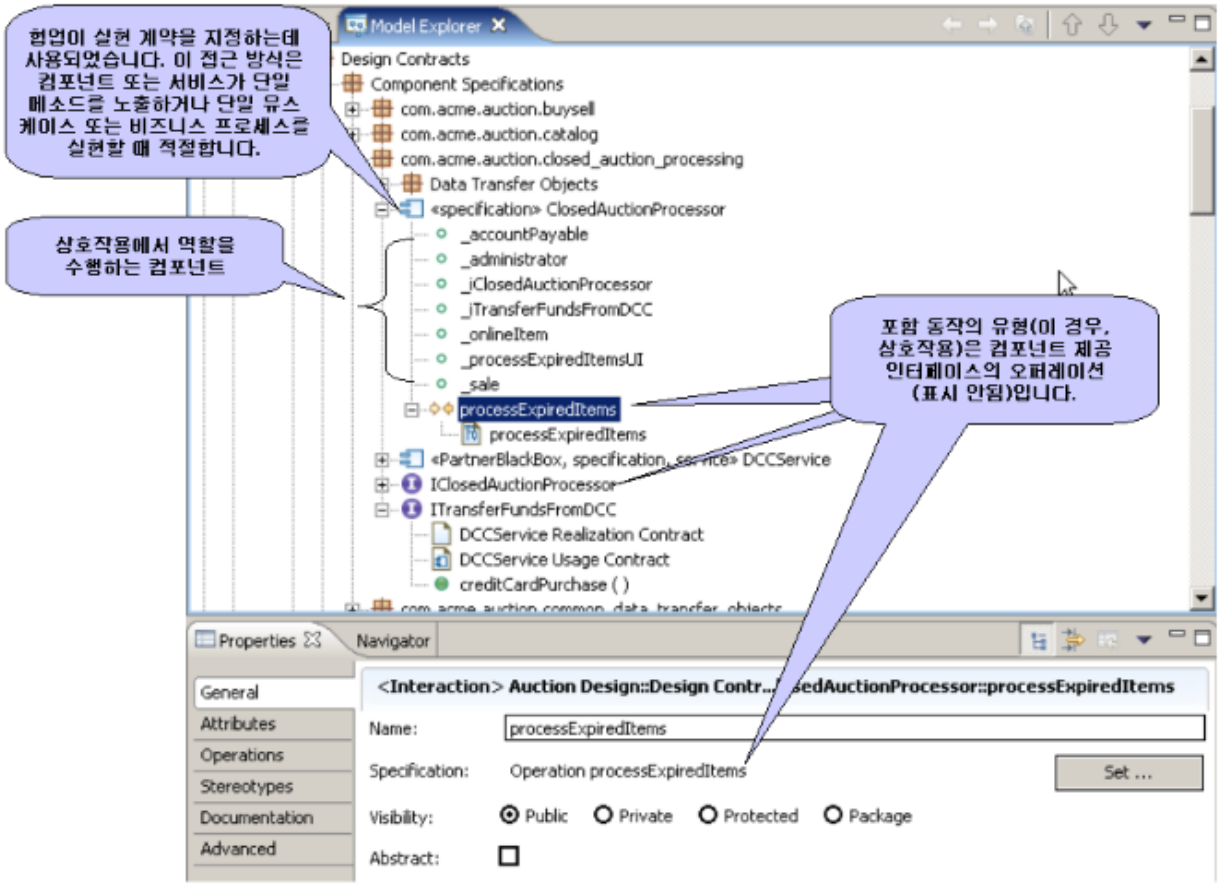


그림 7-4

- 구현 디자인 지정의 가능한 접근 방식이 **그림 7-5**에 표시됩니다. 구현 구조는 오퍼레이션을 포함하는 단순한 클래스를 사용하여 정의합니다. 이 접근 방식은 UML 1.x를 사용하여 작성한 매우 전형적인 디자인 모델입니다. UML 2의 목표를 보다 제대로 유지하는 두 번째 가능한 접근 방식이 **그림 7-6**에 표시됩니다. 여기서, 클래스 대신에 컴포넌트가 사용되며 컴포넌트가 오퍼레이션을 소유하지 않는 대신 동작을 소유함을 알 수 있습니다(이 경우, 상호작용).

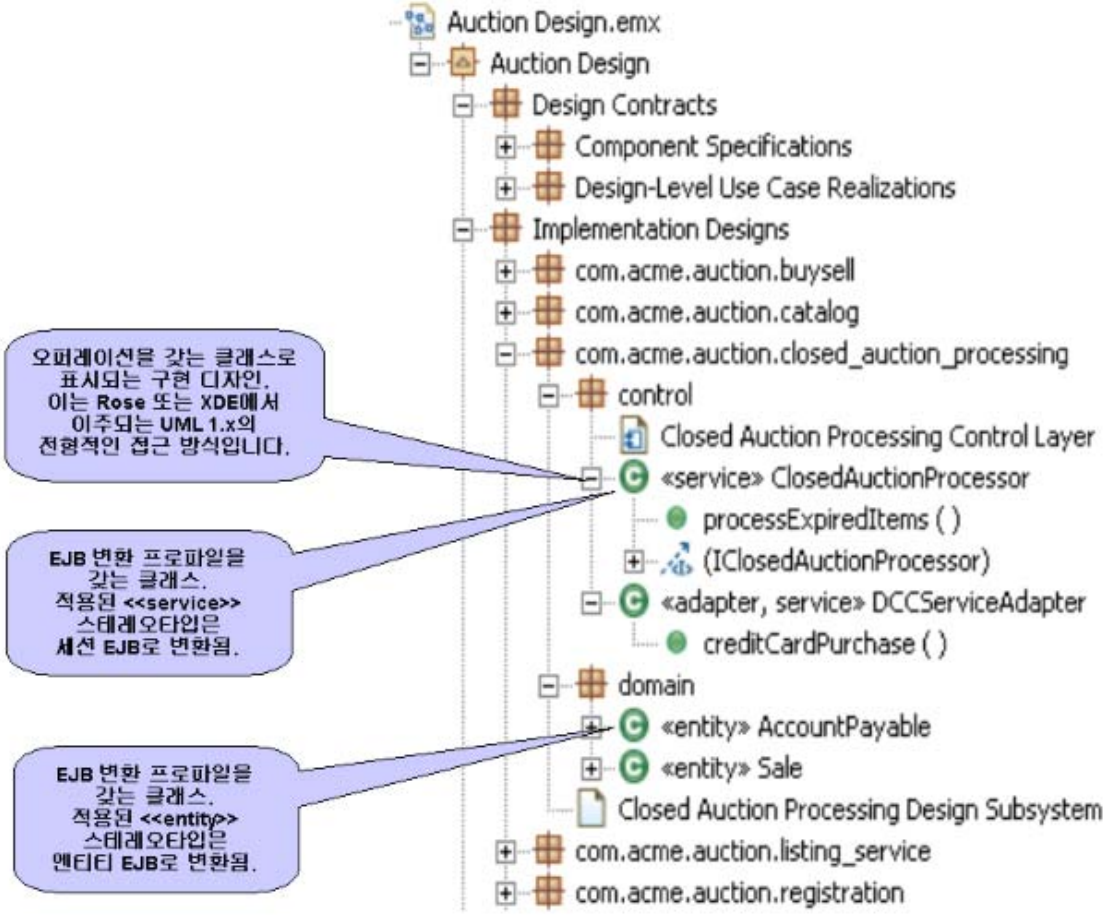

Auction Design.emx
Auction Design
Design Contracts
Component Specifications
Design-Level Use Case Realizations
Implementation Designs
com.acme.auction.buysell
com.acme.auction.catalog
com.acme.auction.closed_auction_processing
control
Closed Auction Processing Control Layer
«service» ClosedAuctionProcessor
processExpiredItems ( )
(IClosedAuctionProcessor)
«adapter, service» DCCServiceAdapter
creditCardPurchase ( )
domain
«entity» AccountPayable
«entity» Sale
Closed Auction Processing Design Subsystem
com.acme.auction.listing_service
com.acme.auction.registration
오퍼레이션을 갖는 클래스로 표시되는 구현 디자인. 이는 Rose 또는 XDE에서 이주되는 UML 1.x의 전형적인 접근 방식입니다.
EJB 변환 프로파일을 갖는 클래스. 적용된 <<service>> 스테레오타입은 세션 EJB로 변환됨.
EJB 변환 프로파일을 갖는 클래스. 적용된 <<entity>> 스테레오타입은 엔티티 EJB로 변환됨.

그림 7-5

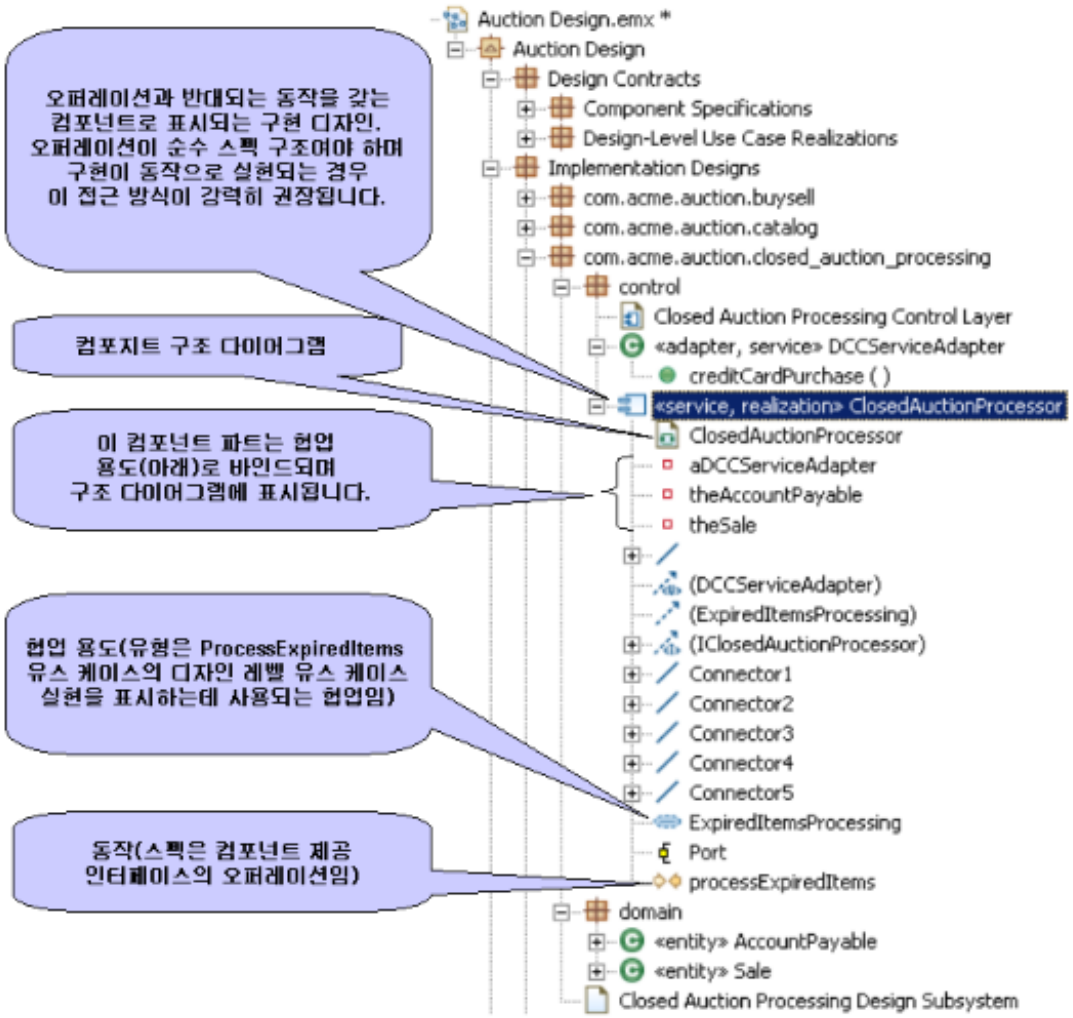


그림 7-6

8. 구현 개요 모델의 내부 조직용 가이드라인

> *XDE/Rose*
> XDE 모델 구조 가이드라인에서 구현 개요 모델이 구현의 서브시스템 레벨 개요를 제공하는 장치로 권장됩니다.그런 다음 각 서브시스템의 세부사항이 서브시스템을 구현한 프로젝트의 코드 모델에 지정됩니다.

엄격히 말해서, RSx에서는 구현 개요 모델을 사용할 필요가 없어야 합니다. 디자인 모델 조직 가이드라인을 따르는 경우 (마지막) 디자인 모델의 조직이 자연적으로 컴포넌트 주변에 구체화되어야

합니다(보다 무거운 <<subsystem>>및 보다 분배 가능한 <<service>> 종류 포함). 그런 다음 변환을 통해 디자인의 패키지가 프로젝트에 맵핑될 수 있습니다. 예를 들어, J2EE 구현의 경우, 여러 Java, EJB, 웹, J2EE 응용프로그램 및 구현이 개발되는 기타 프로젝트에 맵핑됩니다. (사실상 이러한 프로젝트는 이 문서의 기본 개념 및 용어 섹션에서 참고 설명한 바와 같이 솔루션의 구현 모델을 표시함.)

상향식 관점에서 보는 경우, 프로젝트 및 폴더의 관점에서 구현이 어떤 방식으로 조직될지 고려하고, 이를 디자인 모델의 조직의 요소로 포함시켜 직접적인 변환 맵핑이 가능하도록 해야 합니다. 하향식 관점에서 보는 경우, 디자인 노력의 과정에서 디자인 모델의 조직이 전개됨에 따라, 필요한 구현 모델(프로젝트)의 세트를 판단해야 합니다. 어느 방식으로든 디자인 모델의 최종 조직은 프로젝트 및 하위 프로젝트의 개요에 대한 요구를 다뤄야 합니다. 즉, 디자인 모델에서 패키지를 표시하는 다이어그램은 프로젝트 및 해당 하위 폴더의 개요와 같아야 합니다.

그러나 사용자가 계속 초기 단계에 프로젝트 구조를 스케치하려고 할 수도 있으며 시각적으로 보다 명확하게 프로젝트 구조를 설명하려고 할 수도 있습니다(예: 프로젝트 및 폴더를 표현하는 아티팩트가 <<project>> 및 <<folder>> 또는 <<EJB Project>> 및 <<Web Project>>로서 명시적으로 키워드화된 경우). 또 다른 고려사항은 정밀한 구현 아티팩트(예: JAR)를 설명하는 것이 디자인 모델(Rational Software Architect에서 오퍼레이션 이론이 플랫폼 중립적임)에 적합하지 않다는 것입니다. 그러나 이러한 아티팩트는 구현 개요 모델의 포함사항으로 완전히 채택 가능합니다. 따라서 구현 개요 모델을 사용하려는 특정 이유가 있습니다. **그림 8-1**은 샘플 구현 개요 모델을 설명합니다.

마지막으로 구현 개요 모델이 솔루션의 다양한 양상의 비정규 다이어그램을 캡처하는 좋은 위치가 될 수도 있습니다. **그림 8-2**는 이 문서의 대부분의 샘플이 기반으로 하는 경매 시스템의 비정규 상위 개념 다이어그램을 표시합니다.

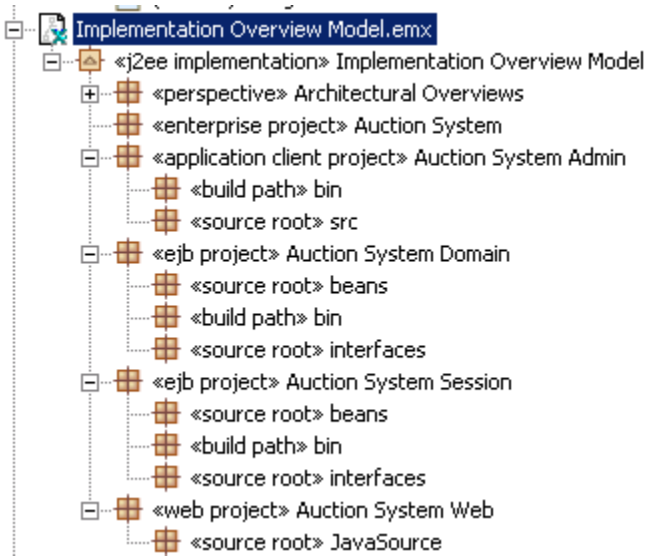


그림 8-1

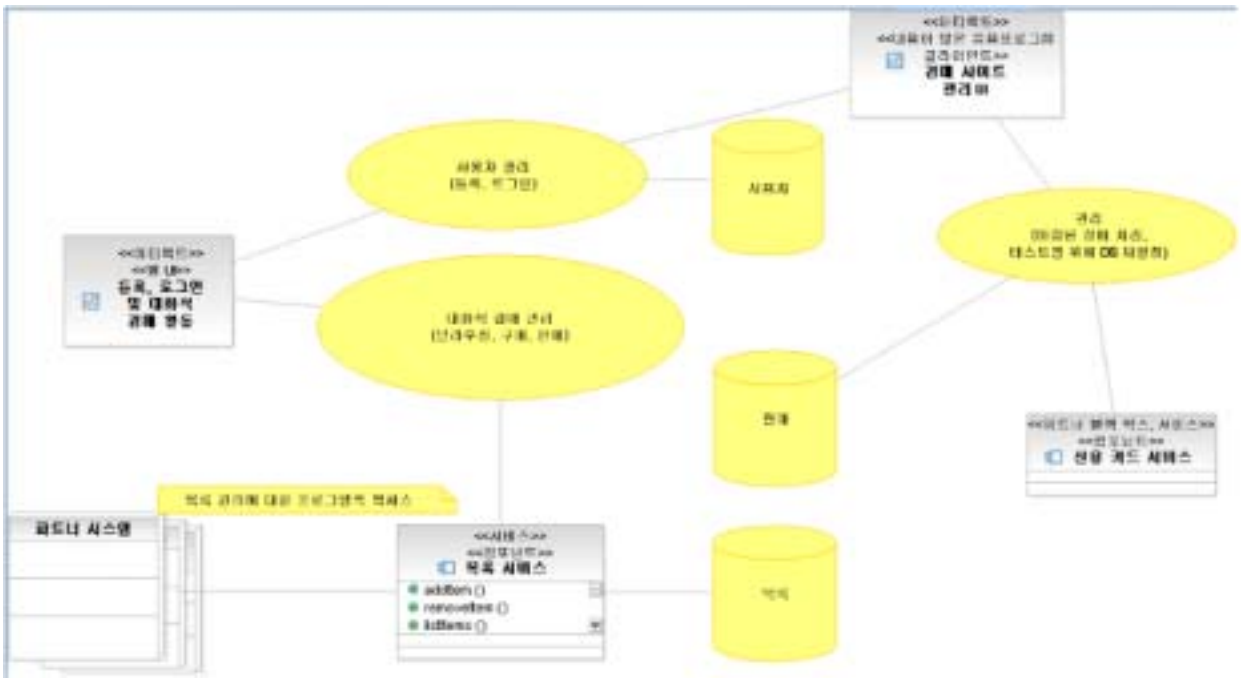

사용자 관리
(등록, 로그인)
사용자
등록, 로그인
및 대화식
권한 부여
대화식 권한 관리
(인증우회, 구축, 관리)
권한
파일나 시스템
목록 서비스
addItem()
removeItem()
isEmpty()
목록

그림 8-2

9. 배치 모델의 내부 조직용 가이드라인

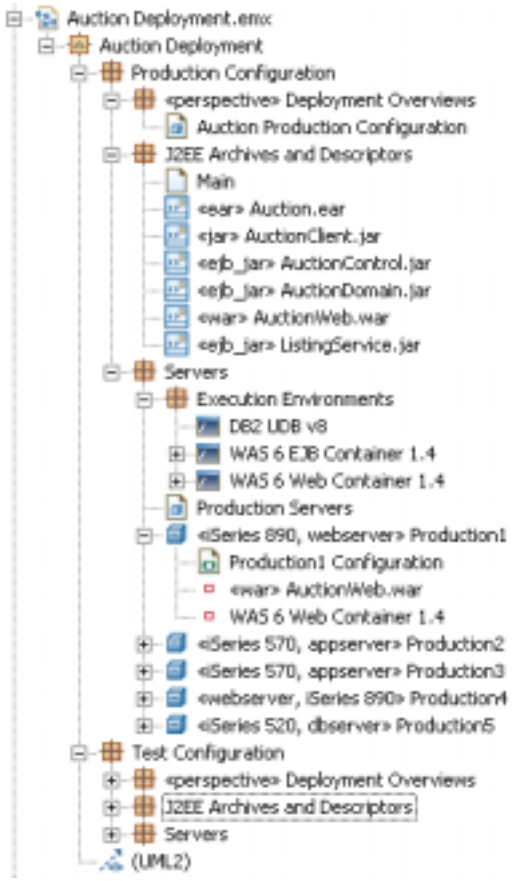


그림 9-1

이 문서에서 언급된 임의의 기타 모델보다 배치 모델에 대해 설명할 필요가 적습니다. 일반적으로 배치 모델링 조직 및 컨텐츠 선택에는 매우 적은 하향 관계가 있으므로 적절한 조치를 수행하십시오. 그렇지만 사용자의 사고를 돕기 위해 가능한 전략 및 약간의 대표 컨텐츠가 위의 **그림 9-1**에 표시됩니다. 이 예에서 다음에 주의하십시오.

1. 제품 구성의 스펙이 테스트 구성의 스펙에서 분리되었습니다.

2. 개요(예: 클러스터, 데이터 센터 또는 엔터프라이즈의)가 <<perspective>> 패키지로 유지보수됩니다.

3. 노드 및 아티팩트의 전문화 및 분류와 관련하여 다음과 같은 간단한 접근 방식이 채택됩니다. 패키징 조합 및 키워드 사용 보다 복잡한 접근 방식은 사용자 고유 환경에서 사용한 자원의 유형을 설명하고 문서화하는데 적합한 특성과 특수 스테레오타입을 정의하는 특수 UML 프로파일을 개발하는 것입니다.

10. 모델링 파일을 사용하여 소프트웨어 아키텍처 문서 표시

모델을 체계화하는 주어진 도구로서(예: 다이어그램 링크와 교차 모델 참조를 포함하는 여러 모델 파일 지원) RUP 소프트웨어 아키텍처 문서와 "4+1 아키텍처 보기"를 표시하는 모델을 작성하는 것은 사실상 아주 쉬운 일이 됩니다.

가장 단순하게는, 다음 **그림 10-1**의 선을 따라 작업을 수행할 수도 있습니다. 모델링 파일을 작성하고 4+1 보기에 해당하는 패키지의 단순 세트로 채우십시오. (이 예의 시스템은 보통 동시에 발생하는 방식으로 표시되지 않으므로 예에서는 프로세스 보기의 패키지 없이 표시됩니다.)

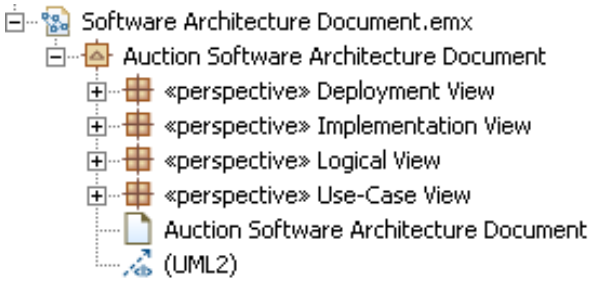


그림 10-1

그런 후, 다음 **그림 10-2**에 제시된 선을 따라 기본 다이어그램을 구성하십시오. 이 다이어그램에 참고사항 또는 텍스트를 추가할 수도 있습니다.

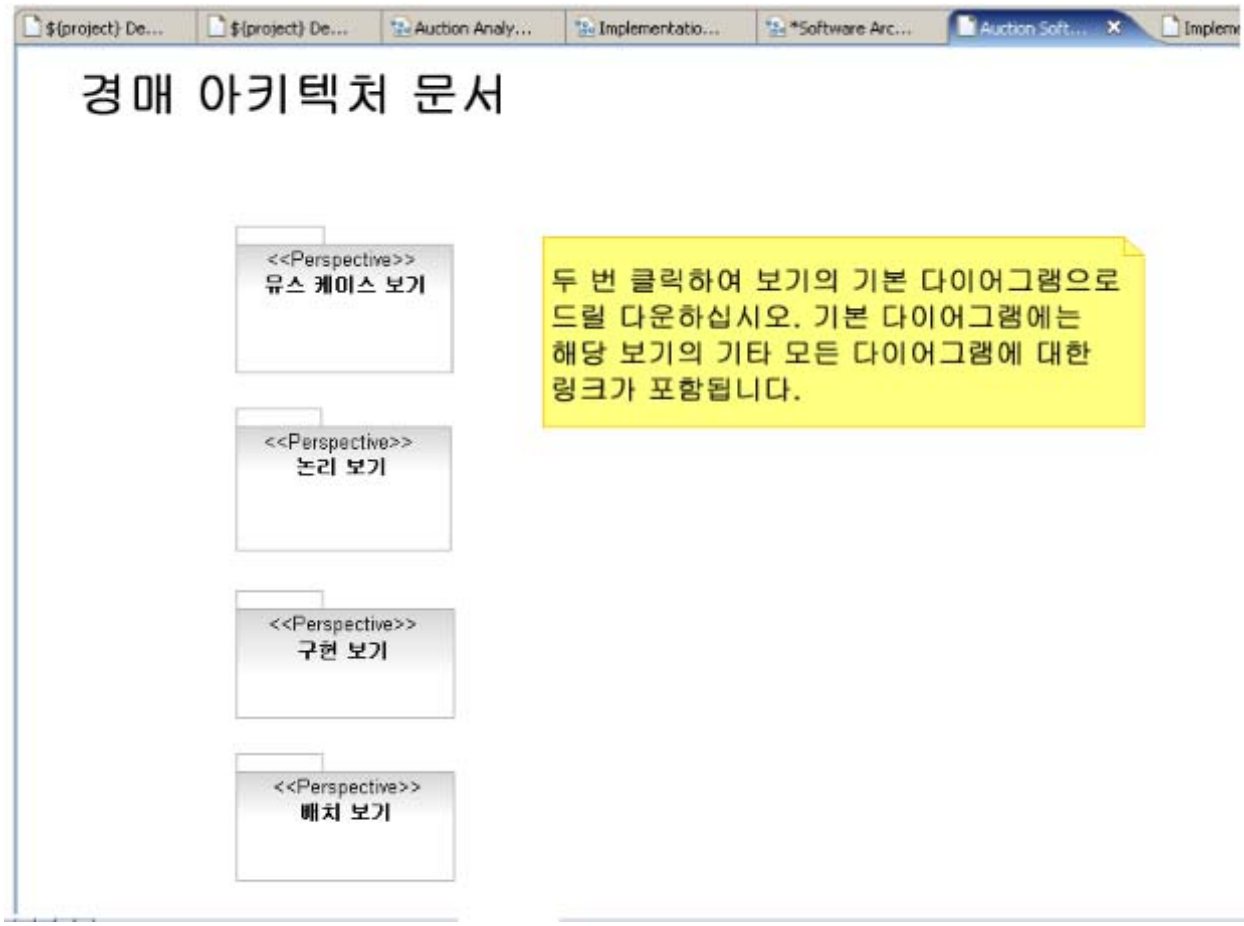

${project} De...
${project} De...
Auction Analy...
Implementatio...
*Software Arc...
Auction Soft...
경매 아키텍처 문서
<<Perspective>>
유스 케이스 보기
<<Perspective>>
논리 보기
<<Perspective>>
구현 보기
<<Perspective>>
배치 보기
두 번 클릭하여 보기의 기본 다이어그램으로 드릴 다운하십시오. 기본 다이어그램에는 해당 보기의 기타 모든 다이어그램에 대한 링크가 포함됩니다.

그림 10-2

그리고 나서, 다음 접근 방식을 사용하여 소프트웨어 아키텍처 문서 모델링 파일에 다이어그램을 작성하십시오.

- 기타 모델링 파일의 UML 시맨틱 요소를 사용하여 작성되고 해당 기타 모델링 파일에 없지만 아키텍처 문서의 일부로 필요한 새 보기를 설명하는 다이어그램을 작성하십시오.

- 소프트웨어 아키텍처 문서 모델링 파일에 있는 기하학적 모양 및/또는 "임시" UML 요소로 구성된 다이어그램을 작성하십시오. (이러한 UML 요소는 문서 또는 설명 용도로만 사용되어야 하며 설명되는 솔루션의 실제 구현에는 시맨틱적 중요성이 없어야 합니다.)

- 단지 기타 모델링 파일의 기존 다이어그램으로의 링크를 포함하는 다이어그램을 작성하십시오. (이 기법은 아키텍처 문서 모델링 파일이 독자에 의해 사용되는 기타 모델링 파일과 함께 분배되는 경우 제대로 작동합니다. 그렇지 않고 아키텍처 문서가 웹으로 출판될 예정인 경우, 기타 접근 방식 중 하나를 대신 따르십시오.)

11. 팀 개발 및 모델 관리 고려사항

이 섹션에서는 모델을 여러 모델링 파일로 파티션하는 시기와 이유에 대한 일부 고려사항을 소개합니다. 이러한 문제에 대한 보다 포괄적인 설명은 RSx 온라인 도움말을 참조하십시오. 여기서는 사용자가 병렬 개발에 대한 개념 및 아티팩트의 다중 사본에 병렬로 작성된 변경사항의 병합 개념에 익숙한 것으로 가정합니다.

첫 번째, 빠른 검토: "기본 개념 및 용어"에서 RUP 가 인식하는 유스 케이스, 분석 및 디자인과 같은 여러 모델 유형에 대해 논의했습니다. RSx 에서의 경우를 설명하기 위해 예제가 주어졌습니다…

- 여러 응용프로그램을 빌드하는 경우 각 유형의 모델이 여러 개 있을 수 있습니다(예: 여러 유스 케이스 모델, 여러 분석 모델 등).

- 하나의 모델(논리적 개념)이 하나 이상의 모델링 파일로 유지될 수 있습니다(예: " 응용프로그램 'X'의 디자인 모델"이 단일 모델링 파일 또는 여러 모델링 파일의 컬렉션으로 유지됨).

파티션 모델

모델을 복수 모델링 파일로 파티션하는 기법은 RSx 온라인 도움말에 포함되며 여기에서는 설명되지 않습니다. 여기서는 파티션하는 시기와 이유에 대해 다룹니다. 특정 모델을 여러 모델링 파일로 유지보수하도록 선택하는 경우는 다음 두 가지입니다.

1. 모델이 유지보수할 수 없을 정도의 크기가 되었을 때, 또는 패키징 구조가 유지보수할 수 없는 수준일 때[15]

2. 모델링 파일에 동시에 너무 많은 변경사항이 전달되어 두 개보다 많은 변경 컨트리뷰터[16]를 병합해야 하는 경우. (이 설명을 이해하기 어려운 경우 아래를 계속 읽으십시오.)

팀의 모델링

구성 관리 정책에서 모델의 병렬 개발을 사용 가능하게 할 경우[17], 파일에 조정되지 않은 변경사항이 작성됩니다(그 결과 파일이 표시하는 논리 모델 또는 모델 서브세트에 작성됨). 언젠가는 변경사항이 병합되어야 합니다. 변경사항 중 일부가 서로 충돌할 경우, 해당 병합은 "**특별**"한 것으로 고려됩니다. 왜냐하면 누군가 충돌하는 변경사항 중 어떤 것을 "선택"할지 결정해야 하기 때문입니다.('평범'한

---

[15] 사용자 커뮤니티에서 사용 중인 시스템에서 파일이 너무 커졌을 때 파티션해야 합니다. 예를 들어, 디스크 크기가 30MB 까지 증가한 모델은 1GB RAM 시스템으로 일일 작업을 하기가 어렵습니다. 이러한 시스템에서는 RAM 에서 5 – 10MB 정도의 모델을 항상 유지하려고 할 수 있습니다. 또는 RAM 을 업그레이드할 수 있습니다. 이는 상대적으로 저렴하면서 효과가 좋은 솔루션입니다. 스왑 파일이 없는 2GB RAM 시스템은 거의 모든 Eclipse 오퍼레이션을 더 빨리 수행하며 결과적으로 큰 모델도 다시 잘 동작하게 해줍니다.

[16] RSx 모델 병합 도구는 최대 세 개의 컨트리뷰터 병합을 지원합니다(두 개의 '변경' 컨트리뷰터와 한 개의 '기본' 컨트리뷰터(즉, 공통 상위)). '변경' 컨트리뷰터가 두 개 이상인 경우 병합하면 계단식으로 되며, 그 시점부터 두 '변경' 컨트리뷰터 중 하나는 세션에서 이전에 완료한 병합의 결과 세트가 됩니다.

[17] "병렬" 개발 정책의 예:

- 비독점적 액세스에 대해 모델링 파일을 체크아웃할 수 있음
- 여러 종사자(practitioner)의 개발 스트림에서 병렬로 모델링 파일에 대해 작업

병합이란 조정되지 않은 변경사항이 충돌되지 않으며, 사람의 간섭 없이 모델 병합 엔진에서 병합을 수행할 수 있는 것입니다.)
특별한 병합은 수행하기 어렵습니다. 어떻게 하면 이를 줄일 수 있을까요?

두 가지 기본 방법으로 충돌을 피하고 결과적으로 특별한 병합을 피할 수 있습니다. 첫 번째 방법은 "강력한 아키텍처"입니다. 두 번째 방법은 "강력한 소유권"입니다. 이는 밀접한 관계를 가집니다. 즉, 강력한 아키텍처가 *강력한* 소유권을 가능하게 합니다.

방법 #1: 강력한 아키텍처

여기서 "강력한 아키텍처"는 주로 분해를 의미합니다. 여기에 적용되는 *아키텍처* 분해의 원칙은 오브젝트 지향 개발, 컴포넌트 기반 디자인 및 서비스 지향 아키텍처에서 적용되는 것과 동일합니다.

- 비즈니스 기능을 최대한 분리하기 위한 노력
- 밀접하게 연결되어야 하는 사항을 함께 그룹화. 그룹 간의 분리
- 분해에 많은 수의 '조직'이 있는 경우, 인력 구성 모델에 따라(강력한 아키텍처 및 강력한 소유권은 서로 밀접한 관계임에 주의) 해당 조직을 보다 밀접한 묶음으로 그룹화하십시오(모델링 용어로는 UML 패키지를 의미함).
- 언제나 많은(*일부의 경우 전부*) 분해 단위로 처리해야 하는 사항이 생깁니다. 이들을 '공통' 패키지에 그룹화하고 각 개발 반복을 계획하여 반복의 시작에 "공통" 부분을 안정화하는 데 초점을 맞추는 "폭포(waterfall)"를 포함하도록 하십시오.
- 시간 요소도 있습니다. 솔루션에 대해 추상적인 이해부터 보다 구체적인 이해로 발전함에 따라, 아키텍처(및 모델)에 대한 최적의 조직에 대한 개념이 전개됩니다. 각 단계에서 다음 단계로 전이할 때 모델 리팩토링(재구성) 활동에 대한 계획을 세워야 합니다(비즈니스 분석, 요구사항, 응용프로그램 분석, 상위 레벨 디자인 등).

솔루션이 상호 종속적이고 밀접하게 연관된 경우, 사용자의 아키텍처에 작업이필요하거나 문제점 도메인의 특성에 문제가 있어 해당 문제점을 분해할 수 없음을 의미합니다. 두 경우 모두 다음과 같은 선택사항이 있습니다.

- 기타 아티팩트에 영향을 줄 수 있는 모든 변경사항에 대해 실제 공간을 공유하고 매우 활동적으로 커뮤니케이션하는 아주 작은 팀에 프로젝트를 지정하여 해결하십시오.
- 특별한 병합을 많이 수행할 수 있도록 준비하십시오.

XDE/Rose

Rose 또는 XDE 사용자인 경우 특별한 모델 병합을 경험했을 수 있습니다. RSx 에서의 특별한 병합은 이에 비해 훨씬 간단합니다. 한 가지 주요 차이점은 RS*x*가 페탈 파일 또는 하위 단위의 *계층 구조*와 반대로, 관련된 모델링 파일을 *뭉뚱그려* 조작한다는 점입니다. 이 결정적인 차이 덕분에 RS*x*의 실제 레벨에서 상위 레벨 논리 분해가 더 잘 지원됩니다.

또한 **많은** 병합 경험과 더 좋은 도구로 비교 병합을 지원합니다.

- RS*x* 백엔드 병합 엔진은 XDE 백엔드보다 10,000 배나 더 빠릅니다.
- 또한 "컴포지트 델타" (다이어그램으로 변경사항을 정렬하고 그룹 및/또는 원자성 오퍼레이션을 사용하기 위해 큰 다이어그램 작성을 그룹화하는 그룹핑 메커니즘), "모델 무결성 보호", "원자성" 및 "계단식 델타 처리"와 같은 여러 기술을 구현합니다. 이는 병합에 의한 손상 가능성을 파일의 단순 편집에서와 같은 레벨로 줄여줍니다.
- 이제 레이아웃/표면 충돌을 해결하기 위해 GUI 를 병합하는 실제 비주얼 다이어그램이 제공됩니다.

방법 #2: 강력한 소유권

일단 강력한 아키텍처 분해를 설정하면 아키텍처 컴포넌트의 "강력한" 소유권을 개별 종사자(practitioner) 또는 작은 팀에 맵핑하는 것은 상당히 간단합니다(전문화된 스킬 세트 제외). 모델에서 각 논리 패키지(또는 분기)에 대해 한 명의 종사자(practitioner)가 독점적으로 작업할 수 있으면, 해당 모델로 수행된 병합은 (모델이 단일 또는 복수 모델링 파일로 저장되었는지 여부와 관계 없이) 대체로 평범한 병합이 됩니다. 이는 활동에 대해 비정규 커뮤니테이션을 많이 하는 경향이 있는 구성원이 속하는 작은 팀이 배타적으로 각 분기를 작업하는 경우에도 똑같이 적용됩니다.

모델을 여러 모델링 파일로 분할하면 특별한 병합을 피할 수 있을까요? 그렇지 않습니다. **아키텍처 상호종속성은 논리 현상이지 실제 현상이 아닙니다**. 한 모델을 여러 모델링 파일로 파티션하면, 요소 상호 종속성의 표시가 파일 내 참조 대신 파일 간 참조가 됩니다. 그러면 충돌 해결이 전혀 쉬워지지 않습니다(실제로는 더 어렵게 만듭니다). 그리고 파일 간 참조를 사용하면 잠재적인 파손 지점을 사용하는 것입니다(사이드바 참조).

사이드바: 상호 모델링 파일 참조

두 개의 모델링 요소가 다른 모델링 파일에 있으며 이들 사이에 관계를 작성할 때는 항상 "상호 모델링 파일 참조"를 작성합니다. 모델링 파일(.emx 파일)이 호스트 OS 파일 시스템에 노출되어 있고, 모델링 파일을 이동하거나 이름을 바꿀 수 있으며, 그렇게 하지 않을 경우 Eclipse 환경 밖에서 수정할 수 있기 때문에 이들 참조는 잠재적 파손 지점을 표시합니다. 그러나 모델링 파일을 항상 Eclipse 환경을 통해 수정하고 변경 관리하며 다음 가이드라인을 준수하는 한 파손이 발생하지 않습니다.

모델링 파일(상호 참조하는 모델링 파일의 컬렉션)의 '종결'을 작업(편집)할 때마다 해당 종결의 모든 모델링 파일을 작업 영역에 표시해야 합니다. 이는 종결 시 모든 모델링 파일이 같은 프로젝트 안에 있어야 함을 의미하지는 않습니다. 그러나 단일 프로젝트를 사용하면 표준 CM 이 워크플로우에서 모든 모델 파일이 같은 프로젝트에서 함께 이동하기 때문에 일반적으로 모든 모델이 표시됩니다.

요약:

- 강력한 아키텍처가 없거나 강력한 아키텍처가 있지만 강력한 소유권이 없는 경우, 아무리 파티션해도 해결할 수 없는 특별한 병합이 자주 발생합니다.

- 강력한 아키텍처과 강력한 소유권이 있는 경우, 특별한 병합의 빈도를 많이 줄일 수있습니다(그러나 아주 없애지는 못함). 컴포넌트 상호 종속성이 항상 존재하기 때문에 특별한 병합을 제거할 수는 없습니다. 앞에서 언급된 '공통' 요소는 이런 예제 중 하나입니다.

- 모델을 여러 파일로 파티션하는 것은 충돌하는 변경사항 없이 복수 종사자(practitioner)가 병렬로 모델링 파일에 대해 작업할 수 있도록 논리적으로 모델을 구조화하는 것보다는 중요하지 않습니다.

- RS*x*를 사용하면 다른 어떤 모델링 도구를 사용할 때보다 더 빠르고 더 효과적으로 모델을 병합할 수 있습니다.

그럼 언제 모델을 파티션해야 합니까? 모델의 크기가 하드웨어의 한계를 넘어서기 시작하는 경우, 그리고 작성한 모델링 파일을 일반적으로 *배타적*(임의 시점에 오직 한 명의 팀 구성원만 파일을 체크아웃함) **및** *분리적*(관련된 모델 요소를 포함한 기타 파일에 액세스하지 않고도 파일에 대부분의 변경사항을 작성할 수 있음)으로 작업하는 경우가 아니면 모델 파티션을 하지 마십시오.

후기: RSx 의 버전 6.x 및 7.x 사이의 변경사항

RSx 가 처음 릴리스되었을 때에는(버전 6.x)Rose 및 XDE 가 지원하는 '하위 단위'라는 개념을 지원하지 않았습니다 . 여기서 하위 단위란 모델의 서브세트를 포함하는 모델링 파일로 모델 탐색기 보기에 표시된 모델의 논리적 보기에 표시되지 않는다는 의미에서 '투명'합니다. 대신 모델 파티션이 여러 최상위 레벨 모델을 정의하는 것으로 제한되었습니다. (여기서 "최상위 레벨"이란 모델 탐색기 보기에서 각 모델링 파일이 별개의 최상위 레벨 항목으로 표시됨을 의미합니다.)

또한 RSx 는 기존 모델의 UML 패키지를 선택하고 해당 패키지를 기반으로 "모델을 작성"할 수 있는 기능을 제공합니다. 즉, 이는 새 모델링 파일을 작성하기 위해 패키지를 "잘라내는" 것과 같습니다. 그 결과로 모델 탐색기 보기에서 패키지가 새 상위 레벨 논리 모델로 표시됩니다. 본래 계층 구조 포함 구조의 가시성은 없어지지만 이런 방법으로 패키지가 "잘린" 위치에서 모델링 파일을 열 때마다 모든 "잘린" 모델들도 열리게 됩니다. 이렇게 하면 불필요한 체크아웃이 생길 뿐만 아니라 모델 탐색기 보기가 복잡해지고 편집기 분할창에 많은 수의 탭이 표시되여 결국 탐색이 어려워집니다.

RSx 의 7.0 버전에서의 하위 단위 지원의 추가로 계층 구조에 대한 가시성의 손실 없이 모델을 여러 파일로 파티션할 수 있으며, 편집기 분할창에서 "모델 편집기" 탭을 제거하여 기타 탐색 문제점을 극복할 수 있습니다.

그러나 이전 6.x 모델 구조화에서 경험한 내용 중 오늘날에도 여전히 적용되는 부분이 있습니다.위에서 언급한 대로 패키지가 잘린 위치에서 모델링 파일을 열면 모든 잘린 모델이 열립니다. "모델 작성" 기능을 사용하는 대신 미리 파티션 전략을 계획하여 이런 문제점을 피할 수 있습니다.

"모델 작성" 기능을 따르는 경우, 일반적으로 단일 모델링 파일로 시작하여 솔루션 아키텍처의 조직을 표시하는 패키지를 작성합니다. 예를 들어, 각 주요 컴포넌트 또는 솔루션의 서브시스템에 대한 패키지가 있습니다(여기서 '컴포넌트' 및 '서브시스템'은 정규 UML 시맨틱 정의가 아닌 이전의 컴퓨팅 시점으로 거슬러 올라가는 비정규 용어의 사용을 반영함). 또한 "공통", "유틸리티" 또는 "프레임워크" 컴포넌트에 대한 패키지가 있습니다. 이러한 모델이 증가함에 따라 응용프로그램 특정 컴포넌트에 해당하는 패키지가 별개의 모델링 파일로 "잘려" 해당 컴포넌트를 빌드하는 팀이 (이론상) 분리적으로 해당 모델 파일에 대해 작업할 수 있습니다. 그러나 실제로는 컴포넌트 특정 모델 파일을 열 경우 본래의'마스터' 모델(여기에 '공통' 부분이 있음)이 열리게 되며, 그 결과 다른 응용프로그램 특정 모델까지 모두 열리게 됩니다. 그 결과 앞에서 설명한 대로 불필요한 체크아웃을 하게 되며 탐색이 어려워집니다.

보다 바람직한 접근 방식은 각 응용프로그램 특정 컴포넌트에 대한 별도의 모델 정의를 '공통' 컴포넌트(또는 구현 아키텍처의 추상 계층을 반영하는 공통/재사용가능 컴포넌트의 연속 계층을 정의하는 다중 모델)에 대한 모델과 함께 미리 계획하는 것입니다. 그러면 모든 응용프로그램 특정 컴포넌트 모델을 이를 소유하는 팀이 열 수 있으며, 해당 응용프로그램 특정 모델이 종속되는 '공통' 모델만 열리게 됩니다.