Rational Unified Process 를 사용하여 대규모 시스템 개발

Maria Ericsson

Rational Software 백서

TP 156

Rational®
the software development company

목차

히스토리 1

상호 연결된 복합 시스템 1

소프트웨어 개발 라이프사이클 2

시스템 개발 워크플로우 및 아티팩트 3

상호 연결된 복합 시스템 개발 4

- 분해 기준 4
- 조직 5
- 상위 시스템의 라이프사이클 5
- 하위 시스템의 라이프사이클 8

상호 연결된 복합 시스템 유스 케이스 11

상호 연결된 복합 시스템 디자인 모델 12

상호 연결된 복합 시스템 정보 세트 13

상호 연결된 복합 시스템 아키텍처 14

시스템 간 관계 15

응용프로그램 영역 16

- 대규모 시스템 16
- 분산 시스템 17
- 레거시 시스템의 재사용 17
- 사전 조정된 패키지 사용 17

요약 17

참조 18

히스토리

이 문서는 Ivar Jacobson, Karin Palmkvist 및 Susanne Dyrhage 가 1995 년 ROAD 5, 6 월호에 기고한 "Systems of Interconnected Systems"[1]의 내용을 발췌한 것입니다. 이 문서는 여러 대규모 시스템 개발 프로젝트 내용을 기반으로 하며 RUP 버전 5.1([2])과 UML([3])에 모두 적용할 수 있습니다.

상호 연결된 복합 시스템

대규모 시스템을 개발하는 경우 복잡도가 상당히 증가합니다. 좀 더 복잡한 아티팩트 세트를 이해해야 하며, 대규모 자원 세트 관리를 위한 오버헤드도 발생합니다. 이 백서에서는 추가된 복잡도 오버헤드를 제어하는 데 사용되는 아키텍처 패턴에 대해 설명합니다. [4]에서 논의된 다른 내용 중 아키텍처 패턴을 **상호 연결된 복합 시스템**이라고 합니다.

이 구조는 명령 및 제어 시스템이나 고도의 통합 IT 솔루션과 같은 대규모 또는 복잡한 시스템을 빌드할 때 유용합니다. 이러한 유형의 "수퍼 시스템"은 대부분의 경우 여러 별도 파트로 분할되며 각 파트는 별도 시스템으로서 독립적으로 개발됩니다. 수퍼 시스템은 한 세트의 상호 연결된 시스템으로 구현되며 각 시스템은 상호 통신을 통해 수퍼 시스템의 기능을 수행합니다. 이러한 시스템 중 전체 기능을 나타내는 특정 시스템을 **상위(superordinate) 시스템***이라고 합니다.* 전체 기능 중 일부를 나타내는 나머지 시스템은 **하위(subordinate) 시스템**이라고 합니다. 상위 시스템은 해당 시스템을 구현하는 하위 시스템과 명확히 구분됩니다. 다른 유형의 시스템 간의 관계는 명확합니다. 즉, 상위 시스템의 관점에서 볼 때 하위 시스템은 서브시스템입니다(그림 1 참조).

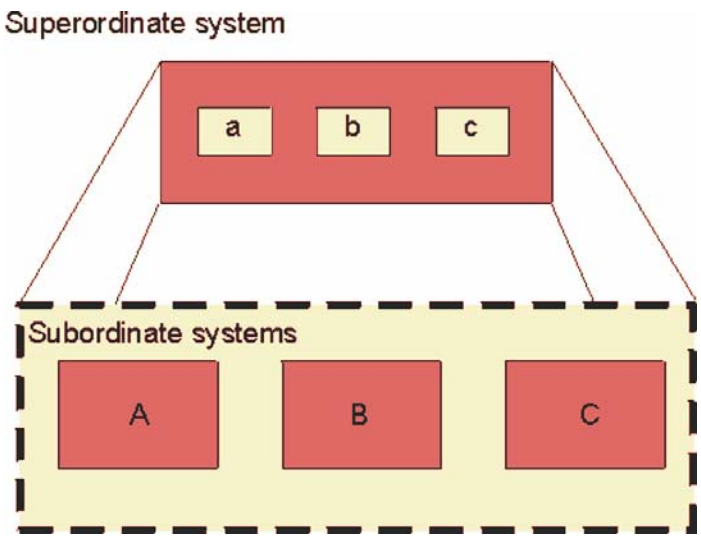

그림 1. 상위 시스템의 스펙은 상호 연결된 복합 시스템에 의해 구현됩니다. 즉, 시스템 A, B, C 는 각각 상위 시스템의 서브시스템 a, b, c 의 구현입니다.

상위 시스템과 하위 시스템을 구분하는 데 따른 이점은 다음과 같습니다.

- 하위 시스템은 판매 및 전달을 포함한 모든 라이프사이클 활동에서 개별적으로 관리할 수 있습니다.
- 특정 하위 시스템을 상호 연결된 시스템의 다른 시스템에 플러그인함으로써 다른 상위 시스템을 쉽게 구현할 수 있습니다.

- 시스템을 처음 빌드할 때부터 해당 시스템이 상호 연결된 복합 시스템인지 여부를 항상 알 수 있는 것은 아닙니다. 즉, 처음에는 "단순" 시스템 보기 작업으로 시작하여 라이프사이클 후반부에 상호 연결된 시스템 패턴의 시스템을 적용해야 하는지 여부를 결정할 수 있습니다.
- 상위 시스템의 새 버전을 개발하지 않고 하위 시스템을 내부적으로 변경할 수 있습니다. 상위 시스템의 새 버전은 주요 기능이 변경될 때만 필요합니다.

각 하위 시스템에는 연관된 아티팩트 세트가 있으며 해당 아티팩트 간에는 명확한 추적성이 존재합니다. 하위 시스템의 아티팩트 세트와 상위 시스템의 해당 아티팩트 세트 간에도 추적성이 유지보수됩니다. 각 하위 시스템은 자체 라이프사이클 단계(도입/인식(Inception), 정제(Elaboration), 구현/구축(Construction) 및 전이(Transition))를 갖는 별도 개발 프로젝트로 관리할 수 있습니다.

대규모 "수퍼 시스템"을 빌드하는 경우 하위 시스템을 보다 세분화하여 상호 연결된 복합 시스템으로 만들어야 합니다.

소프트웨어 개발 라이프사이클

RUP 에서는 개발 라이프사이클을 관리 관점과 개발 관점에서 나타내고 설명합니다(그림 2 참조).

관리 관점에서 볼 때 네 가지 라이프사이클 단계를 통해 시스템을 개발하거나 새 시스템을 생성합니다. 개발 관점에서는 시스템 버전을 반복 개발함으로써 시스템이 점진적으로 완벽해집니다. RUP 에서는 반복에서 수행하는 활동이 원칙 세트로 그룹화됩니다. 각 원칙은 시스템의 특정 측면에 대해 중점 설명하며 이를 통해 시스템 모델 또는 문서 세트가 생성됩니다.

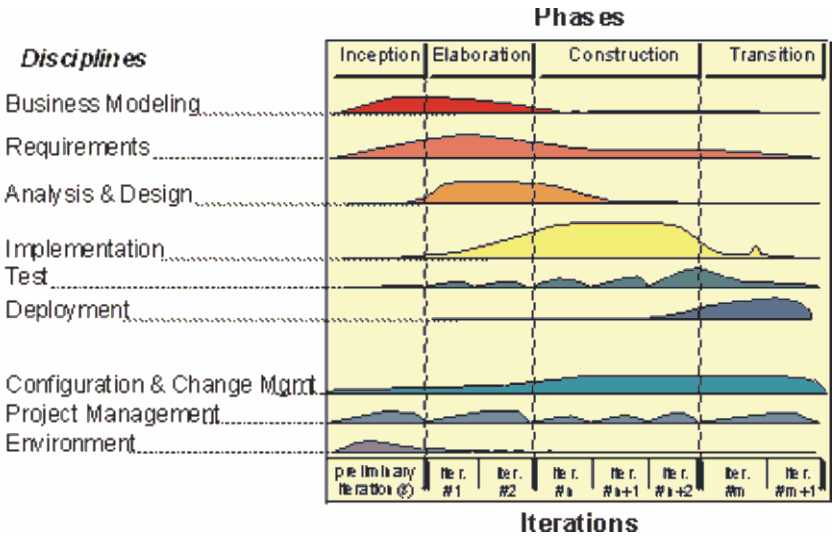

그림 2. 반복 모델

이를 상호 연결된 복합 시스템에 적용함으로써 상위 시스템은 물론 해당 하위 시스템 각각이 모두 자체 라이프사이클을 갖게 되며 일반적으로 별도 프로젝트로 간주됩니다.

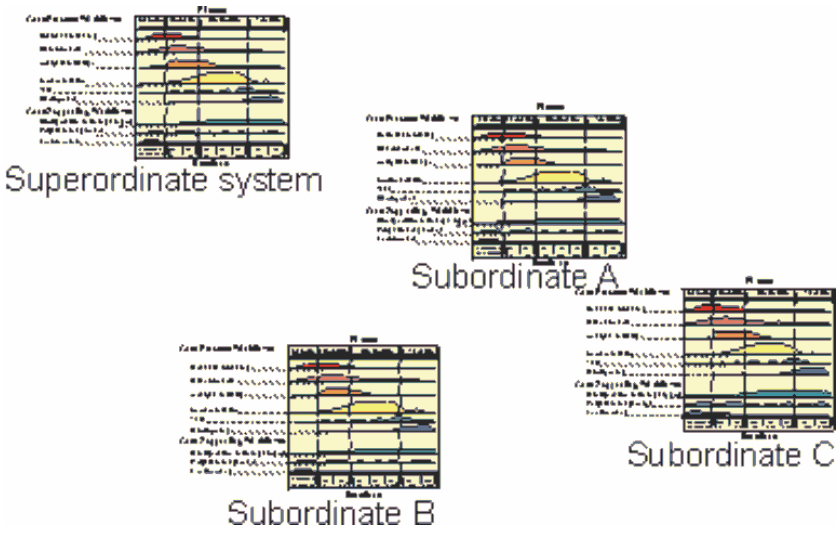

그림 3. 각 시스템(상위 시스템 및 하위 시스템)은 자체 라이프사이클을 갖습니다.

이러한 라이프사이클은 물론 종속성을 갖습니다. 이러한 종속성을 올바르게 관리하는 것은 상호 연결된 복합 시스템을 개발하는 데 있어 중요한 과제 중 하나입니다. 해당 종속성의 예는 다음과 같습니다.

- 라이프사이클의 시간 종속성이 존재합니다. 상위 시스템의 라이프사이클이 먼저 시작됩니다. 상위 시스템에서 하나 이상의 반복이 진행되고 하위 시스템의 인터페이스가 상대적으로 안정되면 하위 시스템의 라이프사이클을 시작할 수 있습니다. 실제로, 상위 시스템에서 하나 이상의 반복이 진행되어야 해당 하위 시스템을 파악할 수 있습니다.
- 하위 시스템의 인터페이스가 안정되면 상위 시스템의 라이프사이클을 유지보수할 수 있습니다. 이는 문제가 발생하는 경우에만 하위 시스템의 인터페이스를 변경해야 하는 적극적인 개발 활동이 수행됨을 의미합니다.
- 하위 시스템의 인터페이스는 상위 시스템 개발자가 소유합니다. 인터페이스에 대한 자세한 사항은 [3] 및 [5]를 참조하십시오.
- 하위 시스템의 인터페이스를 구현하는 클래스는 하위 시스템 개발자가 소유합니다.

시스템 개발 워크플로우 및 아티팩트

일반적으로, 상위 시스템과 하위 시스템 모두 동일한 아티팩트 세트로 개발할 수 있으며 이 때 비컴포지트 시스템에 일반적인 동일한 워크플로우를 사용하여 개발하는 것으로 가정합니다. 이 내용을 설명하려면 먼저 해당 아티팩트와 워크플로우에 대해 설명해야 합니다. RUP 에서는 다음과 같은 다섯 가지 원칙을 사용합니다(그림 4 참조).

- 비즈니스 모델링 – 시스템이 해결해야 하는 요구와 문제점을 보다 잘 이해하기 위해 시스템을 사용할 조직을 평가합니다. 이 원칙을 적용하면 비즈니스 유스 케이스 모델과 비즈니스 분석 모델이 생성됩니다. 이 원칙은 선택적입니다. 시스템을 채택할 조직이 매우 단순한 조직인 경우에는 많은 이점을 제공하지 않습니다.
- 요구사항 – 사용성에 중점을 두고 요구사항을 캡처하고 평가합니다. 이 원칙을 적용하면 액터가 시스템과 통신하는 외부 단위를 나타내고 유스 케이스는 트랜잭션 시퀀스를 나타내며 액터에 측정 가능한 결과 값을 제공하는 유스 케이스 모델이 생성됩니다.

- 분석 및 디자인 – 원하는 구현 환경과, 시스템 구현/구축(Construction)에 해당 환경이 미칠 영향을 연구합니다. 이 원칙을 적용하면 오브젝트가 유스 케이스 플로우 수행을 위해 통신하는 방법을 보여주는 유스 케이스 실현(realization)을 포함하여 오브젝트 모델(디자인 모델)이 생성됩니다. 이 모델에는 제공된 오퍼레이션 관점에서 해당 책임을 지정하는 클래스 및 서브시스템에 대한 인터페이스 정의가 포함될 수 있습니다. 이 오브젝트 모델은 또한 구현 언어, 분배 등의 관점에서 구현 환경에 맞게 수정할 수 있습니다. 경우에 따라 별도 모델의 분석 결과를 고려해야 하는 경우도 있으며 이러한 경우 분석 모델이라고 합니다.

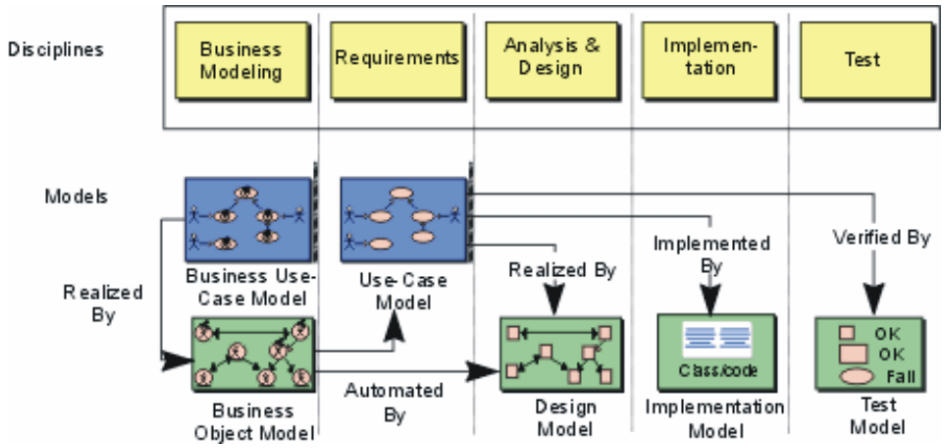

그림 4. 각 원칙은 특정 모델 세트와 연관됩니다.

- 구현 – 규정된 구현 환경에서 시스템을 구현합니다. 이 원칙을 적용하면 소스 코드, 실행 파일 및 파일이 생성됩니다.
- 테스트 – 시스템이 원하는 시스템이며 구현 시 오류가 없는지 확인합니다. 이 원칙을 적용하면 전달할 수 있는 인증 시스템이 생성됩니다.

상호 연결된 복합 시스템 개발

무엇보다 시스템 책임을 여러 시스템에 분배할 수 있는 방법을 정의해야 합니다. 각 시스템은 잘 정의된 책임의 서브세트를 담당해야 합니다. 즉, 이러한 하위 시스템 간의 인터페이스를 정의해야 함을 의미합니다. 이 작업을 수행하면 "분할 정복" 원칙에 따라 각 하위 시스템에 대해 나머지 작업을 개별적으로 수행할 수 있습니다. 따라서 이 작업은 구현 완료 후 테스트와 별도로 시스템 전체에 대해 수행해야 합니다.

분해 기준

시스템을 상호 연결된 복합 시스템으로 분해할지 여부를 결정해야 하며 이를 위해 다음과 같은 몇 가지 사항을 고려해야 합니다.

- 복잡한 대규모 시스템의 경우 문제점을 한 번에 쉽게 이해할 수 있도록 세분화할 수 있습니다.
- 실제 개별 시스템을 처리하는 경우 일반적으로 레거시 시스템 또는 레거시 아키텍처 작업을 수행하는 경우입니다.
- 분해를 통해 시스템 파트 간 꼭 필요한 특정 인터페이스를 정의할 수 있습니다.
- 일부 주요 COTS(Commercial-Off-The-Shelf) 제품을 사용하여 특정 시스템 파트를 구현할 수 있습니다. 이러한 분해를 통해 원하는 COTS 제품 사용 방법을 명확히 할 수 있습니다.

- 파티션을 통해 분산 개발 조직을 최대한 활용할 수 있으며 지리적으로 분산되어 있는 여러 팀 간에 작업을 명확히 분배할 수 있습니다.

이 때 고려해야 할 위험성은 다음과 같습니다.

- 분해 방식을 너무 많이 사용하면 전체적인 문제점과 그 세부사항을 정확히 파악하지 못할 수 있습니다.
- 실제 별도 시스템 또는 실제 별도 팀을 사용함으로써 재사용 기회를 활용하지 못해 융통성이 부족한 구식 시스템으로 전락할 수 있습니다.

조직

위에서 언급한 위험성을 완화하려면 전체 개발 노력을 감독하는 사람으로 구성된 그룹을 지정해야 합니다. 이러한 그룹은 일반적으로 아키텍처 팀이라고 하며 초점을 맞추어야할 핵심 관심사항은 다음과 같습니다.

- 전체 아키텍처가 정의되어 있으며 해당 아키텍처를 하위 시스템에서 준수하는지 여부
- 하위 시스템 간 경험에 대한 재사용 및 공유를 위해 노력하는지 여부
- 생성될 아티팩트와, 하위 시스템 아티팩트 및 상위 시스템 아티팩트 간 관계를 명확하게 이해하고 있는지 여부
- 효과적인 변경 관리 전략이 정의되어 있으며 모든 팀에서 해당 전략을 따르는지 여부

항상 그런 것은 아니지만 일반적으로 아키텍처 팀은 상위 시스템 개발을 소유할 수 있습니다. 조직에 대한 자세한 설명은 [6]을 참조하십시오.

상위 시스템의 라이프사이클

먼저, 시스템 컨텍스트를 보다 잘 이해하기 위해 선택적으로 **비즈니스 모델링**을 수행할 수 있으며 다음과 같은 경우 많은 이점을 얻을 수 있습니다.

- 개발자가 조직을 보다 잘 이해해야 하는 경우
- 조직에서 해당 비즈니스 수행 방법과 용어 및 프로세스 조정 방법이 다른 경우
- 소프트웨어 엔지니어링 노력과 비즈니스 리엔지니어링 노력을 함께 수행하는 경우

[6]도 참조하십시오.

이 노력을 수행하면 비즈니스 유스 케이스 모델 및 비즈니스 분석 모델이 생성됩니다. 또는 비즈니스 도메인의 핵심 개념만을 고려한 제한된 비즈니스 엔지니어링을 수행하고 해당 핵심 개념을 비즈니스 분석 모델에 문서화할 수 있습니다. 이 방법을 일반적으로 도메인 모델링이라고 합니다.

비즈니스 모델 세트가 구축되면 전체 시스템에 대한 **요구사항** 도출을 시작할 수 있습니다. 상호 연결된 복합 시스템에 대한 요구사항 모델링에도 다른 시스템과 동일한 요구가 존재합니다. 유스 케이스 모델은 해당 결과를 효과적으로 나타낼 수 있는 방법입니다([7] 참조). 이러한 상위 유스 케이스 모델을 가장 정확하게 고려할 수 있는 방법은 해당 모델이 시스템의 동작 요구사항을 완벽하게 캡처하는 것으로 가정하는 것입니다. 그러나 이 방법은 일반적인 방법은 아닙니다. 시스템은 다른 시스템에서도 구현해야 하므로 전체 시스템은 일반적으로 매우 복잡합니다. 따라서 이 레벨에서 모든 요구사항을 처리하려는 것은 올바른 방법이 아닙니다. 이에 따라 상위 유스 케이스 모델은 일반적으로 시스템의 기능 요구사항을 완벽하면서도 간단하게 나타냅니다. 즉, 세부 모델링은 각 하위 시스템 구현 작업에서 수행되므로 이 레벨에서 지나치게 세부적으로 다루지 않아도 됩니다. 또한 여러 서브시스템과 관련된 상위 유스 케이스에 많은 요구사항이 표시되지 않을 수도 있습니다. 이러한 요구사항을 서브시스템의 "로컬" 요구사항이라고 합니다.

분석 및 디자인의 목적은 견고한 시스템 아키텍처를 달성하는 것이며 이는 상호 연결된 복합 시스템에 있어 매우 중요한 부분입니다. 상위 시스템의 개발자는 내부 구조에 아무런 영향을 주지 않으면서 견고한 하위 시스템

구조를 달성해야 합니다. 따라서 서브시스템을 사용하여 특정 시스템 분할을 더 작은 파트로 모델링합니다. 또한 올바른 서브시스템 세트와, 상위 시스템의 책임을 해당 서브시스템에 분배하는 방법에 대한 초기 아이디어를 위해 분석 모델을 개발합니다. 분석 클래스는 상위 레벨 유스 케이스를 수행할 때 시스템에서 수행하는 역할을 나타내야 합니다. 따라서 분석 모델은 전체 오브젝트 구조를 상위 레벨 유스 케이스 모델과 유사하게 간단한 그림으로 나타냅니다.

기능상 관련이 있는 분석 클래스는 함께 서브시스템으로 그룹화됩니다. 따라서 기능상의 기준만을 기반으로 한다는 점에서 이상적인 서브시스템 구조를 얻게 됩니다. 예를 들어, 분배 요구사항은 고려하지 않은 경우입니다. 일반적으로 가장 큰 영향을 주는 요소는 레거시 시스템의 존재입니다. 레거시 시스템은 분석 모델에 정의된 일부 또는 많은 책임을 이행합니다. 이러한 시스템이 존재하는 경우, 기존 기능을 최대한 재사용하려면 분석에서 발견된 책임을 다시 파티션해야 하는 경우가 발생할 수 있습니다.

디자인 결과는 분석 시 기능상의 기준을 기반으로 정의한 구조와 현저히 다른 서브시스템 구조일 수 있습니다. 따라서 각각 하위 시스템으로 구현될 디자인 서브시스템 구조가 생성됩니다(그림 5 참조). 그러한 각 시스템에 대해 개별적으로 개발 작업을 계속 수행하기 위해 각 서브시스템별로 인터페이스가 정의됩니다. 인터페이스는 하위 시스템의 개발 규칙을 제공하므로 인페이스 정의는 실제로 상위 레벨에서 수행되는 가장 중요한 활동입니다. 디자인 클래스는 정의되지 않으며 디자인 서브시스템의 인터페이스만 정의하면 됩니다.

시스템의 특정 기술 측면을 파악하기 위한 일부 프로토타입 생성 작업 이외에 상위 시스템 라이프사이클의 파트로 수행되는 **구현은** 없습니다.

최종 원칙은 **테스트**입니다. 여기서 테스트란 다양한 하위 시스템을 어셈블하는 경우 통합 테스트를 의미하며, 모든 상위 유스 케이스가 해당 스펙에 따라 상호 연결된 시스템의 협력으로 수행되는 테스트를 의미합니다.

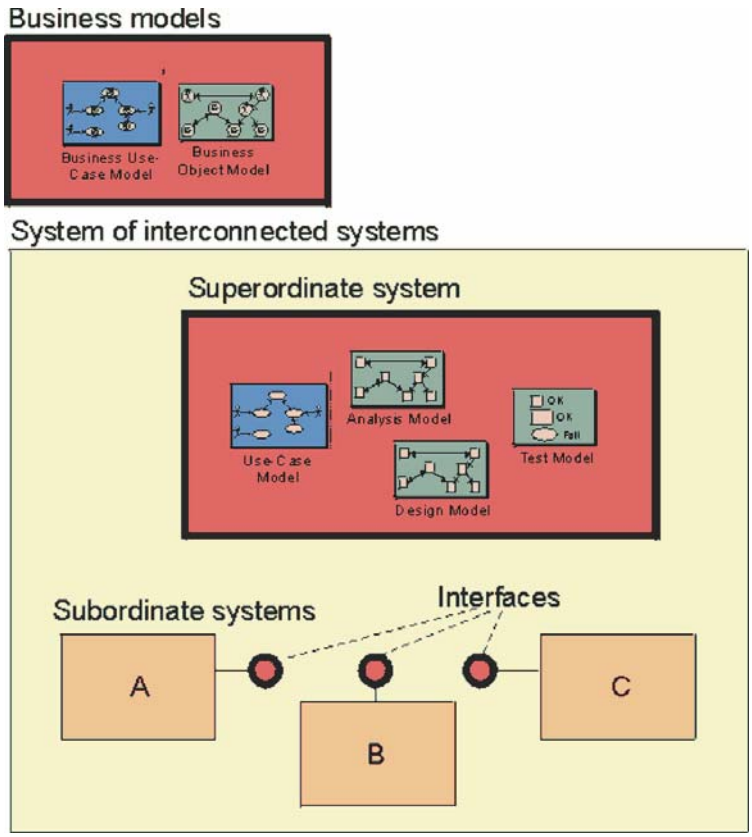

그림 5. 상위 시스템은 상위 레벨 디자인 모델에 정의된 서브시스템이 하위 시스템에 의해 구현되는 모델 세트로 설명합니다. 하위 시스템의 인터페이스는 상위 시스템이 소유합니다.

다음은 상위 시스템 관련 작업 방법을 나타내기 위한 샘플 반복 계획입니다. 첫 번째는 상위 시스템 라이프사이클에서 도입/인식(Inception) 단계의 반복 계획이며 두 번째는 정제(Elaboration) 단계의 반복 계획입니다. 반복 계획을 설명하기 위해 활동 다이어그램을 사용합니다. 이 다이어그램의 조치 상태는 RUP 에 정의된 활동과 일치합니다.

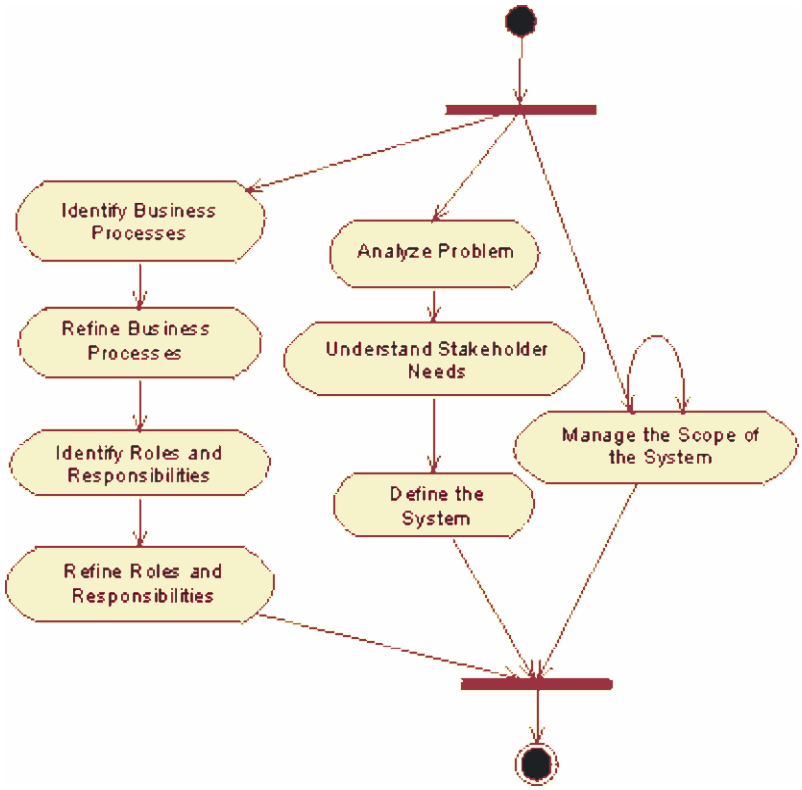

그림 6. 상위 시스템의 도입/인식(Inception) 반복 계획 예제를 설명하는 활동 다이어그램

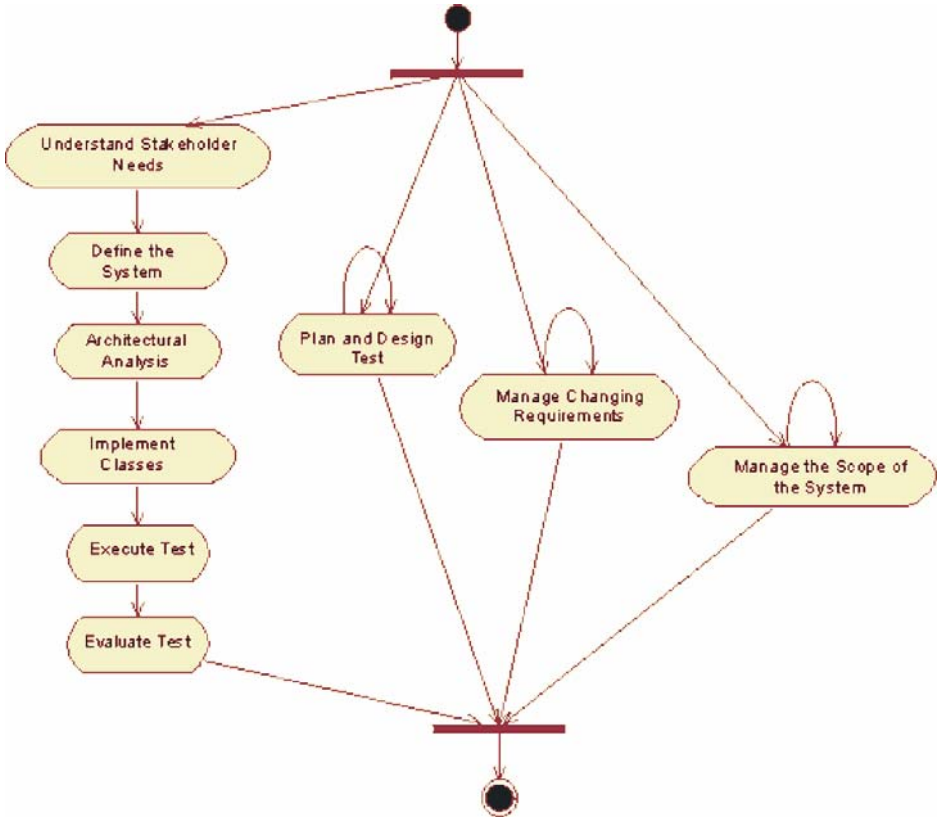

그림 7. 상위 시스템의 정제(Elaboration) 반복 계획 예제를 설명하는 활동 다이어그램. "클래스 구현" 조치 상태는 시스템의 기술적 측면을 파악하기 위해 일부 제한된 프로토타입 구현을 수행할 수 있으므로 여기에 표시됩니다.

하위 시스템의 라이프사이클

각 하위 시스템은 액터로서 통신하는 다른 시스템을 고려하여 블랙 박스로서 일반적인 방식으로 개발됩니다. 위에서 설명한 대로 각 해당 시스템에 대해 일반적인 여러 활동을 수행하고 역시 일반적인 여러 모델을 개발합니다. 상위 레벨의 모델이 모든 세부사항을 반영하여 정의되면 다른 레벨의 모델 간에서 완벽하게 반복될 수 있지만 앞에서 설명한 대로 이러한 경우는 실제로 거의 발생하지 않습니다.

하위 시스템의 경우 **요구사항** 관련 활동을 수행합니다. 하위 시스템의 경계와 해당 액터를 이해하려면 상위 시스템의 인터페이스 및 유스 케이스를 먼저 파악해야 합니다.

하위 시스템에 대한 **분석 및 디자인**을 수행하는 경우 상위 레벨 유스 케이스와 함께 상위 시스템에 정의된 인터페이스가 "경계 조건"이 됩니다.

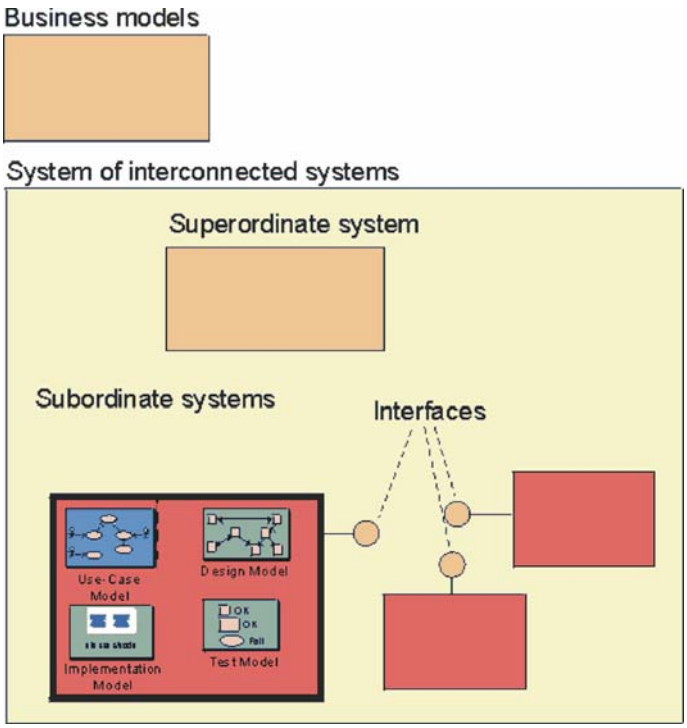

그림 8. 하위 시스템은 고유한 모델 세트로 설명됩니다.

다음은 하위 시스템 관련 작업 방법을 나타내기 위한 해당 라이프사이클의 두 가지 샘플 반복 계획입니다.

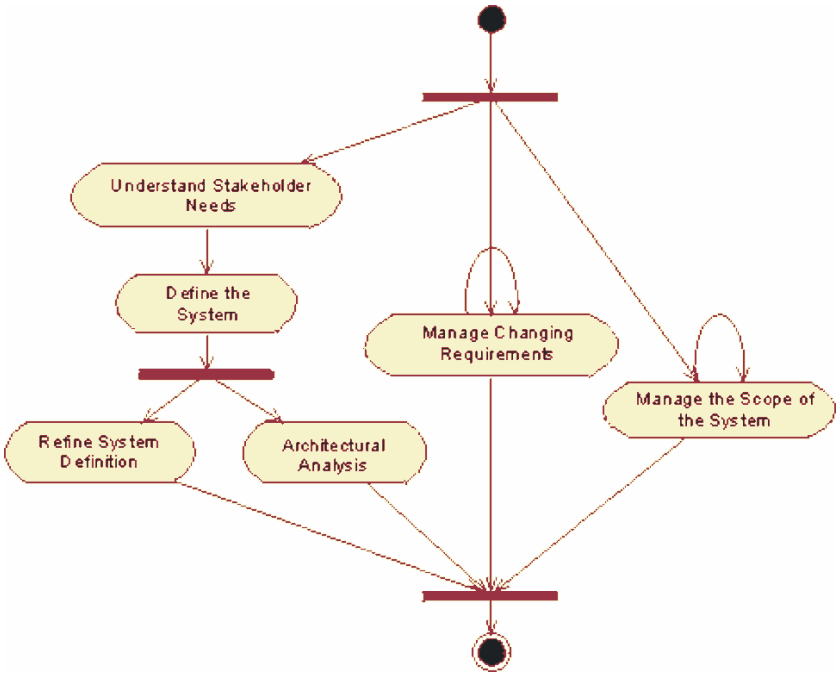

그림 9. 하위(subordinate) 시스템의 샘플 도입/인식(Inception) 반복 계획. 실행 파일이 생성되지 않으므로 불완전한 반복입니다.

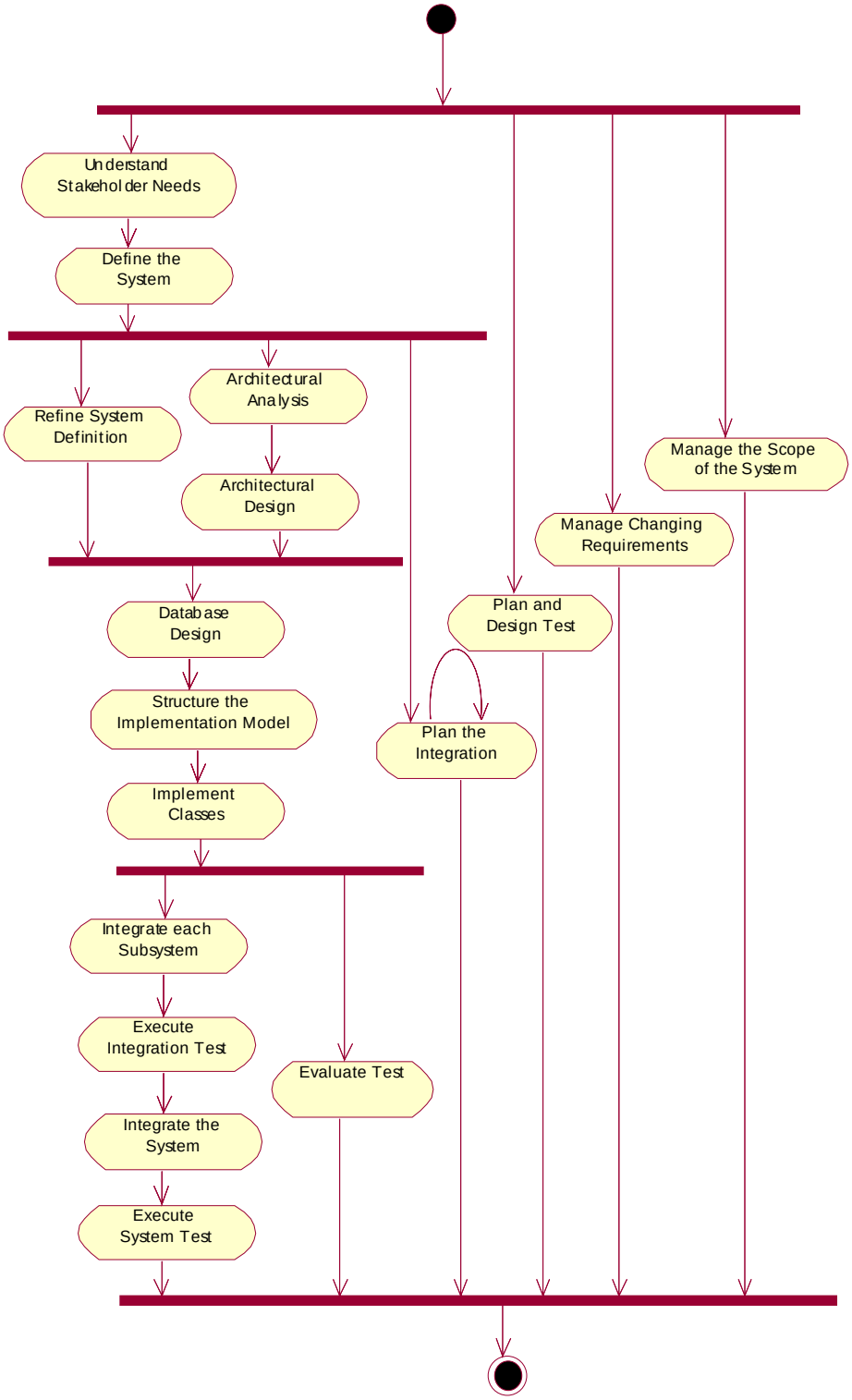

그림 10. 하위 시스템의 샘플 정제(Elaboration) 반복 계획. 정제 단계의 핵심은 정제된 시스템 정의 및 아키텍처의 완료입니다.

상호 연결된 복합 시스템 유스 케이스

상호 연결된 복합 시스템에는 상위 시스템과 하위 시스템 모두 각 시스템마다 하나의 유스 케이스 모델을 빌드해야 합니다. 각 모델은 다음과 같은 방식으로 종속됩니다(그림 11 참조).

- 상위 시스템의 상위 레벨 유스 케이스는 항상 그렇지는 않지만 일반적으로 서브시스템으로 분할됩니다. '분할된' 각 시스템은 해당 하위 시스템 모델에서 유스 케이스가 됩니다(그림 11 참조).
- 특정 하위 시스템의 관점에서 볼 때 나머지 하위 시스템은 해당 유스 케이스 모델의 액터입니다(그림 12 참조).

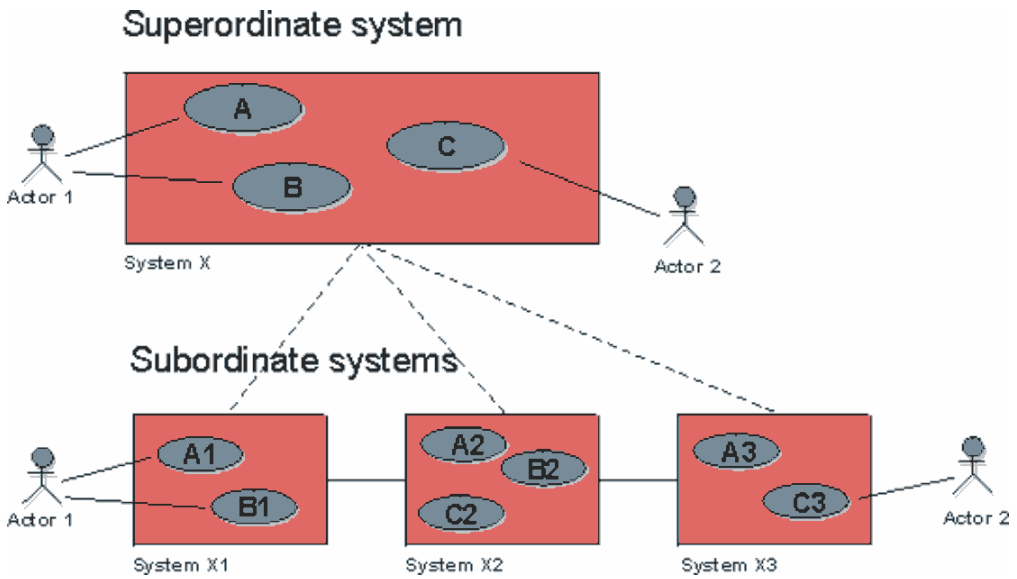

그림 11. 상위 시스템의 상위 유스 케이스와 하위 시스템의 세부 유스 케이스 간 관계

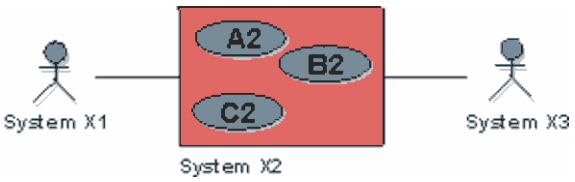

그림 12. 하위 시스템 X2 의 유스 케이스 모델에서 나머지 하위 시스템 X1, X3 은 액터로 표시됩니다.

상위 시스템에 대해 설명하는 유스 케이스에 대해 일부 특수 고려사항이 있습니다. 어떤 의미에서 각 하위 시스템의 모든 요구사항을 다시 설명하게 되므로 이러한 유스 케이스를 지나치게 세분화해서는 안됩니다. 일반적으로 상위 레벨 유스 케이스의 이벤트 플로우에 단계별 아웃라인만 작성하면 되며 텍스트 설명으로 세부적으로 나타낼 필요는 없습니다.

이 유스 케이스 모델에서는 유스 케이스 관계(일반화, 확장, 포함)를 사용해서는 안됩니다. 일반적으로 다음과 같은 이유로 별다른 이점이 없습니다.

- 상위 레벨 유스 케이스를 자세히 설명하지 않으므로 텍스트를 여러 곳에 표시하지 않아도 됩니다.
- 상위 레벨 유스 케이스를 하위 시스템으로 "분할"할 때 정보를 구조화하므로 다른 구조화 메커니즘과 함께 사용하는 경우 혼동될 수 있습니다.

그러나 상호 연결된 복합 시스템에서 재사용가능 컴포넌트를 찾으려는 경우에는 예외입니다. 일반 유스 케이스를 찾기 위해 상위 유스 케이스 모델을 구조화함으로써 재사용가능한 컴포넌트를 효과적으로 찾을 수 있습니다. 이 주제에 대한 자세한 내용은 [6]을 참조하십시오.

상호 연결된 복합 시스템 디자인 모델

상호 연결된 복합 시스템에서 각 시스템에는 상위 시스템 및 하위 시스템 모두 고유한 디자인 모델이 필요합니다. 각 디자인 모델은 다음과 같은 방식으로 연관됩니다.

- 상위 시스템의 디자인 모델에서 서브시스템은 하위 시스템의 경계를 정의합니다.
- 상위 시스템의 서브시스템에 대해 정의하는 오퍼레이션은 하위 시스템의 인터페이스를 정의하는 데 사용됩니다.

상위 시스템의 디자인 모델은 하위 디자인 모델보다 자세히 설명하지 않으며 다음 내용을 생성합니다.

- 서브시스템 – 간단히 설명합니다.
- 유스 케이스 실현(realization) – 서브시스템의 협업 관점. 이러한 상위 레벨 유스 케이스 실현(realization)을 문서화하는 일반적인 방법은 시퀀스 다이어그램을 작성하는 것입니다. 즉, 하위 시스템으로의 상위 레벨 유스 케이스 "분할"을 정의하는 시퀀스 다이어그램을 생성합니다(그림 13 참조).
- 서브시스템 오퍼레이션
- 서브시스템에 대한 인터페이스 정의

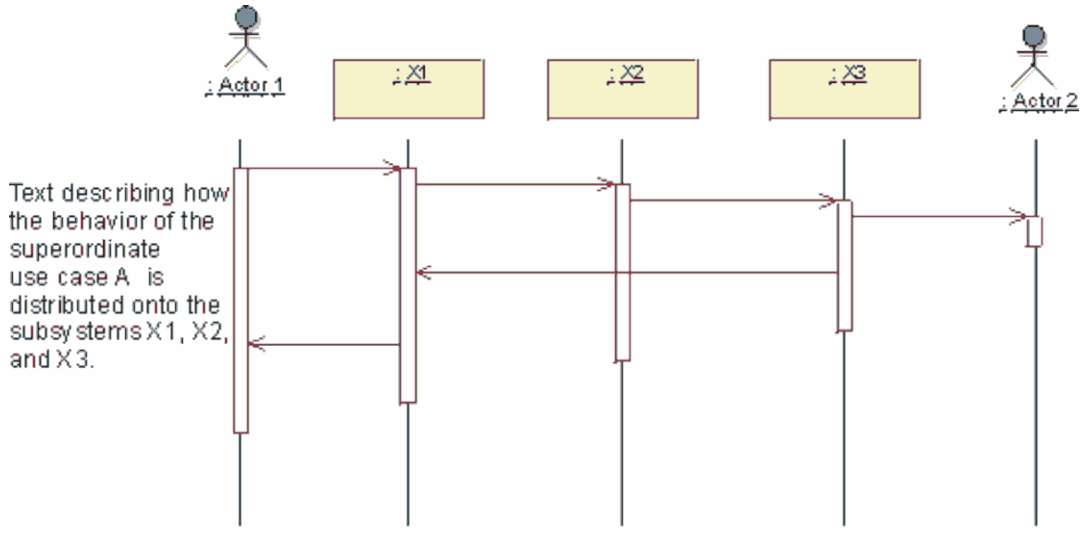

그림 13. 상위 유스 케이스 A 의 실현(realization)에 대한 시퀀스 다이어그램

상호 연결된 복합 시스템 정보 세트

대부분의 조직에서 많은 노력을 기울이는 영역 중 하나는 해당 아티팩트를 관리하는 방법과 해당 종속성을 정확하게 이해하는 것입니다. 이전 섹션에서 상위/하위 유스 케이스 모델과 디자인 모델 간의 종속성에 대해 구체적으로 설명했습니다. 이 밖에 일반 종속성 문제도 고려해야 합니다.

시스템 라이프사이클이 진행됨에 따라 정보 세트 [8]로 구성될 수 있는 아티팩트가 생성됩니다(그림 14 참조). 이러한 정보 세트는 "함께" 연관되는 아티팩트에 따라 구성됩니다.

- 빌드 대상 응용프로그램의 유형에 따라 각 세트의 정확한 컨텐츠를 사용자 정의할 수 있지만 해당 세트는 그대로 유지됩니다.
- 세트 간 종속성을 이해해야 아티팩트 간 추적성을 효과적으로 유지보수할 수 있습니다.

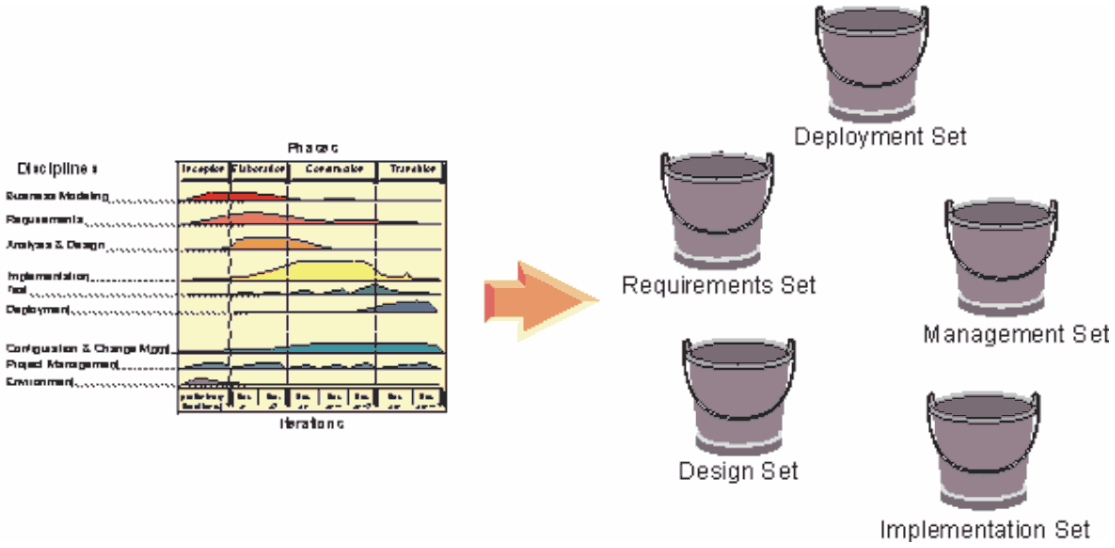

그림 14. 시스템 라이프사이클에서 정보 세트가 생성됩니다.

상호 연결된 복합 시스템에서는 상위 시스템과 각 하위 시스템이 고유한 정보 세트를 생성합니다(그림 15 참조).

- 하위 정보 세트는 해당 상위 정보 세트에 종속됩니다.
- 응용프로그램 유형이 다를 수 있으므로 하위 시스템 간 해당 정보 세트에서 컨텐츠 유형이 다를 수 있습니다.
- 해당 하위 정보 세트는 상위 시스템에 정의된 동일한 서브시스템 인터페이스를 이행한다는 점을 제외하고는 독립적이어야 합니다.

상위 시스템의 아티팩트와 하위 시스템의 아티팩트 간에 추적성을 유지보수하기 위한 노력은 최소화하고 시스템 내 추적성을 유지보수하는 데 우선순위를 두어야 합니다.

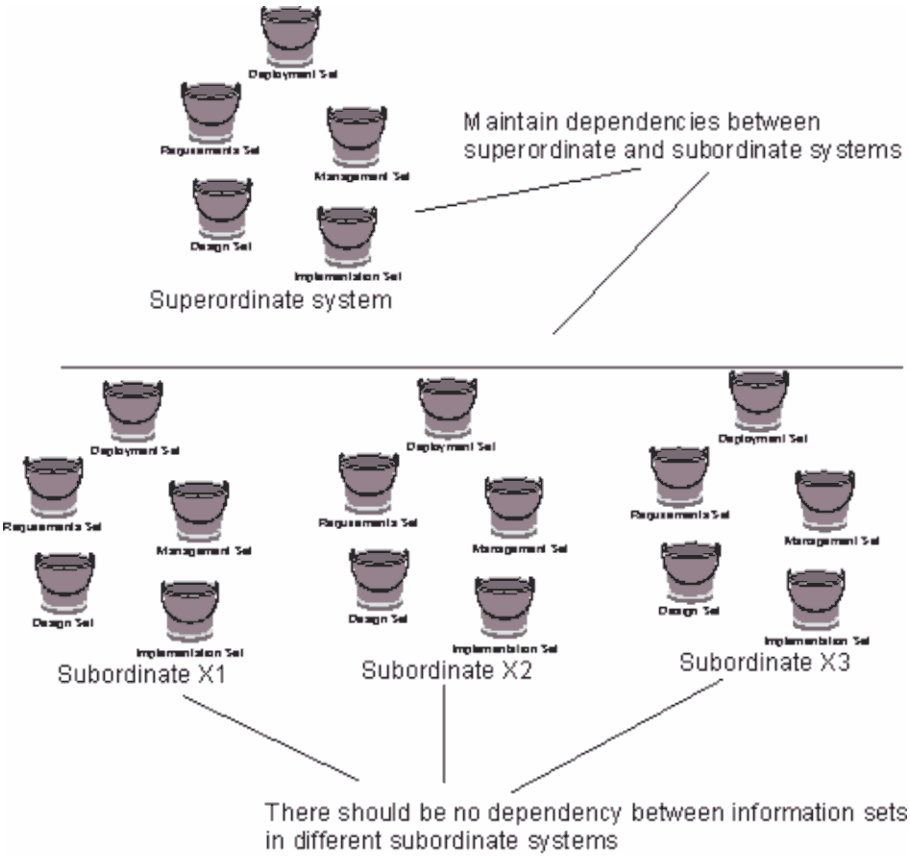

그림 15. 상호 연결된 복합 시스템 내 각 시스템은 고유한 정보 세트를 생성합니다.

상호 연결된 복합 시스템 아키텍처

상호 연결된 복합 시스템에서 각 시스템에는 상위 시스템 및 하위 시스템 모두 해당 아키텍처를 정의해야 합니다.

상위 시스템의 아키텍처 문서에는 다음 내용이 포함되어야 합니다.

- 상위 시스템의 핵심 유스 케이스 또는 시나리오
- 상호 연결된 복합 시스템 계층화
- 하위 시스템 간 재사용 처리 방법 및 재사용 대상
- 모든 하위 시스템에서 사용할 수 있는 일반적인 핵심 메커니즘 및 해당 구현. 예를 들어, 모든 하위 시스템은 통신, 오류 보고 및 결함 관리를 위해 공통 메커니즘을 사용해야 하며 그렇지 않은 경우 상위 시스템이 동종 시스템과 같은 동작을 나타내지 않습니다.

하위 시스템의 아키텍처 문서에는 다음 내용을 명시해야 합니다.

- 상호 연결된 복합 시스템에서 하위 시스템의 역할
- 하위 시스템의 핵심 유스 케이스 또는 시나리오
- 하위 시스템에서 상호 연결된 복합 시스템에 대해 정의된 계층화된 구조를 사용하는 방법. 즉, 상위 시스템의 계층화된 아키텍처에서 정의된 역할을 하위 시스템이 이행하는 방법을 정의해야 합니다.

- 사용될 일반 핵심 메커니즘과 해당 방법 및 추가되는 응용프로그램 특정 핵심 메커니즘
- 재사용 적용 방법. 특히 두 개 이상의 하위 시스템과 공통적으로 연관되는 서브시스템과, 하위 시스템 통신을 위해 빌드되는 메커니즘

시스템 간 관계

일반 시스템 개발 활동을 상호 연결된 복합 시스템이 구현하는 시스템에도 적용할 수 있음을 앞에서 설명했습니다. 이러한 경우 다른 시스템에서 사용한 방법과 유사한 방법으로 시스템을 처리할 수 있다는 점에서 유용합니다. 또한 상위 시스템을 다른 하위 시스템의 형태로 해당 구현과 분리할 수 있습니다. 상호 연결된 복합 시스템에서 각 시스템은 자체 라이프사이클을 갖습니다. 각 시스템마다 특성이 다르므로 이를 생성하려면 다양한 개발 프로세스를 사용해야 합니다. RUP [2] 용어로 표현하면 각 시스템마다 다른 개발 사례를 갖게 됩니다.

상호 연결된 복합 시스템에 포함된 시스템 간 독립성에 대한 마지막 참고 사항은 다음과 같습니다.

먼저 하위 시스템의 경우 각 시스템은 하위 시스템의 디자인 모델에서 하나의 서브시스템을 구현합니다. 서브시스템은 해당 상대 서브시스템이 아닌 해당 인터페이스에 종속됩니다(그림 12 참조). 따라서 새 서브시스템이 동일한 인터페이스를 준수하는 경우 다른 서브시스템에 영향을 주지 않고 특정 서브시스템을 새 버전으로 교환할 수 있습니다. 하위 시스템 간 정확히 동일한 관계가 구축될 수 있습니다. 각 하위 시스템은 주변 영역을 인터페이스 세트로 봅니다. 이는 새 시스템이 다른 시스템에 대해 동일한 역할을 수행하는 경우, 즉 새 시스템을 동일한 인터페이스 세트로 나타낼 수 있는 경우 특정 시스템을 다른 시스템과 교환할 수 있음을 의미합니다. 시스템은 상위 모델에서 서브시스템과 인터페이스 간의 해당 관계에서 지정하는 대로 상대 시스템의 인터페이스를 참조합니다.

하위 시스템의 유스 케이스 모델에서, 해당 하위 시스템이 상호작용하는 다른 하위 시스템의 인터페이스는 액터로 표시됩니다. 하위 시스템은 해당 액터가 제공하는 다른 시스템의 인터페이스를 참조하므로 다른 시스템을 직접 참조하지 않아도 됩니다(그림 16 참조). 그림 12 에서 여러 위치에 표시되는 인터페이스 B 는 상위 시스템의 서브시스템과 해당 하위 시스템이 참조하는 인터페이스와 실제로 동일한 인터페이스임을 나타냅니다.

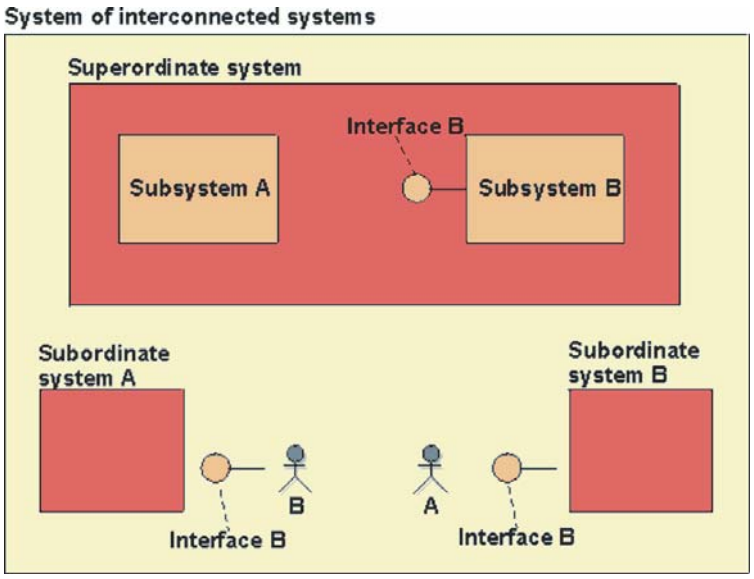

그림 16. 상위 시스템의 서브시스템은 해당 인터페이스를 통해서만 상대 서브시스템에 종속됩니다. 따라서 구현 하위 시스템은 같은 유형의 독립성을 갖습니다. 상위 시스템 모델에서 서브시스템 B 는 다른 서브시스템에 인터페이스 B 를 제공합니다. 따라서 해당 하위 시스템 B 는 다른 하위 시스템에도 동일한 인터페이스 B 를 제공해야 합니다.

상위 시스템과, 해당 하위 시스템에 대한 상위 시스템의 관계도 고려해야 합니다. 이는 다음과 같은 의미에서 해당 구현 시스템과 독립적입니다. 즉, 각 시스템은 상위 시스템 모델에 지정한 내용의 구현일 뿐이며 해당 스펙에는 포함되지 않습니다. 또한 실질적인 이유로, 요구사항을 추적하기 위해 다른 레벨의 시스템 간 추적성 링크를 정의해야 하며, 이를 수행하기 위한 가장 "적절한" 방법은 인터페이스 간에만 해당 링크를 정의하는 것입니다(그림 11 참조). 실제로, 하위 시스템은 상위 모델에 정의된 인터페이스를 제공하는 구현일 뿐이라고 할 수도 있습니다.

그러나 이는 불완전하고 간단한 예제 시스템에만 해당되는 내용입니다. 인터페이스는 특정 상호작용 지점에서 수행되는 내용만 지정합니다. 하나의 하위 시스템에 100 개의 인터페이스가 존재할 수 있으며 각 인터페이스별로 열 개의 오퍼레이션이 포함될 수 있습니다. 인터페이스 설명에는 특정 인터페이스의 입력과 다른 인터페이스의 하나 이상의 출력을 연결할 수 없습니다. 따라서 하위 시스템의 시맨틱을 설명하기 위한 유스케이스가 필요합니다.

상호 연결된 복합 시스템이 시스템을 구현하는 경우 관련된 각 시스템은 상대 시스템에는 종속되지 않지만 해당 인터페이스에는 종속됩니다. 이를 통해 하위 시스템의 병렬 개발을 위한 올바른 플랫폼이 제공됩니다.

응용프로그램 영역

상호 연결된 복합 시스템에 대한 아키텍처 및 모델링 기법은 다음과 같은 다양한 시스템 유형에 사용할 수 있습니다.

- 분산 시스템
- 대규모 또는 복잡한 시스템
- 여러 비즈니스 영역을 결합하는 시스템
- 다른 시스템을 재사용하는 시스템
- 시스템의 분산 개발

반대의 경우도 가능합니다. 즉, 기존 시스템 세트에서 시스템을 어셈블하여 상호 연결된 복합 시스템을 정의할 수 있습니다. 실제로, 이러한 방식으로 대규모 시스템이 초기 발전 단계에서 전개되는 경우도 있습니다. 상호 연결될 수 있는 시스템을 실현함으로써 두 개의 별도 시스템 보다 많은 가치를 추가하는 "대규모 시스템"이 작성됩니다.

다양한 시스템 파트를 자체 시스템으로 볼 수 있는 시스템의 경우 상호 연결된 복합 시스템으로 정의하는 것이 좋습니다. 현재 단일 시스템이라도 예를 들어 분산 개발 또는 재사용 이유로 인해 또는 고객이 특정 파트만 구입해야 하는 경우 나중에 해당 시스템을 여러 개의 개별 제품으로 분할해야 하는 경우도 있습니다.

마지막으로, 상호 연결된 복합 시스템 아키텍처를 사용할 수 있는 여러 경우에 대해 살펴봅니다. 각 예제별로, 해당 시스템을 단일 시스템 *및* 개별 시스템 세트로 고려해야 함을 나타냅니다. 이 시스템은 상호 연결된 복합 시스템이 구현하는 상위 시스템으로 간주해야 합니다.

대규모 시스템

전화 네트워크는 세계에서 가장 큰 상호 연결된 복합 시스템이라고 할 수 있습니다. 따라서 복잡도를 관리하기 위해 두 가지 이상의 시스템 레벨이 필요한 적합한 예제입니다. 이는 또한 최상위 레벨 상위 시스템을 표준화 주체가 소유하고 다른 경쟁 회사에서 이 표준을 준수해야 하는 하나 이상의 하위 시스템을 개발하는 경우의 예제입니다. 여기서는 대규모 시스템을 상호 연결된 복합 시스템으로 구현하는 데 따른 이점을 보여주기 위해 이동 전화 네트워크 GSM(Global System of Mobile Telephony)에 대해 설명합니다.

대규모 시스템의 기능성은 일반적으로 여러 비즈니스 영역이 결합된 결과입니다. 예를 들어, GSM 표준은 호출 등록자로부터 호출을 받는 등록자에 이르기까지 전체 시스템에 적용됩니다. 즉, 이동 전화 동작과 네트워크 노드 동작이 모두 포함됩니다. 각 시스템 파트는 개별 고객이 개별적으로 구입하는 독립 제품이므로 독립 시스템으로 간주되어야 합니다. 예를 들어, 전체 GSM 시스템을 개발하는 회사의 경우 등록자에게는 이동 전화를 판매하고 전화 교환수에게는 네트워크 노드를 판매합니다. 이는 GSM 시스템의 각 파트를 다른 하위 시스템으로 간주해야

하는 이유 중 하나입니다. GSM 과 같이 복잡한 대규모 시스템을 하나의 단일 시스템으로 개발하는 데 많은 시간이 소요되는 것 또한 한 가지 이유입니다. 이러한 경우 각 파트를 여러 개발 팀에서 병렬로 개발해야 합니다.

또한 GSM 표준은 전체 시스템에 적용되므로 해당 시스템을 전체로서, 즉 상위 시스템으로 간주해야 합니다. 이를 통해 개발자는 문제점 도메인과, 서로 다른 파트가 서로 연결되는 관계를 이해할 수 있습니다.

분산 시스템

여러 컴퓨터 시스템에 분산되는 시스템의 경우, 상호 연결된 복합 시스템 아키텍처가 매우 적합합니다. 분산 시스템은 정의대로 항상 두 개 이상의 파트로 구성됩니다. 분산 시스템에는 인터페이스가 잘 정의되어야 하므로, 이러한 시스템은 병렬로 작업을 수행하는 여러 자율 개발 팀에서 분산 방식으로 *개발하는* 데 매우 적절합니다. 또한 분산 시스템의 하위 시스템은 독립 제품으로 판매될 수도 있으므로 분산 시스템을 한 세트의 독립 시스템으로 간주할 수 있습니다.

분산 시스템의 요구사항에는 일반적으로 전체 시스템의 기능성이 포함되므로 경우에 따라 다른 파트 간의 인터페이스가 사전 정의되지 않습니다. 또한 문제점 도메인이 개발자에게 친숙하지 않은 경우 분산 방식에 관계없이 먼저 전체 시스템의 기능성을 고려해야 합니다. 이러한 사항이 단일 시스템으로 보아야 하는 두 가지 중요한 이유입니다.

레거시 시스템의 재사용

일반적으로 대규모 시스템은 레거시 시스템을 재사용합니다. 레거시 시스템은 하위 시스템으로 설명할 수 있습니다. 또한 상위 시스템의 대규모 컨텍스트에서의 작동 방법을 이해하기 위해 레거시 시스템의 유스 케이스 모델과 분석 모델을 리엔지니어링합니다. 이렇게 리엔지니어링된 모델은 완전한 모델은 아니더라도 최소한 상호 연결된 시스템의 나머지 시스템 기능성에 직접적인 영향을 주거나 수정이 필요한 레거시 시스템 기능성을 포함해야 합니다.

사전 조정된 패키지 사용

시스템은 두 개 이상의 사전 조정된 패키지의 통합 또는 사용자 정의 시스템일 수 있습니다. 대표적인 예로 엔터프라이즈 자원 조달 계획(ERP) 시스템을 들 수 있습니다. 일반적으로 ERP 시스템은 원재료 조달 계획(MRP), 재고 관리, 공급 체인 관리 등과 같은 하위 시스템의 컴포지션입니다. 인적 자원 또는 급여 관리 응용프그램과 같은 다른 영역에도 유사한 컴포지션을 사용할 수 있습니다. 이러한 시스템은 완전한 시스템을 얻기 위해 다른 표준 패키지를 전문화하고 상호 연결되어야 하는 사전 조정된 시스템과 같습니다. 패키지 세트가 협력하여 수행하는 기능을 이해하려면 상위 시스템이 필요하며 이는 오늘날 재무 분야의 많은 고객이 직면해 있는 상황입니다.

요약

이 백서에서는 상호 연결된 복합 시스템에 대한 아키텍처 패턴을 소개합니다. 이 구조는 단일 모델에서의 순환만 허용하지는 않습니다. 각 서브시스템을 독립 시스템으로 간주하며 각 시스템의 모든 아티팩트 세트 간에 순환이 허용됩니다. 소개된 아키텍처는 여러 통신 시스템이 구현하는 시스템에 사용됩니다. 각 관련 시스템은 다른 시스템의 모델과 별도로 고유한 모델 세트로 설명됩니다.

이 기법을 사용하는 데 따른 이점은 복잡한 문제점에 접근할 수 있고 "분할 정복" 기법을 사용하여 해당 문제점을 이해할 수 있다는 것입니다. 그러나 더 많은 오버헤드 위험성이 있고 스케줄이 일치하지 않는 단점이 있습니다. 또한 앞에서 조직이 상위 시스템에 반복 라이프사이클을 채택하기 어려워 상위 시스템 라이프사이클 종료 시점에 위험성이 증가하는 예를 살펴보았습니다. 또한 효과적이고 합당한 재사용 전략을 수행함으로써 "구식" 시스템 세트를 개발하지 않도록 유의해야 합니다.

이 문서에 제공된 예제는 상호 연결된 복합 시스템에 대한 모델링 아키텍처가 여러 다른 응용프로그램 영역에도 유용함을 보여줍니다. 실제로, 각 시스템 파트를 독립 시스템으로 간주할 수 있는 모든 시스템에 이 아키텍처를 사용할 수 있습니다.

참조

[1] Jacobson, I.; Palmkvist, K.; Dyrhage, S.; *Systems of Interconnected Systems*, ROAD, 2(1), 1995.

[2] *Rational Unified Process* 버전 5.1

[3] Rumbaugh, J.; Booch, G.; Jacobson,I., *UML Reference Manual*, Addison Wesley Longman, 1999.

[4] Herbert A. Simon, *The Sciences of the Artificial*, MIT Press, 1981.

[5] Jacobson, I.; Bylund, S.; Jonsson, P., *Using Contracts and Use Cases to Build Plugable Architectures*, Journal of Object-Oriented Programming, 1995 년 5, 6 월호

[6] Jacobson, J.; Griss, M.; Jonsson, P., *Software Reuse - Architecture, Process and Organization for Business Success*, Addison Wesley Longman, 1997.

[7] Jacobson, I., *Use Cases in Large-Scale Systems*, ROAD, 1(6), 1995.

Rational®
the software development company

본사 안내:

Rational Software
18880 Homestead Road
Cupertino, CA 95014
전화번호: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
전화번호: (781) 676-2400

수신자 부담 전화번호: (800) 728-1212
전자 우편: info@rational.com
웹: www.rational.com
전세계 지사 안내: www.rational.com/worldwide

Rational, Rational 로고 및 Rational Unified Process 는 미국 또는 기타 국가에서 사용되는 Rational Software Corporation 의 등록상표입니다. Microsoft, Microsoft Windows, Microsoft Visual Studio, Microsoft Word, Microsoft Project, Visual C++ 및 Visual Basic 은 Microsoft Corporation 의 상표 또는 등록상표입니다. 기타 다른 이름들은 식별용으로만 사용되며 해당 회사의 상표 또는 등록상표입니다. ALL RIGHTS RESERVED. Made in the U.S.A.

© Copyright 2002 Rational Software Corporation.
본 내용은 통지 없이 변경될 수 있습니다.