Machine Translated by Google

제7장

건축적 특징의 범위

소프트웨어 아키텍트의 사고방식은 생태계와 함께 발전해야 하며, 특히 아키텍처 특성을 정의하는 데 있어 이러한 변화가 두드러집니다. 기존의 아키텍처 특성 결정 프레임워크들은 시스템 전체에 하나의 아키텍처 특성만 적용한다는 치명적인 결함을 가지고 있었습니다. 물론 이러한 가정이 여전히 유효한 경우도 있지만, 마이크로서비스와 같은 많은 최신 아키텍처는 서비스 수준과 시스템 수준에서 서로 다른 아키텍처 특성을 포함하고 있습니다.

아키텍처 특성의 범위는 건축가에게 유용한 척도이며, 특히 구현의 출발점으로 사용할 가장 적절한 아키텍처 스타일을 결정하는 데 중요합니다. 저희가 저서 『진화적 아키텍처 구축(Building Evolutionary Architectures)』을 집필할 당시, 특정 아키텍처 스타일의 구조적 진화 가능성을 측정할 기법이 필요했습니다. 기존의 어떤 측정 방법도 필요한 수준의 세부 정보를 제공하지 못했습니다. 83페이지의 "구조적 측정" 섹션 에서는 건축가가 아키텍처의 구조적 측면을 분석할 수 있도록 하는 다양한 코드 수준 메트릭을 다루지만, 이러한 메트릭은 범위를 반영하지 못합니다. 코드에 대한 저수준 세부 정보는 보여주지만, 코드베이스 외부에 있는 종속 구성 요소(예: 데이터베이스)는 평가할 수 없습니다. 이러한 구성 요소는 특히 운영상의 특성을 비롯한 여러 아키텍처 특성에 영향을 미칩니다. 건축가가 성능과 탄력성을 고려하여 코드베이스를 설계하는 데 아무리 많은 노력을 기울이더라도 시스템의 데이터베이스가 이러한 특성에 부합하지 않으면 그 노력은 성공적이지 못할 것입니다.

적절한 범위 측정 기준을 찾지 못해 자체적으로 개발했습니다. 이를 '양자 아키텍처'라고 부릅니다.

Machine Translated by Google

건축 양자와 세분성

구성 요소 수준의 결합만이 소프트웨어를 묶는 유일한 요소는 아닙니다. 많은 비즈니스 개념들이 시스템의 각 부분을 의미론적으로 연결하여 기능적 응집력을 만들어냅니다. 소프트웨어를 성공적으로 설계, 분석 및 발전시키려면 아키텍트와 개발자는 문제가 발생할 수 있는 모든 결합 지점을 고려해야 합니다.

라틴어에서 유래한 전문 용어의 복수형

양자(quantum)는 라틴어에서 유래했으며, 따라서 복수형은 'a'로 끝납니다. 양자가 두 개 이상일 경우 'quanta'가 되는데, 이는 라틴어에서 유래한 'data'와 같은 의미입니다. 건축가들은 'datum'이라는 단어를 하나만 언급하는 경우는 드뭅니다. 시카고 스타일 매뉴얼(Chicago Manual of Style)에서는 "원래 이 단어는 'datum'의 복수형이었지만, 현재는 일반적으로 불가산 명사로 취급되어 단수 동사와 함께 사용된다"고 명시하고 있습니다(제18판, 2024년, 336쪽).

과학적 소양을 갖춘 많은 건축가들은 물리학에서 양자(quantum)라는 개념을 알고 있습니다. 물리학에서 양자는 어떤 것의 최소량, 일반적으로 에너지의 양을 의미합니다. 양자라는 단어는 라틴어에서 유래했으며, "얼마나 큰가" 또는 "얼마나 많은가"라는 뜻입니다. 일반적으로는 "작고 깨지지 않는 것"을 의미하며, 이것이 바로 우리가 아키텍처 양자를 정의하는 방식입니다. 아키텍처 양자에 대한 또 다른 비공식적인 정의는 "시스템에서 독립적으로 실행되는 가장 작은 부분"입니다. 예를 들어, 마이크로서비스는 종종 아키텍처 양자를 구성하며, 이는 위의 정의를 잘 보여줍니다. 서비스는 자체 데이터 및 기타 종속성을 포함하여 아키텍처 내에서 독립적으로 실행될 수 있습니다.

아키텍처 양자는 일련의 아키텍처적 특징에 대한 범위를 설정합니다.

이 제품의 특징은 다음과 같습니다:

- 아키텍처의 다른 부분과 독립적인 배포 • 높은 기능적 응집도 • 낮은 외부
  구현 정적 결합도 • 다른 양자
  (quanta)와의 동기식 통신

이 정의는 여러 부분으로 구성되어 있습니다. 하나씩 살펴보겠습니다.

일련의 건축적 특징에 대한 범위를 설정합니다.

건축가들은 건축 양자(architecture quantum)를 건축적 특성, 특히 운영적 특성을 규정하는 경계로 사용합니다. 건축 양자는 독립적으로 배포 가능하고 높은 기능적 응집력(다음에서 논의)을 가지므로 건축 모듈성을 측정하는 유용한 척도가 됩니다.

Machine Translated by Google

독립적으로 배포 가능한 아

키텍처 퀀텀은 아키텍처의 다른 부분과 독립적으로 작동하는 데 필요한 모든 구성 요소를 포함합니다. 예를 들어, 애플리케이션이 데이터베이스를 사용하는 경우 해당 데이터베이스는 퀀텀의 일부입니다. 데이터베이스 없이는 시스템이 작동하지 않기 때문입니다. 이러한 요구 사항으로 인해 단일 데이터베이스를 사용하여 배포된 거의 모든 레거시 시스템은 정의상 '하나의 퀀텀'을 구성합니다.

하지만 마이크로서비스 아키텍처 스타일에서는 각 서비스가 자체 데이터베이스를 포함합니다 (18 장 에서 자세히 설명하는 경계 컨텍스트 기반 철학의 일부). 이로 인해 각 서비스가 고유한 아키텍처 특성 범위를 가지게 되므로 아키텍처 내에 여러 양자가 생성됩니다.

높은 기능적 응집도 컴포넌

트 설계에서 응집도는 포함된 코드가 목적에 있어 얼마나 통일되어 있는지를 나타냅니다. 예를 들어, 모든 속성과 메서드가 고객 엔티티와 관련된 고객 컴포넌트는 높은 응집도를 보입니다. 반면, 무작위로 다양한 메서드를 모아놓은 유틸리티 컴포넌트는 그렇지 않습니다. 높은 기능적 응집도는 아키텍처 구성 요소가 특정한 목적을 수행한다는 것을 의미합니다. 이러한 구분은 단일 데이터베이스를 사용하는 기존의 모놀리식 애플리케이션에서는 큰 의미가 없습니다. 이러한 애플리케이션에서는 응집도가 사실상 시스템 전체에 해당하기 때문입니다. 그러나 이벤트 기반 아키텍처나 마이크로서비스 아키텍처와 같은 분산 아키텍처에서는 아키텍트가 각 서비스를 단일 워크플로(도메인 주도 설계의 경계 컨텍스트에서 설명하는 경계 컨텍스트)에 맞춰 설계하는 경우가 많으므로, 해당 서비스는 높은 기능적 응집도를 나타냅니다.

도메인 주도 설계(DDD)의 경계 컨텍스트는 에릭 에반스의 저서 『도메인 주도 설계(DDD)』(Addison-Wesley Professional, 2003)에 담겨 있으며, 이 책은 현대 건축 사고에 지대한 영향을 미쳤습니다. DDD는 건축가가 복잡한 문제 영역을 체계적으로 분해할 수 있도록 하는 모델링 기법입니다. DDD는 경계 컨텍스트를 정의하는데, 이 경계 컨텍스트에서는 영역의 일부와 관련된 모든 것이 내부적으로는 보이지만 다른 경계 컨텍스트에서는 불투명하게 처리됩니다.

DDD(도메인 주도 설계) 이전에는 아키텍트들이 조직 내 공통 엔티티 전반에 걸쳐 코드를 전체적으로 재사용하려고 했습니다. 그러나 공통으로 공유되는 아티팩트를 생성하면 결합도 증가, 조정 어려움, 복잡성 증가 등 여러 문제가 발생한다는 것을 알게 되었습니다. 경계 컨텍스트(BC) 개념은 각 엔티티가 지역화된 컨텍스트 내에서 가장 잘 작동한다는 점을 인식합니다. 따라서 조직 전체에 걸쳐 통일된 Customer 클래스를 만드는 대신, 각 문제 영역은 자체적인 Customer 클래스를 만들고 다른 영역과의 통신 지점에서 차이점을 조정할 수 있습니다.

Machine Translated by Google

정의의 다음 부분을 명확히 하기 위해 결합 유형에 대해 몇 가지 세부적인 구분을 해야 합니다.

의미적 결합이란 아키
텍트가 솔루션을 구축하는 문제의 본질적인 결합을 설명합니다. 예를 들어, 주문 처리 애플리케이션의 내재적 결합에는 재고, 카탈로그, 장바구니, 고객, 판매 등이 포함됩니다. 소프트웨어 솔루션 구축을 촉발하는 문제의 본질이 이러한 결합을 정의합니다. 아키텍트는 도메인 변경이 시스템 전체에 파급되는 것을 막을 수 있는 기술이 거의 없습니다. 도메인(따라서 의미론)의 변경은 시스템 요구 사항의 변경을 의미하기 때문입니다. 아키텍트는 이러한 변화에 적응할 수 있지만, 핵심 문제의 변경이 아키텍처에 영향을 미치는 것을 막을 수 있는 마법 같은 아키텍처 패턴은 없습니다.

구현 결합도(Implementation
coupling)는 아키텍트와 팀이 특정 종속성을 구현하는 방식을 결정하는 과정을 설명합니다. 주문 처리 애플리케이션에서 팀은 도메인 경계를 설정할 때 다양한 제약 조건을 고려해야 합니다. 예를 들어, 모든 데이터를 단일 데이터베이스에 저장해야 할까요, 아니면 확장성이나 가용성을 높이기 위해 일부 데이터를 분산시켜야 할까요? 모놀리식 아키텍처를 구축해야 할까요, 아니면 분산 아키텍처를 구축해야 할까요? 이러한 질문에 대한 답은 시스템의 의미적 결합도에는 큰 영향을 미치지 않지만, 아키텍처 결정에는 상당한 영향을 미칩니다.

정적 결합이란 아
키텍처의 "연결" 방식, 즉 서비스들이 서로 어떻게 의존하는지를 나타냅니다. 두 서비스가 동일한 결합 지점에 의존하는 경우, 두 서비스는 같은 아키텍처 퀀텀에 속합니다. 예를 들어, 카탈로그 서비스 와 배송 서비스 라는 두 마이크로서비스가 주소 정보를 공유해야 한다고 가정해 보겠습니다. 그러면 두 서비스는 공통 구성 요소에 대한 의존성을 생성합니다. 두 서비스가 모두 해당 의존성에 결합되어 있으므로, 두 서비스는 같은 아키텍처 퀀텀에 속합니다.

소프트웨어 아키텍처에서 결합도를 쉽게 이해하는 방법은 다음과 같습니다. 하나를 변경하면 다른 하나가 제대로 작동하지 않을 수 있다면 두 가지는 결합되어 있다고 할 수 있습니다. 정적 결합도는 아키텍처에서 범위 의존성을 정의합니다. 예를 들어, 여러 서비스가 동일한 관계형 데이터베이스를 사용하는 경우, 이들은 동일한 퀀텀에 속합니다.

Machine Translated by Google

동적 결합이란 아키텍
처 구성 요소들이 서로 통신해야 할 때 발생하는 힘들을 설명합니다. 예를 들어, 두 서비스가 실행 중일 때, 워크플로우를 형성하고 시스템 내에서 작업을 수행하기 위해 서로 통신해야 합니다.

분산 아키텍처에서 서비스 간 통신을 할 때 설계자는 절충점을 고려해야 하며, 이는 15장에서 자세히 다룹니다.

이러한 정의들을 바탕으로, 우리는 아키텍처 양자 정의를 완성할 수 있습니다.

낮은 외부 구현 정적 결합도. 아키텍처 구성 요소 간
의 구현 결합도는 낮아야 합니다.
이러한 특징은 경계 컨텍스트 간의 낮은 결합도를 중시하는 DDD(Domain-Driven Design) 철학에서 비롯됩니다. 퀀타는 컴포넌트보다 한 단계 높은 추상화 수준에 위치하는 아키텍처의 운영 구성 요소 역할을 합니다. 퀀타는 서비스 경계와 겹치는 경우가 많습니다. 이러한 목표는 아키텍처의 여러 부분 간의 느슨한 결합을 선호하는 아키텍트의 일반적인 성향을 반영합니다.

일반적으로 서비스나 서브시스템 내부처럼 높은 응집력이 요구되는 경우에는 긴밀한 결합이 바람직합니다. 단 하나의 구현 변경으로 인해 예상치 못한 연쇄적인 부작용이 발생하여 겉보기에는 관련이 없어 보이는 여러 요소에까지 영향을 미칠 수 있는 아키텍처를 "취약하다"고 합니다. 시스템의 결합 범위가 넓을수록 느슨한 결합은 아키텍처의 취약성을 줄이는 데 도움이 됩니다. 예를 들어, 아키텍트가 서비스 호출에서 필드 이름을 State 에서 StateCode 로 변경하면서 특정 호출자에게만 영향을 미칠 것이라고 생각했지만, 예상치 못하게 다른 여러 종속성이 깨져 버린 경우를 생각해 볼 수 있습니다.

범위가 좁을수록 더 높은 커플링이 허용되고, 범위가 넓을수록 커플링은 더 느슨해야 합니다.

동기식 통신

통신은 동적 결합, 즉 아키텍처 양자가 서로를 호출하는 것을 의미하며 이는 분산 아키텍처에서 흔히 발생합니다. 이는 특히 59페이지의 "운영 아키텍처 특성" 에서 설명하는 운영 아키텍처 특성군과 관련이 있는데 , 이러한 특성 은 분산 아키텍처에서 중요한 타이밍 및 블로킹을 결정하는 경우가 많기 때문입니다.

Machine Translated by Google

예를 들어, 결제 서비스와 경매 서비스 로 구성된 마이크로서비스 아키텍처를 생각해 보겠습니다 . 경매가 종료되면 경매 서비스는 결제 정보를 결제 서비스로 동기적으로 전송합니다. 하지만 결제 서비스는 500ms마다 한 건의 결제만 처리할 수 있다고 가정해 봅시다 . 이때 많은 경매가 동시에 종료되면 어떻게 될까요? 두 서비스는 운영 아키텍처 측면에서 서로 다른 특성을 가지고 있습니다. 첫 번째 호출은 정상적으로 처리되겠지만, 이후에는 결제 서비스 가 요청량을 감당하지 못해 실패하기 시작할 것입니다. 결제 서비스는 경매 서비스 만큼 확장성이 뛰어나지 않기 때문입니다 .

여기서 동기 통신을 언급하는 이유는 비동기 통신이 잠재적 영향이 더 적기 때문입니다. 예를 들어, 경매 서비스가 메시지 큐를 사용하여 결제 서비스를 비동기적으로 호출하는 경우, 큐는 두 시스템이 작동할 수 있도록 버퍼 역할을 할 수 있습니다. 물론, 경매 서비스가 결제 서비스가 처리할 수 있는 양보다 더 많은 메시지를 지속적으로 보내면 결국 메시지 큐가 오버플로우될 것입니다. 하지만 메시지가 한꺼번에 쏟아지는 경우에는 수신자가 준비될 때까지 큐가 대기 중인 메시지를 보관할 수 있습니다. 동기 통신은 분산 아키텍처, 특히 아키텍처의 각 부분이 서로 다른 아키텍처 특성을 가질 경우 매우 취약합니다. 15장에서는 이벤트 기반 아키텍처에서의 다양한 통신 유형을 자세히 다룹니다.

양자 아키텍처라는 개념은 범위에 대해 새롭게 생각하는 방식을 제시합니다. 현대 시스템에서 아키텍트는 시스템 수준이 아닌 양자 수준에서 아키텍처 특성을 정의합니다. 이는 새로운 문제 영역을 분석할 때 중요한 정보를 제공합니다.

범위 설정의 영향

건축가는 건축적 특성의 범위를 활용하여 그림 7-1 의 의사결정 트리에 전반적으로 나타나 있고 단계별로 더 자세히 설명 된 적절한 서비스 경계를 결정하는 데 도움을 받을 수 있습니다 .

Machine Translated by Google

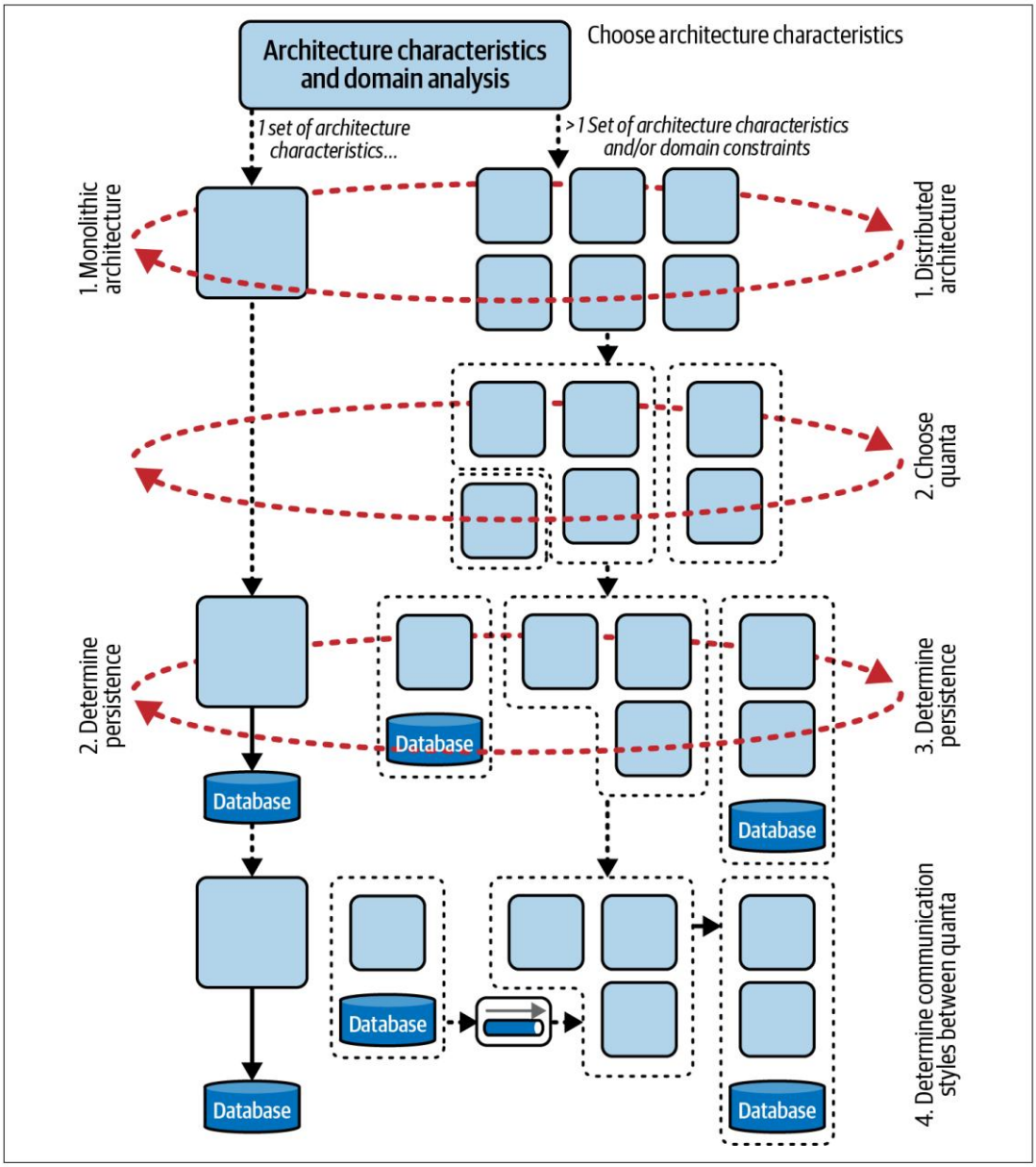

그림 7-1. 건축적 특성의 범위를 활용하여 건축 스타일을 결정하는 데 도움을 주는 의사결정 트리

Machine Translated by Google

범위 설정 및 아키텍처 스타일 문제 영역의 양자적

경계를 결정하는 것은 아키텍처 스타일을 선택하는 데 도움이 됩니다. 단일체 아키텍처가 가장 적합할까요, 아니면 분산 아키텍처가 가장 적합할까요? 그림 7-2에 표시된 의사 결정 차트의 첫 번째 부분을 고려해 보세요.

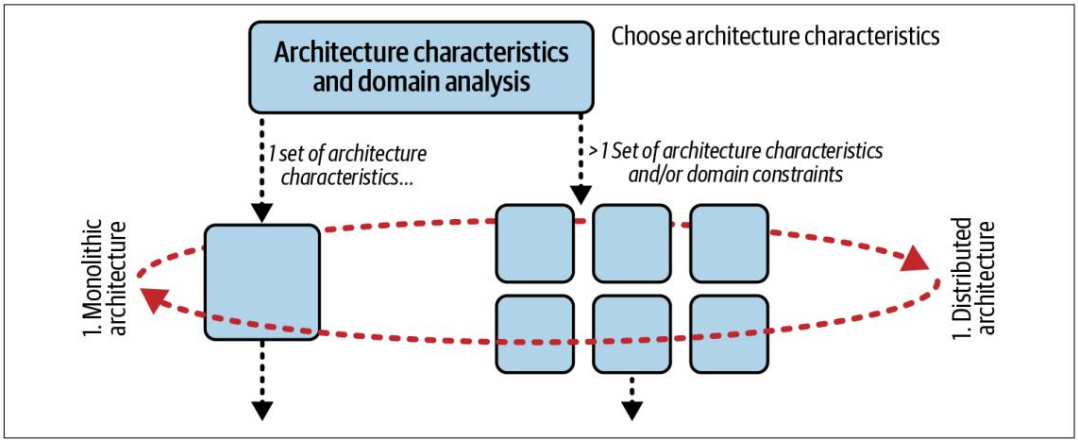

그림 7-2. 건축적 특징에 따른 적절한 건축 양식 선택

4 장 에서 논의했듯이 , 건축가는 가장 적절한 아키텍처 스타일을 결정하기 위해 아키텍처 특성과 해당 영역을 분석해야 합니다. 이러한 분석에는 솔루션에 여러 아키텍처 특성 그룹이 필요한지 여부도 포함됩니다. 그림 7-2 의 1단계에서 건축가는 시스템이 단일 아키텍처 특성 세트로 성공할 수 있는지, 아니면 둘 이상의 그룹이 필요한지 판단합니다(예시는 103페이지의 "Kata: Going Green" 참조).

건축가가 단일 아키텍처 특성 세트로 충분하다고 판단하면 모놀리식 아키텍처를 선택하여 후속 선택 사항의 수를 줄일 수 있습니다. 분산 아키텍처를 선택하는 경우 다음 단계는 그림 7-3에 표시된 것처럼 양자 경계를 결정하는 것입니다.

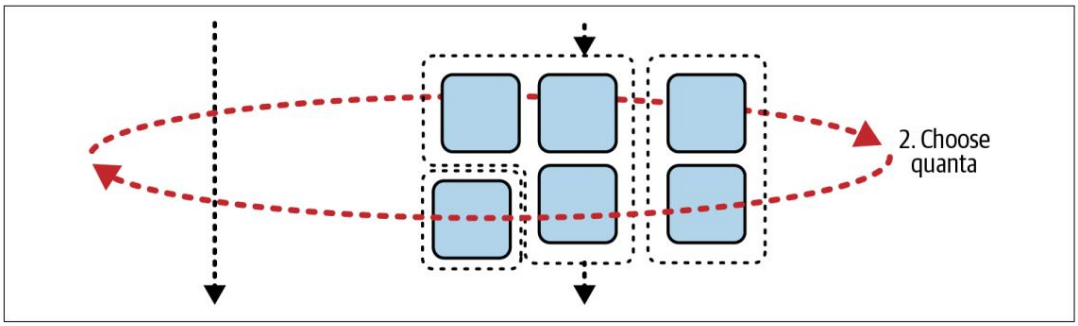

그림 7-3. 분산 아키텍처에서 설계자는 적절한 양자 경계를 선택해야 합니다.

Machine Translated by Google

18장 에서는 세분성을 결정하는 몇 가지 지침을 제공합니다. 다음 단계는 지속성 메커니즘을 선택하는 것인데, 이는 그림 7-4에 나와 있는 두 가지 아키텍처 계열 모두에 해당합니다.

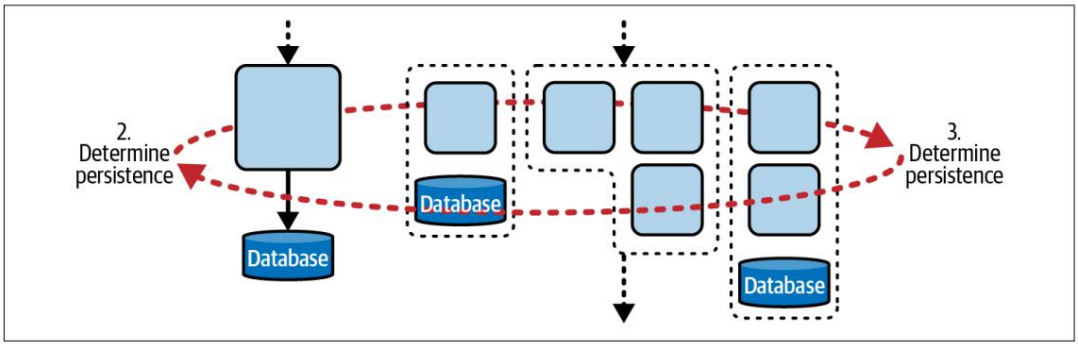

그림 7-4. 두 건축 양식 모두 일반적으로 어떤 형태의 지속성을 필요로 한다.

모놀리식 아키텍처의 경우 일반적으로 단일 모놀리식 데이터베이스가 적합합니다. 아키텍처와 데이터베이스는 함께 개발 및 배포되며, 이러한 과정이 동시에 진행됩니다. 이로써 설계자는 가장 적합한 모놀리식 스타일을 선택하는 단계로 넘어갈 수 있습니다.

반면 분산 아키텍처는 단일 데이터베이스를 사용하거나(이벤트 기반 아키텍처에서 흔히 사용됨) 서비스 세분화 수준에 따라 데이터를 분할할 수 있습니다(마이크로서비스 아키텍처에서처럼). 마지막으로 남은 단계는 각 서비스 간 통신 방식을 동기식 또는 비동기식으로 결정하는 것입니다.

(이 문제는 15 장 에서 자세히 다룹니다.) 일부 시스템에서는 동기식 통신을 선택하면 정적 결합을 통해 설정된 양자 경계가 변경될 수 있다는 점을 기억하십시오. 두 가지 유형의 결합은 빈번하게 상호 작용합니다.

카타: 친환경 생활

아키텍처 양자를 아키텍처 특성의 범위로 사용하여 문제를 분석해 보겠습니다. '친환경'이라는 이 문제는 그림 7-5에 나와 있습니다.

당신은 휴대폰과 같은 오래된 전자제품을 재활용하고 재판매하는 기업인 고잉 그린(GG)과 협력하고 있습니다. GG의 시스템은 공용 키오스크와 웹사이트를 모두 사용하며, 모두 동일한 시스템으로 운영됩니다. 사용자는 자신의 기기 모델 번호와 상태를 업로드할 수 있고, GG는 해당 기기에 대한 입찰가를 제시합니다. 사용자가 입찰가를 수락하면 키오스크에 기기를 반납하거나, 웹사이트를 통해 신청한 경우 GG에서 발송하는 상자를 받아 우편으로 보낼 수 있습니다. GG는 기기를 수령하면 상태를 평가하고 사용자에게 대금을 지급합니다. 그런 다음 GG는 기기의 가치를 평가하여 재활용하거나 재판매합니다. 또한 시스템을 통해 보고서 및 기타 분석 자료를 제공합니다.

Machine Translated by Google

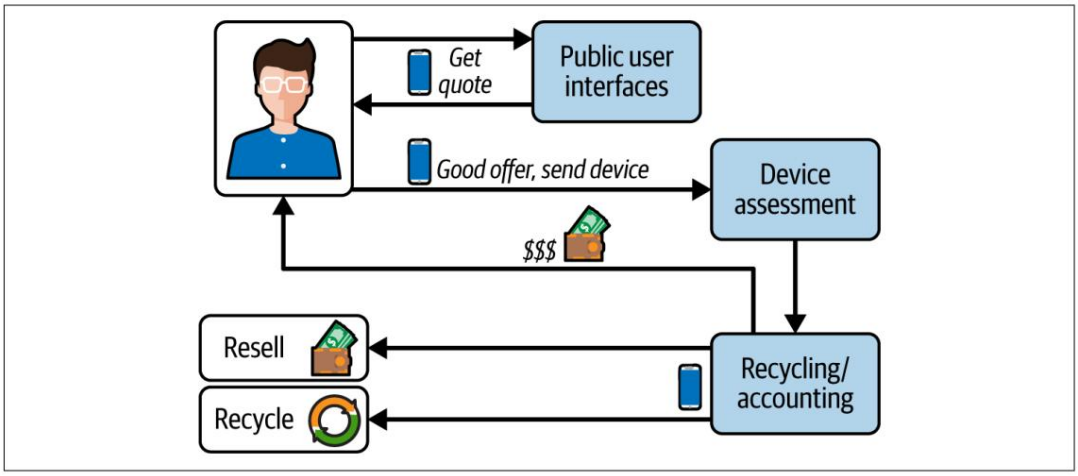

그림 7-5. 친환경 경영을 위한 요구사항 개요

건축적 특성 분석을 수행하면서 그림 7-6에서와 같이 세 개의 뚜렷한 클러스터가 형성되는 것을 확인할 수 있습니다.

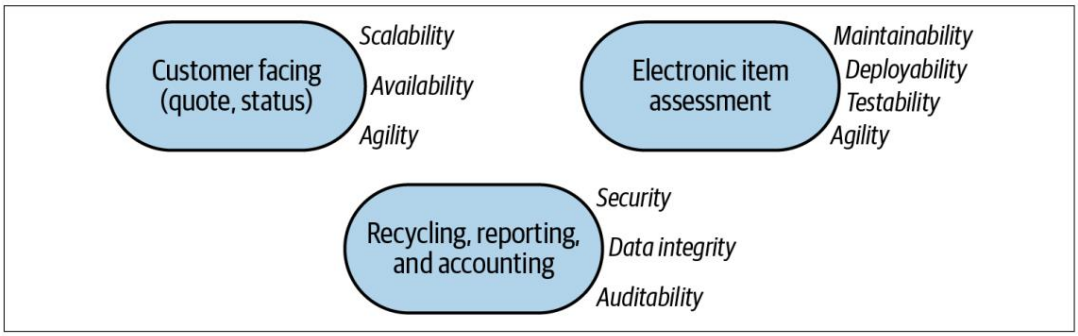

그림 7-6. e GG 아키텍처 특성 분석 결과 세 가지 기능 클러스터가 도출되었습니다.

애플리케이션의 대외적인 부분은 확장성, 가용성 및 민첩성이 필요하고, 백오피스 기능은 보안, 데이터 무결성 및 감사 가능성이 필요하며, 평가 부분은 유지 관리 용이성, 배포 용이성 및 테스트 용이성(민첩성이라는 복합적인 아키텍처 특성을 구성함)이 필요합니다. 평가 부분에 별도의 아키텍처 특성이 필요한 이유는 무엇일까요? 이는 비즈니스 동인과 아키텍처 고려 사항이 어떻게 교차하는지 보여주는 좋은 예입니다. GG의 비즈니스 모델은 가장 가치가 높은 중고 전자 제품을 재판매하는 데 기반을 두고 있으며, 새로운 모델이 끊임없이 출시됩니다.

기기 평가를 더 빨리 업데이트할수록 더 새롭고 (따라서 더 가치 있는) 기기를 재판매할 수 있습니다.

확장성, 가용성, 보안, 데이터 무결성, 감사 용이성, 유지보수성, 배포 용이성, 테스트 용이성 등 모든 기준을 충족하는 시스템을 설계할 수 있을까요? 가능은 하지만 쉽지는 않을 겁니다. 이러한 아키텍처적 특성들은 서로 상충되는 경우가 많습니다. 예를 들어, 빠른 배포를 달성하는 것은 시스템이 확장성, 가용성, 보안, 데이터 무결성, 감사 용이성, 유지보수성, 배포 용이성, 테스트 용이성 등 모든 조건을 만족할 때 더 어려워집니다.

Machine Translated by Google

또한 감사 가능성과 같은 백오피스 관련 사항을 우선시합니다. 그리고 사용자 인터페이스는 시스템의 다른 부분과는 완전히 다른 수준의 확장성을 요구합니다.

모든 것을 다 하려고 하기보다는 아키텍처 특성 클러스터를 기준으로 양적(quanta)을 구분할 수 있습니다. 그림 7-7 에서 점선은 각 아키텍처 양적을 나타냅니다. 특성 범위를 서비스 세분성의 기준으로 삼는 것은 가장 유익한 절충안을 결정하는 좋은 첫걸음입니다. (이 주제는 18 장 에서 더 자세히 다룹니다.)

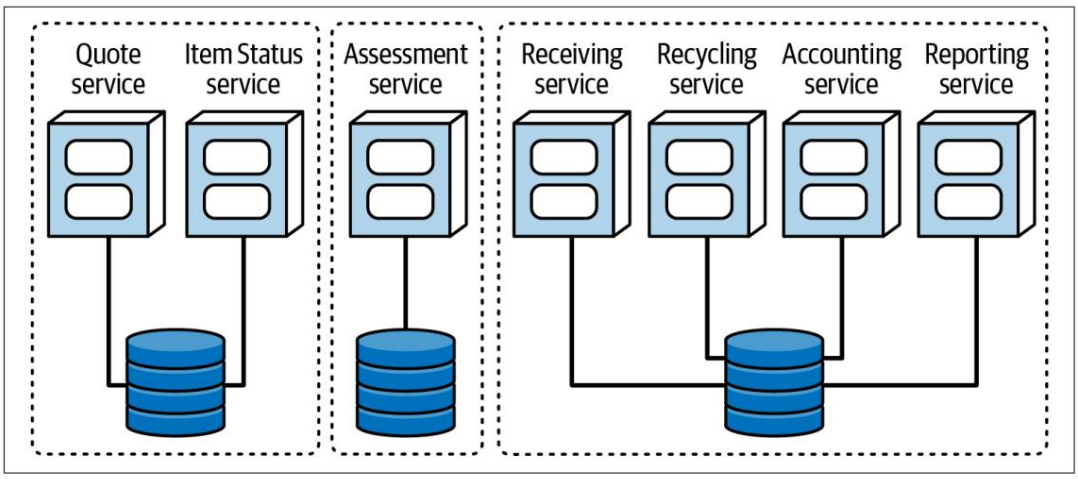

그림 7-7은 GG 아키텍처가 각 아키텍처 특성 집합을 아키텍처 경계로 포착함을 보여줍니다.

범위 설정 및 클라우드

클라우드 기반 리소스는 시스템의 운영 아키텍처 특성을 상당 부분 포함하고 있기 때문에 상황을 더욱 복잡하게 만듭니다. 아키텍트는 배포 모델에 따라 애플리케이션의 전체 또는 일부에 클라우드 기반 리소스를 사용할 때 최소 두 가지 시나리오를 고려해야 합니다.

클라우드를 활용한 컨테이너 호스팅 많은 개발팀
: 은 클라우드를 대체 운영 센터로 활용하여 서버용 컨테이너를 실행(및 오케스트레이션)합니다. 이러한 상황에서 아키텍트는 컨테이너의 아키텍처적 특성과 오케스트레이션 도구(예: Kubernetes)로 인해 발생하는 제약 조건을 고려해야 합니다.

클라우드 제공업체 리소스를 시스템 구성 요소로 활용하기
: 클라우드 기반 시스템의 또 다른 유형은 트리거 함수, 데이터베이스 등과 같은 클라우드 제공업체의 구성 요소를 사용하여 애플리케이션을 조합하는 것입니다. 이 경우 아키텍트는 제공업체가 제공하고 유지 관리하는 기능을 살펴보고 특정 환경에 적합한 기능을 구축하는 방법에 대한 통찰력을 얻어야 합니다.

Machine Translated by Google

오늘날 클라우드 제공업체의 구성 설정으로 알려진 탄력성 같은 많은 기능은 이전 세대의 물리적 시스템 설계자들이 어렵게 얻어 낸 결과물입니다. 당시 그들이 고민했던 주요 사항들은 이제 훨씬 수월해졌지만, 제공업체 가용성이나 강화된 보안 문제와 같은 현대적인 절충안들이 여전히 존재합니다. 소프트웨어 아키텍처의 세부 사항은 끊임없이 변화하지만, 절충안을 분석하는 작업은 변함없이 중요합니다.