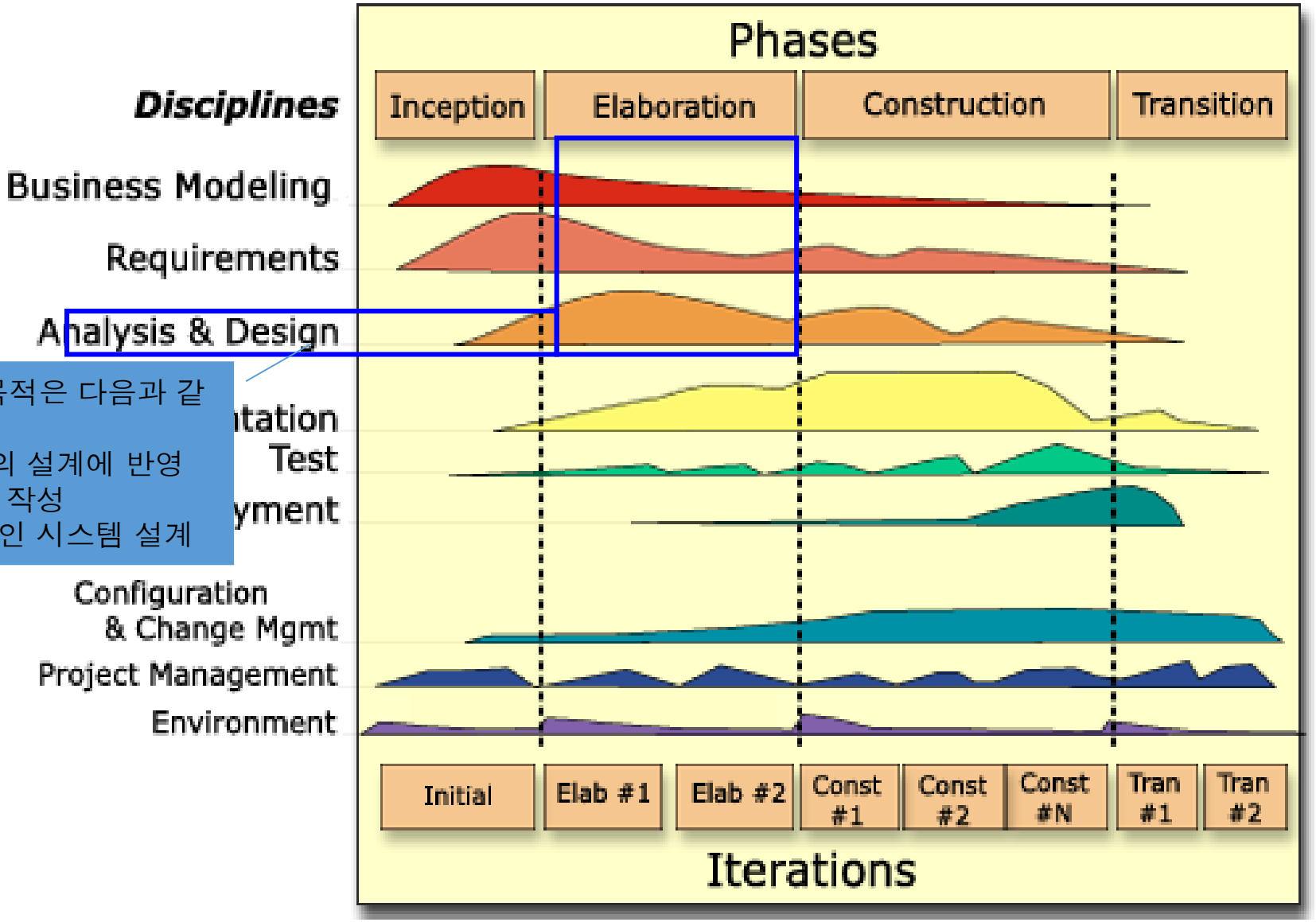
Patterns for KUPE

박문학 임담섭 추로요 남룽아웅

Content

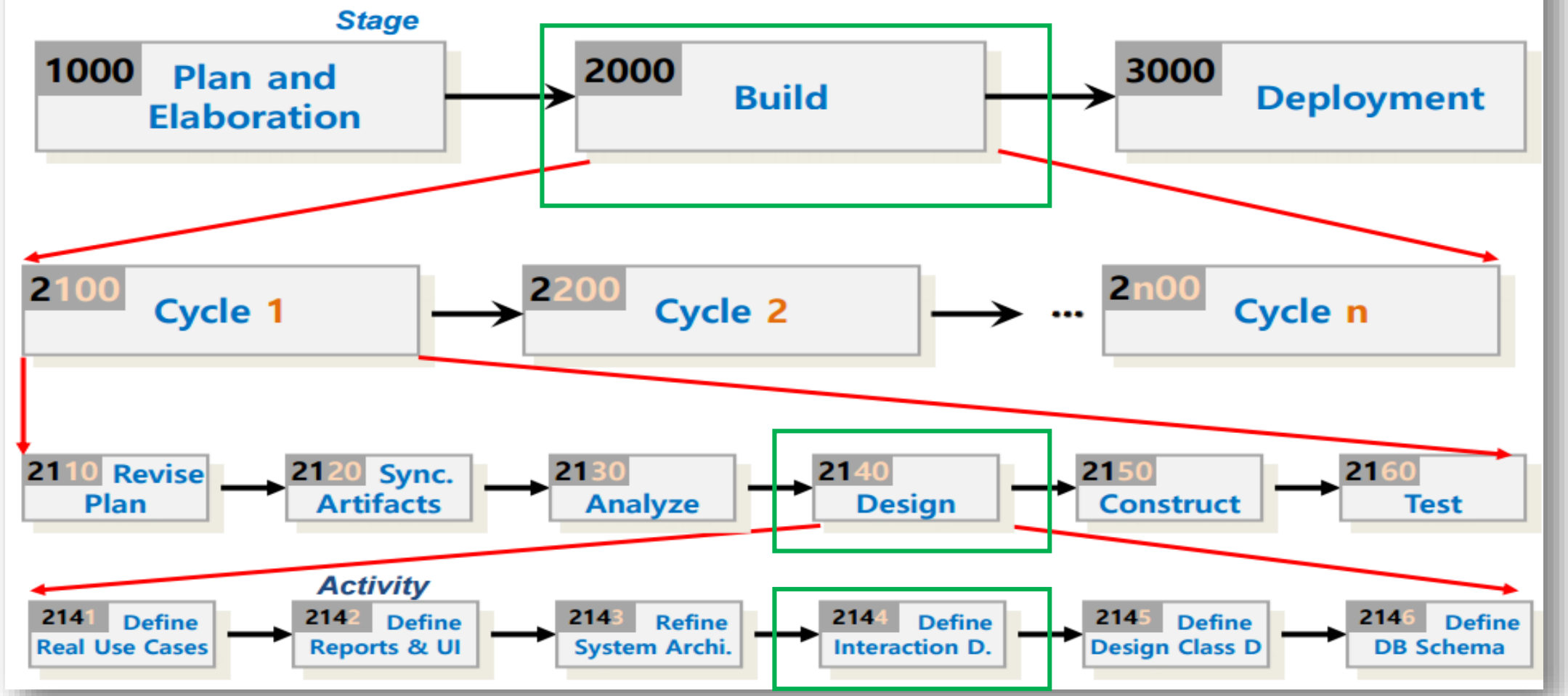
- Grasp pattern
- GOF design pattern

Analysis & Design Overview



Analysis & Design의 목적은 다음과 같다.
- 요구사항들이 시스템의 설계에 반영
- 베이스 라인 아키텍처 작성
- 개발 플랫폼에 독립적인 시스템 설계

4. Architecture of KUPE



GRASP pattern

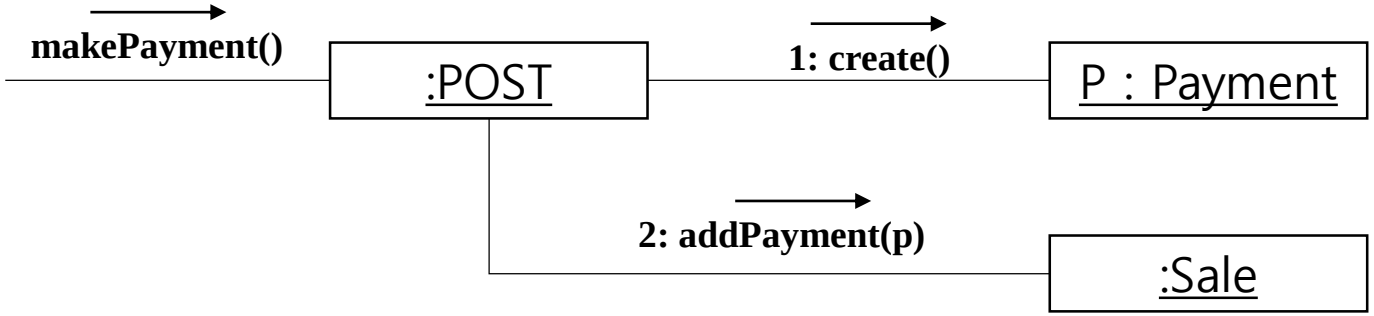
GRASP Design Pattern

- General Responsibility Assignment Software Patterns (GRASP)
  - Fundamental and universal OO design principles in the form of patterns
  - It provides direction for
    - **assigning responsibilities to classes**
    - determining the classes that will be in a design

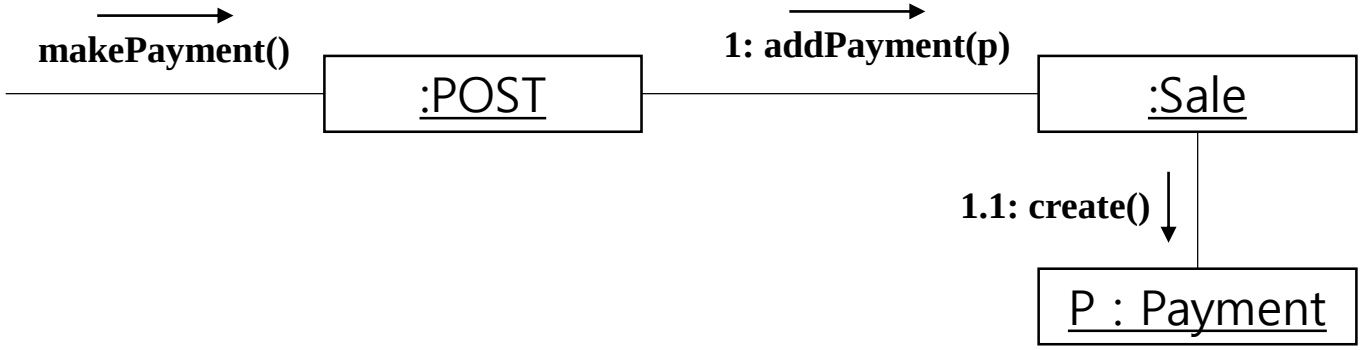
GRASP Patterns

- Low Coupling
- High Cohesion
- Expert
- Creator
- Controller
- Don’t talk to Stranger

POST Example for Low Coupling

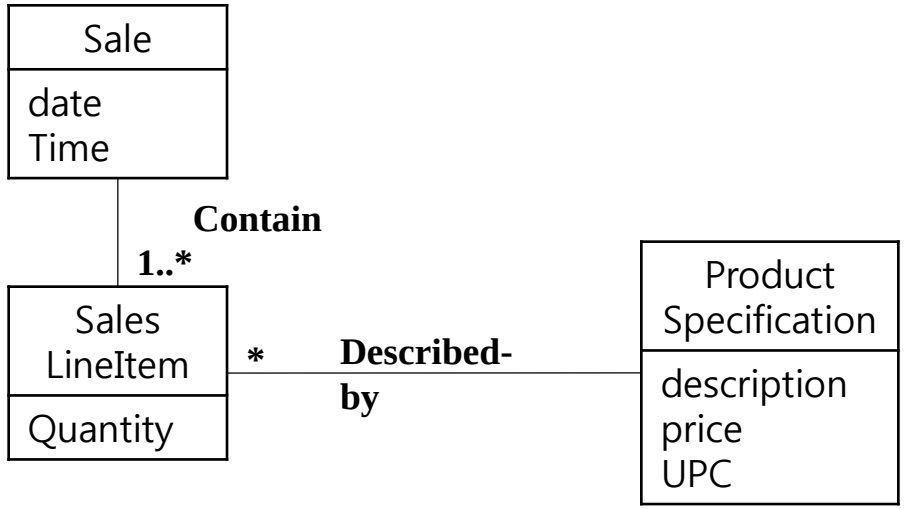


=> This design couples the POST class to knowledge of the Payment class.

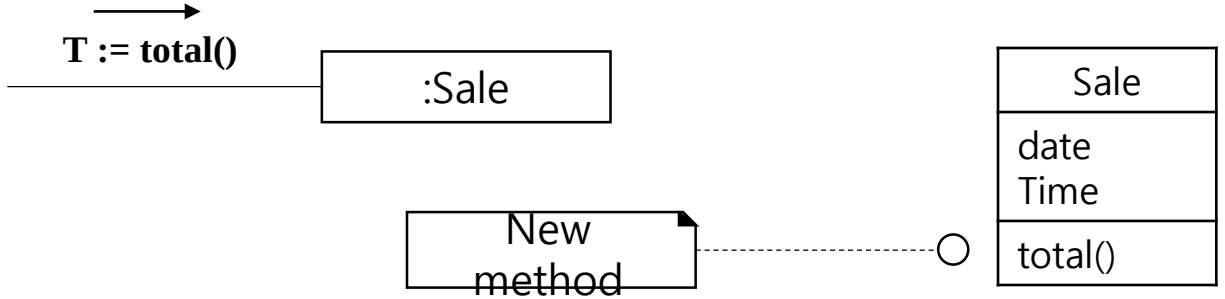


POST Example for Expert

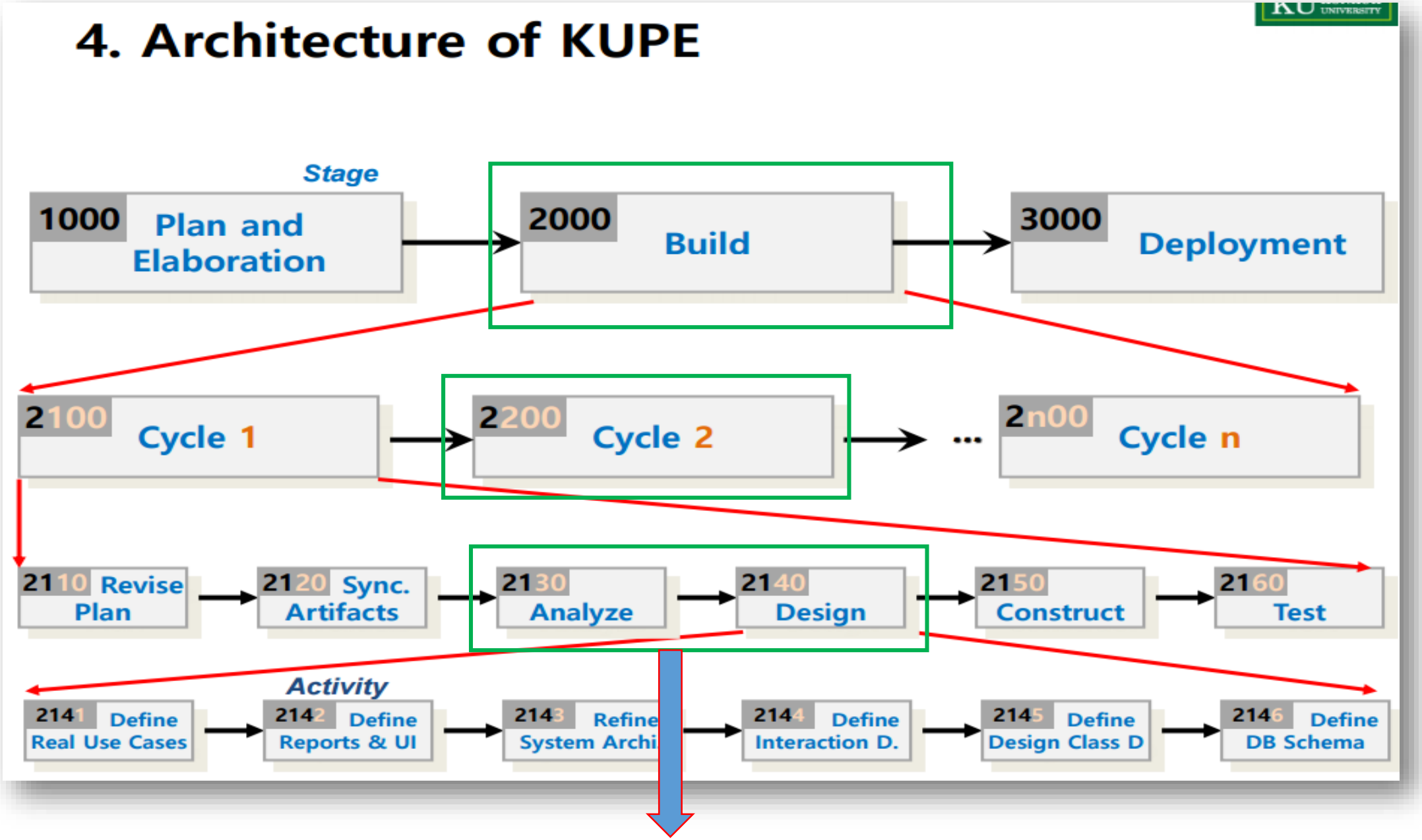
- Conceptual Model



- How to calculate the grand total of the sale



4. Architecture of KUPE

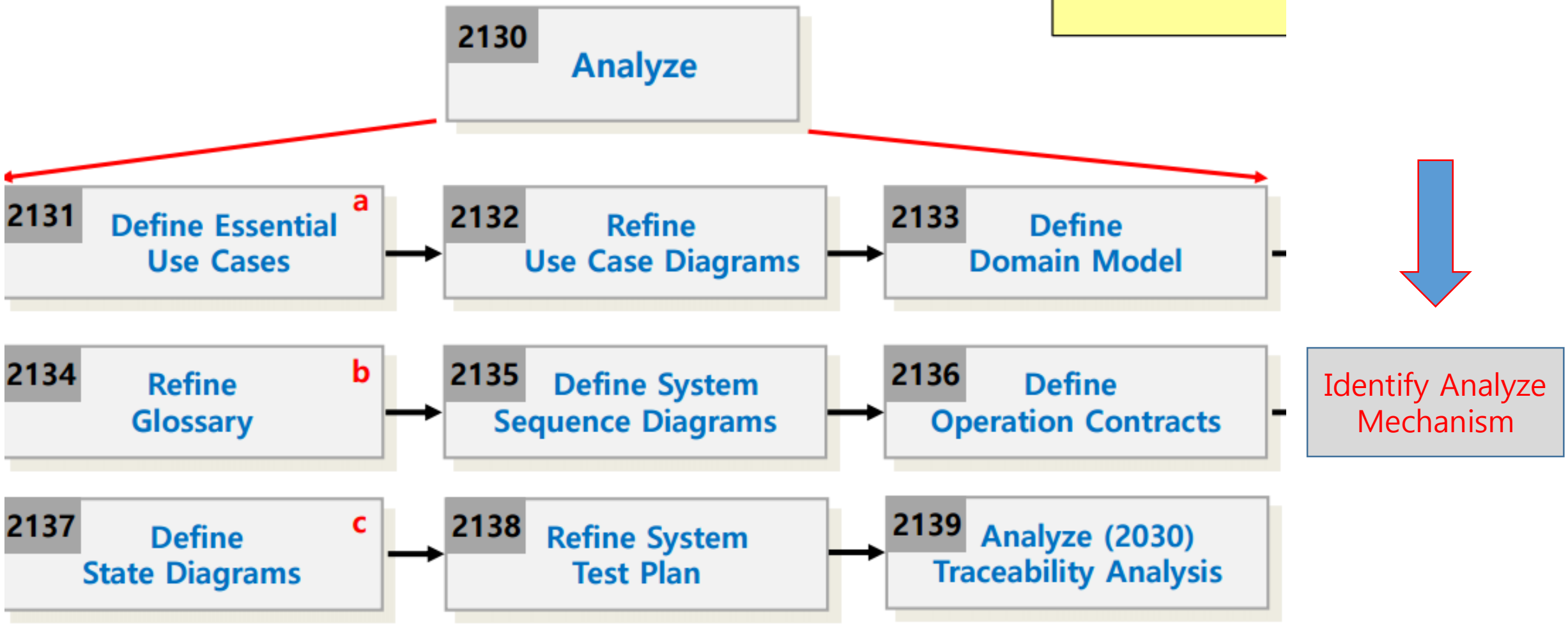


Design pattern

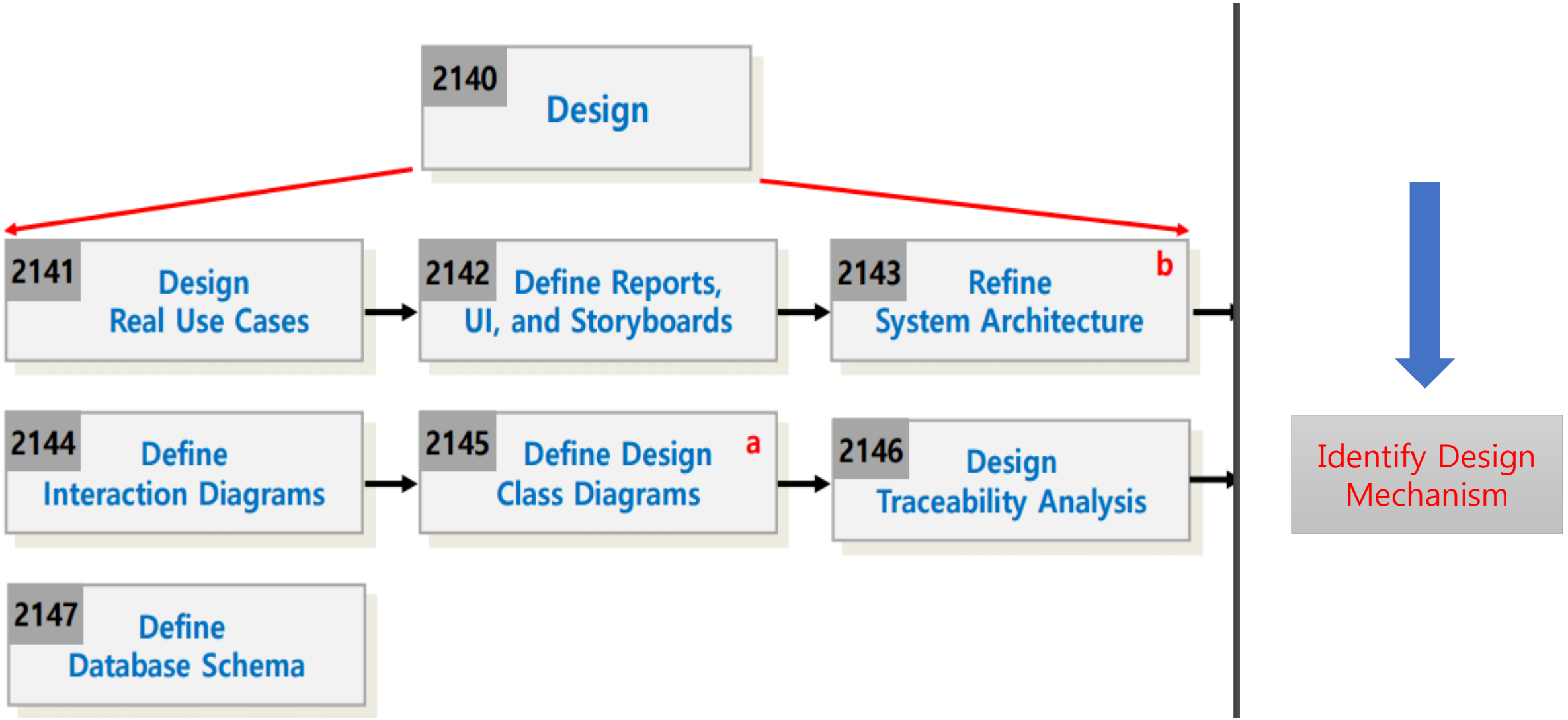
Iteration 2 Analyze

- Phase 2030 Activities

b. ongoing
c. optional



Iteration 2 Design



Architecture Design

- Identify Design Mechanism Overview
  - Architecture Analysis 단계에서 확인한 분석 메커니즘을 구현환경상의 제약조건을 기반으로 설계 메커니즘으로 정제한다.
  - Identify Design Mechanism Steps
    - Categorizes Clients of Analysis Mechanisms
    - Document Architectural Mechanisms

Iteration 2 Requirements

- 1. **Support for variations in third-party external services**. For example, different tax calculators must be connectable to the system, and each **has a unique interface**. Likewise with different accounting systems and so forth. Each will offer a different API and protocol for a core of common functions.

- 2. Complex pricing rules.
- 3. Pluggable business rules.
- 4. A design to refresh a GUI window when the sale total changes.

Requirement

The NextGen POS system needs to support several kinds of external third-party services, including tax calculators, credit authorization services

Identify Analyze Mechanism

Need a adapter interface

Design 단계

Analyze Mechanism

Design Mechanism

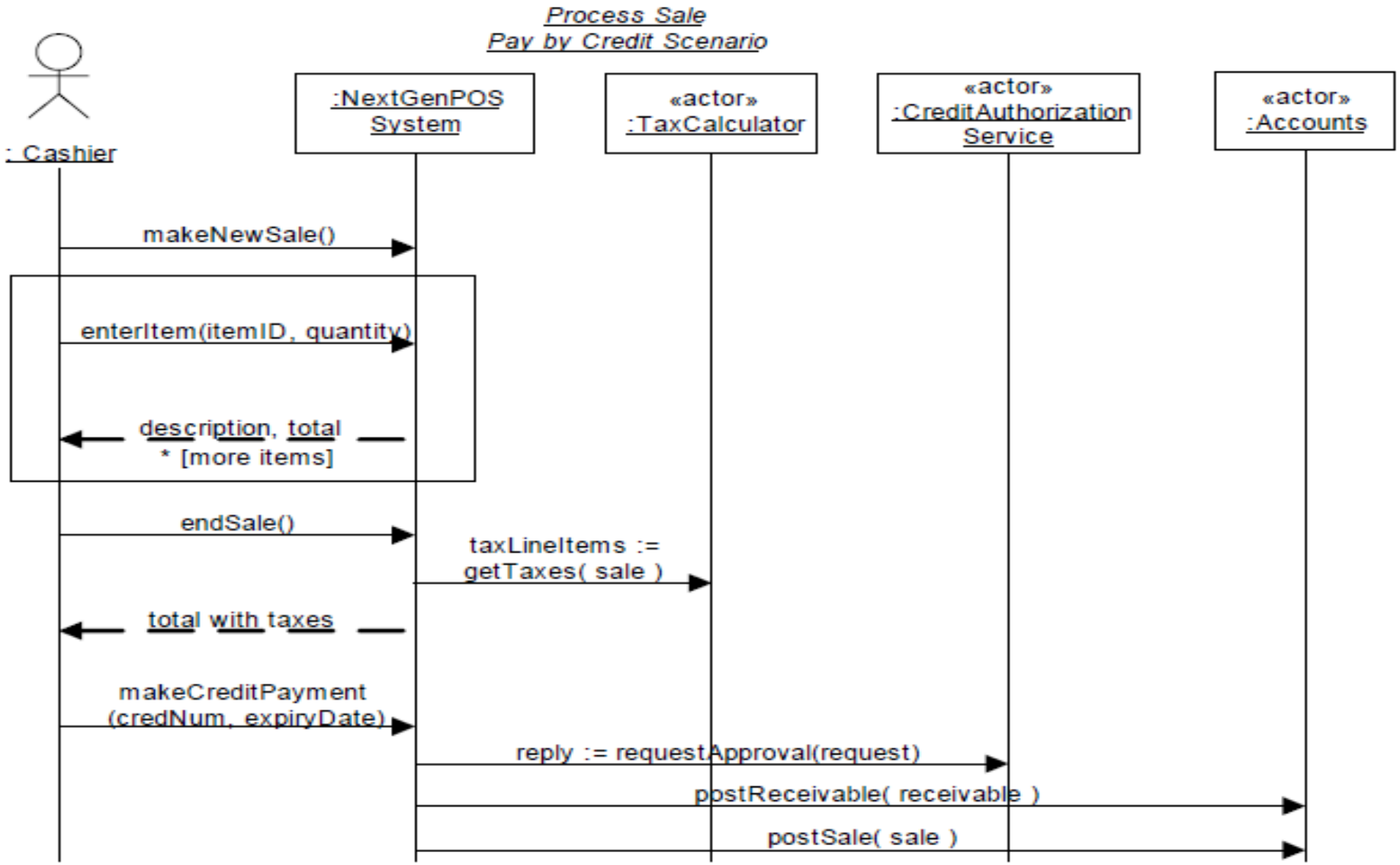
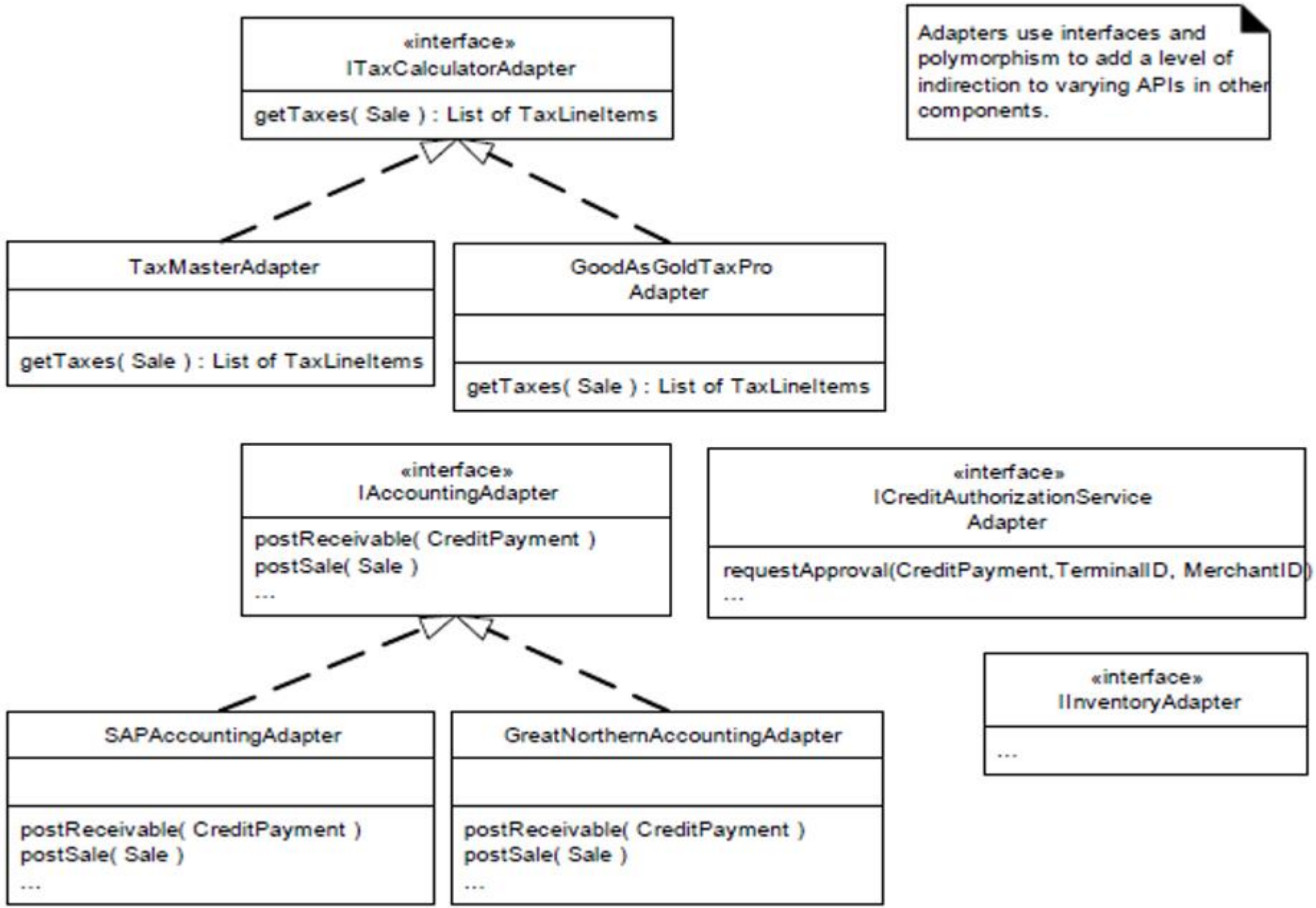
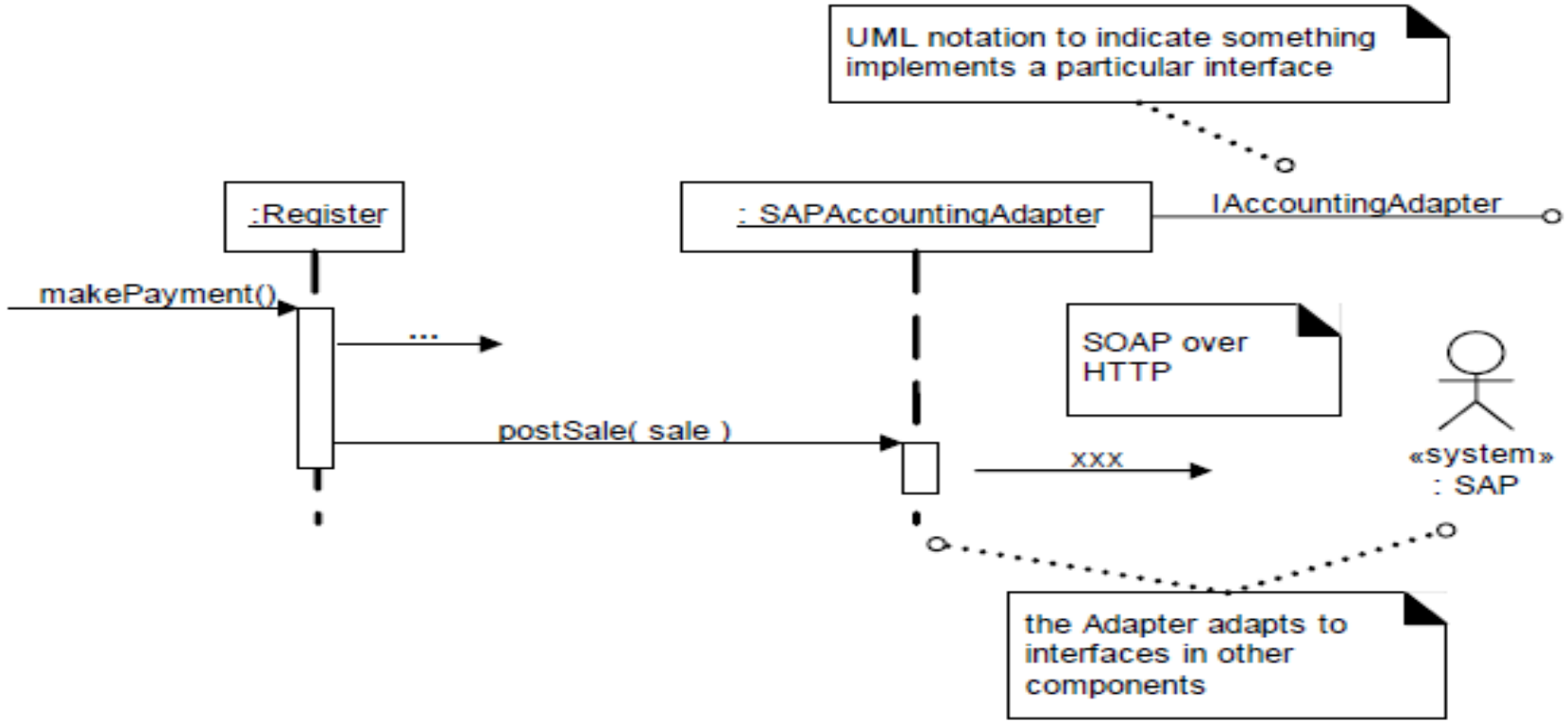


Figure 21.1 An SSD scenario that illustrate some external systems


«interface»
ITaxCalculatorAdapter
getTaxes(Sale) : List of TaxLineItems
Adapters use interfaces and polymorphism to add a level of indirection to varying APIs in other components.
TaxMasterAdapter
getTaxes(Sale) : List of TaxLineItems
GoodAsGoldTaxPro Adapter
getTaxes(Sale) : List of TaxLineItems
«interface»
IAccountingAdapter
postReceivable(CreditPayment)
postSale(Sale)
...
«interface»
ICreditAuthorizationService Adapter
requestApproval(CreditPayment, TerminalID, MerchantID)
...
«interface»
IInventoryAdapter
...
SAPAccountingAdapter
postReceivable(CreditPayment)
postSale(Sale)
...
GreatNorthernAccountingAdapter
postReceivable(CreditPayment)
postSale(Sale)
...

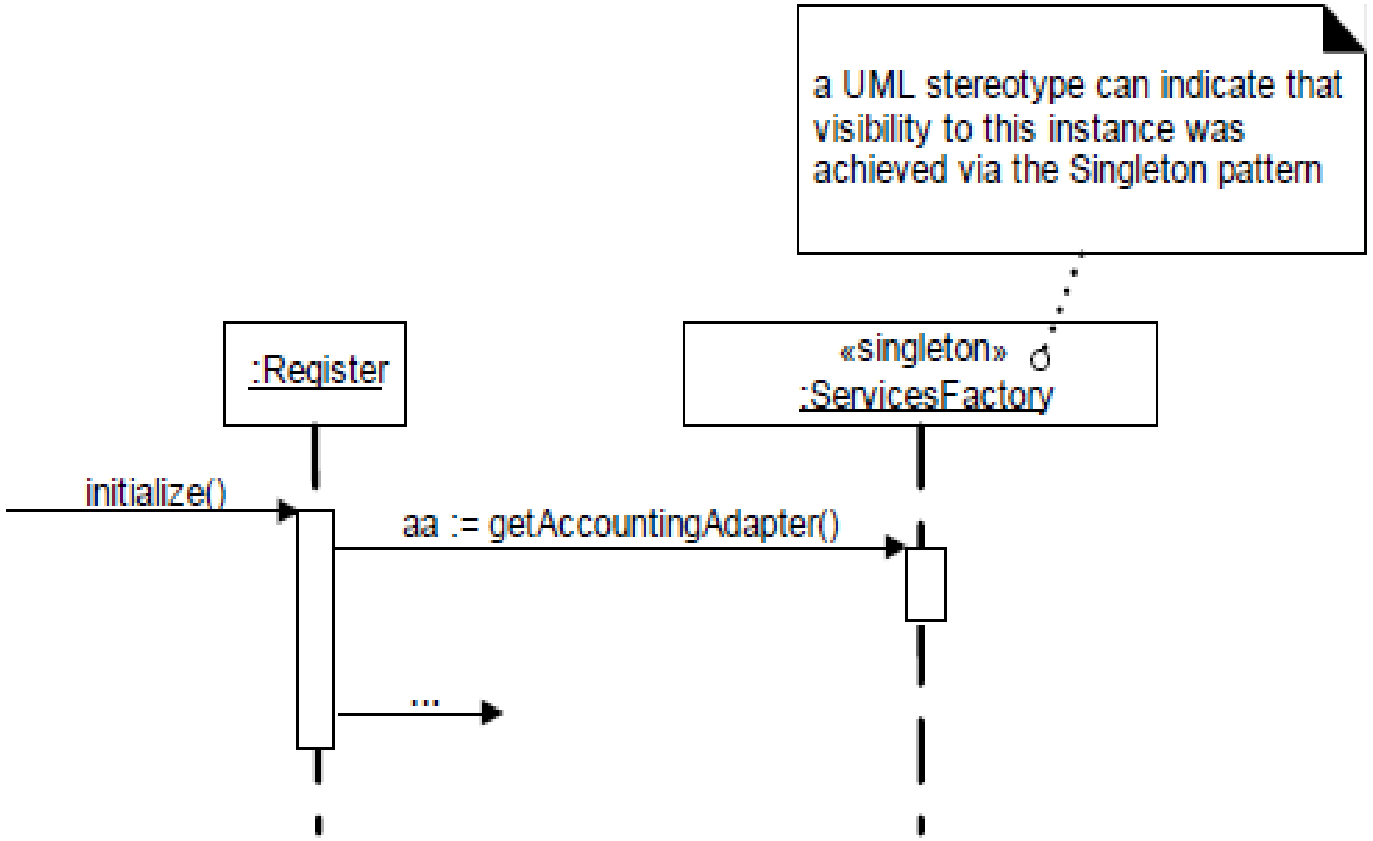
Adapter pattern

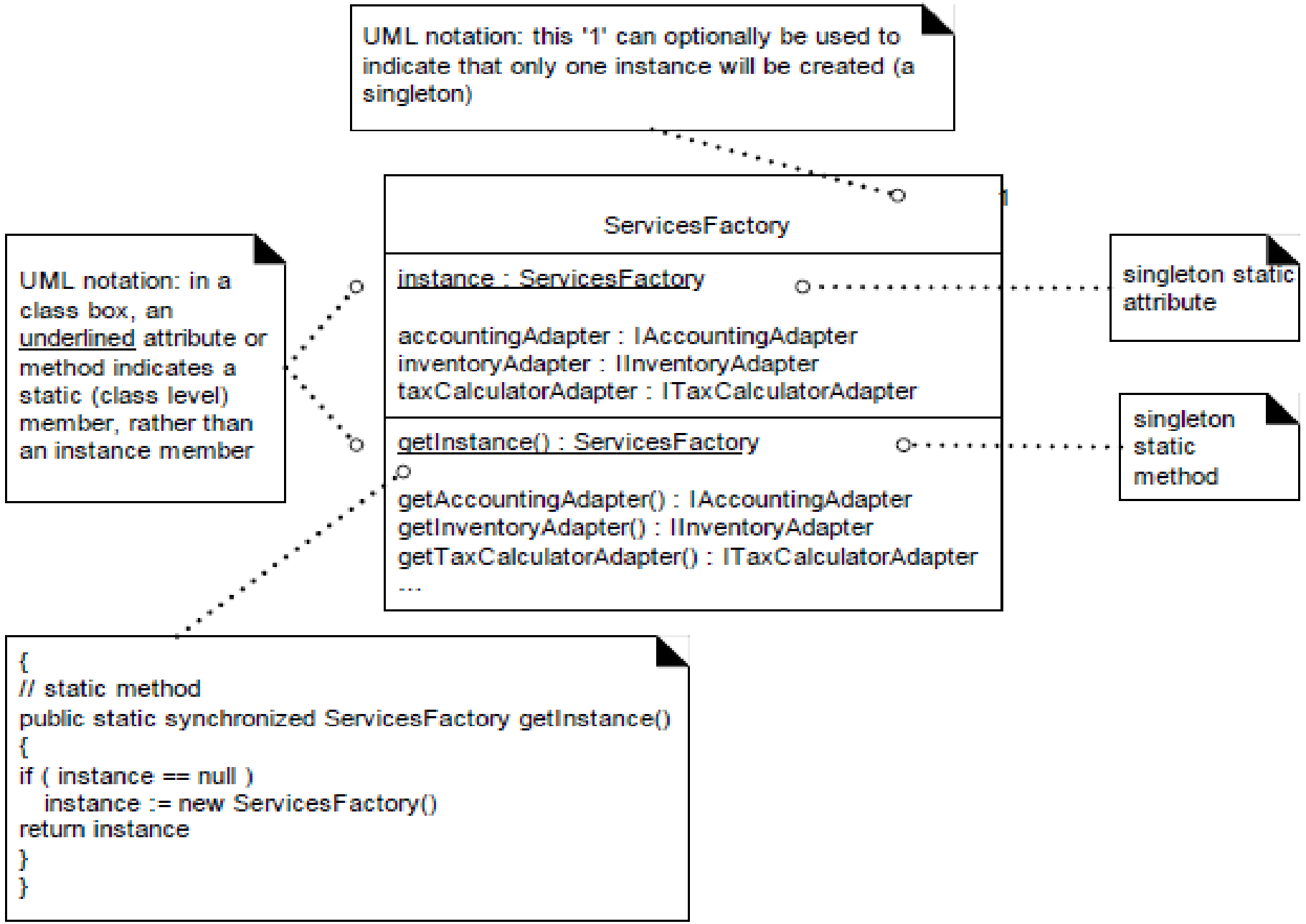


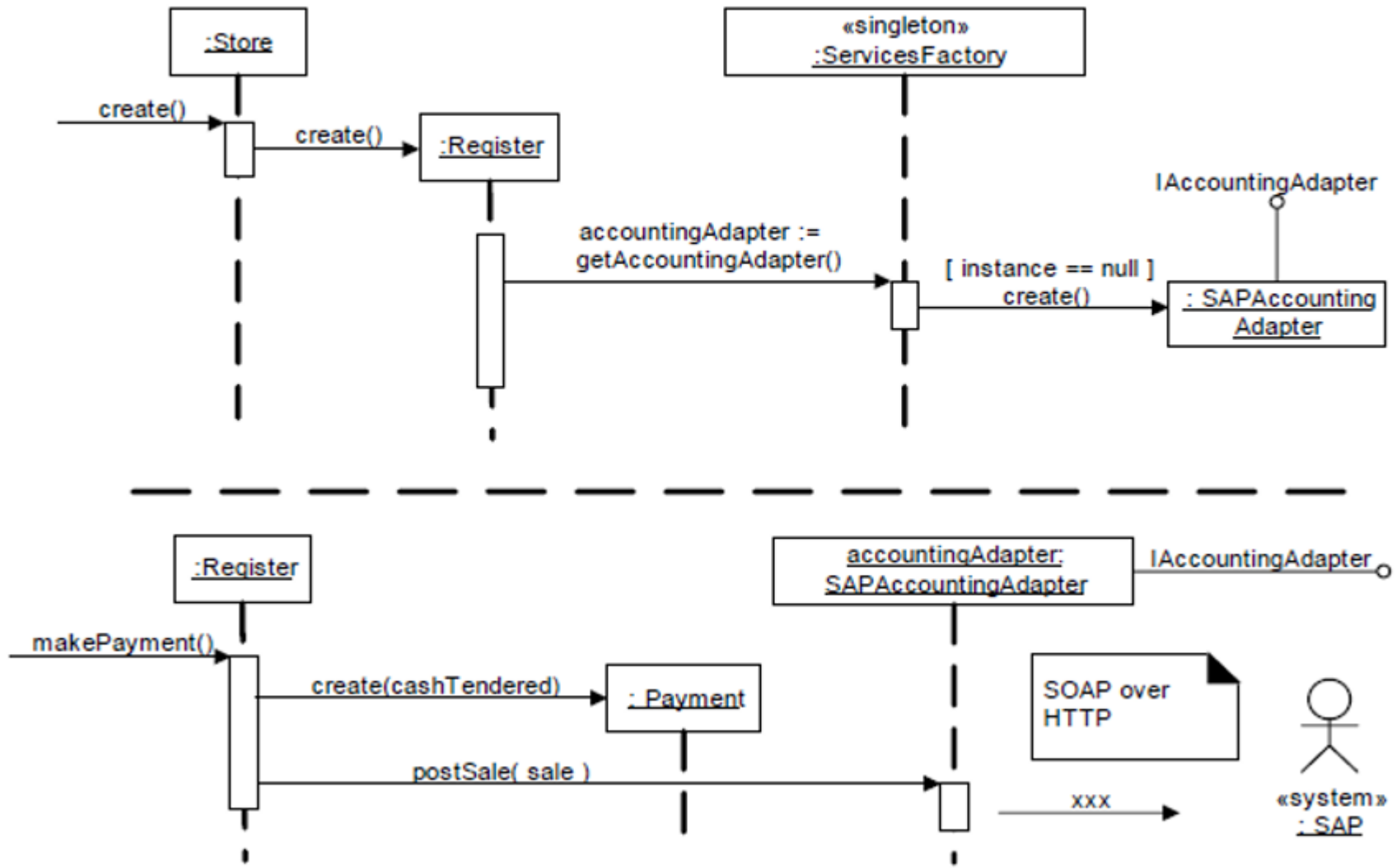
The adapter raises a new problem in the design:

- In the prior Adapter pattern solution for external services with varying interfaces, who creates the adapters?
- And how to determine which class of adapter to create, such as *TaxMaster-Adapter* or *GoodAsGoldTaxProAdapterl*

Factory pattern & singleton pattern




UML notation: this '1' can optionally be used to indicate that only one instance will be created (a singleton)
ServicesFactory
instance : ServicesFactory
accountingAdapter : IAccountingAdapter
inventoryAdapter : IInventoryAdapter
taxCalculatorAdapter : ITaxCalculatorAdapter
getInstance() : ServicesFactory
getAccountingAdapter() : IAccountingAdapter
getInventoryAdapter() : IInventoryAdapter
getTaxCalculatorAdapter() : ITaxCalculatorAdapter
...
singleton static attribute
singleton static method
UML notation: in a class box, an underlined attribute or method indicates a static (class level) member, rather than an instance member
{
// static method
public static synchronized ServicesFactory getInstance()
{
if (instance == null)
instance := new ServicesFactory()
return instance
}
}


:Store
«singleton»
:ServicesFactory
create()
create()
:Register
IAccountingAdapter
accountingAdapter :=
getAccountingAdapter()
[instance == null]
create()
: SAPAccounting
Adapter
:Register
accountingAdapter:
SAPAccountingAdapter
IAccountingAdapter
makePayment()
create(cashTendered)
: Payment
SOAP over
HTTP
postSale(sale)
xxx
«system»
: SAP

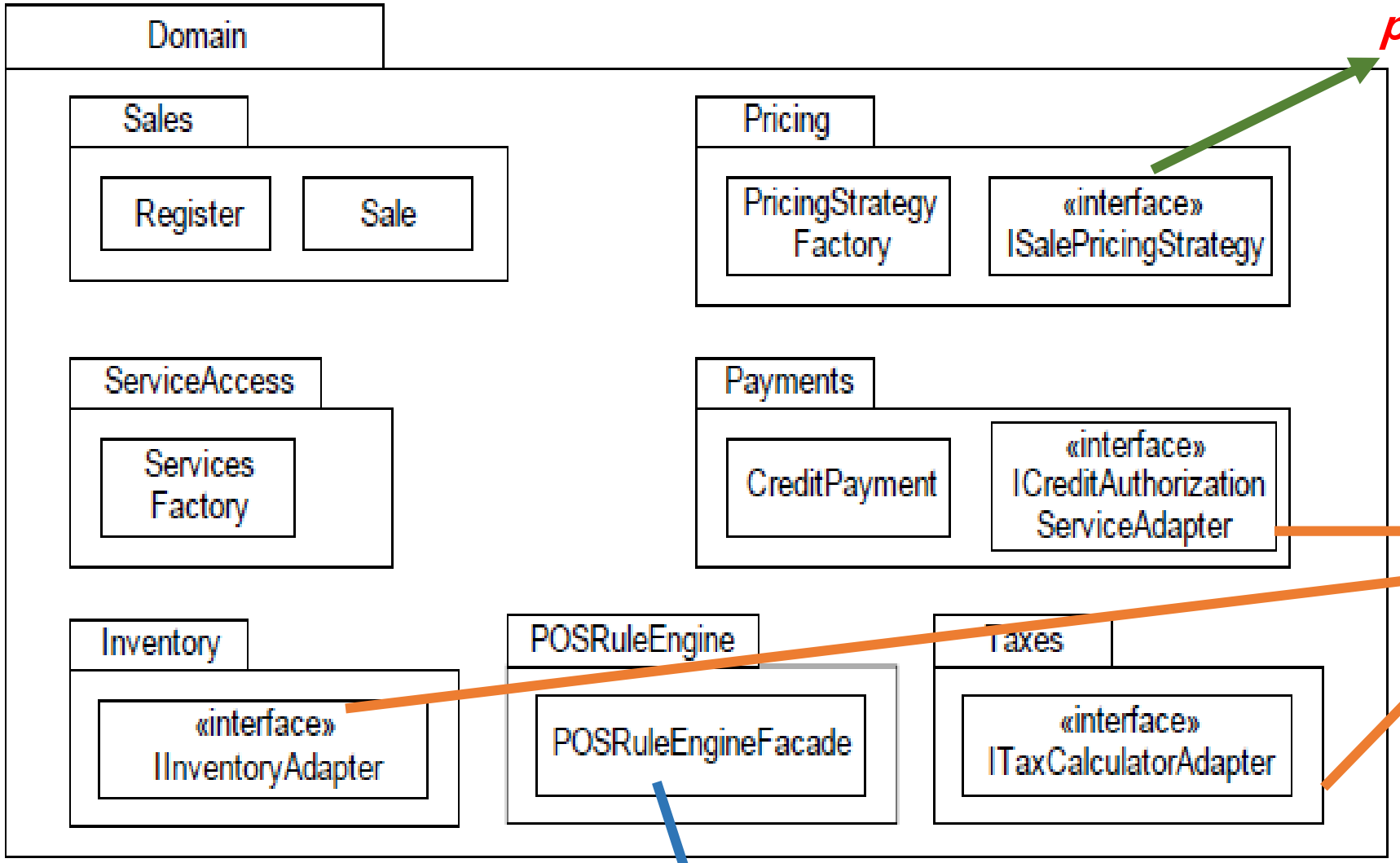
Requirement

- Complex pricing rules.---*Strategy* ,Composite pattern

- Pluggable business rules--- Façade,Singleton pattern

- A design to refresh a GUI window when the sale total changes. ---Observer pattern

수정한 domain model



Strategy & composite pattern

Adapter &factory &singleton pattern

Facade pattern