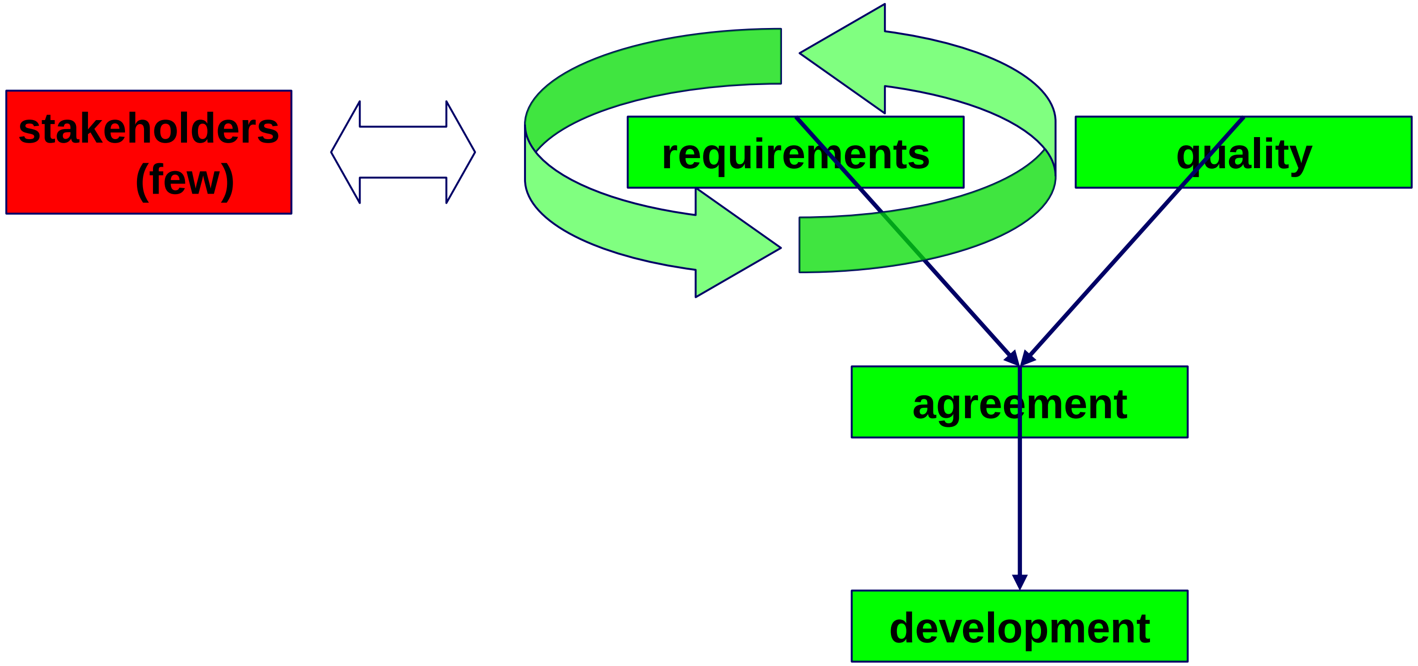
Domain-Specific Software Architecture

Inggriani Liem, Yudistira Asnar

Credit

- Special Thanks to
  - **Zhiying Lin** on Slides in Domain-Specific Software Engineering
  - **Hans van Vliet** on Slides in SE, Software Architecture

Pre-architecture life cycle



Why Is Architecture Important?

Architecture is the vehicle for stakeholder communication

Architecture manifests the earliest set of design decisions

- Constraints on implementation
- Dictates organizational structure
- Inhibits or enable quality attributes

Architecture is a transferable abstraction of a system

- Product lines share a common architecture
- Allows for template-based development
- Basis for training

Software architecture, definition (1)

The architecture of a software system defines that system in terms of computational components and interactions among those components.

(from Shaw and Garlan, *Software Architecture, Perspectives on an Emerging Discipline*, Prentice-Hall, 1996.)

Software Architecture, definition (2)

The software architecture of a system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

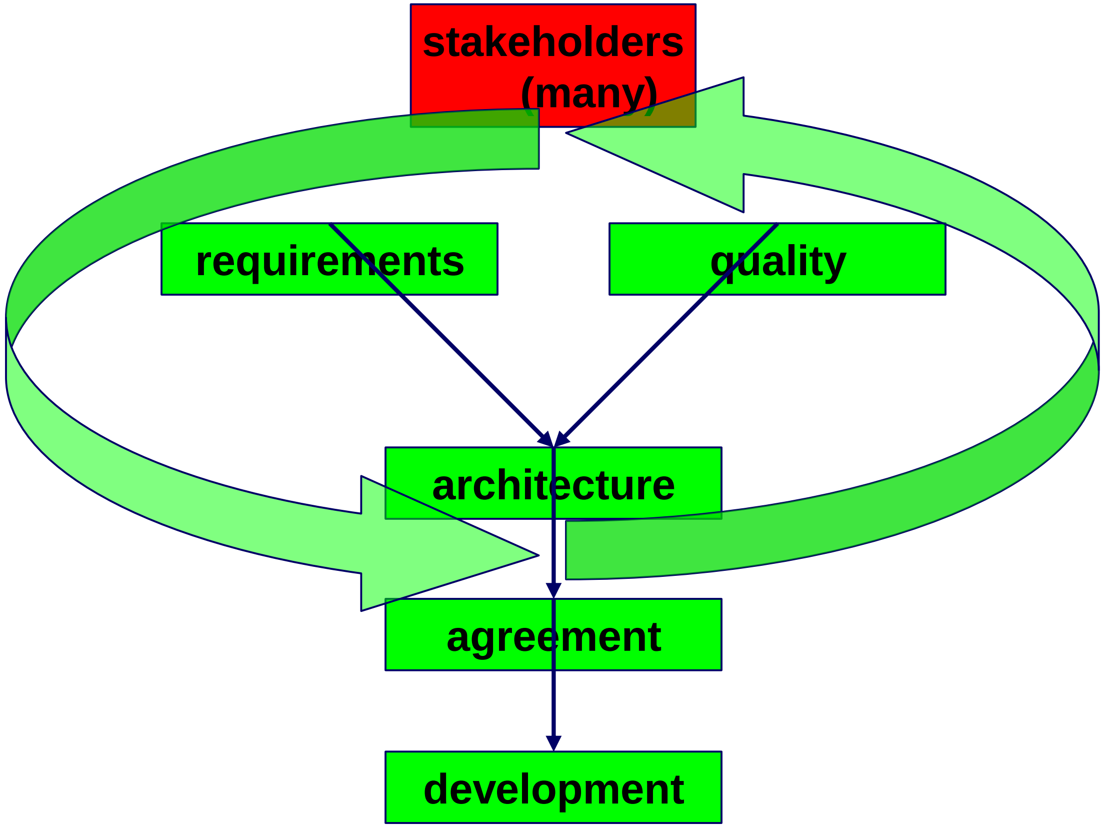
(from Bass, Clements, and Kazman, *Software Architecture in Practice*, SEI Series in Software Engineering. Addison-Wesley, 2003.)

Software Architecture, definition (3)

Architecture is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution

(from IEEE Standard on the Recommended Practice for Architectural Descriptions, 2000.)

Architecture in the life cycle



stakeholders (many)

requirements

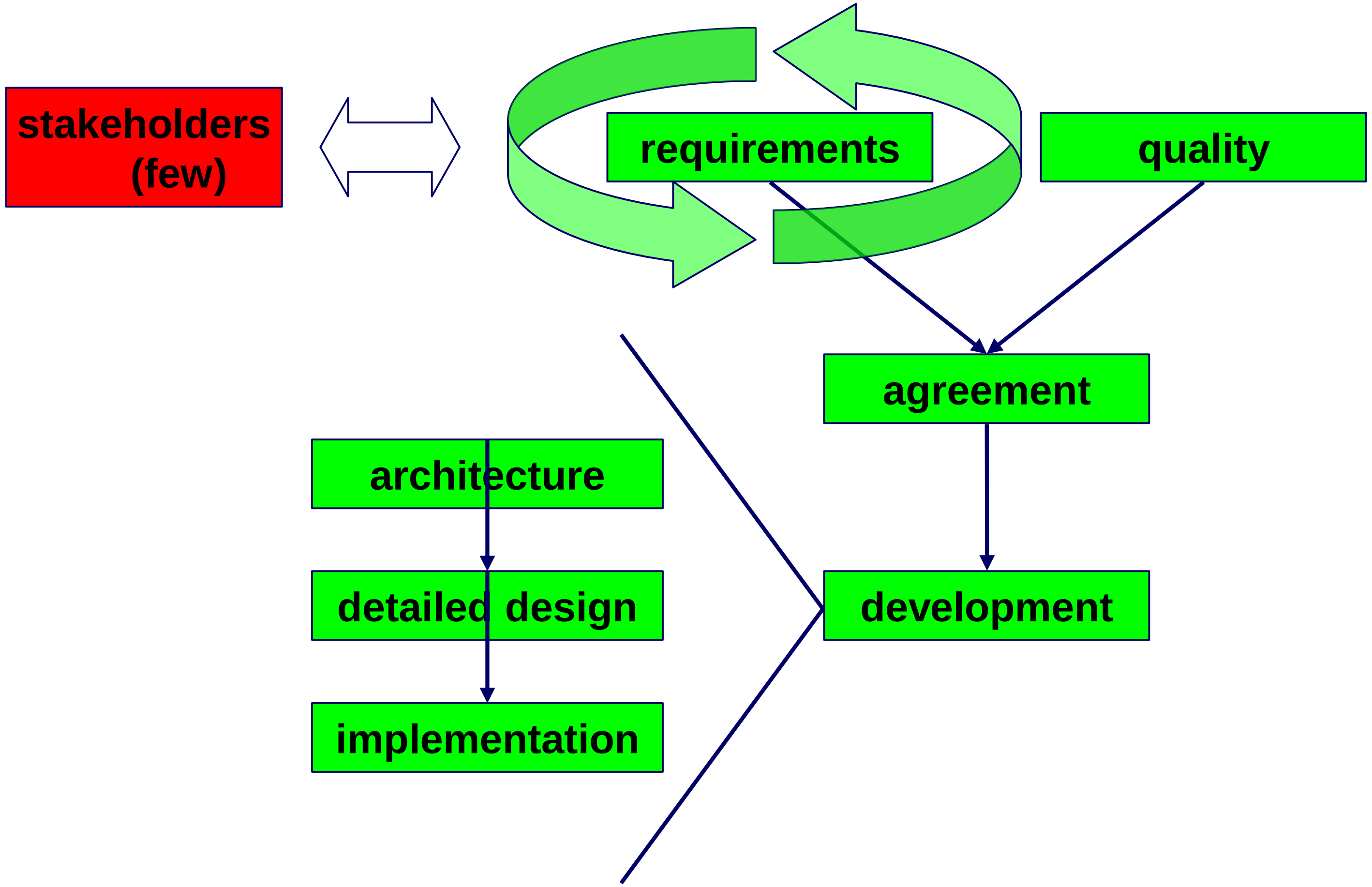
quality

architecture

agreement

development

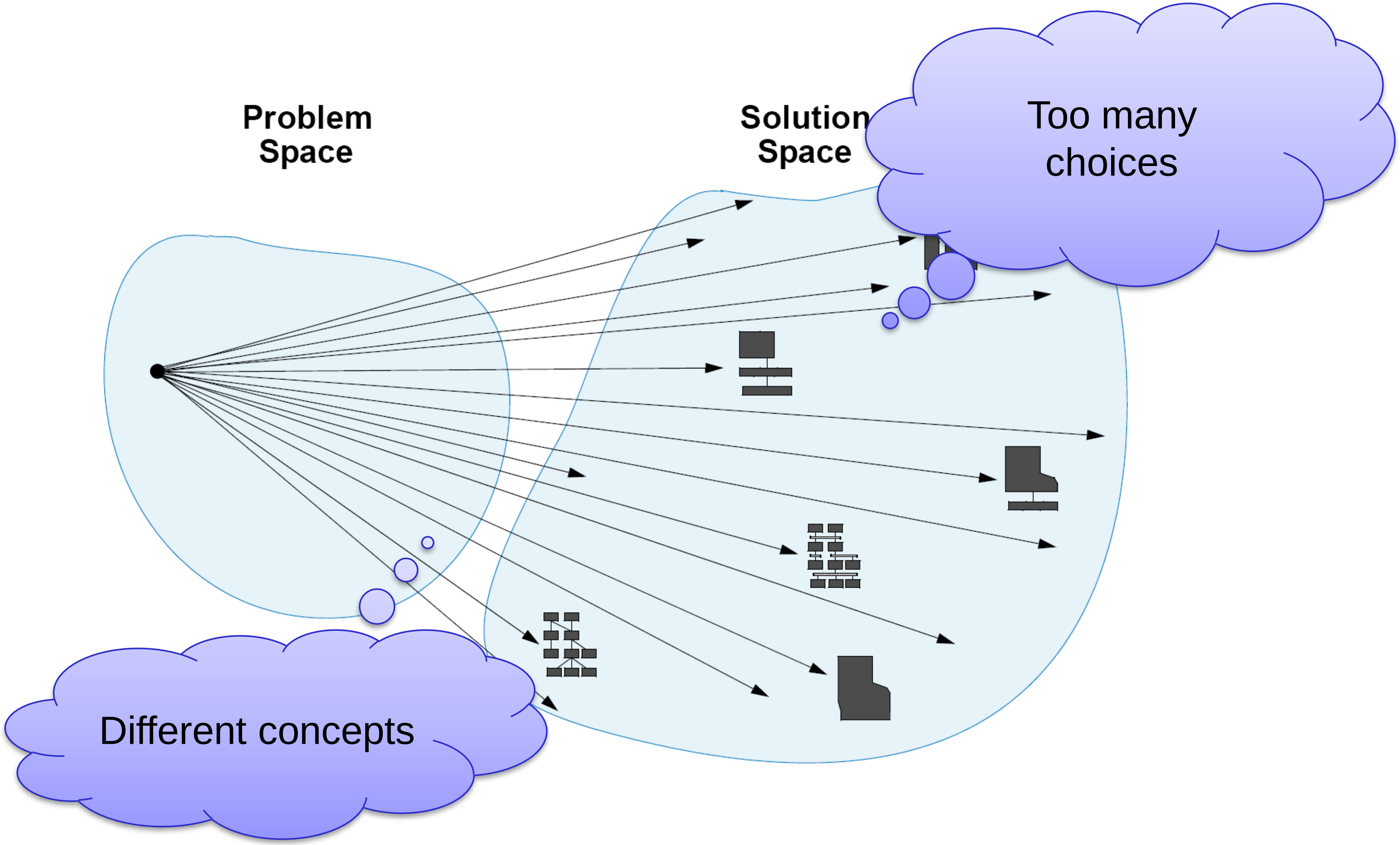
Adding architecture, the easy way



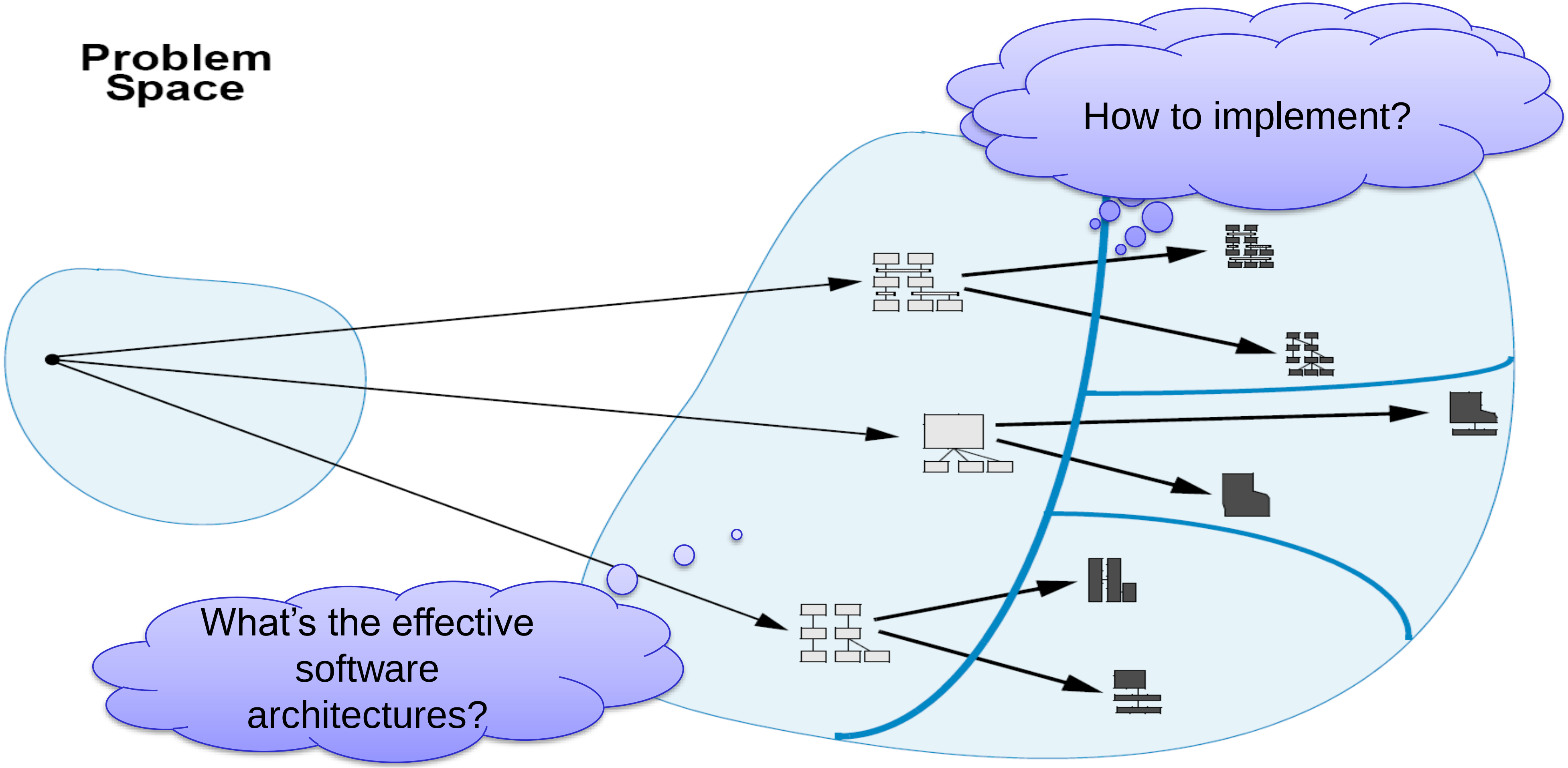
Domain-Specific Software Engineering

An approach to software engineering that is characterized by extensively leveraging existing domain knowledge.

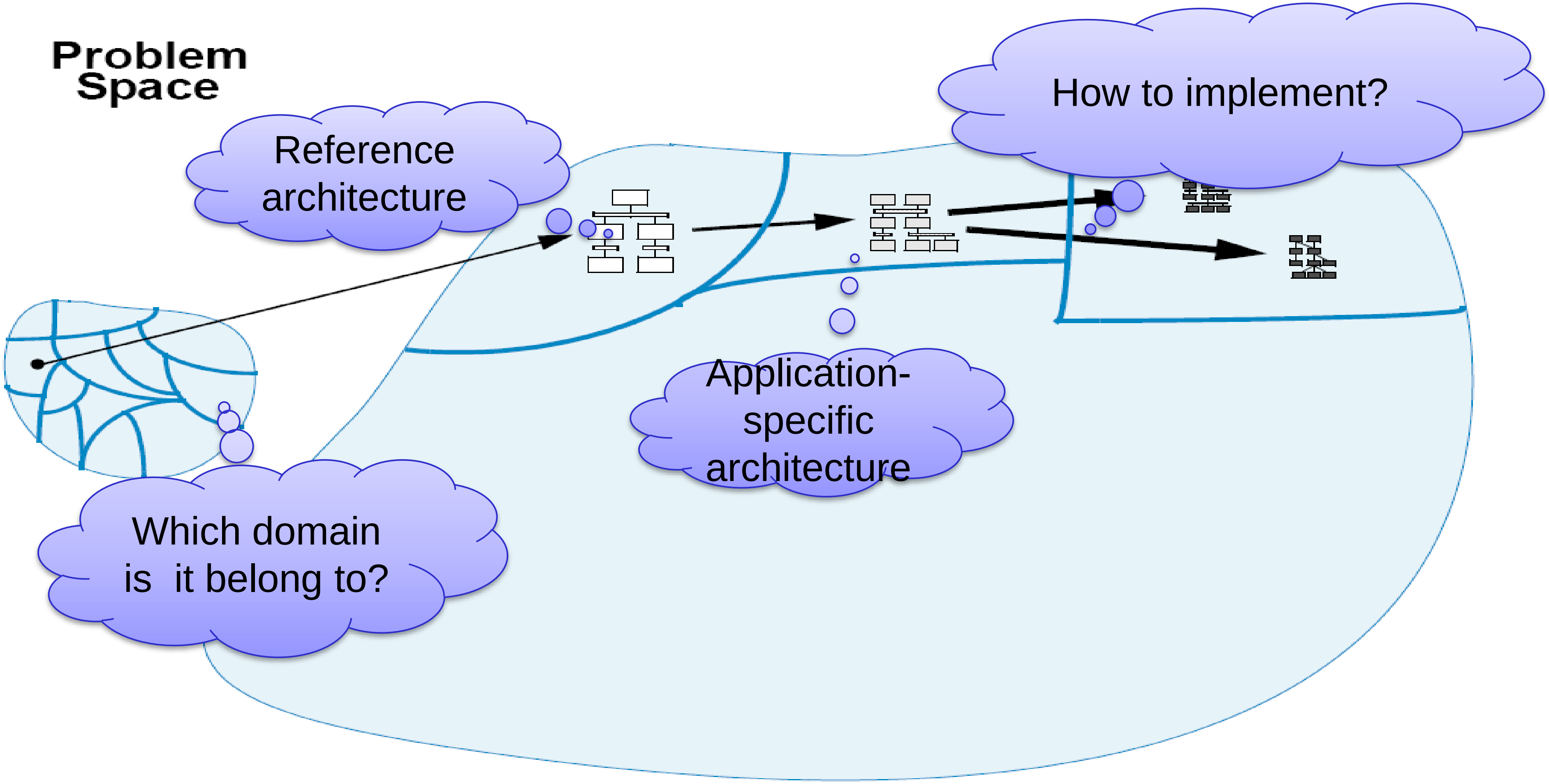
Traditional Software Engineering



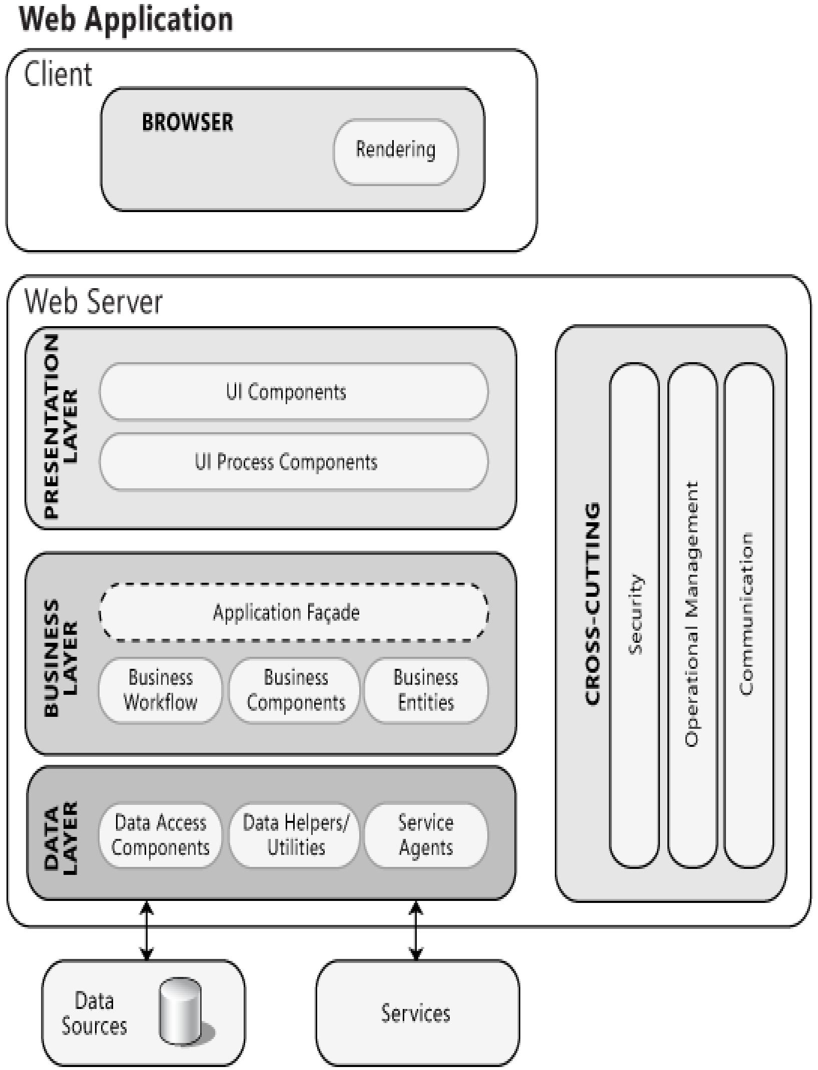
Architecture-Based Software Engineering



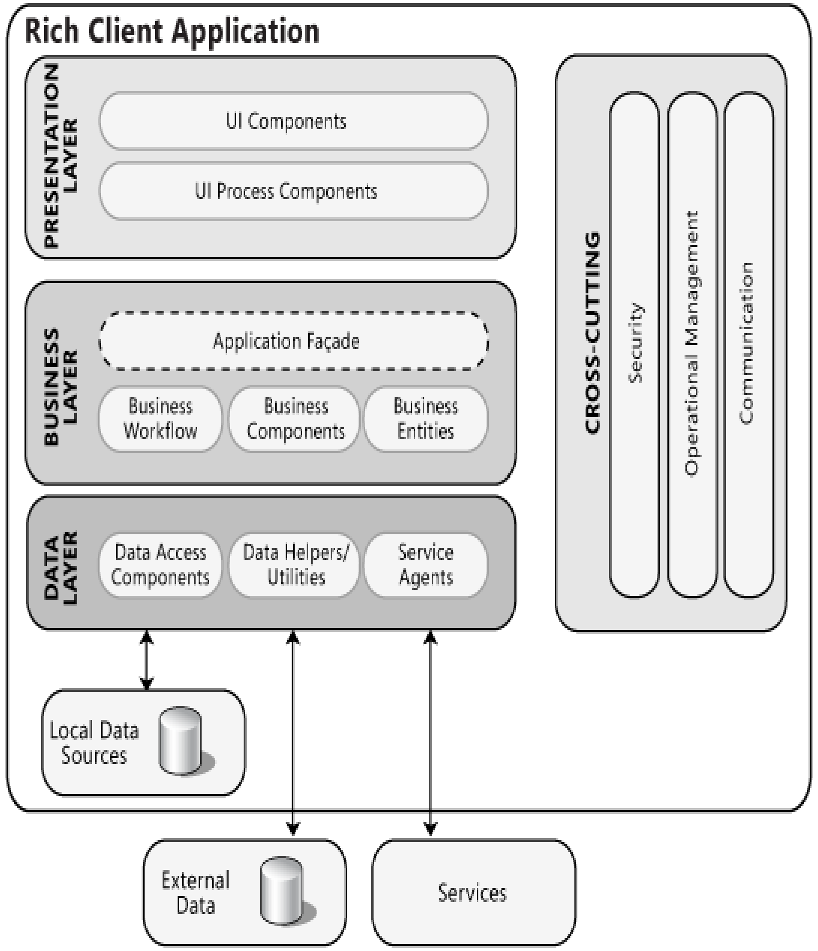
Domain-Specific Software Engineering


Problem Space
Reference architecture
How to implement?
Application-specific architecture
Which domain is it belong to?

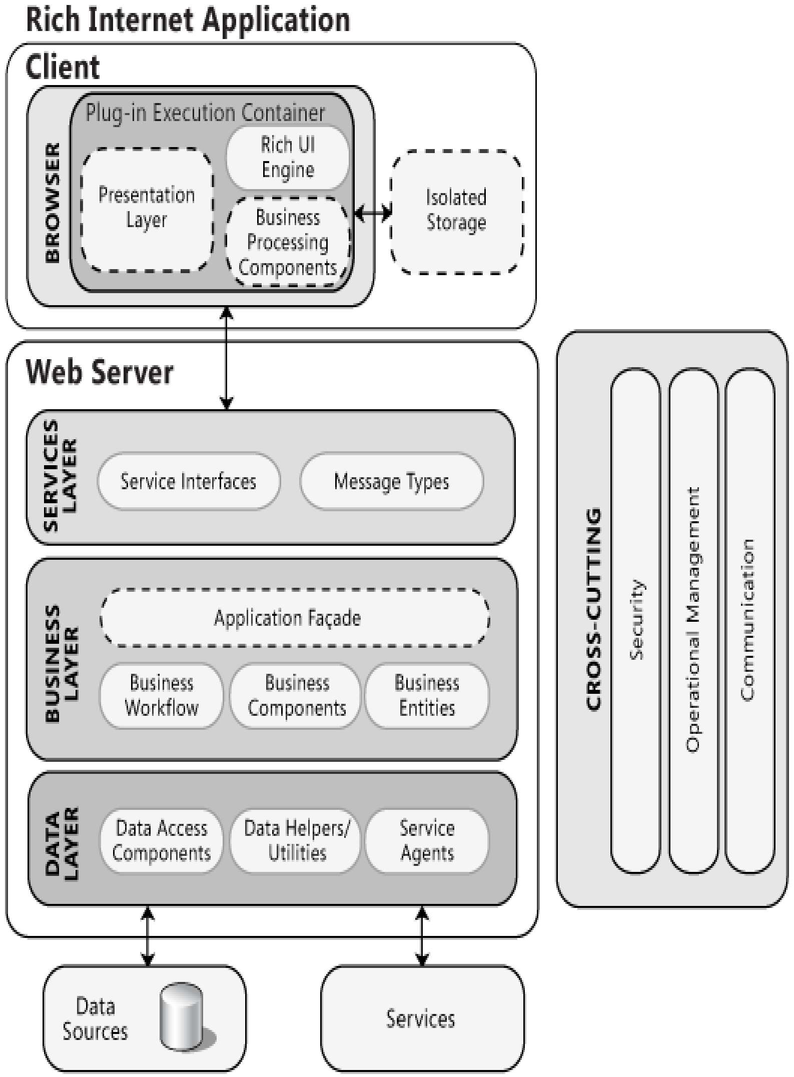
Web Application Architecture



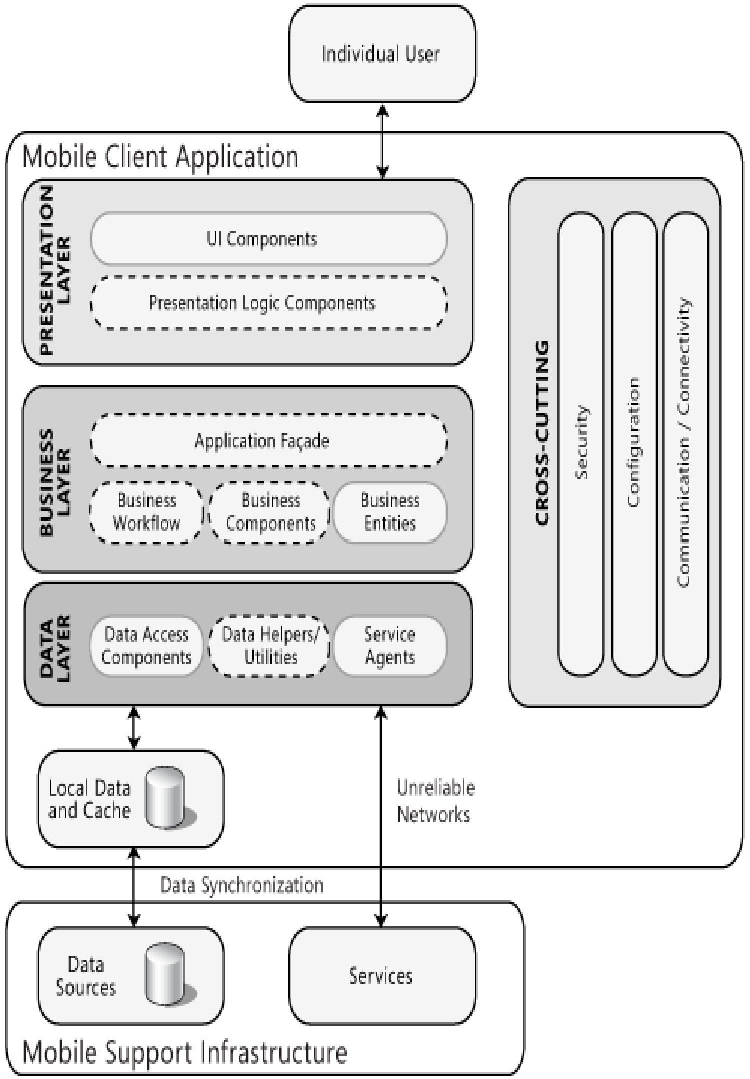
Rich Client Application



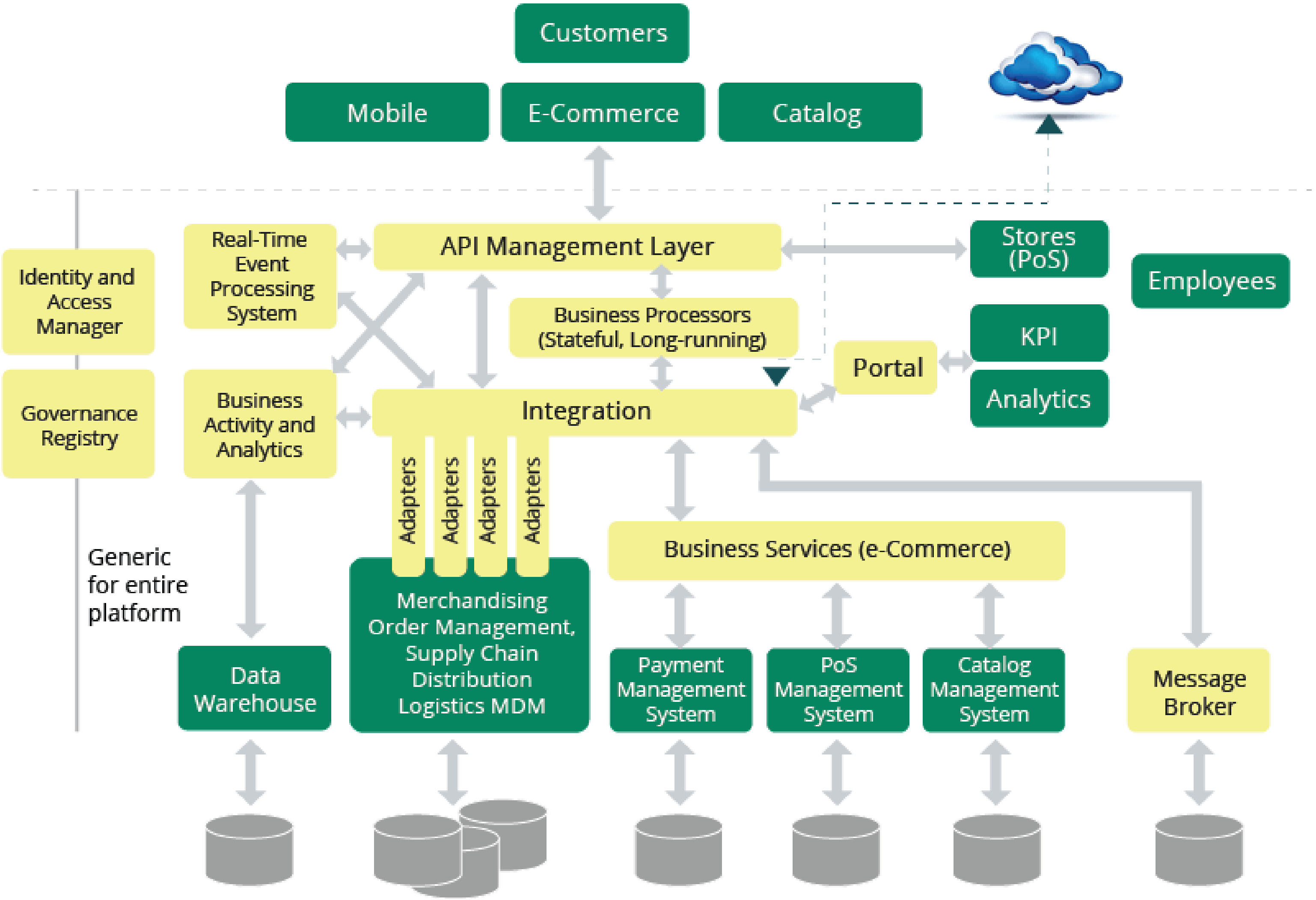
Rich Internet Application



Mobile Apps Architecture

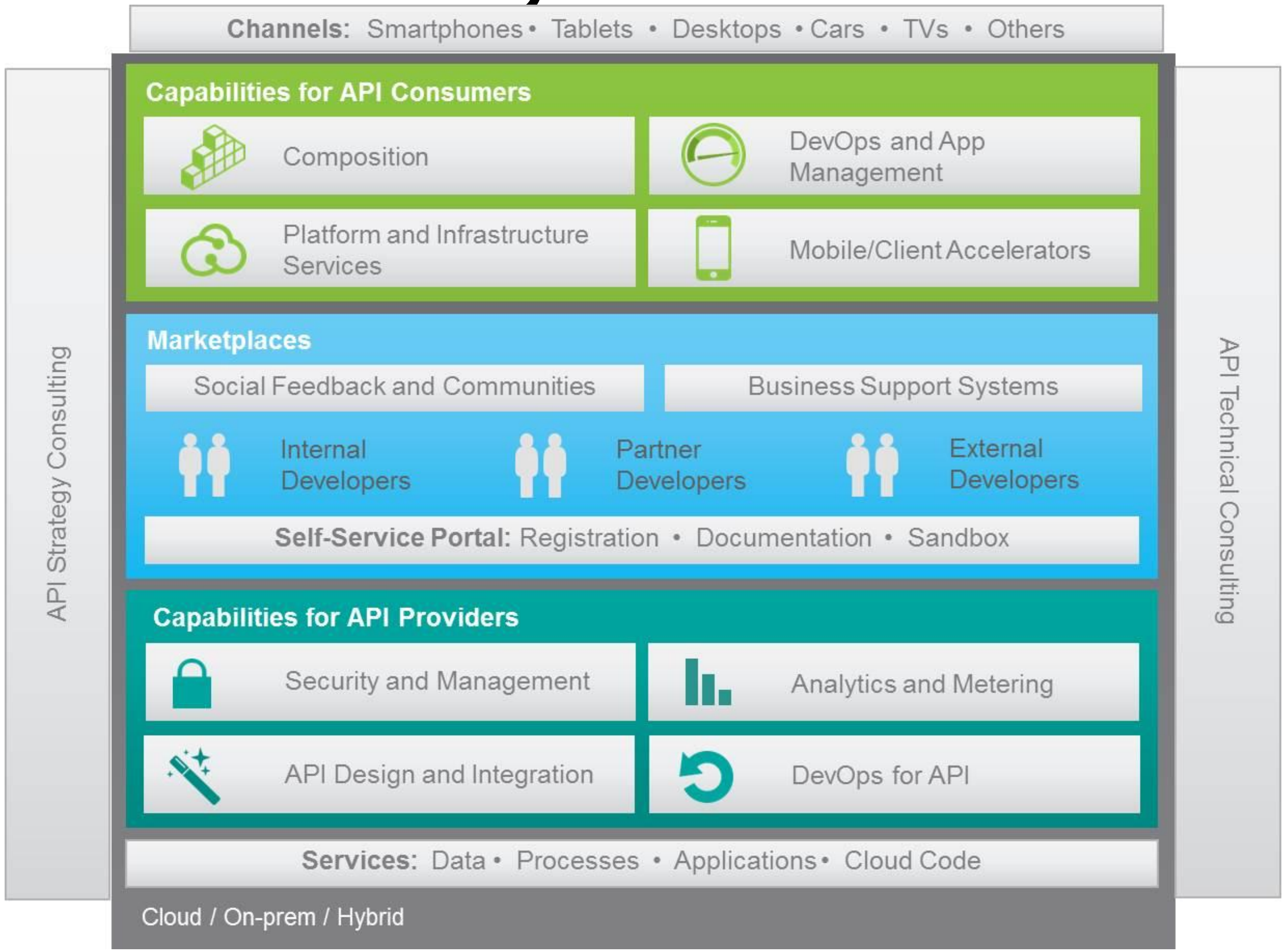


Retail Reference Architecture by WSO2



Customers
Mobile
E-Commerce
Catalog
Identity and Access Manager
Governance Registry
Real-Time Event Processing System
Business Activity and Analytics
API Management Layer
Business Processors (Stateful, Long-running)
Integration
Adapters
Adapters
Adapters
Adapters
Stores (PoS)
Employees
KPI
Analytics
Portal
Business Services (e-Commerce)
Generic for entire platform
Data Warehouse
Merchandising Order Management, Supply Chain Distribution Logistics MDM
Payment Management System
PoS Management System
Catalog Management System
Message Broker

API Management Reference Architecture by IBM



Channels: Smartphones • Tablets • Desktops • Cars • TVs • Others

API Strategy Consulting

Capabilities for API Consumers

- Composition
- DevOps and App Management
- Platform and Infrastructure Services
- Mobile/Client Accelerators

Marketplaces

- Social Feedback and Communities
- Business Support Systems
- Internal Developers
- Partner Developers
- External Developers

Self-Service Portal: Registration • Documentation • Sandbox

Capabilities for API Providers

- Security and Management
- Analytics and Metering
- API Design and Integration
- DevOps for API

Services: Data • Processes • Applications • Cloud Code

Cloud / On-prem / Hybrid

API Technical Consulting

Domain-specific software architecture

DSSA is basically ‘Software Architecture focused on a particular domain.’
- To constraint the problem space
- Facilitate focused development

A DSSA comprises:

- domain model
- reference requirements
- A reference architecture.
- A component library/software elements.
- An application configuration method for selecting and configuring components.
- Externally visible properties of those elements

How DSSA can be processed?

Stage 1. Define the Scope of the Domain

Stage 2. Define/Refine Domain-Specific Concepts/Requirements

Stage 3. Define/Refine Domain-Specific Design and Implementation Constraints

Stage 4. Develop Domain Architectures/Models

Stage 5. Produce/Gather Reusable Workproducts

Stage 1:
Define the Scope of the Domain

Define what can be accomplished

-- emphasis is on the user's needs.

Inputs

1. Experts
2. Existing systems
3. Existing documentation (e.g. textbooks, articles)

Stage 1:
Define the Scope of the Domain

Outputs

1. Block diagram of the domain of interest including inputs and outputs to the domain and high-level relationships between functional units/elements in the domain
2. List of people's names to serve as future references or validation sources
3. List of projects with pointers to documentation and source code
4. List of needs to be met by applications in this domain

Stage 2:
Define/Refine Domain-Specific Concepts/Requirements

Similar to Requirements Analysis

-- emphasis is on the problem space.

Inputs

1. Outputs from Stage 1
2. Selected systems
3. Selected documentation (e.g., textbooks, articles)

Stage 2: Define/Refine Domain-Specific Concepts/Requirements

Outputs

1. Domain Models
   - Scenarios →
   - Domain Dictionary
   - Context Information Diagram
   - Entity/Relationship Diagram
   - Object Diagram
   - Data-Flow Diagram
   - State-Transition Diagram
2. Functional requirements (Reference requirements)

Feature model

Information model

Operational model

Stage 2:
Define/Refine Domain-Specific Concepts/Requirements

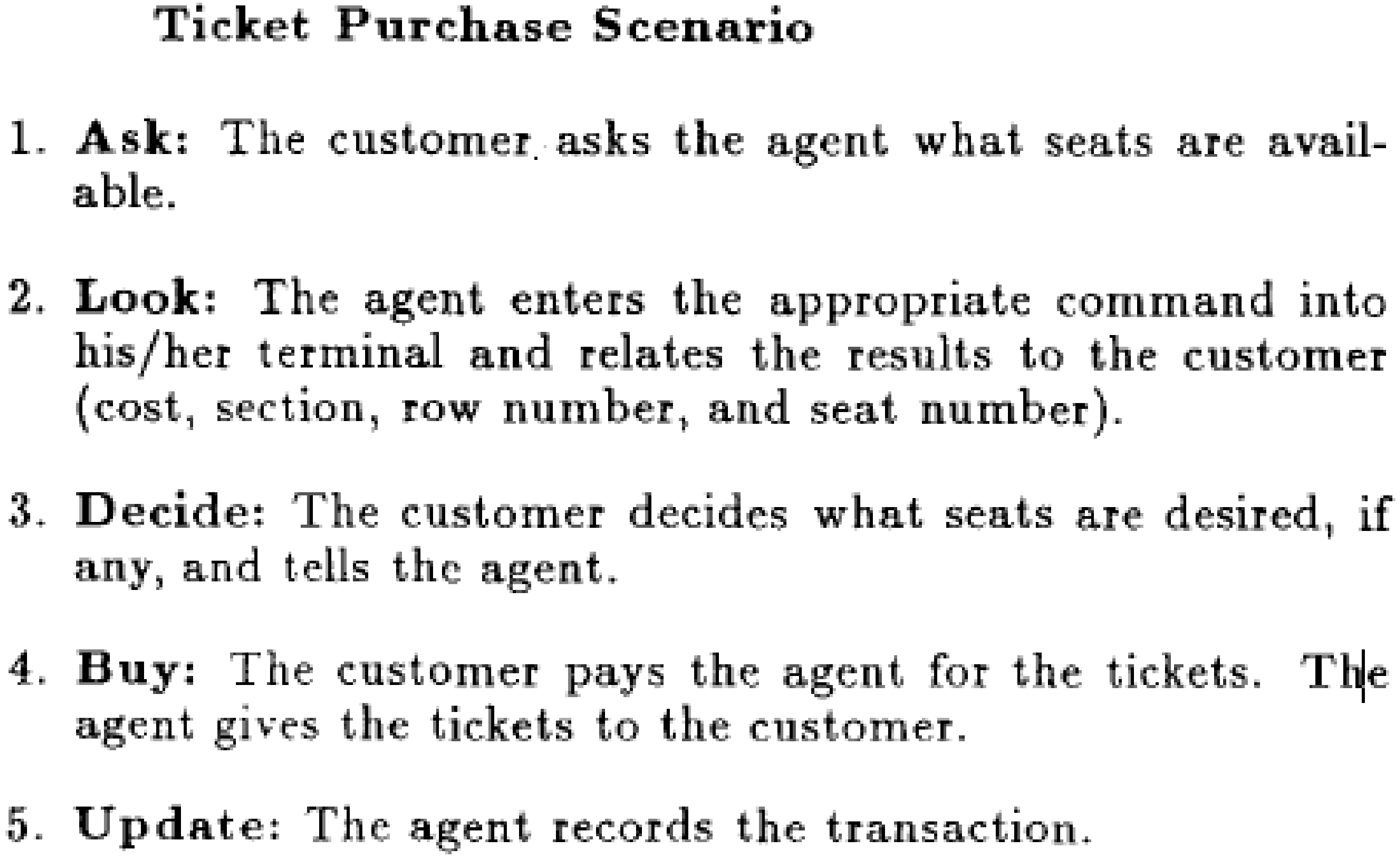
A domain model is a product of context analysis and domain analysis.

Context analysis Define the boundaries of a domain and the relationship of the entities inside the domain to those outside.

Domain analysis Identify, capture, and organize the domain assets.

Scenarios

The scenarios consist of a list of numbered, labeled scenario steps or events followed by a brief description.



Ticket Purchase Scenario

1. **Ask:** The customer asks the agent what seats are available.

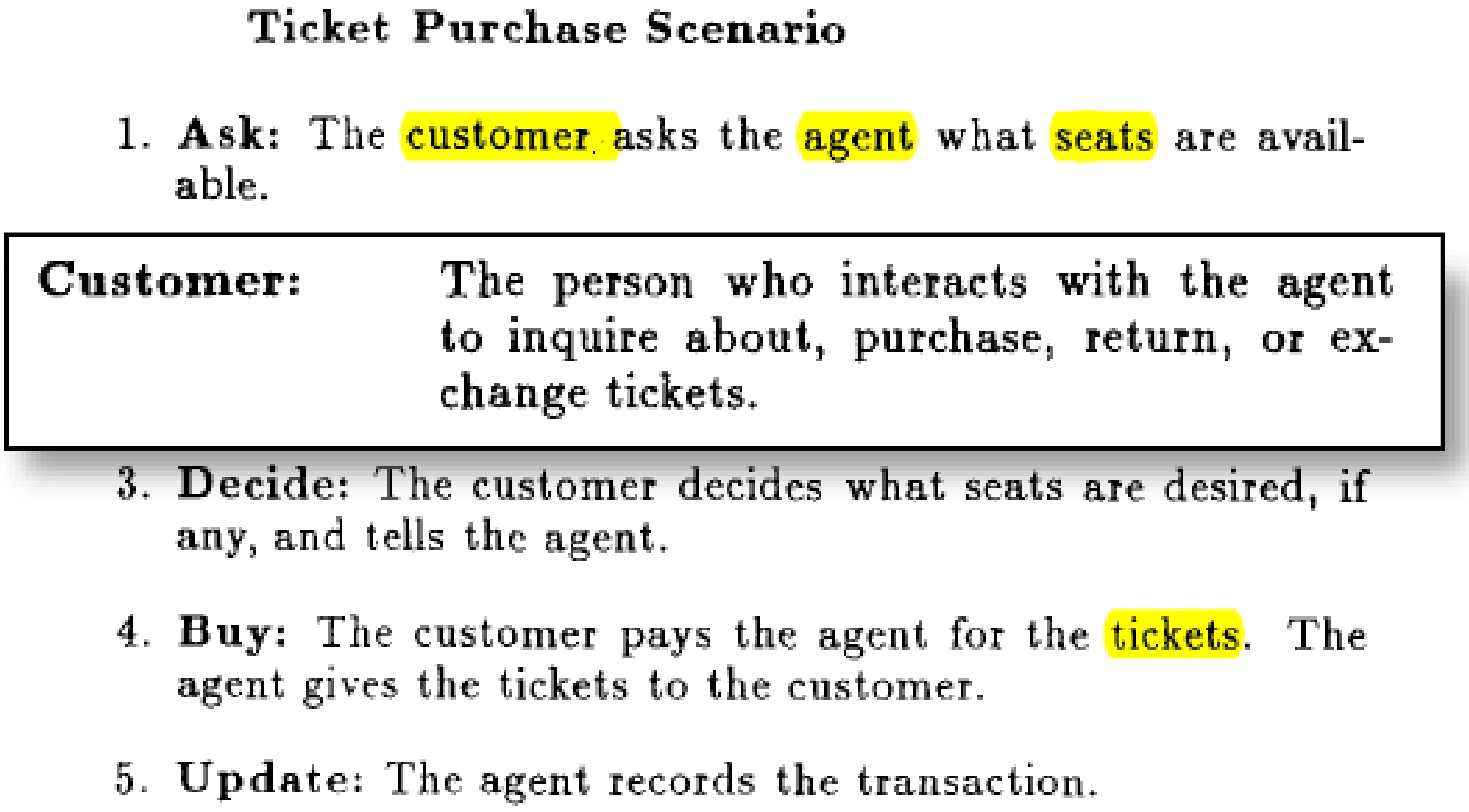
2. **Look:** The agent enters the appropriate command into his/her terminal and relates the results to the customer (cost, section, row number, and seat number).

3. **Decide:** The customer decides what seats are desired, if any, and tells the agent.

4. **Buy:** The customer pays the agent for the tickets. The agent gives the tickets to the customer.

5. **Update:** The agent records the transaction.

Domain Dictionary



Ticket Purchase Scenario

1. **Ask:** The customer asks the agent what seats are available.

Customer: The person who interacts with the agent to inquire about, purchase, return, or exchange tickets.

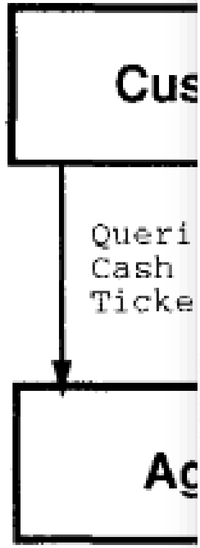
3. **Decide:** The customer decides what seats are desired, if any, and tells the agent.

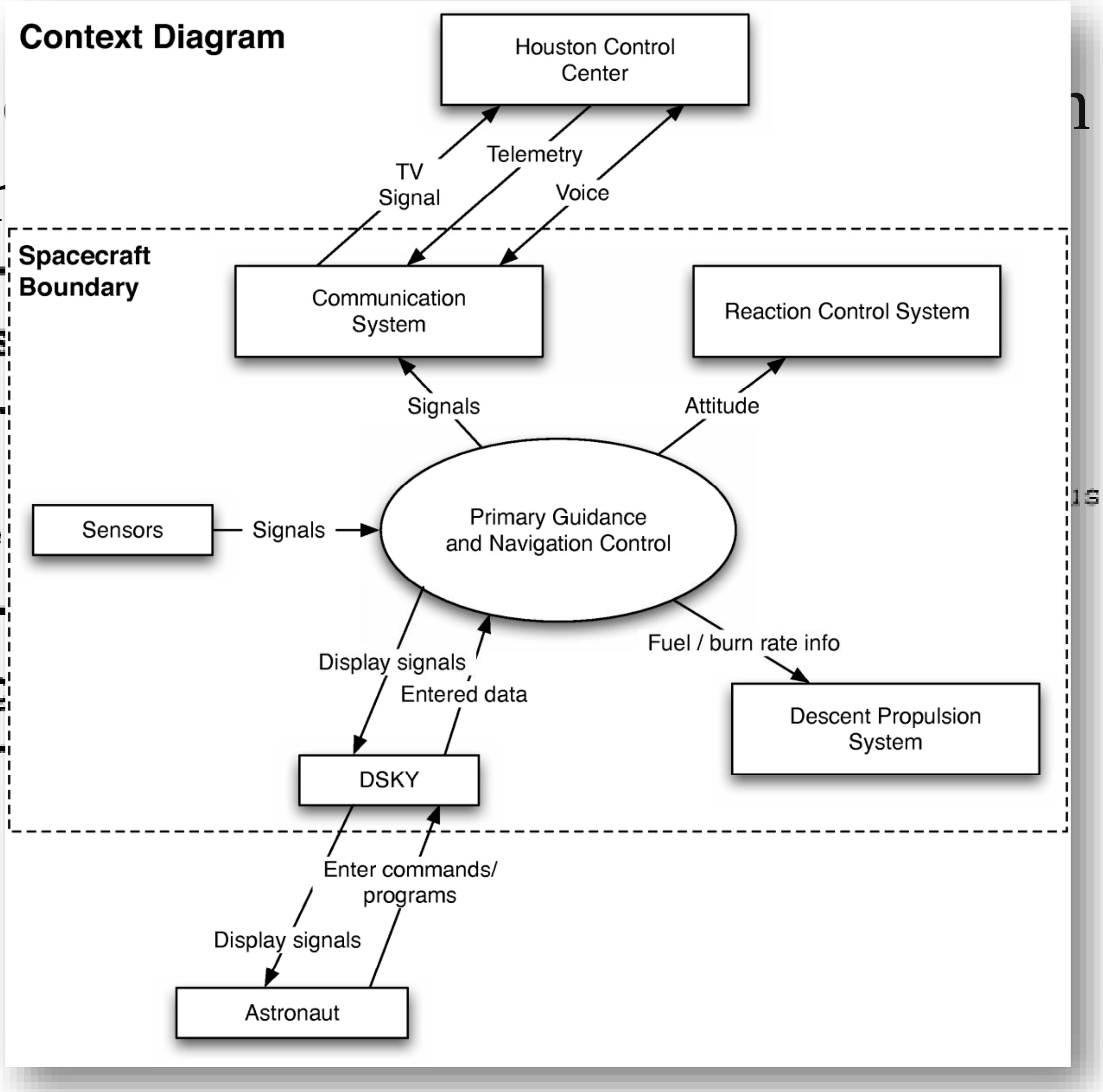
4. **Buy:** The customer pays the agent for the tickets. The agent gives the tickets to the customer.

5. **Update:** The agent records the transaction.

Context Information Diagram

It describ n the
major cor





Entity/Relationship (ER) Diagram

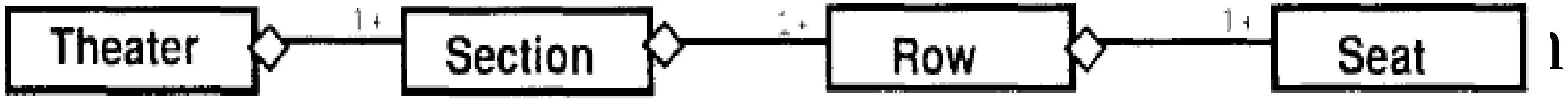


Figure 4: Theater Aggregation Hierarchy

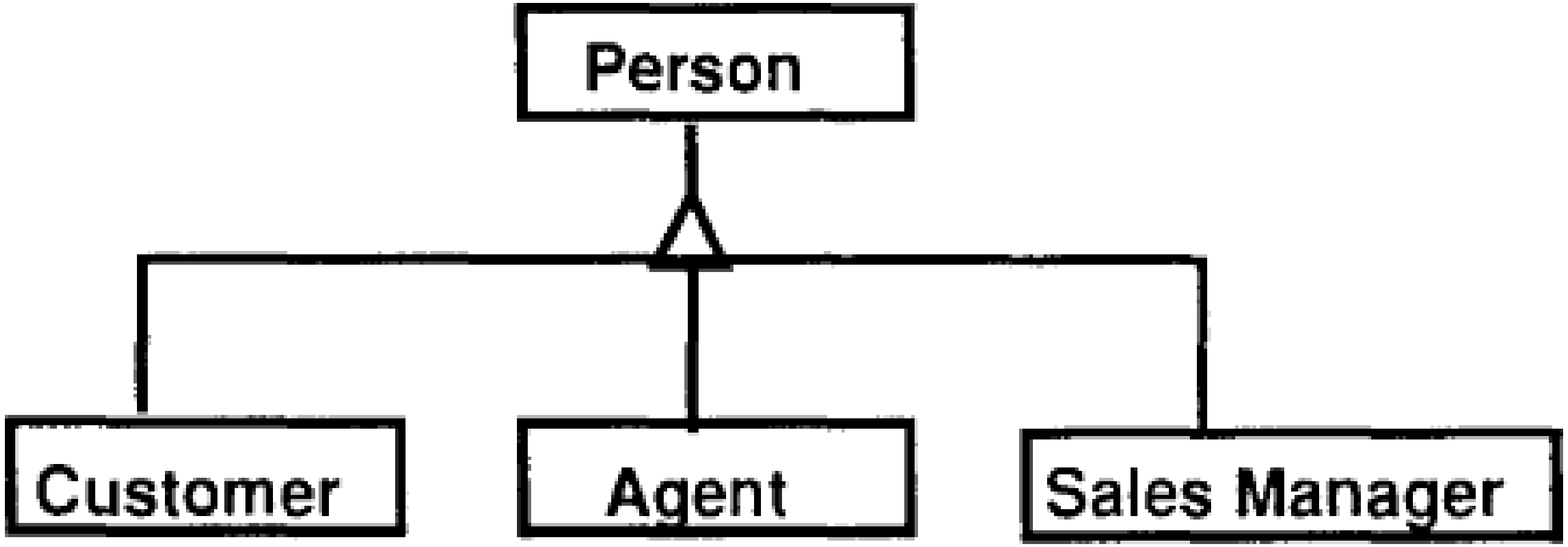
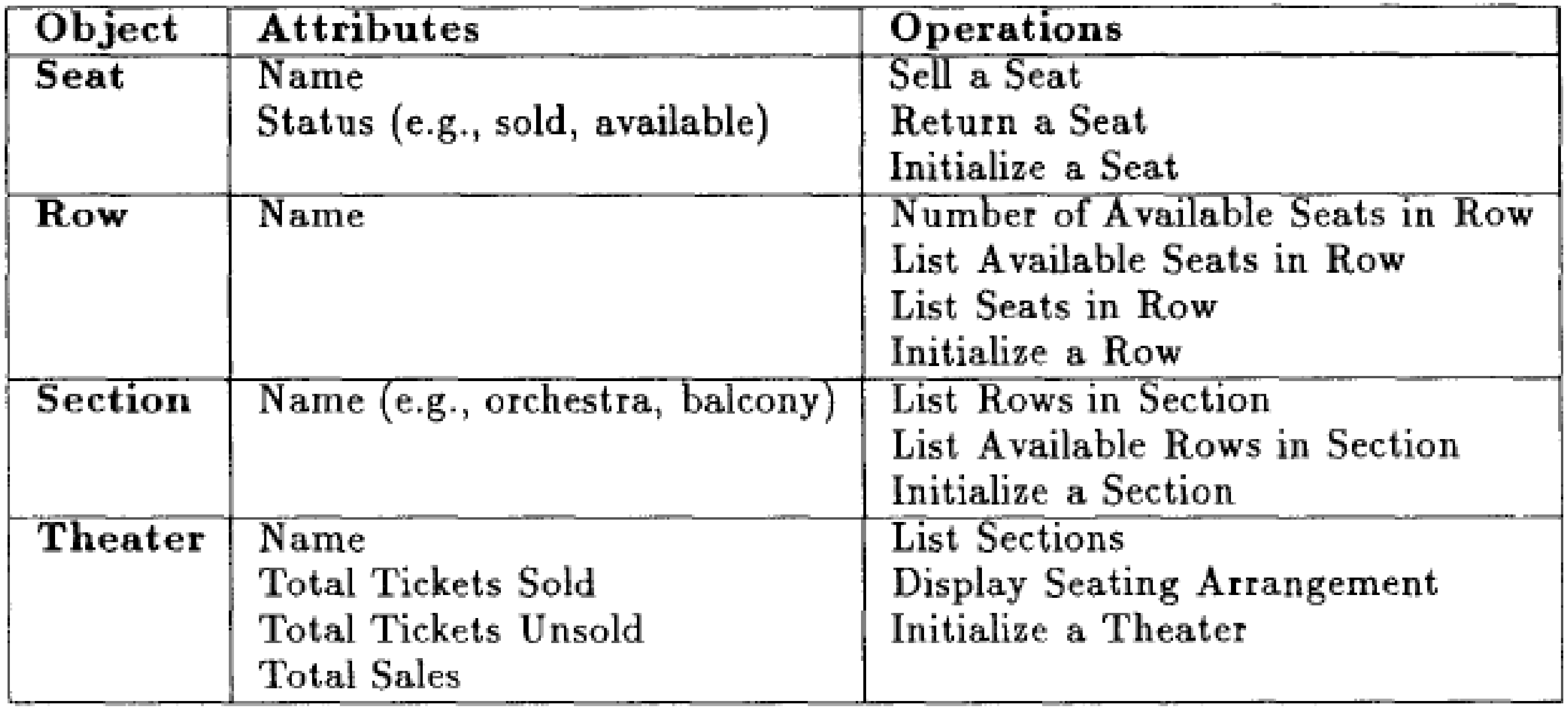


Figure 3: Person Types Taxonomy

Object diagram



Object	Attributes	Operations
Seat	Name Status (e.g., sold, available)	Sell a Seat Return a Seat Initialize a Seat
Row	Name	Number of Available Seats in Row List Available Seats in Row List Seats in Row Initialize a Row
Section	Name (e.g., orchestra, balcony)	List Rows in Section List Available Rows in Section Initialize a Section
Theater	Name Total Tickets Sold Total Tickets Unsold Total Sales	List Sections Display Seating Arrangement Initialize a Theater

Table 1: List of Objects, Operations, and Attributes

Data-Flow Diagram

It focuses on the data exchanged within the system, with no notion of control.

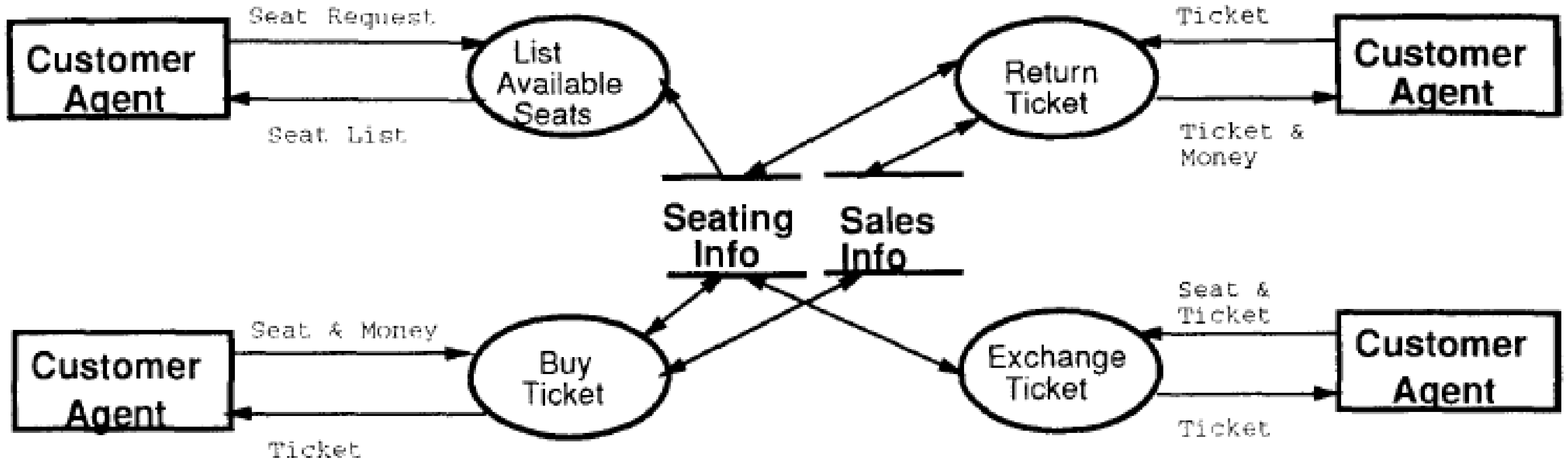


Figure 7: Data Flow Diagrams involving the Customer and Agent

State-Transition Diagram

It describes the events and states that take place in the domain.

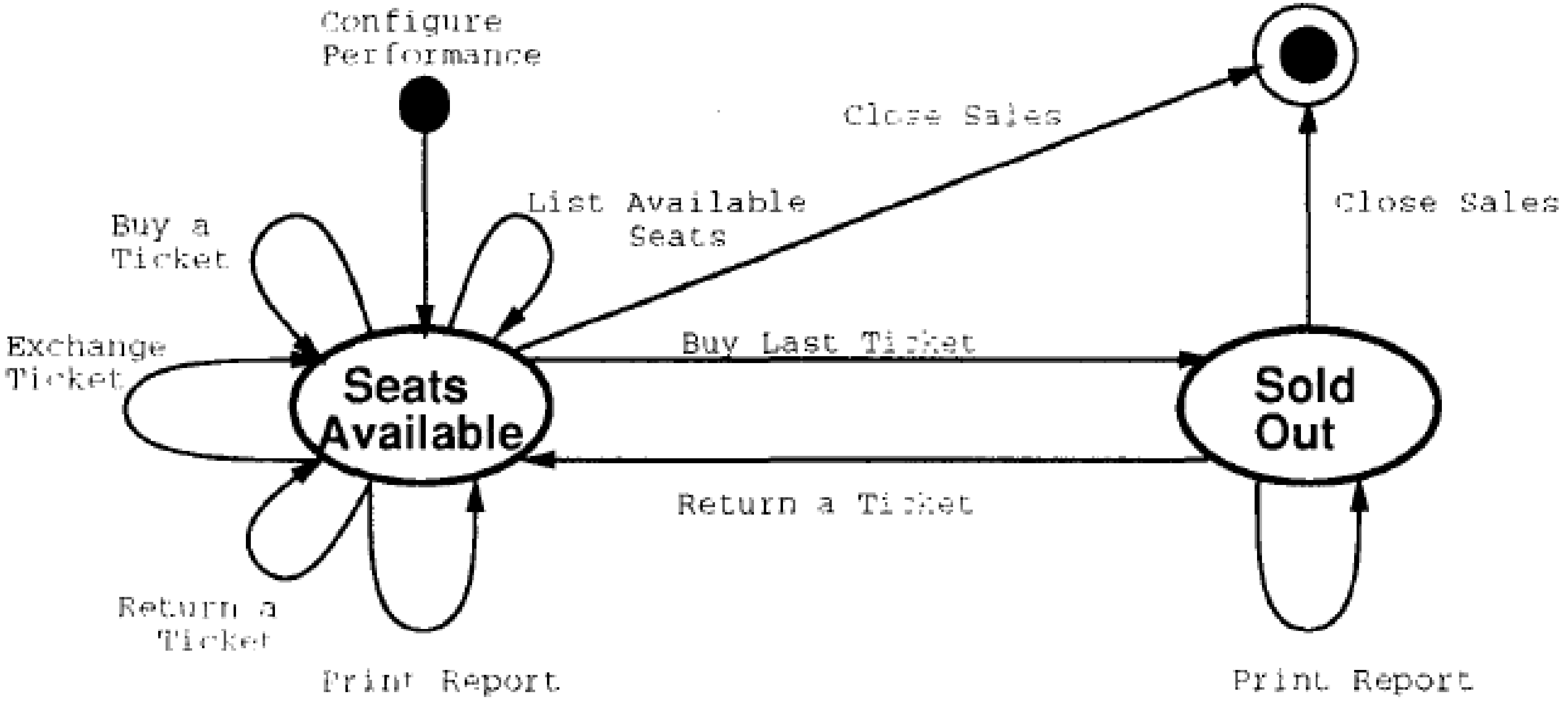


Figure 9: State Transition Diagram

Functional Requirements

It is a p

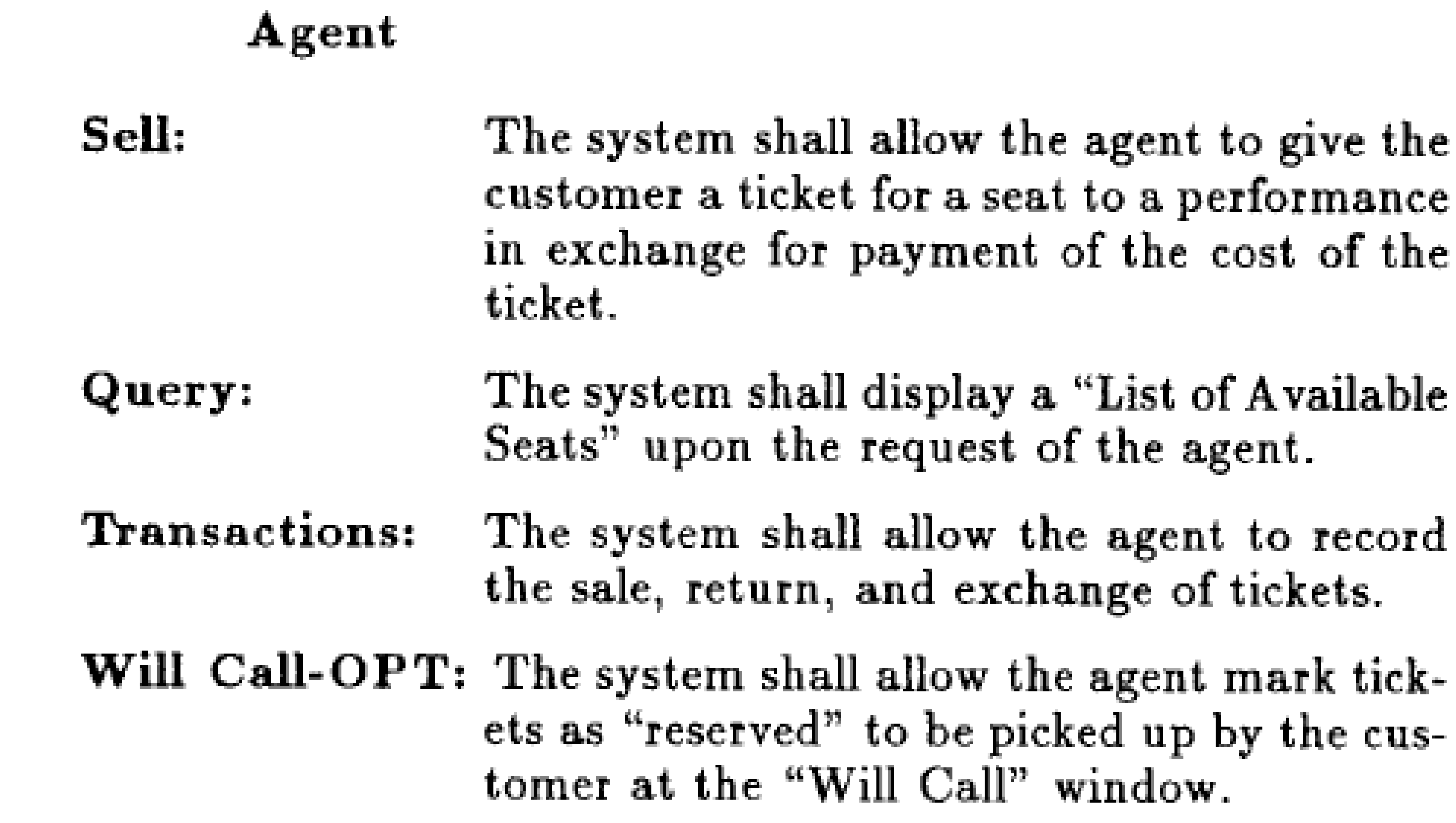
(**Ref** ping of

the r to the

cano

Thre

It defin



Agent

Sell: The system shall allow the agent to give the customer a ticket for a seat to a performance in exchange for payment of the cost of the ticket.

Query: The system shall display a "List of Available Seats" upon the request of the agent.

Transactions: The system shall allow the agent to record the sale, return, and exchange of tickets.

Will Call-OPT: The system shall allow the agent mark tickets as "reserved" to be picked up by the customer at the "Will Call" window.

Stage 3:
Define/Refine Domain-Specific Design and Implementation Constraints

Similar to Requirements Analysis

-- emphasis is on the solution space.

Inputs

1. Outputs from Stage 1, especially the context diagram
2. Outputs from Stage 2, especially control and data flow diagrams, and rationale

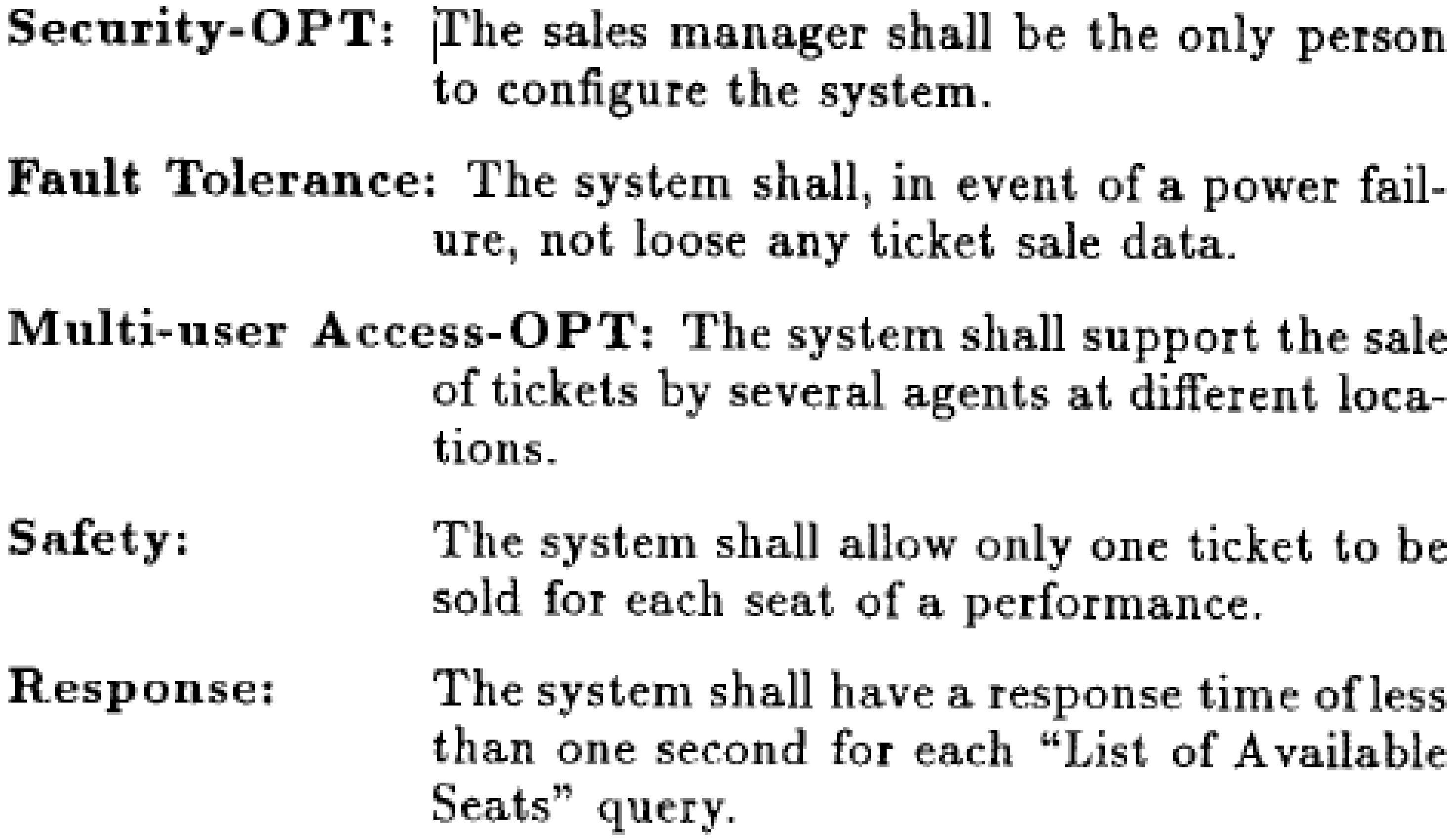
Stage 3:
Define/Refine Domain-Specific Design and Implementation Constraints

Outputs

1. Non-Functional Requirements
2. Design Requirements
3. Implementation Requirements

Non-Functional Requirements

Eg, security, performance, reliability.



Security-OPT: The sales manager shall be the only person to configure the system.

Fault Tolerance: The system shall, in event of a power failure, not loose any ticket sale data.

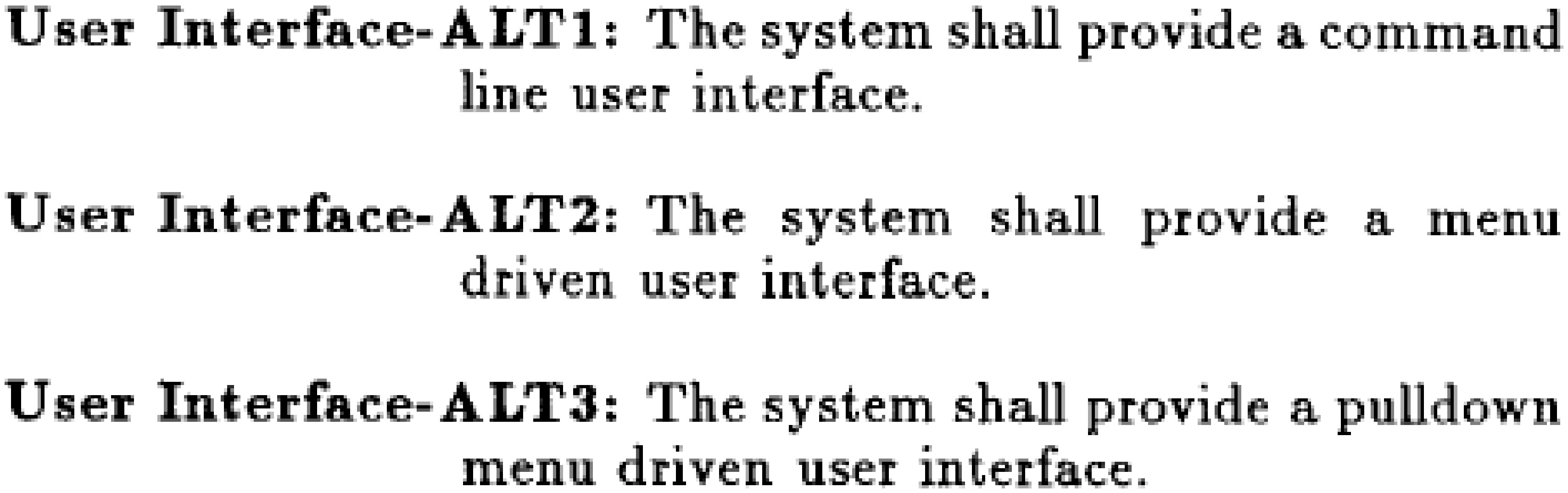
Multi-user Access-OPT: The system shall support the sale of tickets by several agents at different locations.

Safety: The system shall allow only one ticket to be sold for each seat of a performance.

Response: The system shall have a response time of less than one second for each "List of Available Seats" query.

Design Requirements

Eg, architecture style, user interface style.

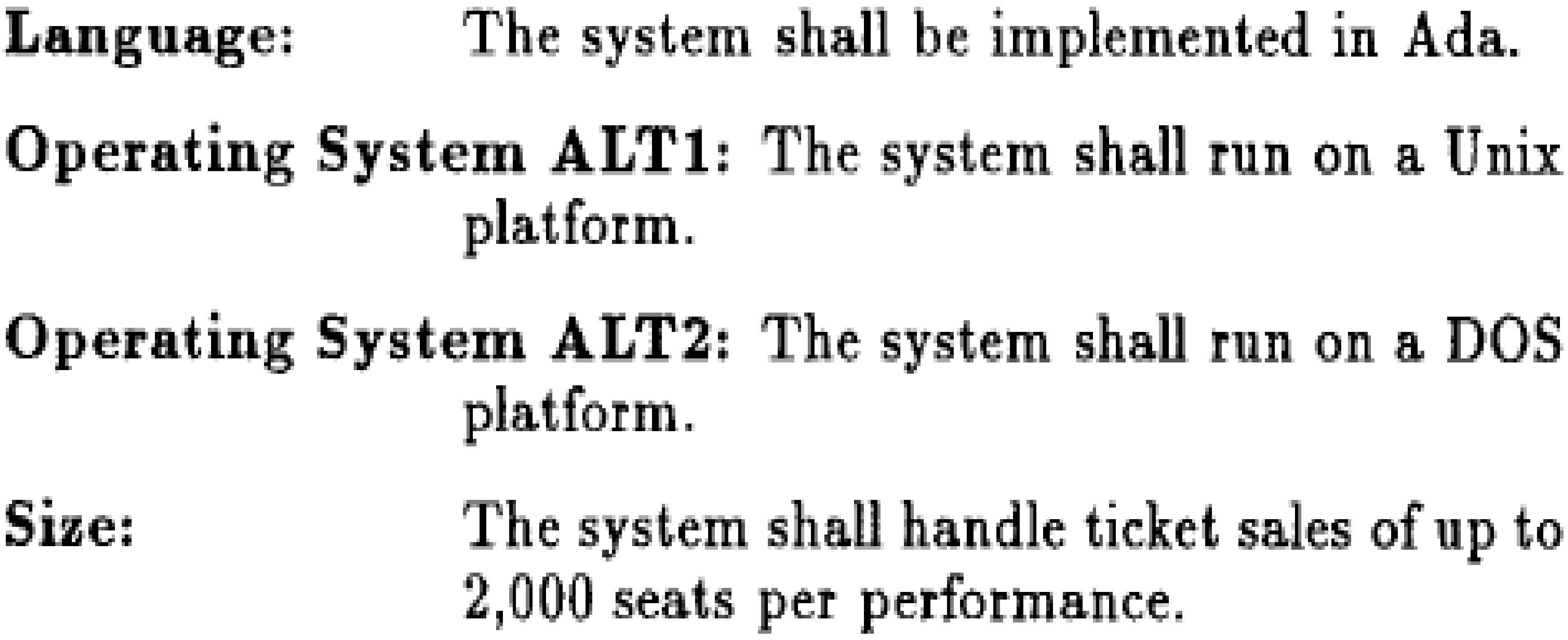


User Interface-ALT1: The system shall provide a command line user interface.

User Interface-ALT2: The system shall provide a menu driven user interface.

User Interface-ALT3: The system shall provide a pulldown menu driven user interface.

Implementation Requirements



Language: The system shall be implemented in Ada.

Operating System ALT1: The system shall run on a Unix platform.

Operating System ALT2: The system shall run on a DOS platform.

Size: The system shall handle ticket sales of up to 2,000 seats per performance.

Stage 4:
Develop Domain Architectures/Models

Similar to High-Level Design

--emphasis is on defining module/model interfaces and semantics.

Input:

The input to this stage consist of the inputs and outputs of the previous stages.

Outputs

A Reference Architecture

Reference Architecture

What is it?

Reference Architecture is the set of principal design decisions that are simultaneously applicable to multiple related systems, typically within an application domain, which explicitly defined points of variation.

When to develop?

Not too-early; not too-late.

Reference Architecture

- Reference Architecture Model
- Configuration Decision Tree
- Architecture Schema/Design Record
- Reference Architecture Dependency Diagram
- Component Interface Descriptions
- Constraints and Rationale

Reference Architecture Model

All designs start out with some simple abstraction based on the architecture style.

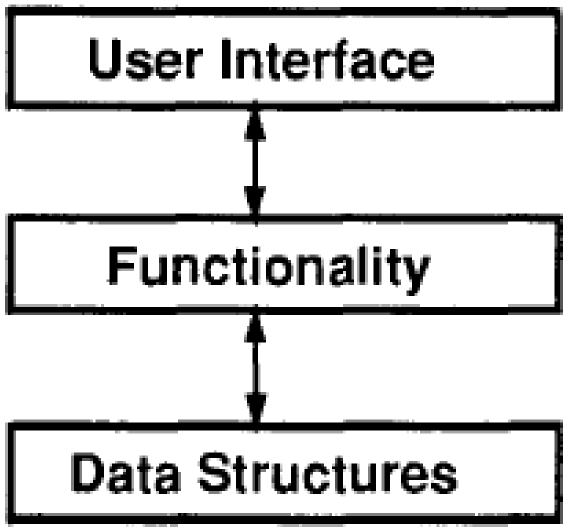
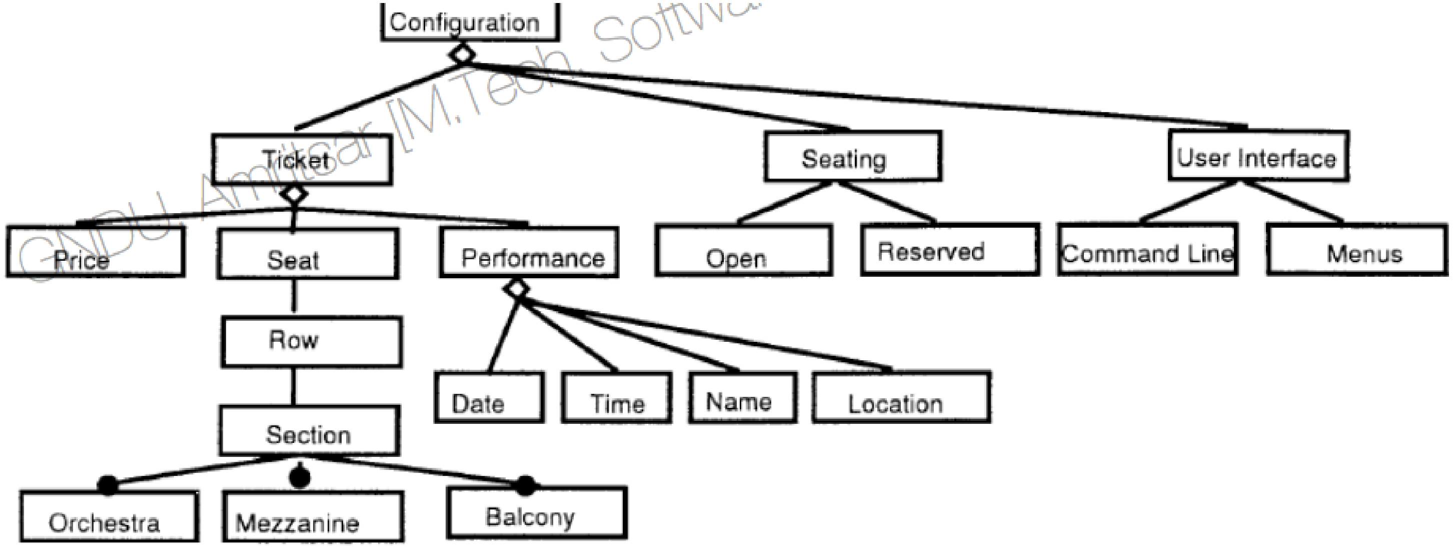


Figure 11: Simple "Layered" Reference Architecture Model

Configuration Decision Tree

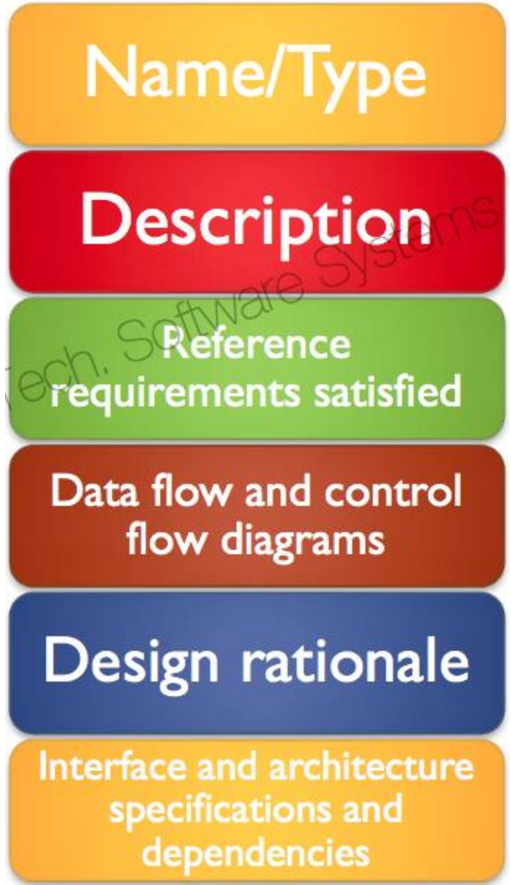
A subset of reference requirements is chosen and a configuration decision tree is made accordingly.

Configuration is done at reference architecture instantiation time.



Architecture Schema

It is a collection point for knowledge about the components that make up a DSSA.



Name/Type

Description

Reference requirements satisfied

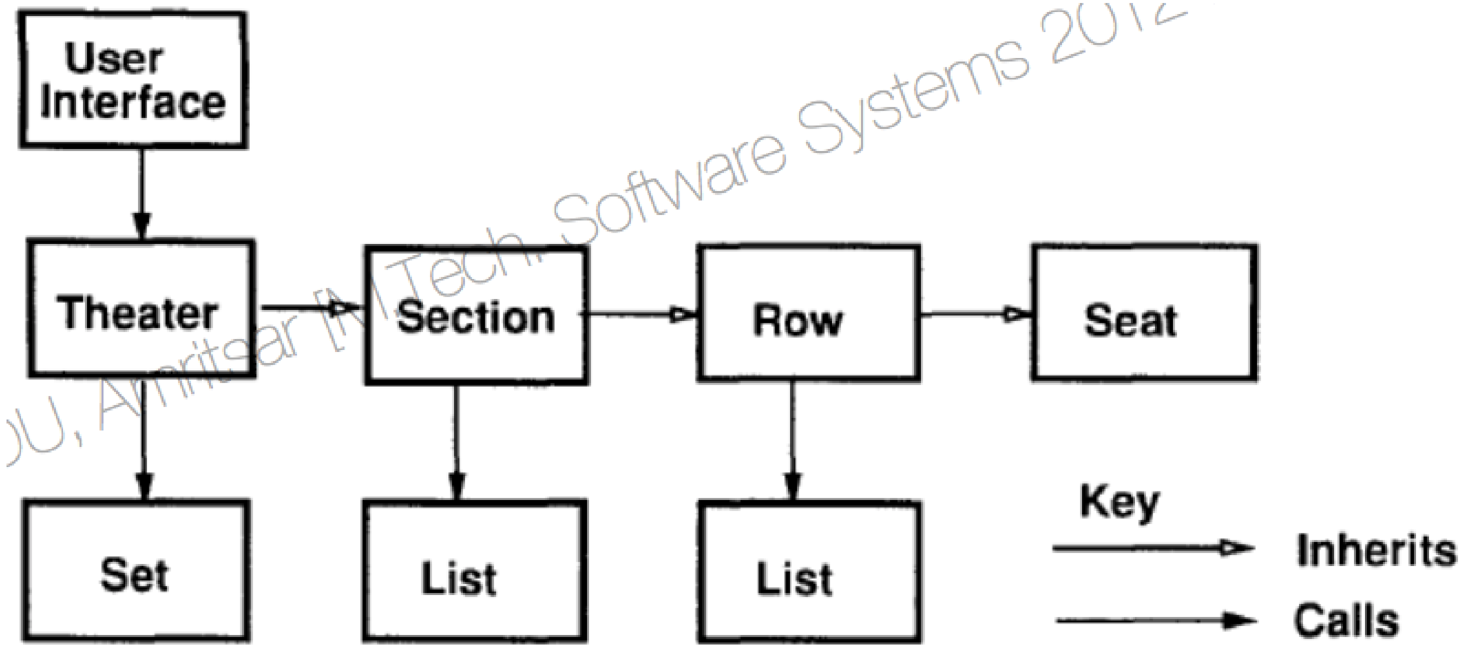
Data flow and control flow diagrams

Design rationale

Interface and architecture specifications and dependencies

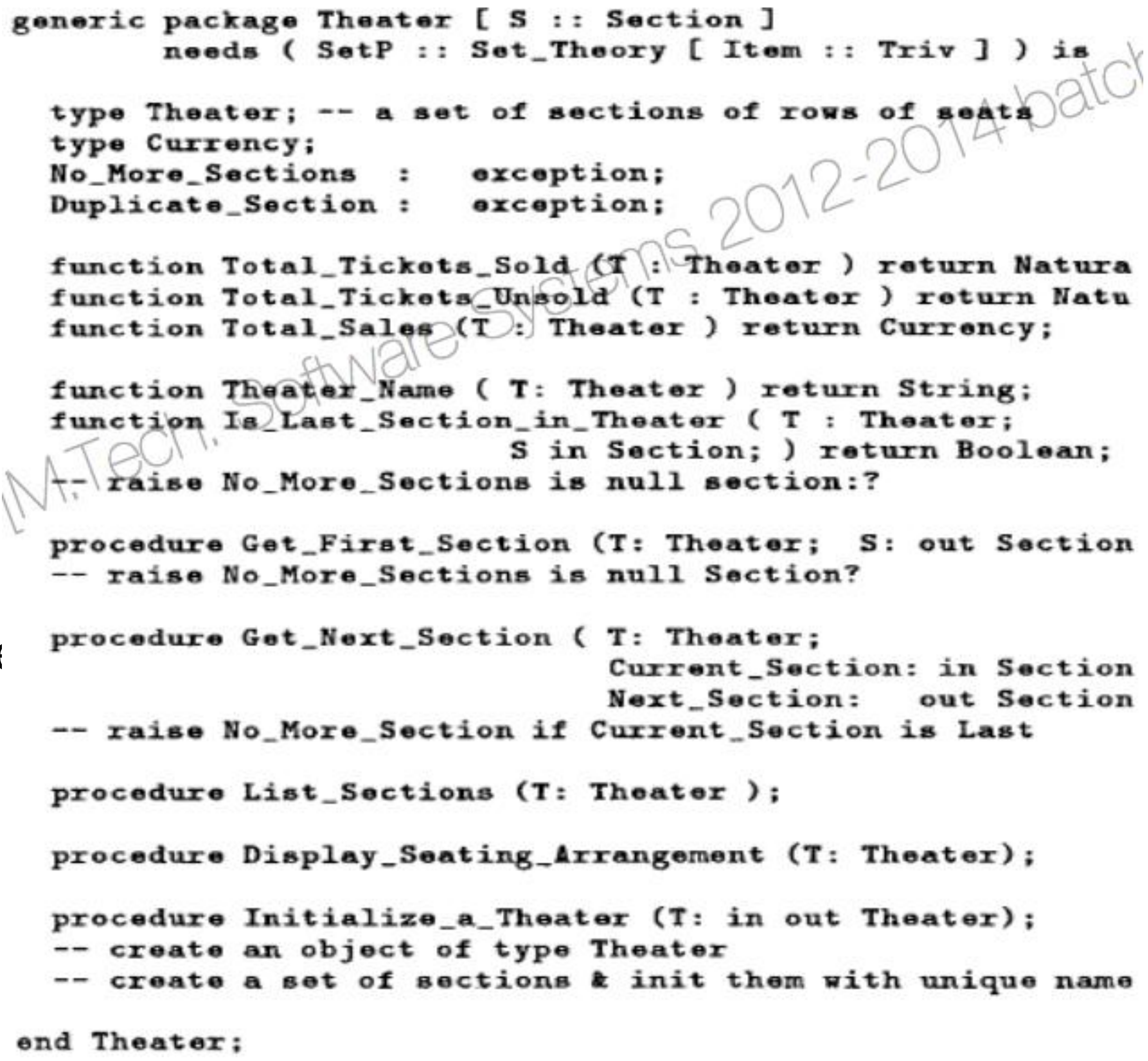
Dependency Diagram

The reference architecture dependency diagram reveals component connections at a level of granularity reflecting the architectural style chosen by the system architect



Component Interface Description

The focus is on, how elements interact with their environments, not on how elements are implemented. An Interface Description Language(IDL) is used to describe the interface as per the syntax of the language chosen.



```
generic package Theater [ S :: Section ]
        needs ( SetP :: Set_Theory [ Item :: Triv ] ) is

  type Theater; -- a set of sections of rows of seats
  type Currency;
  No_More_Sections  :   exception;
  Duplicate_Section :   exception;

  function Total_Tickets_Sold (T : Theater ) return Natura
  function Total_Tickets_Unsold (T : Theater ) return Natu
  function Total_Sales (T : Theater ) return Currency;

  function Theater_Name ( T: Theater ) return String;
  function Is_Last_Section_in_Theater ( T : Theater;
                            S in Section; ) return Boolean;
  -- raise No_More_Sections is null section:?

  procedure Get_First_Section (T: Theater;  S: out Section
  -- raise No_More_Sections is null Section?

  procedure Get_Next_Section ( T: Theater;
                               Current_Section: in Section
                               Next_Section:   out Section
  -- raise No_More_Section if Current_Section is Last

  procedure List_Sections (T: Theater );

  procedure Display_Seating_Arrangement (T: Theater);

  procedure Initialize_a_Theater (T: in out Theater);
  -- create an object of type Theater
  -- create a set of sections & init them with unique name

end Theater;
```

Constraints

Constraints are the **ranges** of parameter values, **relationships** between parameter values or components etc. which have to be considered throughout the development of a system.

Stage 5:
Produce/Gather Reusable Workproducts

Eg, Implementation/collection of reusable artifacts (e.g., code, documentation, etc.).

Input

The interface specifications generated in Stage 4 and related artifacts from existing systems are the primary inputs to this stage.

Output

1. Reusable components and associated test cases and documentation
2. Cross reference of components to requirements, constraints, and architecture

Main Advantages of DSSA

The overall cost is minimized as the assets can be reused.

The market share of the organization can be increased by developing related applications for different users.

Reference

Taylor , R.N; Medvidovic , N.; Dashofy , E.M.; , “Software Architecture: Foundations, Theory, and Practice,” Wiley, 2009.

Will Tracz, “DSSA (Domain-Specific Software Architecture) Pedagogical Example”, Software Engineering Notes vol 20, no 3, Page 49-63, July 1995.

Will Tracz, “Domain-Specific Software Architecture (DSSA) Frequently Asked Questions (FAQ)”, Software Engineering Notes vol 19, no 2, Page 52-57, Apr 1994.

Will Tracz, Lou Coglianese, “A Domain-Specific Software Architecture Engineering Process Outline”, Software Engineering Notes vol 18, no 2, Page 40-50, Apr 1993.