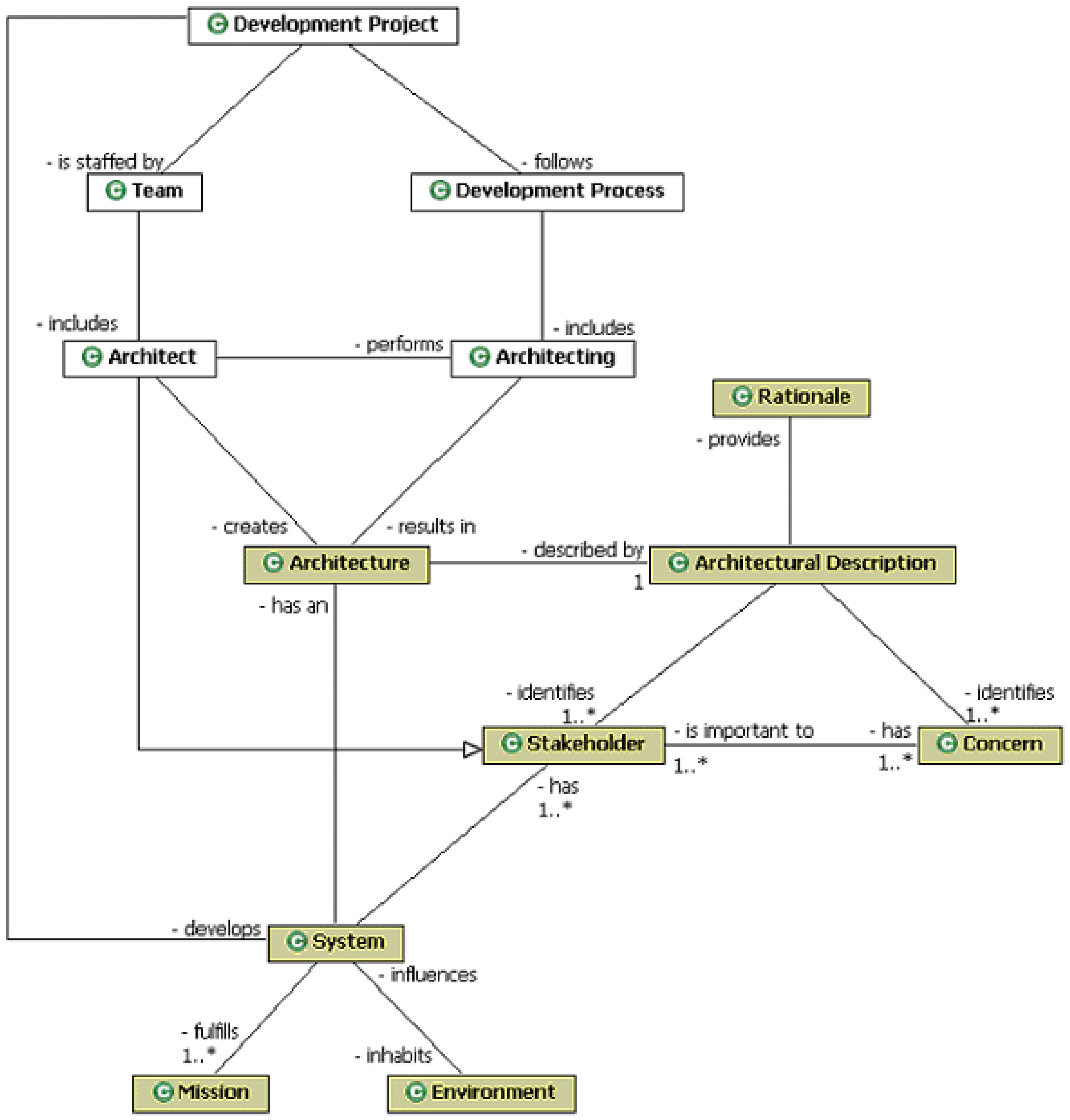
SE 4352
Software Architecture and Design

Fall 2018

Module 6

Architecting Process



In Class Hands On Activity

- Teams of 3 to 4
- What natural language statements are represented by the previous diagram
- Assign a spokesperson

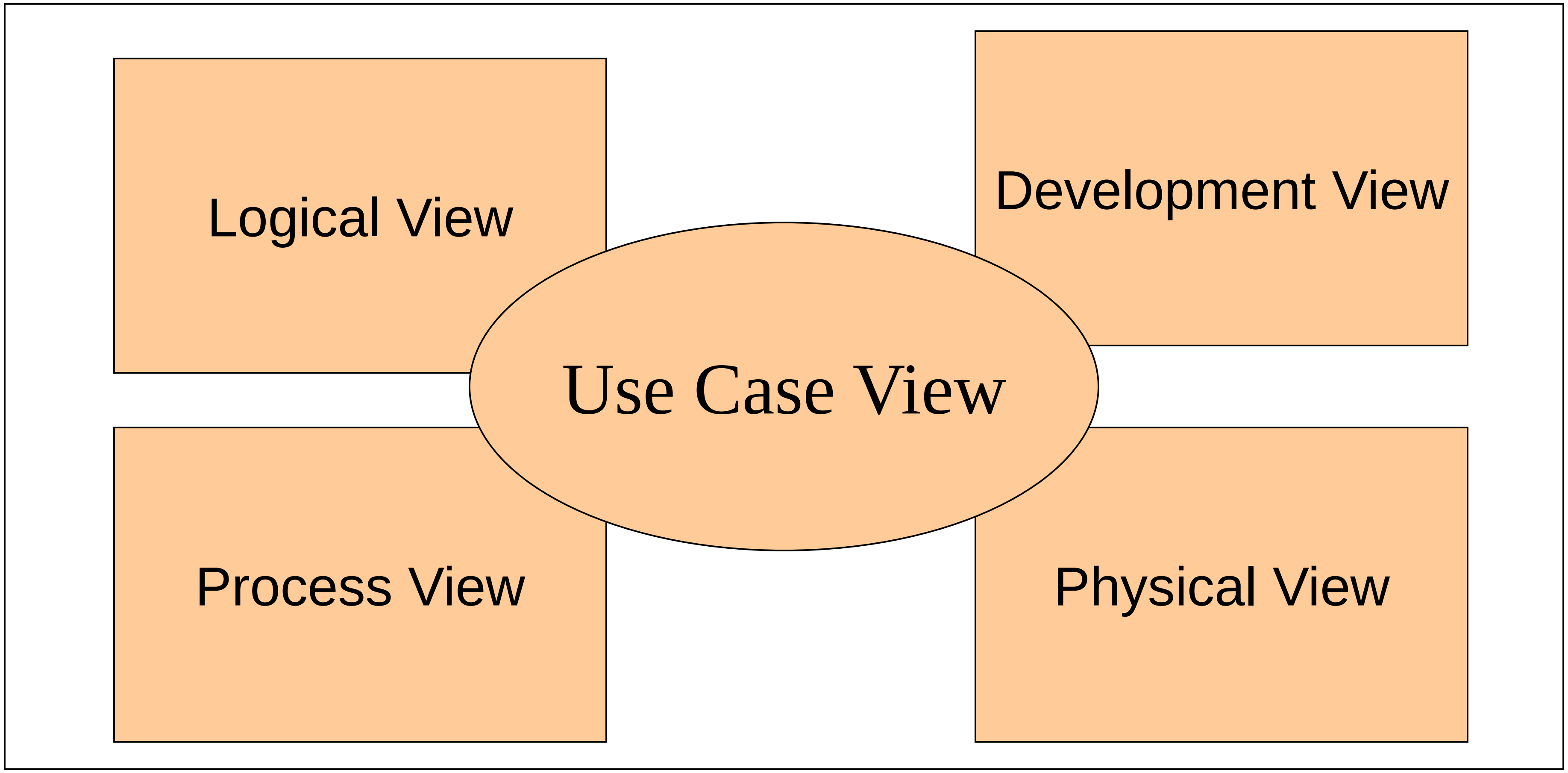
Documenting Software Architecture

- How to communicate software architecture in written format
- Questions to ask
  - Who is the reader?
  - What are they trying to understand?
  - What is important to the reader?
  - What level of detail do they need?
- Key is to have views for different stakeholders showing how goals will be accomplished
- Bridge the gap between white-boarding and downstream needs
- Create tangible artifacts

Good architecture artifacts facilitate:

- Platform for educating new team members about the solution
- Verification & Validation of solution before build
- Explanation of how solution meets business and engineering goals
- Various views specific to the problem domain
- Ability to focus on the view needed by an individual
- Necessary details for downstream activities

The "4+1" View of Software Architecture



The 4+1 view

- Use Case view
  - Understandability
- Logical View (Static Design Model)
  - Functionality
- Process View (Dynamic)
  - Performance
  - Scalable
  - Throughput
- Physical View (HW to SW mapping)
  - Software management
  - System topology
  - Delivery
  - Installation
- Development View (Static organization of development environment)

Recommended Basic Architecture Artifacts

- Business context
- System context
- Architecture overview
- Functional architecture
- Operational architecture
- Architecture decisions

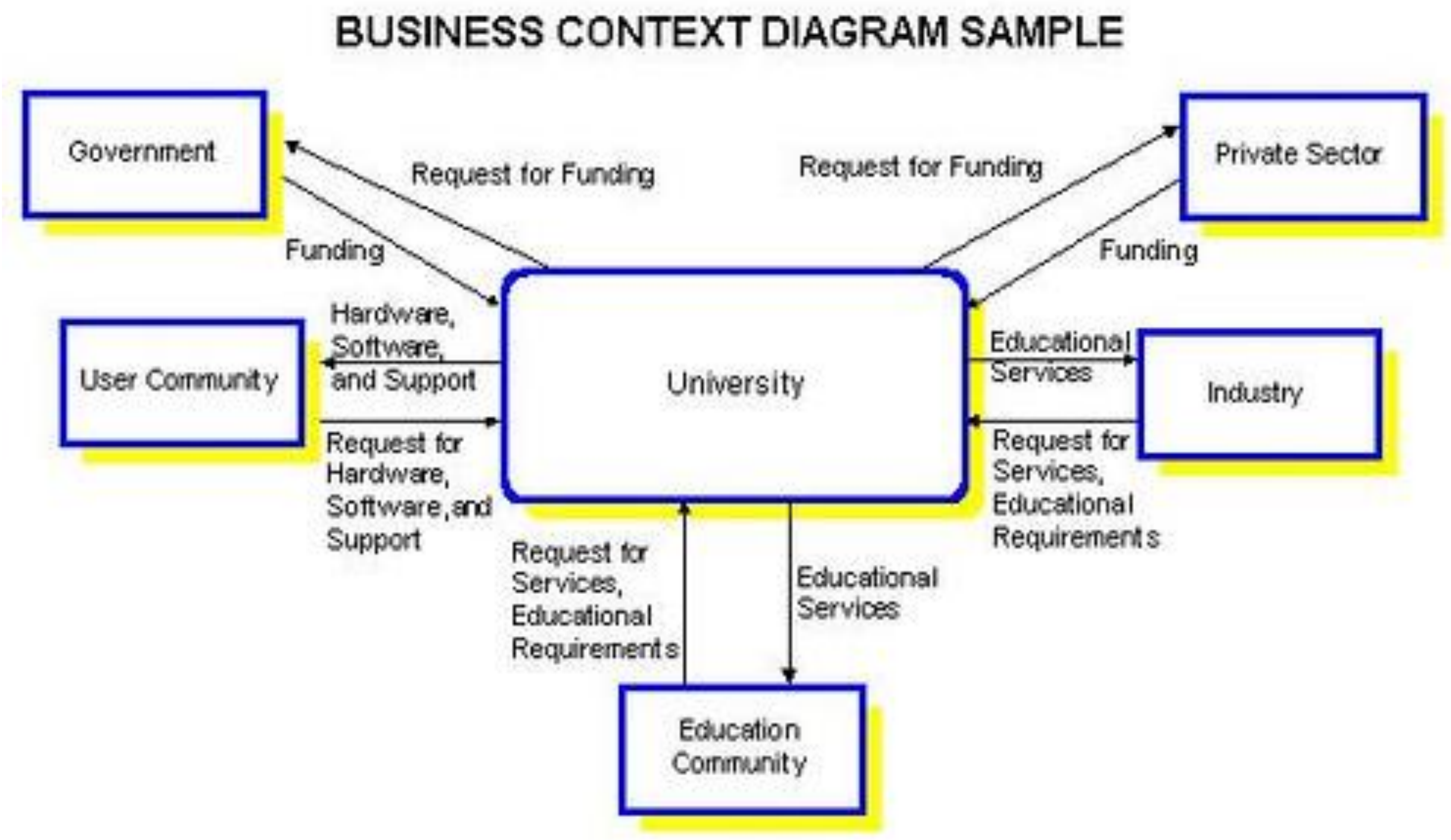
Business Context

- Organizational view of how system interacts with other enterprises to depict the business ecosystem where software will reside
- Particuarly important in systems with a high dependency on external organizations
- High level doesn’t differentiate between roles and users but depicts them as a community that interacts with the business

Example Business Context for a University

- University is central entity
- Dependencies on
  - Government to request funding
  - Government for regulatory conformance
  - IT industry to request research project and educational resources
  - User community needing hardware and software support
  - Other universities in consortium

Example Business Context Diagram


BUSINESS CONTEXT DIAGRAM SAMPLE
Government
Request for Funding
Funding
Request for Funding
Private Sector
Funding
User Community
Hardware, Software, and Support
Request for Hardware, Software, and Support
University
Educational Services
Industry
Request for Services, Educational Requirements
Request for Services, Educational Requirements
Educational Services
Education Community

System context

- Documents how the entire system interacts with external entities
- System shown as black box
- External entities include systems and end users
- Define information and control flows between system and external entities
- Used to clarify, confirm, and document the environment in which the system operates
- Nature of external systems including interfaces and control flows helps downstream specification of technical artifacts

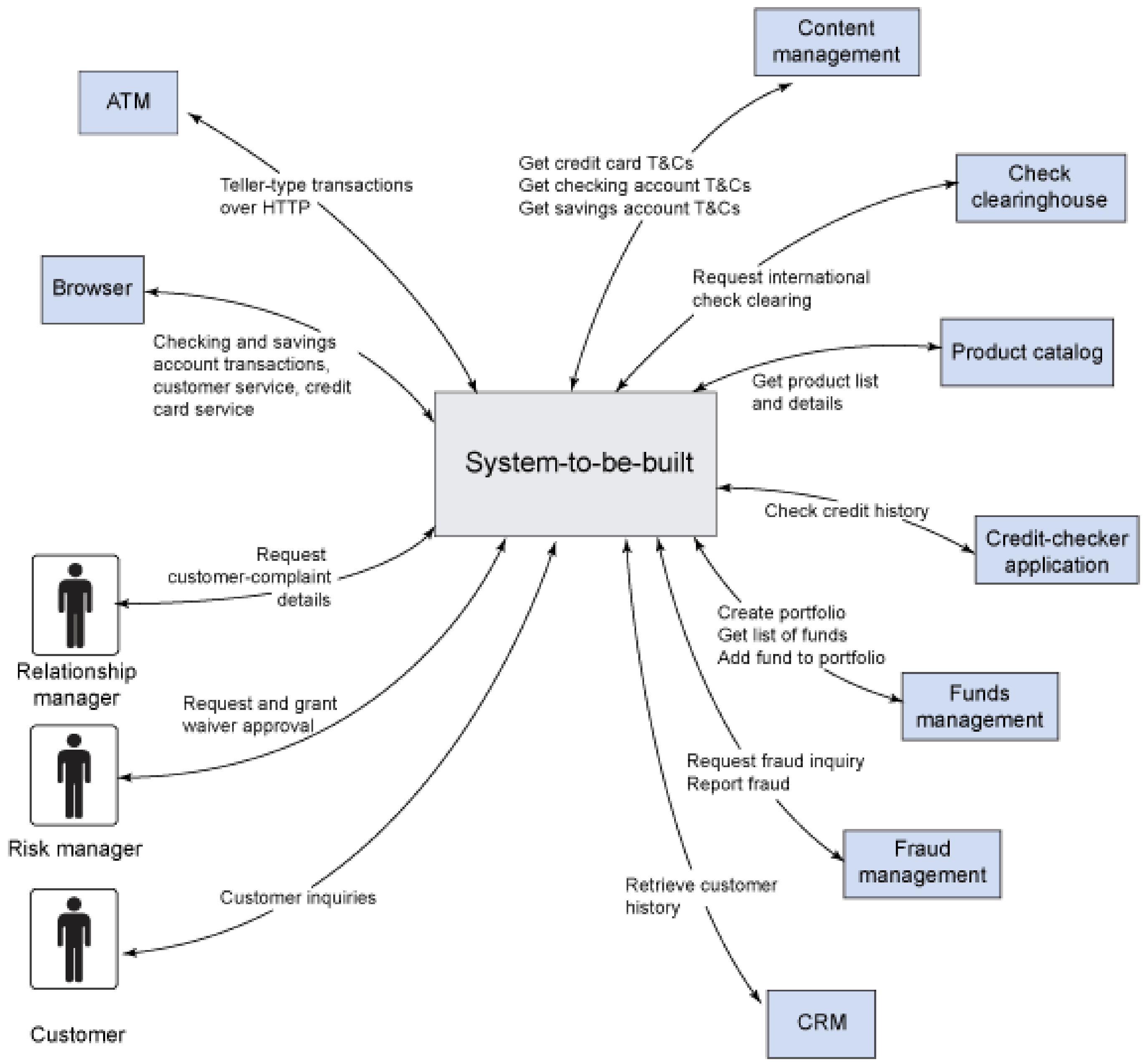
System Context

- Uses business context to identify external organizations
- Identify specific external IT systems and applications for sending and receiving information
  - For each external entity
  - Collectively create system level view
- Shows which external systems are in scope
- Provides a decomposition of business context and provides traceability to business context information

System Context More Details

- Usually developed at two levels
  - Outward facing diagram where software or application represented by black box
  - Software or application represented by set of architecture building blocks
- Identifies key architectural artifacts that will be required to build the complete system
- Includes information flow between system-to-be-built and each external system
- External systems may require adapters
- Also indicates enterprise applications or databases involved

Example System Context Diagram



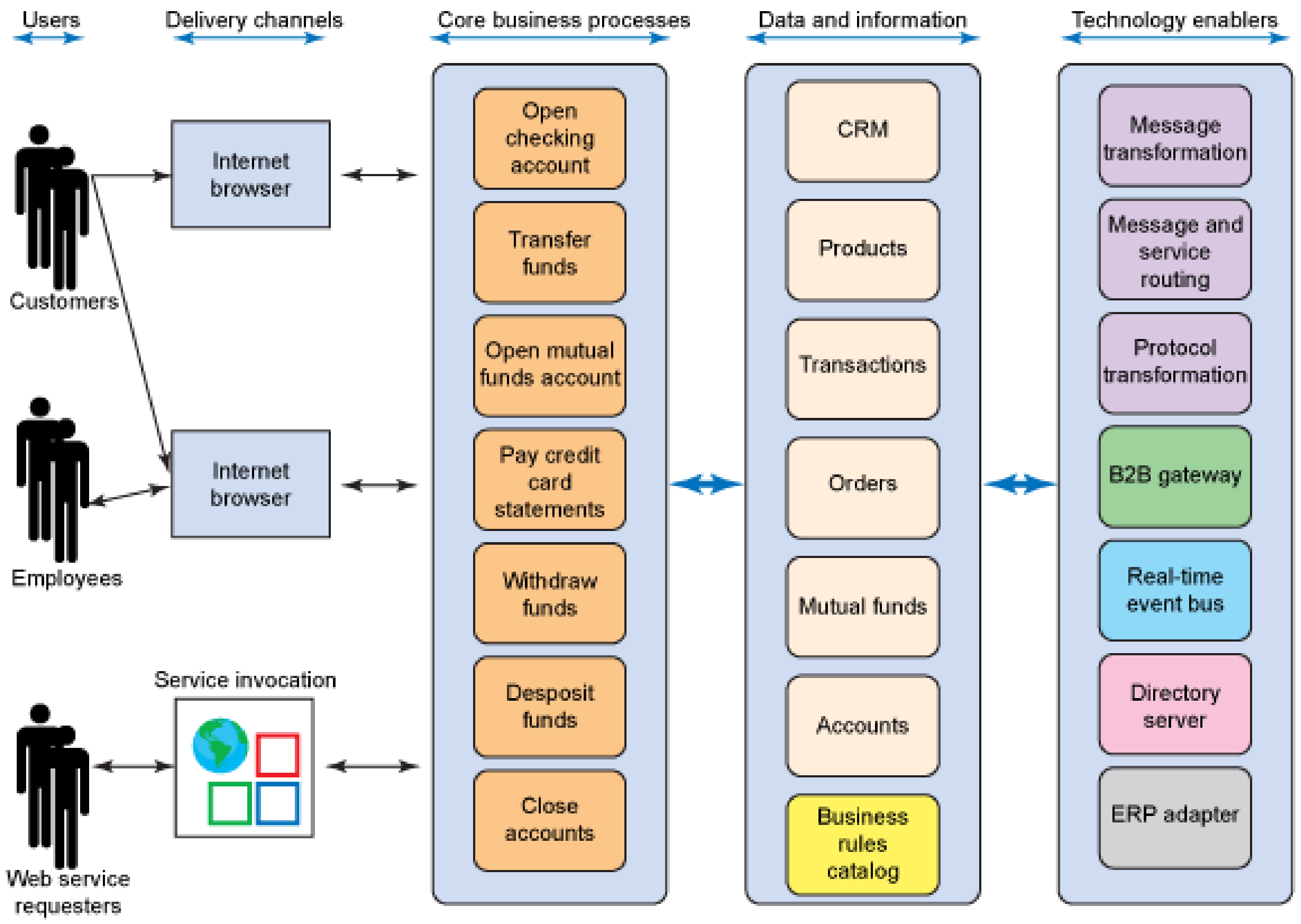
Architecture Overview

- Illustrates main conceptual elements and relationships
- Simple schematic representation
- Enterprise view and IT system view
- Represents business and IT capabilities required
- Provide high-level schematics that are further elaborated and documented in functional and operational architecture
- Depicts strategic direction for enterprise wrt IT systems

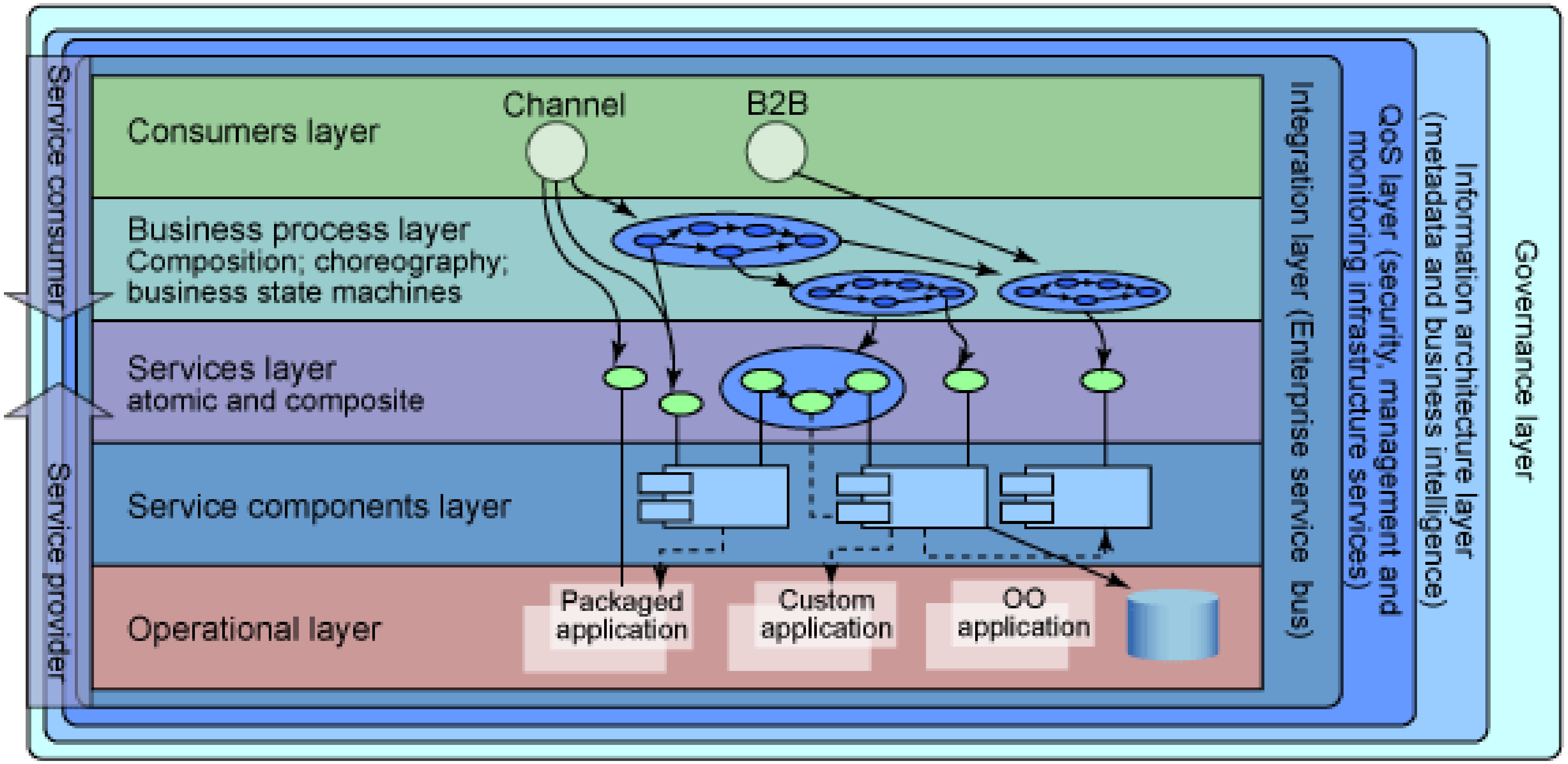
Architecture Overview

- Various views
  - Enterprise
    - Users and delivery channels
    - Core business processes
    - Data and information
    - Technology enablers
  - Layered
  - IT system
- Acts as guide for more elaborate functional and operational architecture
- Communicate conceptual understanding to stakeholders
- Provide a mechanism to evaluate different solutions

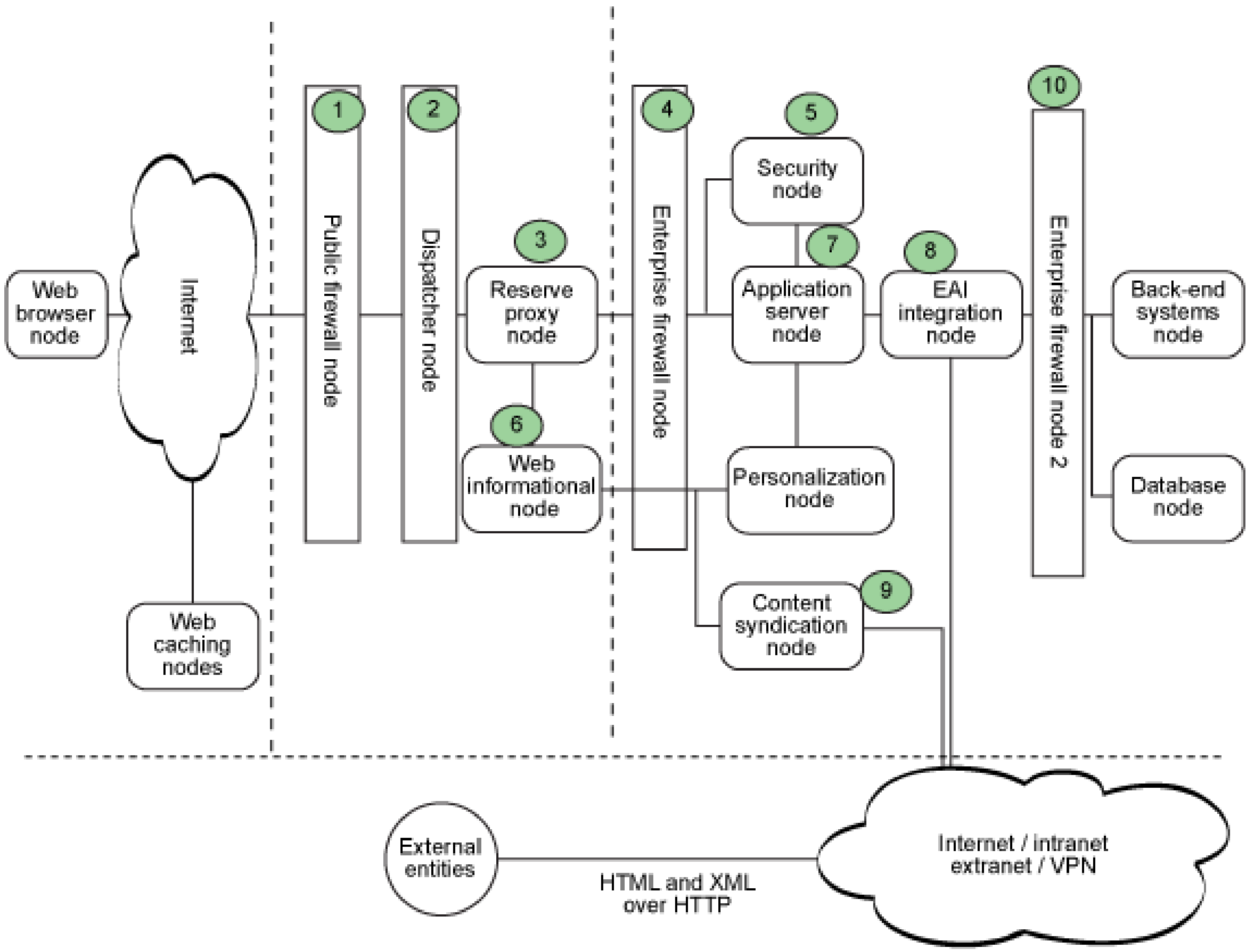
Example Enterprise Architecture Overview



Example Layered Architecture Overview


Consumers layer
Channel
B2B
Business process layer
Composition; choreography; business state machines
Services layer
atomic and composite
Service components layer
Operational layer
Packaged application
Custom application
OO application
Service consumer
Service provider
Integration layer (Enterprise service bus)
QoS layer (security, management and monitoring infrastructure services)
Information architecture layer (metadata and business intelligence)
Governance layer

Example IT System Architecture Overview



Functional architecture

- Also known as component architecture or model
- Used to document how architecture is decomposed into IT subsystems
- Provide a logical grouping of software components
- Describes structure of IT system in terms of its software components with their:
  - responsibilities
  - Interfaces
  - Static relationships
  - Way they collaborate to deliver required functions from the component
- Developed iteratively through various stages of elaboration

Functional Architecture

- Break the problem domain into a set of non-overlapping and collaborating components
- Analysis level modeling (as opposed to design level modeling or implementation level modeling)
- Macro level modeling (design is macro level, implementation is micro design)
- Three levels (From Higher to Lower levels of abstraction)
  - Logical Level
  - Specification Level
  - Physical Level
- “What” should be built and not “How” should it be built

Functional Architecture

- Three levels (From Higher to Lower levels of abstraction)
  - **Logical Level**
    - **Subsystem**
    - **Component**
  - Specification Level
  - Physical Level

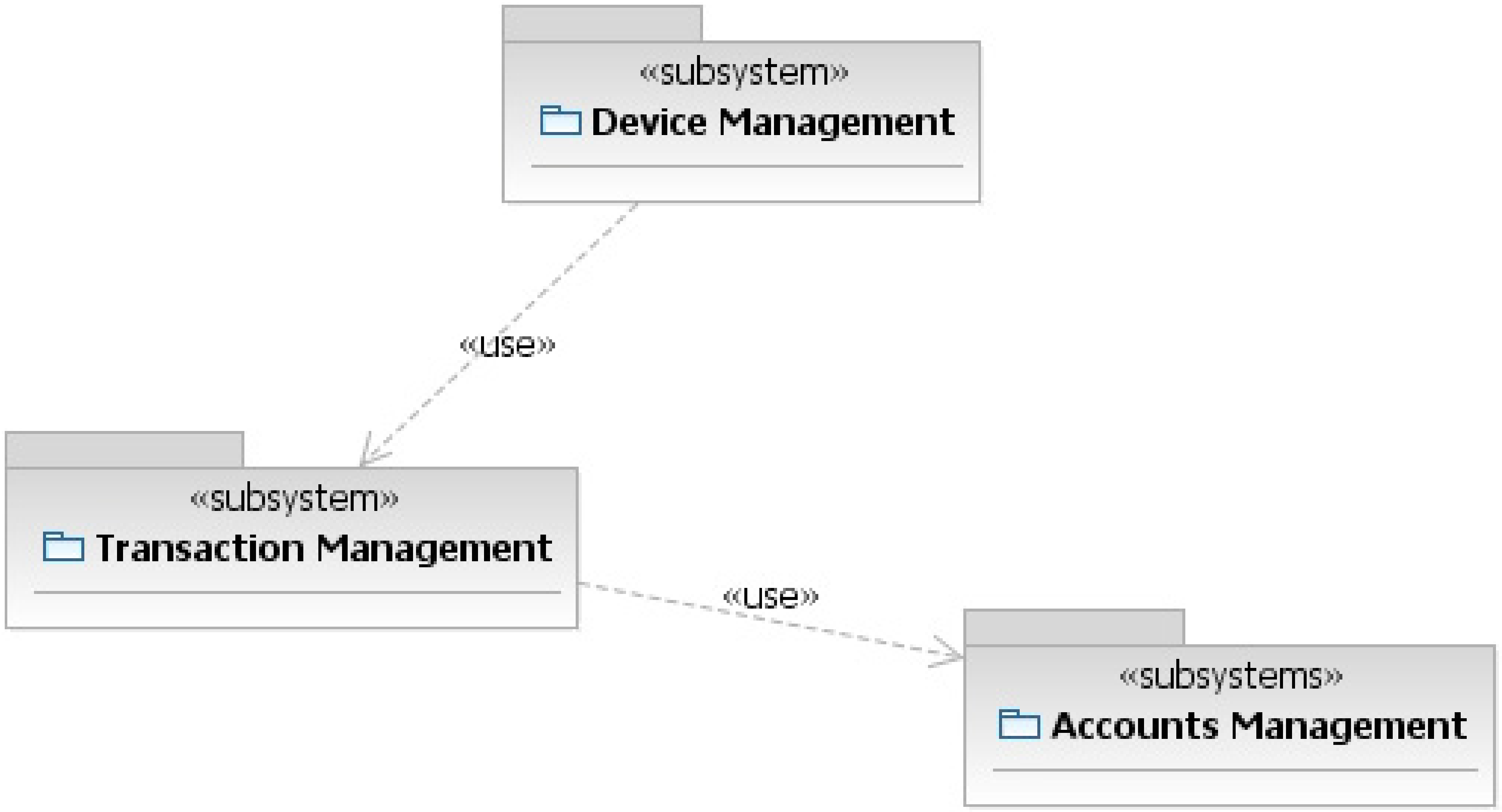
Logical Level Design

- Subsystem Design
  - Functional area in business domain can be represented and realized by one for more IT subsystem
  - Subsystem is grouping of cohesive software components that tend to change together so as to localize changes
  - IT Functions exposed by set of interfaces at the subsystem level
  - Components within subsystem implement interfaces
  - Foster parallel development using external interface contracts
  - First step is to identify IT subsystems and document
    - Identify high-level interfaces
  - Create UML representation of the subsystems and their interdependencies

For each subsystem

- Subsystem ID
  - Unique ID for each subsystem so that it’s easy to reference in the design
- Subsystem Name
  - e.g. Accounts Management, Transaction Management, etc
- Functions
  - List of IT functions the subsystem exposes through its internal implementation
- Interfaces
  - Textual List of interfaces the subsystem supports or exposes

Example Subsystem Dependencies



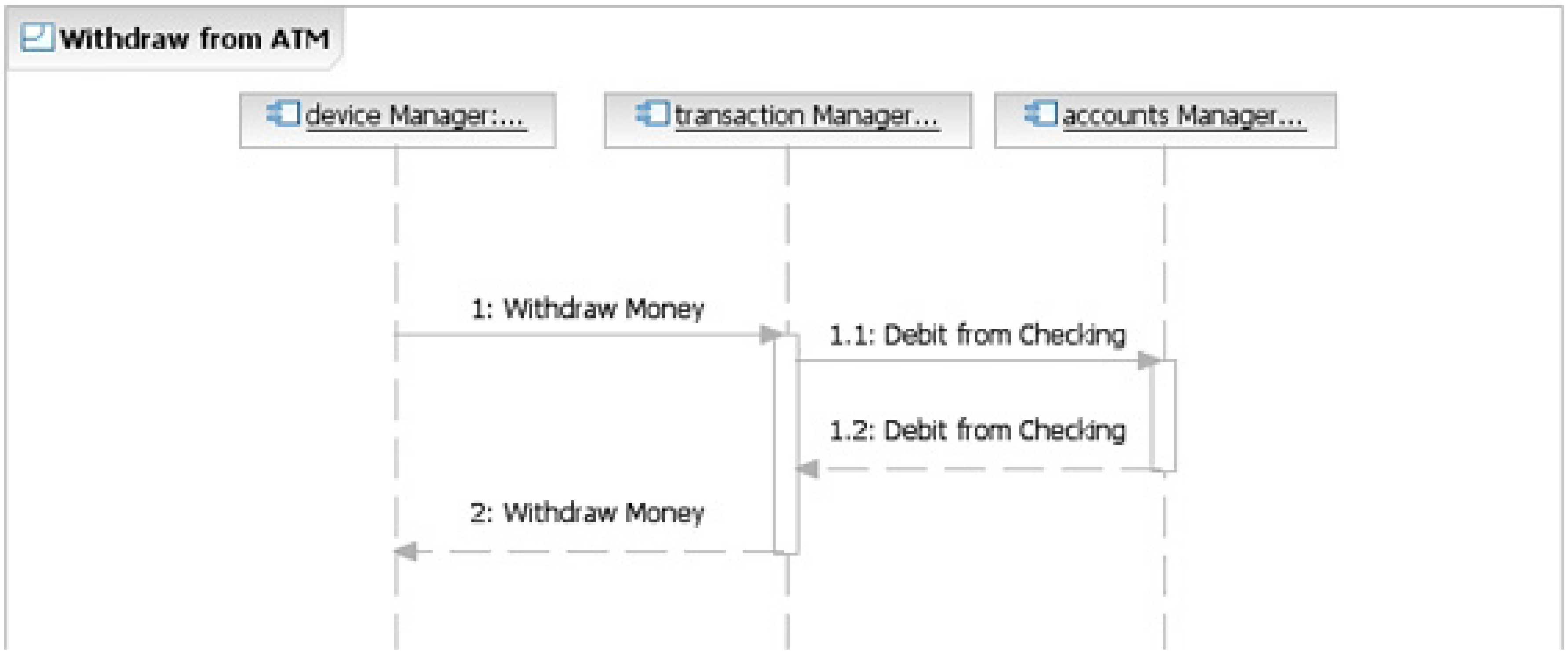
Textual Template -- Subsystem

Subsystem ID	SUBSYS-1
Subsystem Name	My Subsystem
Functions	Func1, Func2, …
Interfaces	INT-1-1, INT-1-2, INT-2-1

Logical Level Design

- Component Design
  - High level software components that collectively realize interfaces exposed by subsystem
  - Each component can be associated with a particular business entity (e.g. Savings account features, CD features, Safety Deposit Box, etc)
  - Identify logical components and then identify architecturally significant use cases
  - Next analyze the use cases and create component interaction diagrams to elaborate on how the use case may be realized through the functions that are exposed by the identified components
  - Collaboration diagram shows how components interact

Example High-Level Component Interaction



Functional Architecture

- Three levels (From Higher to Lower levels of abstraction)
  - Logical Level
  - **Specification Level**
  - Physical Level

Specification Level Design

- Detailed Design
- Add details to logical level design
  - Interfaces well defined
  - Data elements owned by each subsystem identified and detailed
  - Component's responsibilities are fleshed out in detail
- Four steps
  - Component responsibility matrix
  - Interface specification for components
  - Identify an associate data to subsystems
  - Component Interaction Diagram

Component Responsibility Matrix

Subsystem ID	SUBSYS-1
Component ID	COMP-1
Component Name	MyComponent
Component Responsibilities	Funct1, Funct2, Funct3 NFR1 NFR2 BR1 BR2

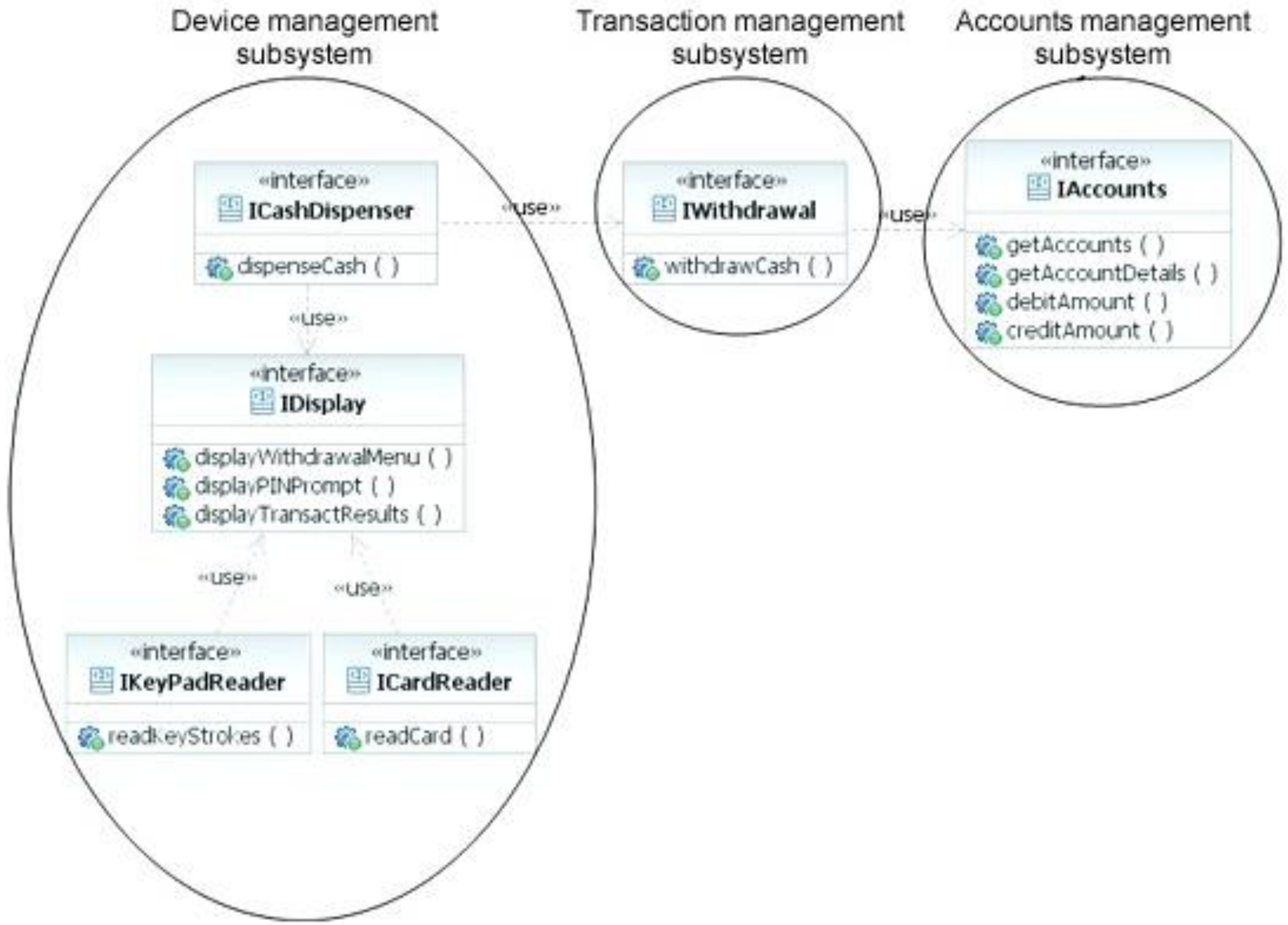
Component Responsibility XREF

	Comp 1	**Comp 2**	**Comp 3**	**…**	**…**	**…**	**…**	**…**	**…**
Funct 1									
Funct 2									
Funct 3									
NFR1									
NFR2									
NFR3									
BR1									
…									

Textual Template – Interface Design

Subsystem ID	SUBSYS-1
Component ID	COMP-1
Interface Name & ID	My Interface, INT-1-1
Interface operation(s)	Signatures for each operation along with descriptions

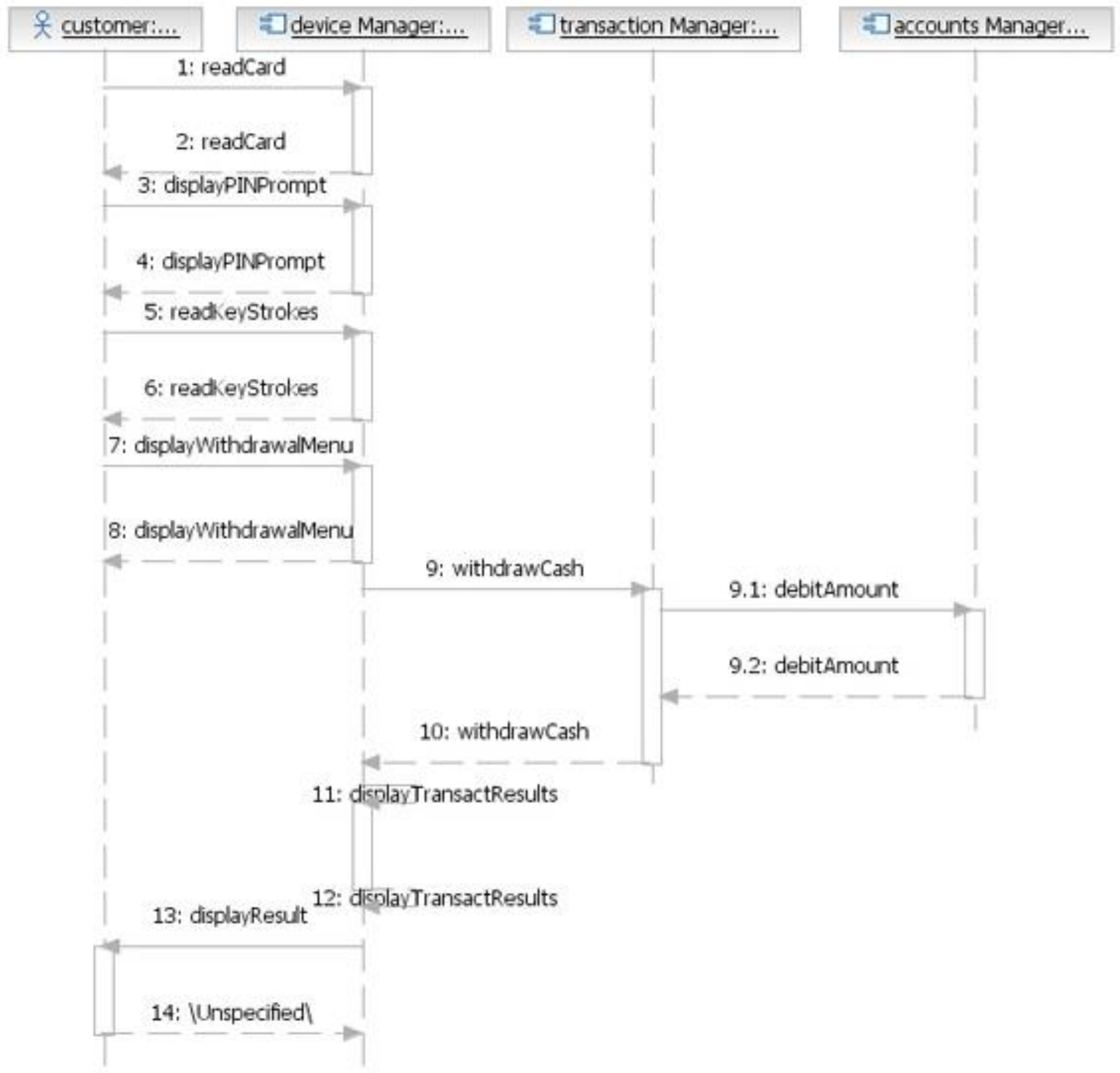
Interface Dependency Between & Inside Subsystems



Identifying & Associating Data to Subsystems

1. Analyze parameter list on the interfaces
2. Map parameters to the closest business entities or data types in the logical data model
3. Repeat steps 1 and 2 for each of the interfaces
4. Keep a running list of the data types identified
5. Repeat steps 1-4 for each component
6. Consolidate the list of data types identified
7. Repeat for all subsystems
8. Map against logical data model
9. Assign primary responsibility to one subsystem

Component Interaction Diagram



Functional Architecture

- Three levels (From Higher to Lower levels of abstraction)
  - Logical Level
  - Specification Level
  - **Physical Level**

Physical Level Design

- Revolves around distribution of application subsystems on the infrastructure
- Subsystems assigned to a hardware node
- Influences technical infrastructure and middleware
- Strong tie to NFRs
- Some overlap with Operational

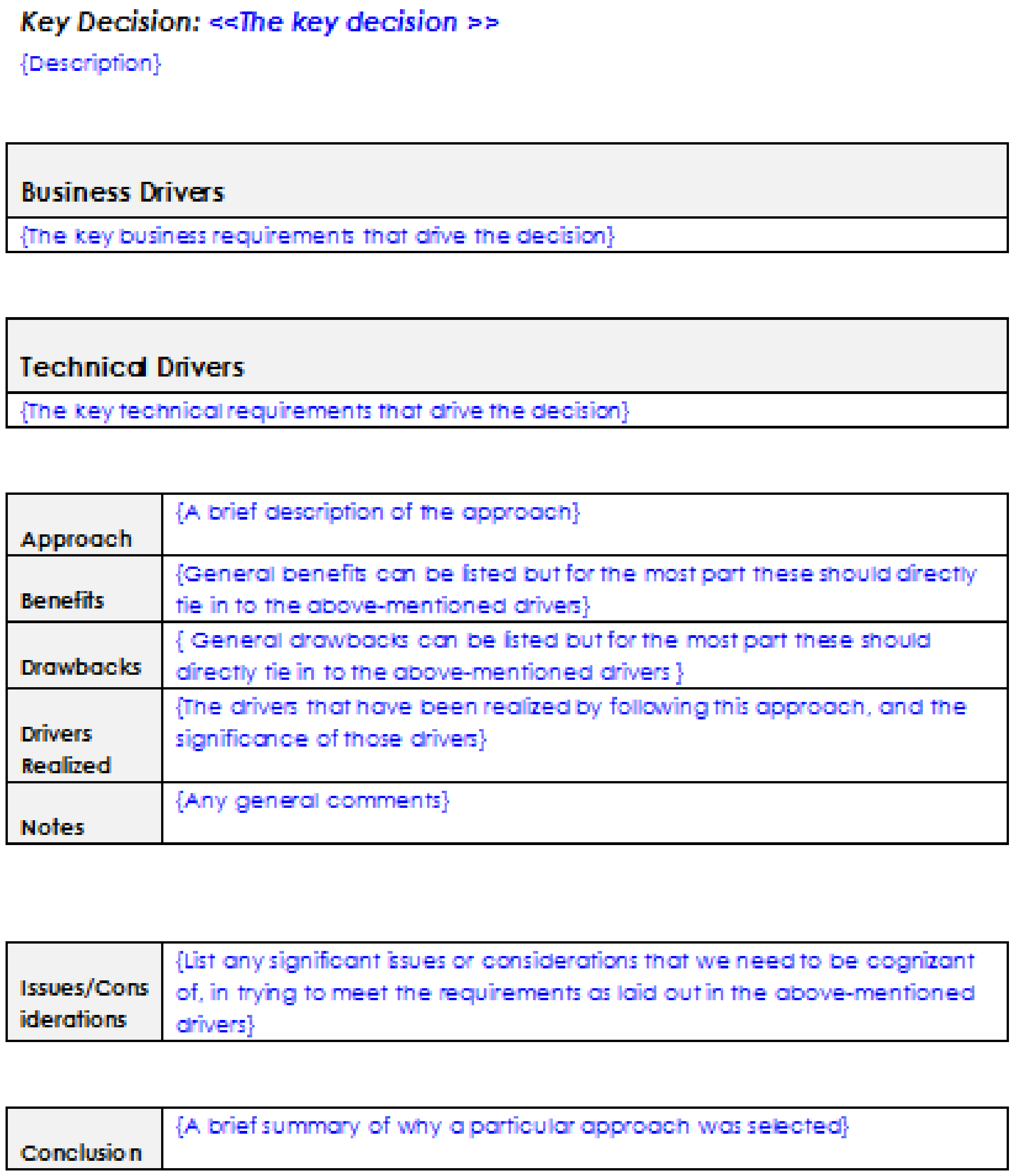
Operational Architecture

- Represents network of computer systems that support
  - Performance
  - Scalability
  - Fault tolerance
  - Etc
- Includes Middleware, systems software, and application software components
- Developed iteratively through various stages of elaboration

Architecture Decisions

- Single place where all architecturally relevant decisions are documented
- Typically include
  - Structure of systems
  - Identification of middleware components to support integration requirements
  - Allocation of functions to each architectural component (building blocks)
  - Allocation of architectural building blocks to various layers
  - Adherence to standards
  - Choice of technology to implement particular building block of functional components
- Involves
  - Identification of problem
  - Evaluation, including pros and cons, of various solutions
  - Selected solution, including adequate justification
- Assists with downstream activities

Template for Architectural Decisions



***Key Decision:** <<The key decision >>*

{Description}

Business Drivers
{The key business requirements that drive the decision}

Technical Drivers
{The key technical requirements that drive the decision}

Approach	{A brief description of the approach}
Benefits	{General benefits can be listed but for the most part these should directly tie in to the above-mentioned drivers}
Drawbacks	{ General drawbacks can be listed but for the most part these should directly tie in to the above-mentioned drivers }
Drivers Realized	{The drivers that have been realized by following this approach, and the significance of those drivers}
Notes	{Any general comments}

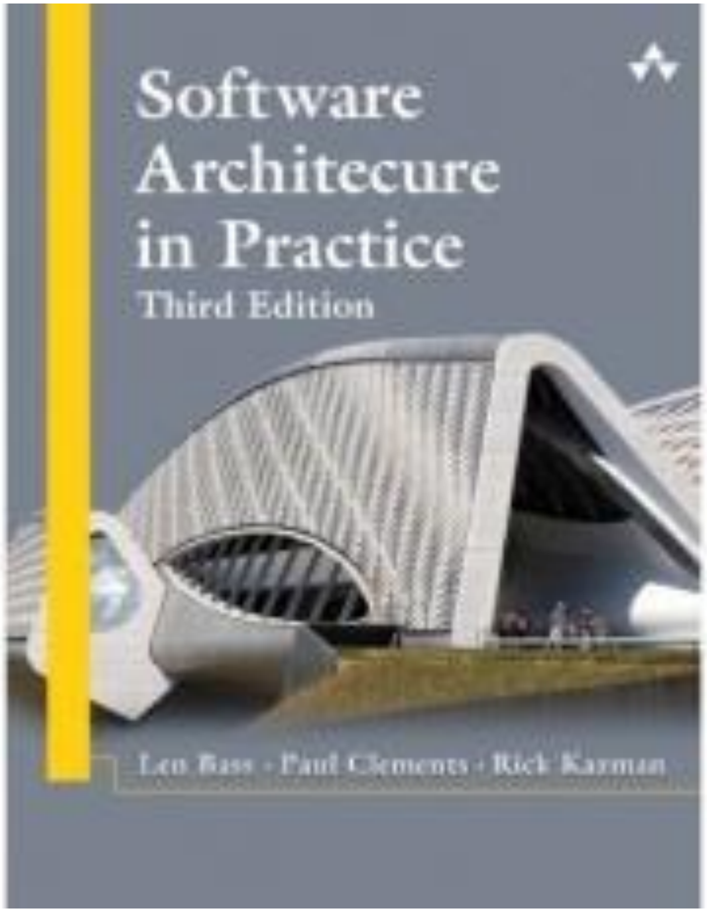
Issues/Considerations	{List any significant issues or considerations that we need to be cognizant of, in trying to meet the requirements as laid out in the above-mentioned drivers}

Conclusion	{A brief summary of why a particular approach was selected}

Recommended Basic Architecture Artifacts

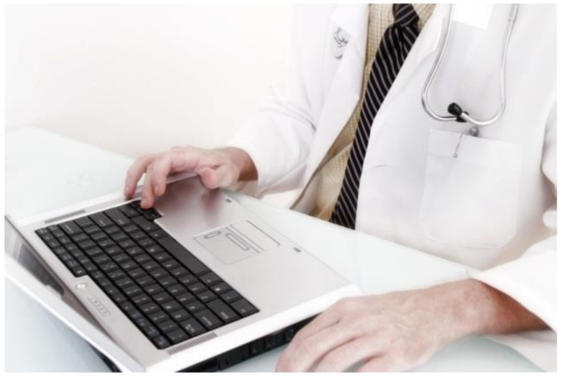
- Business context
- System context
- Architecture overview
- Functional architecture
- Operational architecture
- Architecture decisions

Chapter 18


Software Architecure in Practice
Third Edition
Len Bass · Paul Clements · Rick Kazman

Architecture Documentation

- Even the best architecture will be useless if the people who need it
  - do not know what it is;
  - cannot understand it well enough to use, build, or modify it;
  - misunderstand it and apply it incorrectly.
- All of the effort, analysis, hard work, and insightful design on the part of the architecture team will have been wasted.



Uses and Audience for Architecture Documentation

- Architecture documentation must
  - be sufficiently transparent and accessible to be quickly understood by new employees
  - be sufficiently concrete to serve as a blueprint for construction
  - have enough information to serve as a basis for analysis.
- Architecture documentation is both prescriptive and descriptive.
  - For some audiences, it prescribes what *should* be true, placing constraints on decisions yet to be made.
  - For other audiences, it describes what *is* true, recounting decisions already made about a system’s design.
- Understanding stakeholder uses of architecture documentation is essential as they determine the information to capture.

Three Uses for Architecture Documentation

Education

- Introducing people to the system
  - New members of the team
  - External analysts or evaluators
  - New architect

Primary vehicle for communication among stakeholders

- Especially architect to developers
- Especially architect to future architect!

Basis for system analysis and construction

- Architecture tells implementers what to implement.
- Each module has interfaces that must be provided and uses interfaces from other modules.
- Documentation can serve as a receptacle for registering and communicating unresolved issues.
- Architecture documentation serves as the basis for architecture evaluation.

Notations

- *Informal notations*
  - Views are depicted (often graphically) using general-purpose diagramming and editing tools
  - The semantics of the description are characterized in natural language
  - They cannot be formally analyzed
- *Semiformal notations*
  - Standardized notation that prescribes graphical elements and rules of construction
  - Lacks a complete semantic treatment of the meaning of those elements
  - Rudimentary analysis can be applied
  - UML is a semiformal notation in this sense.
- *Formal notations*
  - Views are described in a notation that has a precise (usually mathematically based) semantics.
  - Formal analysis of both syntax and semantics is possible.
  - Architecture description languages (ADLs)
  - Support automation through associated tools.

Choosing a Notation

- Tradeoffs
  - Typically, more formal notations take more time and effort to create and understand, but offer reduced ambiguity and more opportunities for analysis.
  - Conversely, more informal notations are easier to create, but they provide fewer guarantees.
- Different notations are better (or worse) for expressing different kinds of information.
  - UML class diagram will not help you reason about schedulability, nor will a sequence chart tell you very much about the system's likelihood of being delivered on time.
  - Choose your notations and representation languages knowing the important issues you need to capture and reason about.

Views

- Views let us divide a software architecture into a number of (we hope) interesting and manageable representations of the system.
- Principle of architecture documentation:
    - *Documenting an architecture is a matter of documenting the relevant views and then adding documentation that applies to more than one view.*

Which Views?...The Ones You Need!

- Different views support different goals and uses.
- We do not advocate a particular view or collection of views.
- The views you should document depend on the uses you expect to make of the documentation.
- Each view has a cost and a benefit; you should ensure that the benefits of maintaining a view outweigh its costs.

Method for Choosing the Views

- **Step 1. Build a stakeholder/view table.**
  - Rows: List the stakeholders for your project's software architecture documentation
  - Columns: Enumerate the views that apply to your system.
  - Some views (such as decomposition, uses, and work assignment) apply to every system, while others (various C&C views, the layered view) only apply to some systems.
  - Fill in each cell to describe how much information the stakeholder requires from the view: none, overview only, moderate detail, or high detail.

Method for Choosing the Views

- **Step 2. Combine views to reduce their number**
  - Look for marginal views in the table; those that require only an overview, or that serve very few stakeholders.
  - Combine each marginal view with another view that has a stronger constituency.

Method for Choosing the Views

- **Step 3. Prioritize and stage**.
  - The decomposition view (one of the module views) is a particularly helpful view to release early.
    - High-level decompositions are often easy to design
    - The project manager can start to staff development teams, put training in place, determine which parts to outsource, and start producing budgets and schedules.
  - You don’t have to satisfy all the information needs of all the stakeholders to the fullest extent.
    - Providing 80 percent of the information goes a long way, and this might be “good enough” so that the stakeholders can do their job.
    - Check with the stakeholder if a subset of information would be sufficient.
  - You don’t have to complete one view before starting another.
    - People can make progress with overview-level information
    - A breadth-first approach is often the best.

Building the Documentation Package

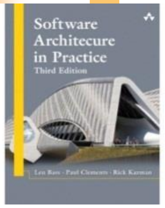
- Documentation package consists of
  - Views
  - Documentation beyond views

Documenting Behavior

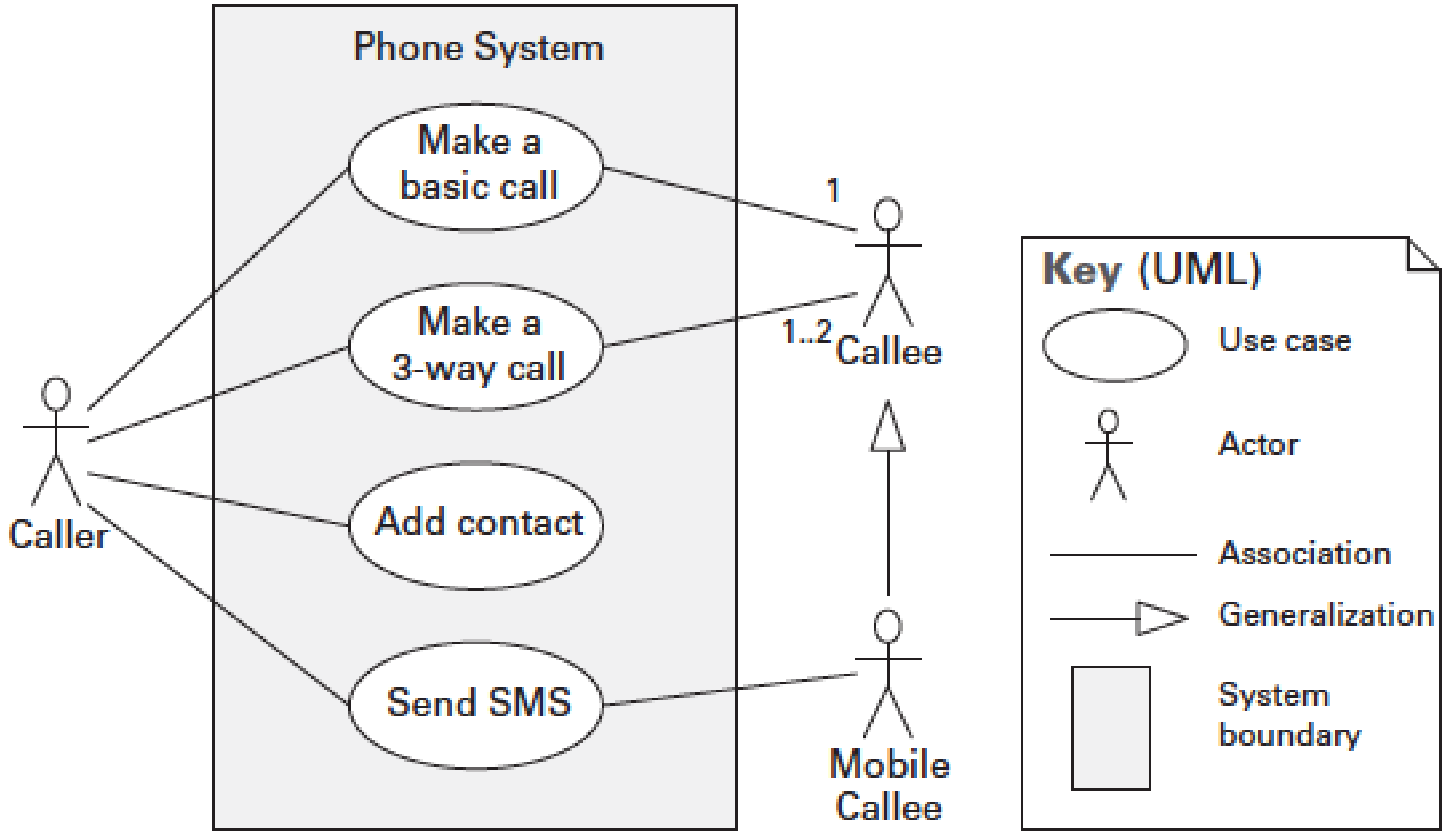
- Behavior documentation complements each view by describing how architecture elements in that view interact with each other.
- Behavior documentation enables reasoning about
  - a system's potential to deadlock
  - a system's ability to complete a task in the desired amount of time
  - maximum memory consumption
  - and more
- Behavior has its own section in our view template's element catalog.

Notations for Documenting Behavior

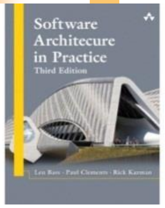
- Trace-oriented languages
  - *Traces* are sequences of activities or interactions that describe the system's response to a specific stimulus when the system is in a specific state.
  - A trace describes a particular sequence of activities or interactions between structural elements of the system.
  - Examples
    - use cases
    - sequence diagrams
    - communication diagrams
    - activity diagrams
    - message sequence charts
    - timing diagrams
    - Business Process Execution Language



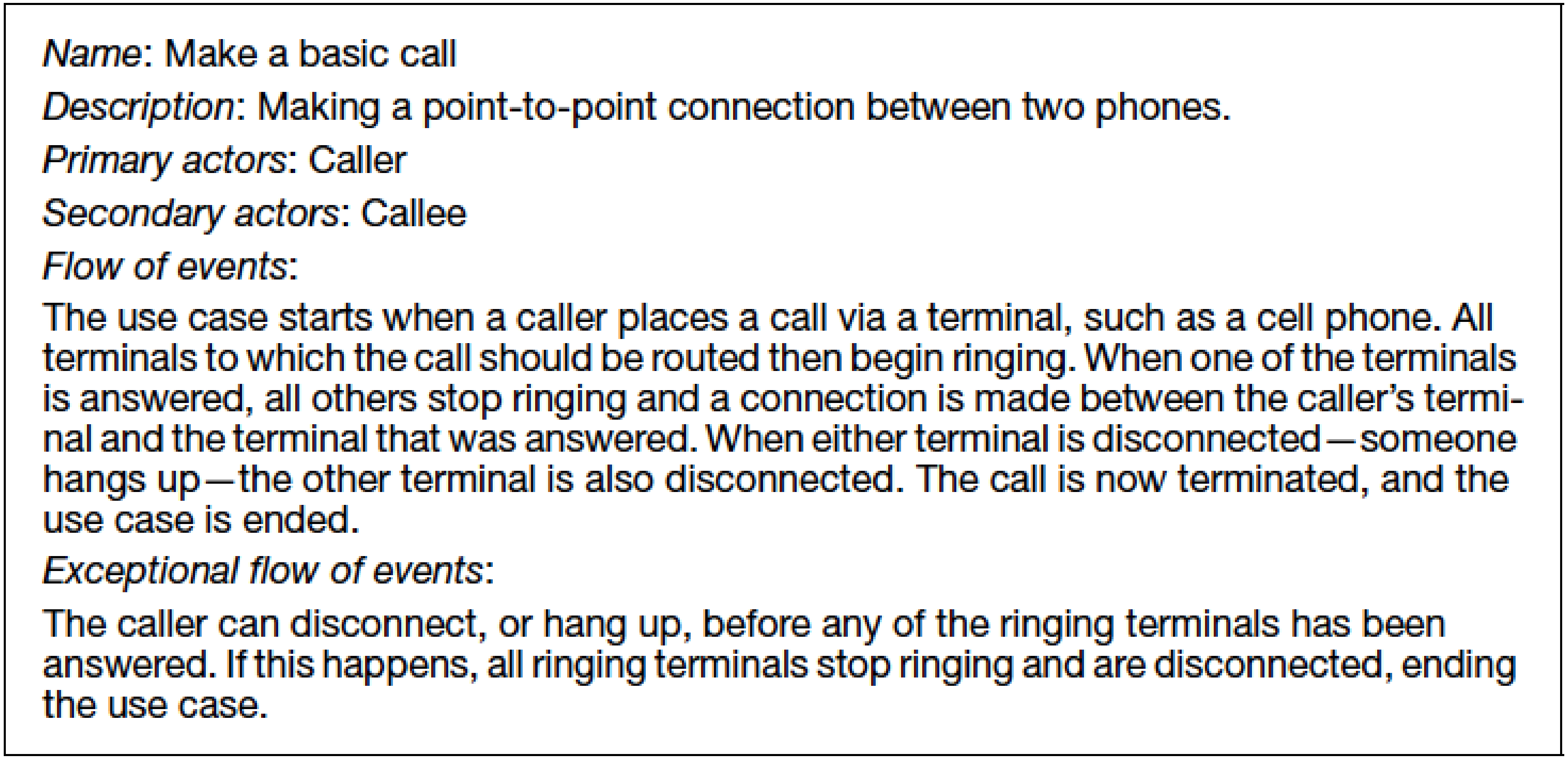
Use Case Diagram



From *Documenting Software Architectures: Views and Beyond*, 2nd edition, Addison Wesley, 2010



Use Case Description



Name: Make a basic call

Description: Making a point-to-point connection between two phones.

Primary actors: Caller

Secondary actors: Callee

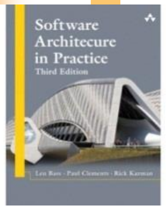
Flow of events:

The use case starts when a caller places a call via a terminal, such as a cell phone. All terminals to which the call should be routed then begin ringing. When one of the terminals is answered, all others stop ringing and a connection is made between the caller's terminal and the terminal that was answered. When either terminal is disconnected—someone hangs up—the other terminal is also disconnected. The call is now terminated, and the use case is ended.

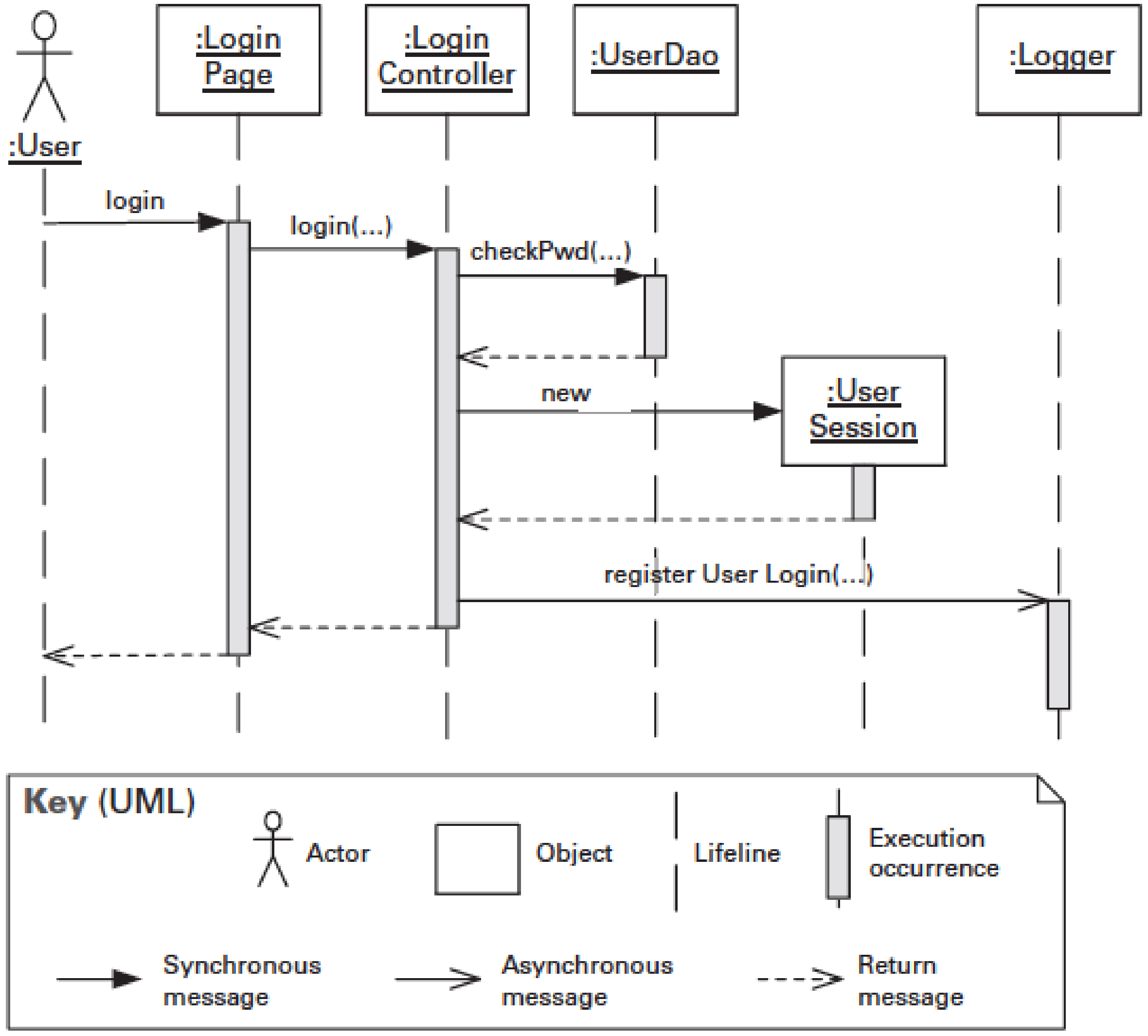
Exceptional flow of events:

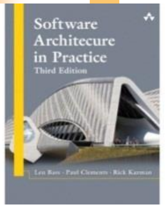
The caller can disconnect, or hang up, before any of the ringing terminals has been answered. If this happens, all ringing terminals stop ringing and are disconnected, ending the use case.

From *Documenting Software Architectures: Views and Beyond*, 2nd edition, Addison Wesley, 2010

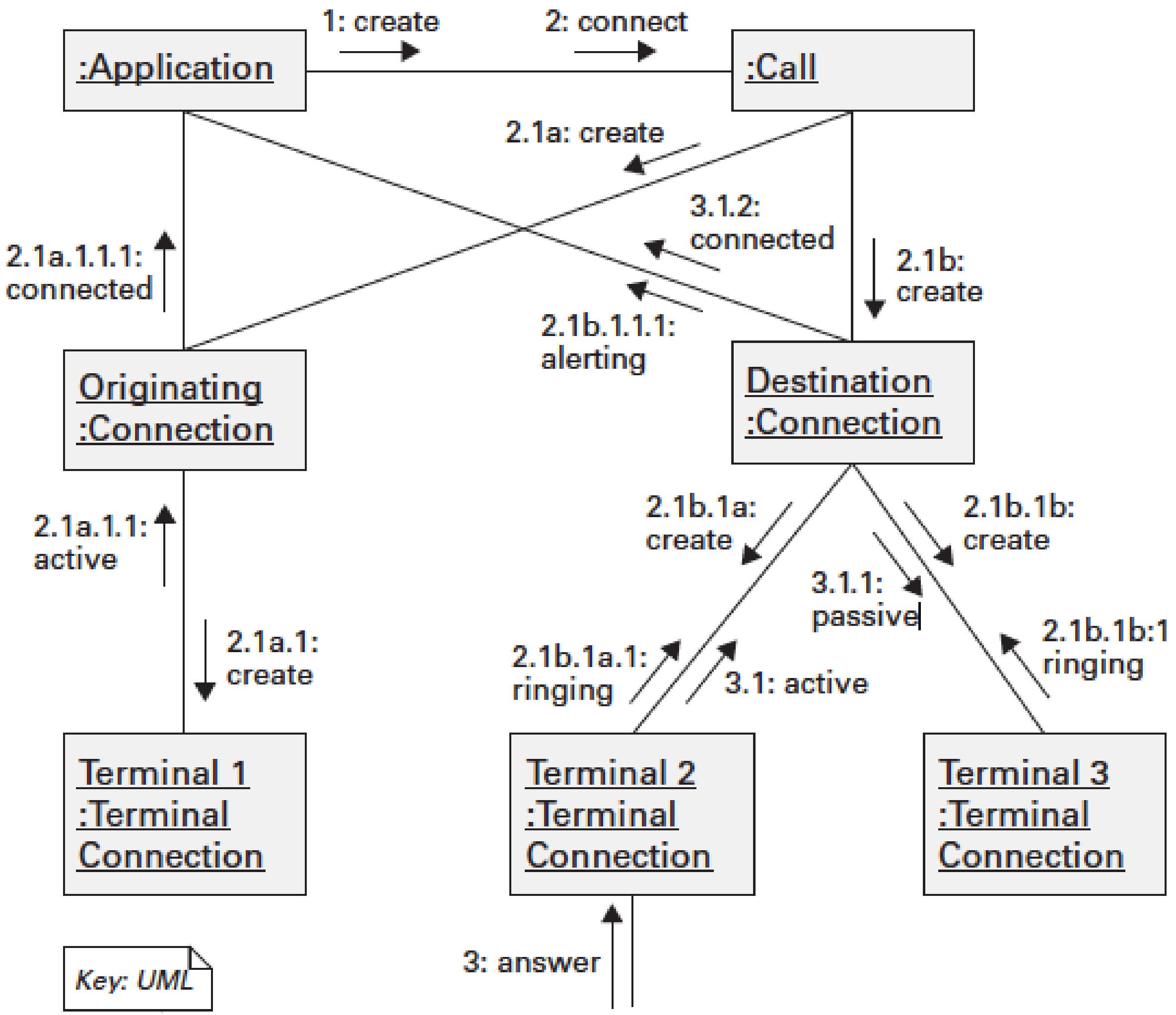


Sequence Diagram

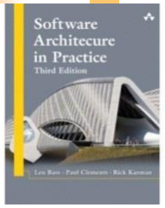




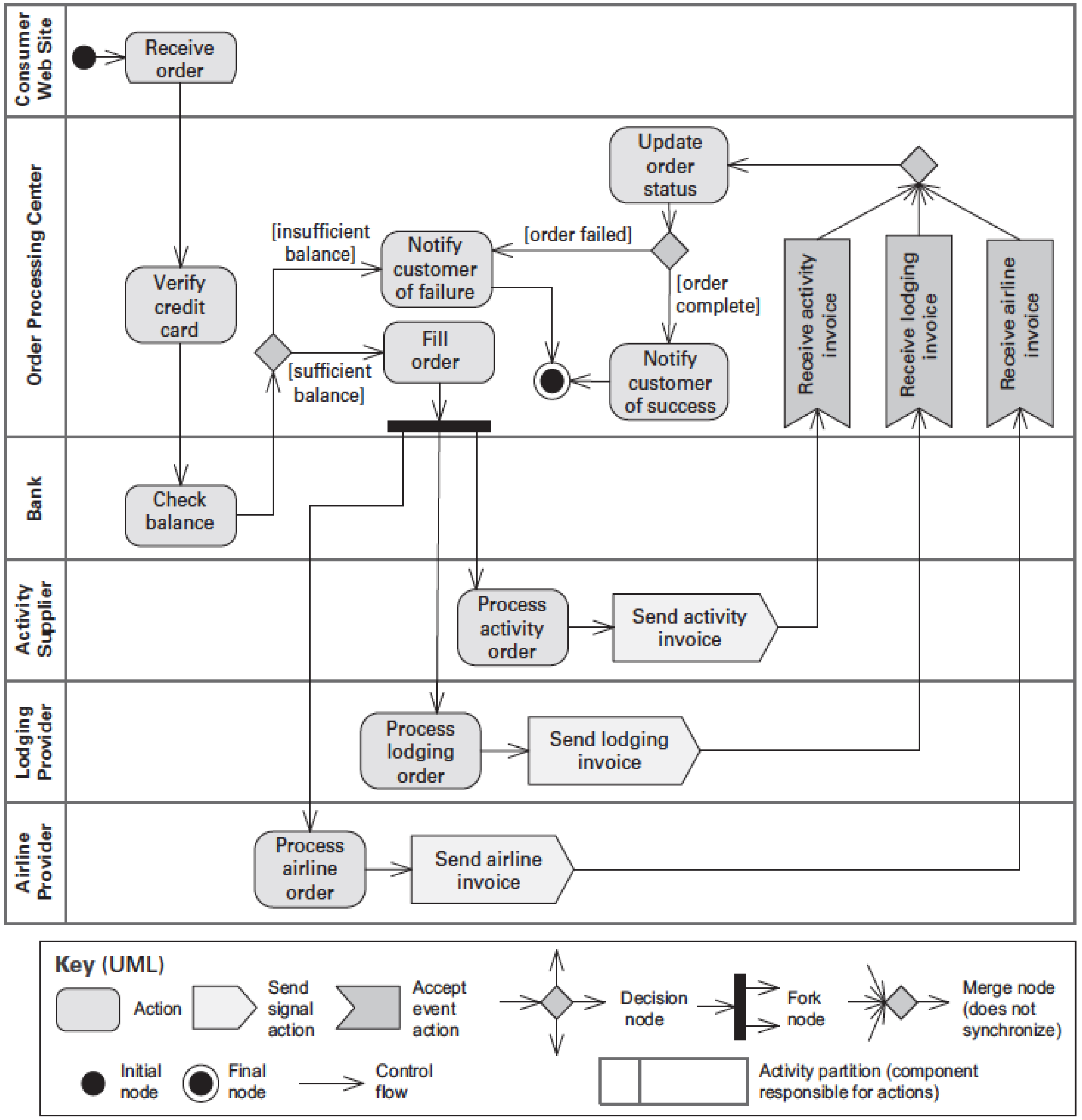
Communication Diagram



From *Documenting Software Architectures: Views and Beyond*, 2nd edition, Addison Wesley, 2010



Activity Diagram



From *Documenting Software Architectures: Views and Beyond*, 2nd edition, Addison Wesley, 2010

Documenting Quality Attributes

- Where do quality attributes show up in the documentation? There are five major ways:
  - Rationale that explains the choice of design approach should include a discussion about the quality attribute requirements and tradeoffs.
  - Architectural elements providing a service often have quality attribute bounds assigned to them, defined in the interface documentation for the elements, or recorded as *properties* that the elements exhibit.
  - Quality attributes often impart a “language” of things that you would look for. Someone fluent in the “language” of a quality attribute can search for the kinds of architectural elements) put in place to satisfy that quality attribute requirement.
  - Architecture documentation often contains a *mapping to requirements* that shows how requirements (including quality attribute requirements) are satisfied.
  - Every quality attribute requirement will have a constituency of stakeholders who want to know that it is going to be satisfied. For these stakeholders, the roadmap tells the stakeholder where in the document to find it.

Summary

- You must understand the uses to which the writing is to be put and the audience for the writing.
- Architectural documentation serves as a means for communication among various stakeholders, not only up the management chain and down to the developers but also across to peers.
- An architecture is a complicated artifact, best expressed by focusing on views.
- You must choose the views to document, must choose the notation to document these views, and must choose a set of views that is both minimal and adequate.
- You must document not only the structure of the architecture but also the behavior.