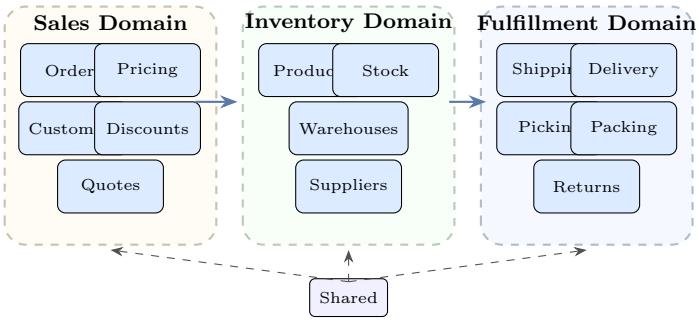
Logical Viewpoint

Architecture Viewpoint Specification

Functional Decomposition, Domain Structure & Capabilities



Version:	2.0
Status:	Release
Classification:	ISO/IEC/IEEE 42010 Compliant
Last Updated:	December 12, 2025

Based on the Views and Beyond approach to software architecture documentation

Contents

1 Viewpoint Name

Viewpoint Identification

Name:	Logical Viewpoint
Synonyms:	Functional Viewpoint, Domain View, Conceptual Architecture View, Business Capability View, Service View, Functional Decomposition View
Identifier:	VP-LOG-001
Version:	2.0

1.1 Viewpoint Classification

The Logical Viewpoint describes the system's functional structure in terms of logical elements, their responsibilities, and relationships—independent of implementation technology. Within the Views and Beyond approach, this aligns with Module styles (particularly Decomposition and Uses) but at a higher abstraction level focusing on functional capabilities rather than code modules. This viewpoint bridges business requirements and technical architecture.

Table 1: Viewpoint Classification Taxonomy

Attribute	Value
Style Family	Module (Logical/Conceptual)
Primary Focus	Functional Elements and Domain Structure
Abstraction Level	High-Level / Conceptual
Temporal Perspective	Static Functional Structure
Related Styles	Decomposition, Layered, Domain-Driven Design
IEEE 42010 Category	Functional/Logical Viewpoint
4+1 View Model	Logical View

1.2 Viewpoint Scope

The Logical Viewpoint encompasses the following aspects:

- **Functional Decomposition:** How system functionality is partitioned into logical elements with clear responsibilities.
- **Domain Structure:** Organization of the system according to business domains and subdomains.
- **Business Capabilities:** Mapping of system elements to business capabilities they enable.
- **Logical Services:** Abstract service definitions independent of implementation.

- **Information Entities:** Key domain entities and their relationships.
- **Functional Dependencies:** How logical elements depend on and interact with each other.
- **Domain Boundaries:** Clear boundaries between different functional areas.
- **Cross-Cutting Concerns:** Functionality that spans multiple domains.

2 Overview

The Logical Viewpoint provides a technology-independent view of the system's functional organization. It serves as a bridge between business requirements and technical implementation, ensuring that the system's structure reflects business needs and domain knowledge.

2.1 Purpose and Scope

The primary purpose of this viewpoint is to establish a clear functional decomposition that stakeholders from both business and technical backgrounds can understand and validate. It captures the essential "what" of the system before diving into the "how" of implementation.

Viewpoint Definition

The Logical Viewpoint defines the system's functional structure in terms of logical elements, their responsibilities, relationships, and organization into domains. It provides a conceptual model that is independent of specific technologies, platforms, or implementation approaches, focusing on functional capabilities and domain concepts that align with business needs.

2.2 Key Characteristics

The Logical Viewpoint exhibits several distinctive characteristics:

Technology Independence: Describes functionality without reference to specific programming languages, frameworks, or platforms.

Business Alignment: Structure reflects business domains, capabilities, and terminology rather than technical concerns.

Stakeholder Accessibility: Understandable by both business and technical stakeholders, using domain language.

Stability: More stable than implementation views as it changes only when business requirements change.

Foundation for Design: Guides subsequent technical architecture decisions and implementation structure.

2.3 Relationship to Other Viewpoints

The Logical Viewpoint connects to other architectural viewpoints:

Table 2: Relationships to Other Viewpoints

Viewpoint	Relationship
Context	Context defines external boundaries; Logical defines internal functional structure.
Development	Logical elements guide code module organization. Domain boundaries inform package structure.
Component-and-Connector	Logical services are realized as runtime components. Functional flows become connectors.
Deployment	Logical boundaries may inform deployment unit boundaries.
Information/Data	Domain entities in logical view are detailed in data models.
Process	Functional elements may map to processes or services at runtime.

2.4 Logical Architecture Overview

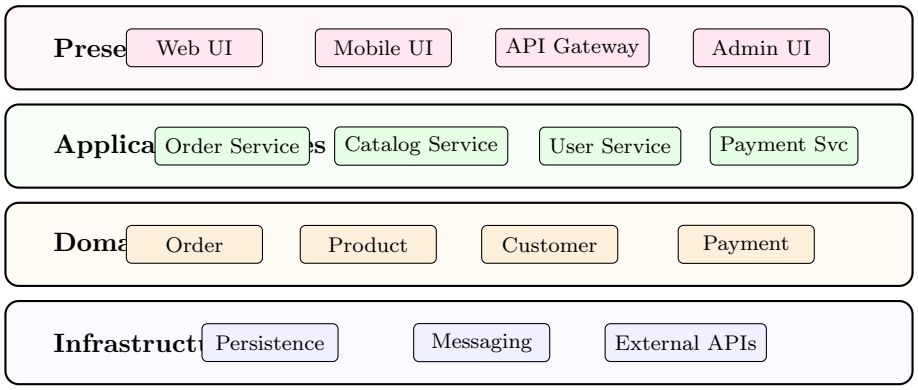


Figure 1: Logical Architecture Layers

3 Concerns

This section enumerates the architectural concerns that the Logical Viewpoint is designed to address.

3.1 Primary Concerns

C1: Functional Decomposition

- How is system functionality partitioned?

- What are the major functional areas?
- What responsibilities does each element have?
- How are responsibilities distributed?
- What is the rationale for the decomposition?

C2: Domain Structure

- What business domains does the system address?
- How are domains bounded and separated?
- What are the core vs supporting domains?
- How do domains relate to organizational structure?
- What ubiquitous language exists in each domain?

C3: Business Capability Mapping

- What business capabilities does the system enable?
- How do functional elements map to capabilities?
- What is the capability maturity in each area?
- How do capabilities align with business strategy?
- What capability gaps exist?

C4: Service Identification

- What logical services exist?
- What operations does each service provide?
- What are service boundaries?
- How autonomous are services?
- What service contracts exist?

C5: Domain Entity Modeling

- What are the key domain entities?
- What are entity relationships?
- What are aggregate boundaries?
- What invariants must be maintained?
- How do entities map to bounded contexts?

C6: Functional Dependencies

- How do functional elements depend on each other?
- What is the dependency direction?
- Are there circular dependencies?
- What coupling exists between elements?
- How are dependencies managed?

C7: Information Flow

- How does information flow between elements?

- What data is shared vs. private?
- What transformations occur?
- What is the source of truth for each data type?
- How is data consistency maintained?

C8: Cross-Cutting Functionality

- What functionality spans multiple domains?
- How is cross-cutting functionality handled?
- What shared services exist?
- How are shared concepts managed?
- What integration patterns are used?

C9: Extensibility and Evolution

- How can new functionality be added?
- What extension points exist?
- How do domains evolve independently?
- What backward compatibility is required?
- How are breaking changes managed?

C10: Business Rules and Logic

- Where do business rules reside?
- How are rules organized and managed?
- What validation logic exists?
- How are policies enforced?
- How do rules vary by context?

3.2 Concern-Quality Attribute Mapping

Table 3: Concern to Quality Attribute Mapping

Concern	Modtic.	Maintain.	Testtic.	Reustic.	Underst.	Flextic.	Interop.	Scaltic.
Decomposition	●	●	●	○	●	○	–	○
Domain Structure	●	●	○	○	●	●	○	●
Capabilities	○	○	–	○	●	●	–	–
Services	●	○	●	●	○	●	●	●
Entities	○	●	○	○	●	○	○	–
Dependencies	●	●	●	○	○	○	–	○
Info Flow	○	○	○	–	○	○	●	○
Cross-Cutting	○	○	○	●	○	○	●	–
Extenstic.	●	○	○	●	–	●	○	●
Business Rules	○	●	●	○	●	○	–	–

● = Primary impact, ○ = Secondary impact, – = Minimal impact

4 Anti-Concerns

Understanding what the Logical Viewpoint is *not* appropriate for helps stakeholders avoid misapplying this viewpoint.

4.1 Out of Scope Topics

AC1: Implementation Technology

- Programming languages and frameworks
- Database technologies
- Messaging platforms
- Cloud services
- Library choices

AC2: Physical Deployment

- Server configurations
- Container orchestration
- Network topology
- Cloud infrastructure
- Environment configurations

AC3: Runtime Behavior

- Process and thread structure

- Concurrency mechanisms
- Performance optimization
- Caching strategies
- Resource management

AC4: Detailed Data Models

- Physical database schemas
- Index definitions
- Storage optimization
- Query patterns
- Data migration scripts

AC5: Operational Concerns

- Monitoring and alerting
- Deployment pipelines
- Incident management
- Backup and recovery
- Security implementation

Common Misapplications

Avoid using the Logical Viewpoint for:

- Specifying implementation details (use Development Viewpoint)
- Defining physical deployment (use Deployment Viewpoint)
- Detailing runtime components (use C&C Viewpoint)
- Specifying database schemas (use Information Viewpoint)
- Documenting API contracts (use Interface Specifications)

5 Typical Stakeholders

The Logical Viewpoint serves stakeholders across business and technical domains.

5.1 Primary Stakeholders

Table 4: Primary Stakeholder Analysis

Stakeholder	Role Description	Primary Interests
Software Architects	Design system structure	Functional decomposition, domain boundaries, dependencies
Business Analysts	Define requirements	Business capability mapping, domain terminology
Domain Experts	Provide domain knowledge	Domain model accuracy, business rule placement
Product Managers	Define product direction	Feature placement, capability roadmap
Development Leads	Guide implementation	Service boundaries, module structure guidance
Enterprise Architects	Manage system portfolio	Capability alignment, reuse opportunities

5.2 Secondary Stakeholders

Table 5: Secondary Stakeholder Analysis

Stakeholder	Role Description	Primary Interests
Developers	Implement features	Understanding functional context, responsibility clarity
QA Engineers	Test functionality	Test scope, functional boundaries
Technical Writers	Document system	Domain terminology, functional descriptions
New Team Members	Onboard to system	System understanding, conceptual overview
Integration Teams	Connect systems	Service boundaries, integration points
Business Sponsors	Fund development	Business alignment, capability coverage

5.3 Stakeholder Concern Matrix

Table 6: Stakeholder-Concern Responsibility Matrix

	Decomp.	Domain	Capab.	Service	Entity	Depend.	InfoFlow	CrossCut	Extens.	Rules
Architect	R	R	C	R	A	R	R	R	R	A
Bus. Analyst	C	A	R	C	C	I	C	C	C	R
Domain Exp.	C	R	C	I	R	I	C	I	I	R
Product Mgr	I	C	A	C	I	I	I	I	A	C
Dev Lead	A	C	I	A	C	A	C	A	C	C
Enterprise	C	C	R	C	I	C	I	C	C	I

R = Responsible, A = Accountable, C = Consulted, I = Informed

6 Model Types

The Logical Viewpoint employs several complementary model types to capture different aspects of functional structure.

6.1 Model Type Catalog

MT1: Functional Decomposition Diagram

- *Purpose:* Show hierarchical breakdown of system functionality
- *Primary Elements:* Functional areas, sub-functions, relationships
- *Key Relationships:* Contains, decomposes-to
- *Typical Notation:* Hierarchical diagrams, tree structures

MT2: Domain Model

- *Purpose:* Capture key domain concepts and relationships
- *Primary Elements:* Entities, value objects, relationships
- *Key Relationships:* Associates, aggregates, inherits
- *Typical Notation:* UML class diagrams (conceptual level)

MT3: Bounded Context Map

- *Purpose:* Show domain boundaries and context relationships
- *Primary Elements:* Bounded contexts, relationships
- *Key Relationships:* Upstream/downstream, shared kernel, ACL
- *Typical Notation:* Context map diagrams (DDD style)

MT4: Business Capability Model

- *Purpose:* Map system to business capabilities
- *Primary Elements:* Capabilities, capability groups

- *Key Relationships:* Enables, supports, requires
- *Typical Notation:* Capability maps, heat maps

MT5: Service Catalog

- *Purpose:* Document logical services and operations
- *Primary Elements:* Services, operations, contracts
- *Key Relationships:* Provides, consumes, depends-on
- *Typical Notation:* Service tables, interface diagrams

MT6: Logical Layer Diagram

- *Purpose:* Show functional layers and their relationships
- *Primary Elements:* Layers, elements within layers
- *Key Relationships:* Uses, calls (top-down only)
- *Typical Notation:* Layer diagrams, stack diagrams

MT7: Functional Flow Diagram

- *Purpose:* Show information and control flow between elements
- *Primary Elements:* Functions, flows, decision points
- *Key Relationships:* Flows-to, triggers, responds-to
- *Typical Notation:* Flow diagrams, sequence diagrams

MT8: Use Case Realization

- *Purpose:* Show how functional elements realize use cases
- *Primary Elements:* Use cases, participating elements
- *Key Relationships:* Participates-in, realizes
- *Typical Notation:* Collaboration diagrams, sequence diagrams

MT9: Cross-Reference Matrix

- *Purpose:* Map elements to requirements, capabilities, or domains
- *Primary Elements:* Elements, mapping targets
- *Key Relationships:* Maps-to, supports, implements
- *Typical Notation:* Matrices, tables

6.2 Model Type Relationships

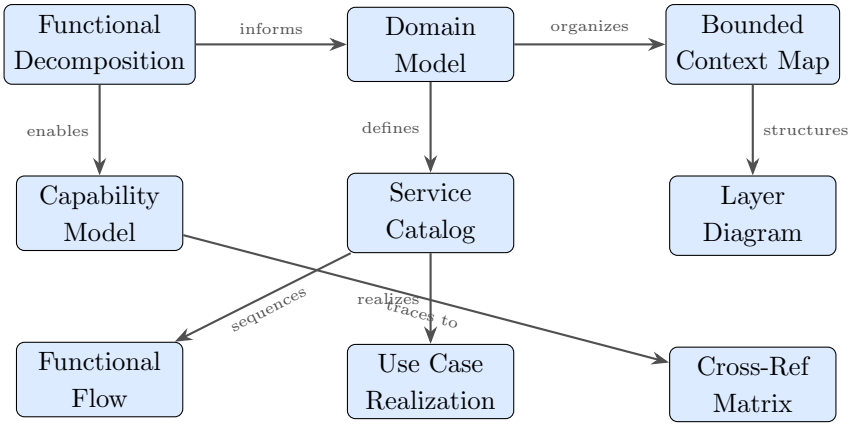


Figure 2: Model Type Dependency Relationships

7 Model Languages

For each model type, specific languages, notations, and techniques are prescribed.

7.1 Logical Element Notation

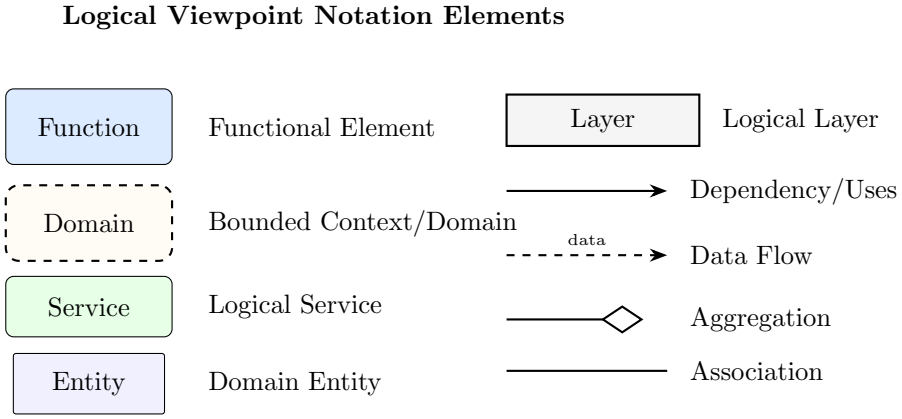


Figure 3: Logical Viewpoint Notation Legend

7.2 Domain-Driven Design Context Relationships

Table 7: DDD Context Relationship Types

Relationship	Description	When to Use
Shared Kernel	Shared subset of domain model	Teams collaborate closely, shared concepts
Customer-Supplier	Upstream serves downstream	Clear dependency direction
Conformist	Downstream conforms to upstream	No control over upstream model
Anti-Corruption Layer	Translation layer between contexts	Protect domain from external model
Open Host Service	Published well-defined protocol	Multiple consumers need access
Published Language	Common interchange format	Cross-system communication
Separate Ways	No integration	Independent development
Partnership	Coordinated evolution	Mutual dependency, joint success

7.3 Functional Element Classification

Table 8: Functional Element Type Classification

Element Type	Description	Examples
Core Domain	Central business differentiator	Order Management, Pricing Engine
Supporting	Supports core without differentiating	Reporting, Notifications
Generic	Commodity, not business-specific	Authentication, File Storage
Application Service	Orchestrates domain operations	OrderProcessingService
Domain Service	Domain logic not in entities	PricingCalculator
Infrastructure	Technical capabilities	EmailSender, DataAccess

7.4 Tabular Specifications

7.4.1 Service Specification Table

Table 9: Example Service Specification Format

Service	Domain	Responsibilities	Key Operations
Order Service	Sales	Order lifecycle management	createOrder, cancelOrder, getOrder
Pricing Service	Sales	Calculate prices and discounts	calculatePrice, applyDiscount
Inventory Svc	Inventory	Stock management	checkStock, reserveStock, release
Fulfillment Svc	Fulfillment	Order fulfillment coordination	pickOrder, packOrder, shipOrder

7.4.2 Domain Entity Specification Table

Table 10: Example Domain Entity Specification

Entity	Domain	Key Attributes	Invariants	Aggregate
Order	Sales	id, customer, items, status	Items > 0, valid status	Order (root)
OrderItem	Sales	product, quantity, price	Quantity > 0	Order
Customer	Sales	id, name, email, tier	Valid email format	Customer (root)
Product	Inventory	sku, name, price, stock	Price > 0	Product (root)

7.4.3 Capability Mapping Table

Table 11: Example Business Capability Mapping

Capability	Sub-Capability	Functional Element	Maturity
Sales Management	Order Processing Pricing Quotations	Order Service Pricing Service Quote Service	High Medium Low
Inventory Mgmt	Stock Control Replenishment	Inventory Service Supplier Service	High Medium
Fulfillment	Shipping Returns	Shipping Service Returns Service	High Medium

8 Viewpoint Metamodels

This section defines the conceptual metamodel underlying the Logical Viewpoint.

8.1 Core Metamodel

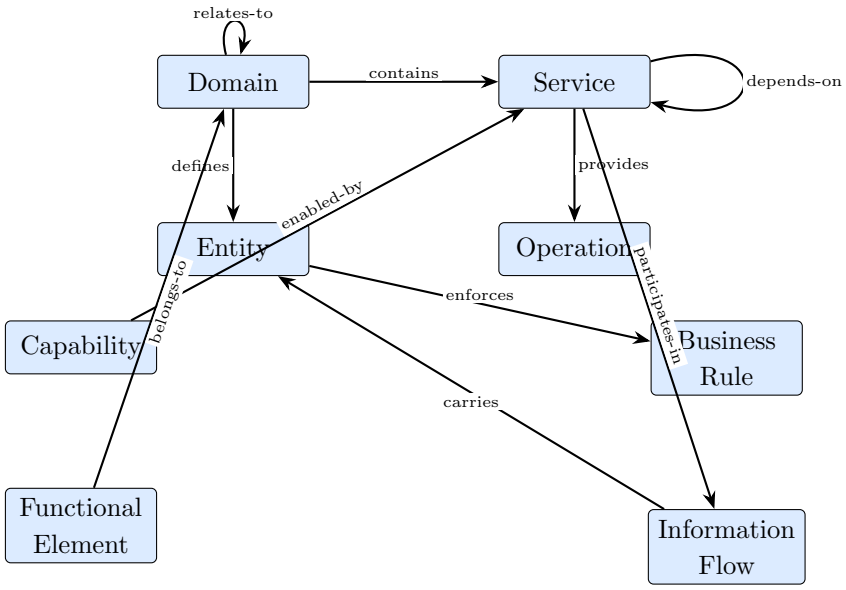


Figure 4: Logical Viewpoint Core Metamodel

8.2 Entity Definitions

Entity: Domain

Definition: A bounded context representing a cohesive area of business functionality with its own ubiquitous language, model, and boundaries.

Attributes:

- `domainId`: Unique identifier
- `name`: Domain name
- `description`: Domain purpose and scope
- `type`: Domain type (core, supporting, generic)
- `ubiquitousLanguage`: Key terms and definitions
- `owner`: Team or individual responsible
- `boundaries`: What is in/out of scope
- `subdomains`: Child domains if decomposed
- `contextRelationships`: Relationships to other domains

Constraints:

- Domain must have clear boundaries
- Ubiquitous language must be documented
- Core domains must be identified and prioritized
- Context relationships must be explicitly defined

Entity: Service

Definition: A logical grouping of related operations that provides a cohesive set of capabilities, encapsulating domain logic behind a well-defined interface.

Attributes:

- `serviceId`: Unique identifier
- `name`: Service name
- `description`: Service purpose
- `domain`: Owning domain
- `type`: Service type (application, domain, infrastructure)
- `operations`: List of operations provided
- `dependencies`: Services this depends on
- `consumers`: Services that consume this
- `contracts`: Interface contracts
- `autonomy`: Level of independence

Constraints:

- Service must belong to exactly one domain
- Operations must be cohesive to service purpose
- Dependencies should follow domain relationships
- Contracts must be versioned

Entity: Domain Entity

Definition: A domain object with identity that encapsulates state and behavior, representing a key concept within a bounded context.

Attributes:

- `entityId`: Unique identifier
- `name`: Entity name
- `description`: Entity purpose
- `domain`: Owning domain
- `attributes`: Key attributes
- `relationships`: Relationships to other entities
- `invariants`: Business rules that must hold
- `lifecycle`: Valid lifecycle states
- `aggregate`: Aggregate this belongs to (if any)
- `isAggregateRoot`: Whether this is an aggregate root

Constraints:

- Entity must have identity
- Invariants must be enforceable
- Aggregate boundaries must be clear
- Lifecycle states must be well-defined

Entity: Operation

Definition: A specific action or behavior that a service can perform, representing a unit of functionality with defined inputs and outputs.

Attributes:

- `operationId`: Unique identifier
- `name`: Operation name (verb-noun format)
- `description`: Operation purpose
- `service`: Owning service
- `inputs`: Input parameters
- `outputs`: Output/return values
- `preconditions`: Required state before execution
- `postconditions`: Guaranteed state after execution
- `exceptions`: Possible failure conditions
- `idempotent`: Whether operation is idempotent

Constraints:

- Operation name should clearly indicate action
- Inputs and outputs must be documented
- Side effects should be explicit

Entity: Capability

Definition: A business ability that the system enables, representing what the business can do (not how it does it).

Attributes:

- `capabilityId`: Unique identifier
- `name`: Capability name
- `description`: Capability purpose
- `level`: Hierarchy level (L1, L2, L3)
- `parent`: Parent capability (if any)
- `children`: Child capabilities
- `enablingServices`: Services that enable this
- `maturity`: Current maturity level
- `strategicImportance`: Business importance
- `owner`: Business owner

Constraints:

- Capabilities should be business-focused, not technical
- Hierarchy should be consistent
- Strategic capabilities should be identified

Entity: Business Rule

Definition: A constraint, policy, or logic that governs business operations and must be enforced by the system.

Attributes:

- `ruleId`: Unique identifier
- `name`: Rule name
- `description`: Rule statement
- `type`: Rule type (constraint, derivation, policy)
- `domain`: Domain where rule applies
- `entities`: Entities affected by rule
- `expression`: Formal rule expression
- `enforcement`: Where/how rule is enforced
- `source`: Business source of rule
- `exceptions`: Allowed exceptions

Constraints:

- Rules must be clearly stated
- Enforcement location must be defined
- Business source should be traceable

Entity: Functional Element

Definition: A logical unit of functionality that contributes to system capabilities, abstracting implementation details.

Attributes:

- `elementId`: Unique identifier
- `name`: Element name
- `description`: Element purpose
- `domain`: Owning domain
- `type`: Element type (core, supporting, infrastructure)
- `responsibilities`: Key responsibilities
- `dependencies`: Elements this depends on
- `interfaces`: Provided/required interfaces
- `constraints`: Applicable constraints

Constraints:

- Element should have single clear responsibility
- Dependencies should be minimized
- Interfaces should be well-defined

Entity: Information Flow

Definition: The movement of data or messages between functional elements, characterizing how information is exchanged.

Attributes:

- `flowId`: Unique identifier
- `name`: Flow name
- `description`: Flow purpose
- `source`: Origin element
- `destination`: Target element
- `dataType`: Type of information carried
- `trigger`: What initiates the flow
- `frequency`: How often flow occurs
- `synchronicity`: Sync vs async
- `transformation`: Any data transformation

Constraints:

- Flows should have clear purpose
- Data types should be well-defined
- Cross-domain flows should be explicit

8.3 Relationship Definitions

Table 12: Metamodel Relationship Definitions

Relationship	Source	Target	Description
contains	Domain	Service	Domain includes this service
defines	Domain	Entity	Domain defines this entity
provides	Service	Operation	Service offers this operation
enforces	Entity	Rule	Entity enforces this business rule
enabled-by	Capability	Service	Capability is enabled by service
belongs-to	Function	Domain	Functional element is in domain
depends-on	Service	Service	Service requires another service
participates-in	Service	Flow	Service is part of information flow
carries	Flow	Entity	Flow transmits entity data
relates-to	Domain	Domain	Domain has relationship to another

9 Conforming Notations

Several existing notations and modeling approaches align with the Logical Viewpoint.

9.1 UML Structural Diagrams

UML provides several diagram types suitable for logical modeling:

Class Diagrams (Conceptual): Domain entities, relationships, and attributes at a conceptual level.

Package Diagrams: Grouping of elements into domains and functional areas.

Component Diagrams: Logical components and their dependencies (abstract level).

Conformance Level: High for domain modeling, medium for service modeling.

9.2 Domain-Driven Design Notations

DDD provides specialized notations for bounded contexts and strategic design:

Context Maps: Show relationships between bounded contexts.

Aggregate Diagrams: Show aggregate boundaries and roots.

Conformance Level: High for domain structure and boundaries.

9.3 ArchiMate

ArchiMate provides enterprise architecture notation including business and application layers:

Business Layer: Capabilities, processes, functions.

Application Layer: Application services, components, interfaces.

Conformance Level: High for capability modeling and layered views.

9.4 Notation Comparison

Table 13: Logical Notation Comparison

Feature	UML	DDD	ArchiMate	BPMN	SysML	Custom
Domain modeling	●	●	○	–	○	●
Service definition	○	○	●	○	○	●
Capability mapping	–	–	●	–	○	●
Context boundaries	○	●	●	–	○	●
Entity modeling	●	●	○	–	●	●
Flow modeling	○	–	○	●	●	●
Layering	○	○	●	–	○	●
Standardized	●	○	●	●	●	–

● = Strong support, ○ = Limited support, – = Not applicable

10 Model Correspondence Rules

Model correspondence rules define how elements in logical models relate to elements in other architectural views.

10.1 Development View Correspondence

Correspondence Rule CR-01: Domain to Package Mapping

Rule: Every domain should map to one or more code packages/modules in the development view.

Formal Expression:

$$\forall d \in Domains : \exists P \subseteq Packages : implements(P, d)$$

Rationale: Ensures domain boundaries are reflected in code structure.

Verification: Package structure review.

Correspondence Rule CR-02: Service to Module Mapping

Rule: Every logical service must be implemented by code modules.

Formal Expression:

$$\forall s \in Services : \exists M \subseteq Modules : realizes(M, s)$$

Rationale: Ensures services have implementation.

Verification: Traceability matrix.

10.2 Component-and-Connector View Correspondence

Correspondence Rule CR-03: Service to Component Mapping

Rule: Logical services should map to runtime components.

Formal Expression:

$$\forall s \in Services : \exists c \in Components : manifests(c, s)$$

Rationale: Ensures logical design is realized at runtime.

Verification: Component traceability review.

10.3 Information View Correspondence

Correspondence Rule CR-04: Entity to Data Model Mapping

Rule: Domain entities should have corresponding elements in data models.

Formal Expression:

$$\forall e \in DomainEntities : \exists d \in DataEntities : represents(d, e)$$

Rationale: Ensures domain concepts are persisted.

Verification: Data model review.

11 Operations on Views

This section defines methods for creating, interpreting, analyzing, and maintaining logical views.

11.1 Creation Methods

11.1.1 View Development Process

Step 1: Identify Business Domains

1. Analyze business processes and organizational structure
2. Identify major functional areas
3. Determine core vs supporting vs generic domains
4. Define domain boundaries
5. Document ubiquitous language for each domain

Step 2: Model Domain Entities

1. Identify key domain concepts in each domain
2. Define entity attributes and relationships
3. Identify aggregate boundaries
4. Document entity invariants
5. Define entity lifecycles

Step 3: Define Services

1. Group related operations into services
2. Define service responsibilities
3. Identify service dependencies
4. Document service contracts
5. Assess service autonomy

Step 4: Map Business Capabilities

1. Define capability hierarchy
2. Map services to capabilities
3. Assess capability maturity
4. Identify capability gaps
5. Prioritize capability development

Step 5: Define Context Relationships

1. Identify domain interactions
2. Choose appropriate relationship types
3. Define integration patterns
4. Document shared concepts
5. Plan translation layers

Step 6: Document Business Rules

1. Gather rules from domain experts
2. Classify rules by type
3. Assign rules to domains/entities
4. Define enforcement mechanisms
5. Document exceptions

Step 7: Validate and Refine

1. Review with domain experts
2. Validate against requirements
3. Check for completeness
4. Assess coupling and cohesion
5. Refine based on feedback

11.1.2 Domain Decomposition Strategies

Strategy: Business Capability Decomposition

Approach: Structure domains around business capabilities.

Process:

1. Define L1 business capabilities
2. Decompose into L2, L3 capabilities
3. Group capabilities into domains
4. Align domains with capability ownership

Use When: Strong alignment with business organization needed.

Strategy: Event Storming

Approach: Discover domains through domain events.

Process:

1. Identify domain events (orange stickies)
2. Group events by aggregate (yellow)
3. Identify bounded contexts from clusters
4. Define context relationships

Use When: Collaborative discovery with domain experts.

Strategy: Value Stream Decomposition

Approach: Align domains with value streams.

Process:

1. Map end-to-end value streams
2. Identify value-adding activities
3. Group activities into domains
4. Optimize for flow

Use When: Focus on customer value delivery.

11.2 Analysis Methods

11.2.1 Coupling and Cohesion Analysis

Cohesion Assessment

Purpose: Evaluate how well elements within a domain belong together.

High Cohesion Indicators:

- Elements share common purpose
- Elements change together
- Elements use shared terminology
- Elements owned by same team

Low Cohesion Indicators:

- Mixed responsibilities
- Unrelated elements grouped together
- Different change drivers
- Terminology confusion

Coupling Assessment

Purpose: Evaluate dependencies between domains/services.

Coupling Types (least to most problematic):

1. **Message coupling:** Communication via messages
2. **Data coupling:** Sharing data structures
3. **Control coupling:** Sharing control logic
4. **Content coupling:** Direct internal access

Reduction Strategies:

- Use anti-corruption layers
- Define explicit contracts
- Prefer asynchronous communication
- Duplicate data to reduce runtime coupling

12 Examples

12.1 Example 1: E-Commerce Domain Model

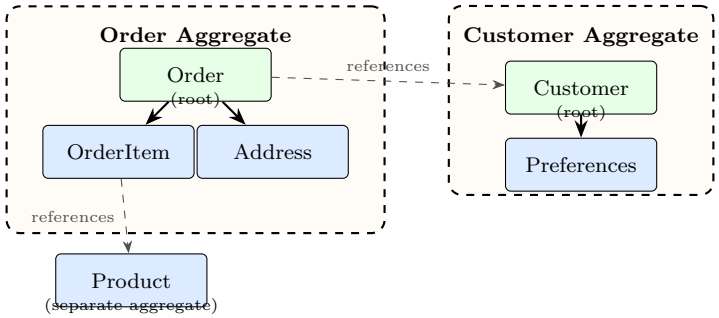


Figure 5: E-Commerce Domain Model with Aggregates

Description: This domain model shows two aggregates: Order and Customer. Order is the aggregate root containing OrderItem and Address. Customer is a separate aggregate referenced by Order (not contained). Product is in a separate aggregate and only referenced by OrderItem via ID.

12.2 Example 2: Bounded Context Map

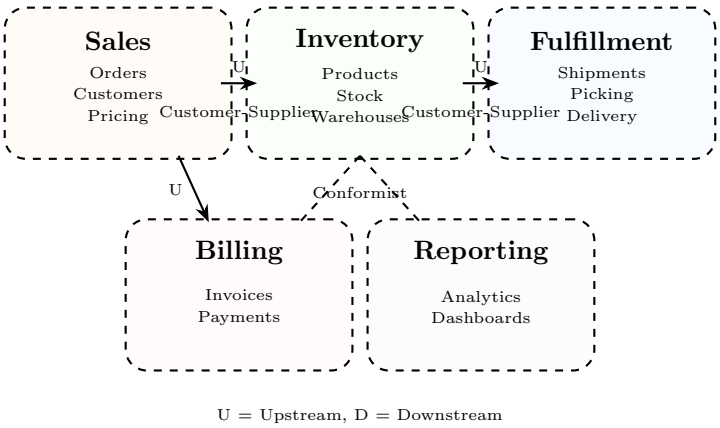


Figure 6: Bounded Context Map

Description: This context map shows five bounded contexts with their relationships. Sales is upstream to Inventory and Billing. Inventory is upstream to Fulfillment. Reporting is a conformist to both Sales and Inventory, meaning it conforms to their models without translation.

12.3 Example 3: Business Capability Heat Map

Table 14: Business Capability Maturity Heat Map

L1 Capability	L2 Capability	Maturity	Strategic	Investment
Customer Mgmt	Acquisition	High	Yes	Maintain
	Retention	Medium	Yes	Grow
	Support	High	No	Maintain
Order Mgmt	Order Entry	High	Yes	Maintain
	Order Tracking	Medium	No	Grow
	Returns	Low	No	Build
Inventory Mgmt	Stock Control	High	No	Maintain
	Forecasting	Low	Yes	Build
	Replenishment	Medium	No	Grow

13 Notes

13.1 Domain-Driven Design Principles

Strategic DDD Guidelines

- **Focus on Core Domain:** Invest most effort in differentiating capabilities
- **Ubiquitous Language:** Use consistent terminology from domain experts
- **Bounded Contexts:** Clearly define model boundaries
- **Context Mapping:** Explicitly define how contexts relate
- **Anti-Corruption Layers:** Protect domain model from external influences
- **Shared Kernel:** Carefully manage shared code between contexts
- **Continuous Refinement:** Evolve model as understanding deepens

13.2 Service Design Principles

Service Design Guidelines

- **Single Responsibility:** Each service has one clear purpose
- **Loose Coupling:** Minimize dependencies between services
- **High Cohesion:** Related operations belong together
- **Explicit Contracts:** Clear interface definitions
- **Encapsulation:** Hide internal implementation
- **Autonomy:** Services can operate independently
- **Statelessness:** Prefer stateless service design where possible

13.3 Common Pitfalls

Common Mistakes to Avoid

1. **Anemic Domain Model:** Entities with only data, no behavior
2. **God Services:** Services doing too much
3. **Leaky Abstractions:** Implementation details in logical models
4. **Ignoring Bounded Contexts:** Trying to create single unified model
5. **Technology-Driven Design:** Letting tools drive domain structure
6. **Missing Ubiquitous Language:** Inconsistent terminology
7. **Premature Decomposition:** Splitting before understanding
8. **Circular Dependencies:** Creating dependency cycles between domains

14 Sources

14.1 Primary References

1. Clements, P., et al. (2010). *Documenting Software Architectures: Views and Beyond* (2nd ed.). Addison-Wesley Professional.
2. Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional.
3. Vernon, V. (2013). *Implementing Domain-Driven Design*. Addison-Wesley Professional.
4. Rozanski, N., & Woods, E. (2011). *Software Systems Architecture* (2nd ed.). Addison-Wesley Professional.
5. Bass, L., Clements, P., & Kazman, R. (2021). *Software Architecture in Practice* (4th ed.). Addison-Wesley Professional.

14.2 Supplementary References

6. Vernon, V. (2016). *Domain-Driven Design Distilled*. Addison-Wesley Professional.
7. Brandolini, A. (2021). *Introducing EventStorming*. Leanpub.
8. Newman, S. (2021). *Building Microservices* (2nd ed.). O'Reilly Media.
9. Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.
10. The Open Group. (2018). *ArchiMate 3.1 Specification*.

14.3 Online Resources

- Domain-Driven Design Reference: https://www.domainlanguage.com/ddd/reference/
- Martin Fowler's Patterns: https://martinfowler.com/

- EventStorming: https://www.eventstorming.com/
- Microservices.io Patterns: https://microservices.io/patterns/
- ArchiMate: https://www.opengroup.org/archimate-forum

A Logical View Checklist

Item	Complete?
Domain Structure	
Business domains identified	☐
Domain boundaries defined	☐
Core/supporting/generic classification done	☐
Ubiquitous language documented	☐
Context relationships mapped	☐
Domain Entities	
Key entities identified	☐
Entity relationships defined	☐
Aggregate boundaries established	☐
Entity invariants documented	☐
Lifecycles defined	☐
Services	
Services identified and documented	☐
Service responsibilities defined	☐
Service dependencies mapped	☐
Contracts documented	☐
Business Alignment	
Capabilities mapped	☐
Business rules documented	☐
Requirements traced	☐
Domain expert review completed	☐
Quality	
Cohesion assessed	☐
Coupling minimized	☐
Dependencies are acyclic	☐
Extension points identified	☐

B Glossary

Aggregate A cluster of domain objects treated as a unit for data changes.

Aggregate Root
The entity that serves as the entry point to an aggregate.

Bounded Context
: A boundary within which a domain model is defined and applicable.

Business Capability
: An ability that a business possesses or needs.

Cohesion
: The degree to which elements of a module belong together.

Context Map
: A diagram showing relationships between bounded contexts.

Coupling
: The degree of interdependence between modules.

Domain
: A sphere of knowledge or activity.

Domain Entity
: An object defined by its identity rather than attributes.

Domain Service
: A service that encapsulates domain logic not belonging to entities.

Functional Element
: A logical unit of system functionality.

Invariant
: A condition that must always be true for an object.

Service
: A logical grouping of operations providing business capabilities.

Ubiquitous Language
: A common language shared by developers and domain experts.

Value Object
: An object defined by its attributes, with no identity.