Software Architecture Documentation

Element Catalog

A Comprehensive Guide to Documenting Architectural Elements, Relations, Interfaces, and Behaviors

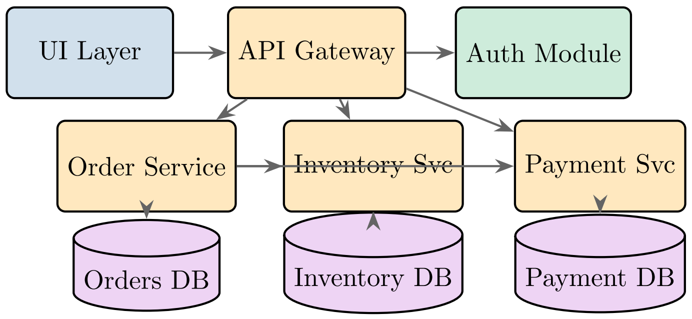
Architecture Documentation Series

Based on SEI Views and Beyond, IEEE 42010, and Industry Best Practices

December 4, 2025

Abstract

The Element Catalog is a critical supporting document that provides detailed specifications for architectural elements depicted in primary architectural views. While diagrams communicate structure and relationships visually, the element catalog captures the rich detail necessary for implementation, analysis, and governance: element properties, interface specifications, behavioral contracts, quality attribute requirements, and traceability information. This comprehensive guide establishes standards and best practices for creating element catalogs that serve as authoritative references throughout the software development lifecycle. The document covers element classification taxonomies, property specification frameworks, relationship documentation patterns, interface contracts, behavioral modeling, and governance processes for maintaining catalog accuracy over time.



Contents

1 Introduction to the Element Catalog

1.1 Definition and Purpose

The Element Catalog is a structured repository of detailed information about architectural elements and their relationships within a specific architectural view. While primary view presentations (diagrams) communicate structure visually, they cannot convey the depth of information necessary for implementation, analysis, and maintenance. The element catalog bridges this gap by providing comprehensive specifications in a consistent, searchable format.

Definition

An **Element Catalog** is a supplementary documentation artifact that records detailed properties, interfaces, behaviors, and relationships for each element appearing in an architectural view, providing the authoritative reference for understanding element specifications and contracts.

The element catalog serves several critical purposes. First, it provides **complete specification** by capturing information that cannot be represented in diagrams, including detailed properties, interface contracts, behavioral constraints, and quality requirements. Second, it enables **analysis support** by providing the data necessary for architectural analysis, including dependency analysis, impact assessment, and quality attribute evaluation. Third, it facilitates **implementation guidance** by giving developers precise specifications for implementing, integrating, and testing architectural elements. Fourth, it supports **governance** by establishing the authoritative source for architectural decisions, enabling conformance checking and change impact analysis. Fifth, it enables **knowledge preservation** by documenting institutional knowledge about element purposes, constraints, and evolution history.

1.2 Relationship to Architectural Views

The element catalog has an intimate relationship with the architectural view it supplements. Every element appearing in a view's primary presentation should have a corresponding entry in the catalog. The view diagram and catalog together form a complete architectural view.

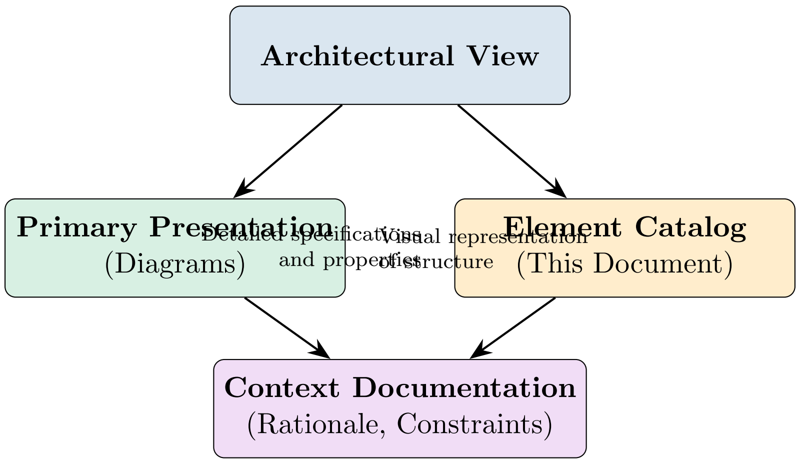


Figure 1: Element Catalog within Architectural View Structure

The catalog complements the primary presentation by providing depth where the diagram provides breadth. A well-designed diagram shows the "what" and "how" of structure at a glance; the catalog explains the "why" and provides the detail necessary for implementation.

1.3 Catalog Scope and Boundaries

Each element catalog is scoped to a specific architectural view. A system with multiple views (module view, component-and-connector view, deployment view) will have multiple element catalogs, each documenting elements appropriate to that view's concerns.

The catalog scope should align with the view scope. For a module decomposition view, the catalog documents modules and their decomposition relationships. For a client-server view, the catalog documents clients, servers, and their interaction protocols. For a deployment view, the catalog documents deployment nodes and artifact mappings.

Key Point

Maintain a one-to-one correspondence between architectural views and element catalogs. Each view has exactly one catalog, and each catalog documents exactly one view. Cross-view relationships should be captured through explicit traceability references rather than duplicating element definitions.

1.4 Standards and Frameworks

The element catalog concept derives from several architectural documentation standards. The SEI "Views and Beyond" approach explicitly defines the element catalog as a required component of every architectural view. IEEE 42010 (Systems and Software Engineering—Architecture Description) establishes the framework for architectural views and viewpoints that catalogs support. ISO/IEC 25010 provides the quality model that informs element quality attribute documentation. ArchiMate provides element taxonomies applicable to enterprise architecture catalogs. The C4 Model defines element hierarchies (System, Container, Component) that inform catalog structure.

2 How to Read and Use This Catalog

2.1 Catalog Organization

This element catalog is organized into four major sections that progressively build understanding of the architectural elements. Section A (Elements and Properties) documents each element's identity, purpose, responsibilities, and key properties. Section B (Relations and Properties) captures the relationships among elements, including their types, directions, and quality requirements. Section C (Element Interfaces) specifies the externally visible contracts through which elements interact. Section D (Element Behavior) describes dynamic aspects including state machines, lifecycle events, and algorithmic responsibilities.

Each section uses consistent formatting and conventions described in this chapter. Readers should familiarize themselves with these conventions before diving into the detailed specifications.

2.2 Naming Conventions

Consistent naming enables efficient navigation and cross-referencing. This catalog uses the following naming conventions.

For **element identifiers**, the format is `<Type>-<Domain>-<Name>-<Version>`. The type prefix indicates the element kind (SVC for service, MOD for module, CMP for component, DAT for data store). The domain indicates the functional area. The name is a descriptive identifier. The version is optional for versioned elements. Examples include `SVC-ORDER-Processing-v2`, `MOD-AUTH-TokenValidator`, and `CMP-UI-ShoppingCart`.

For **interface identifiers**, the format is `<Element>.<Interface>`. Examples include `SVC-ORDER-Processing.`
and `MOD-AUTH-TokenValidator.ValidateToken`.

For **relation identifiers**, the format is `REL-<Source>-<Target>-<Type>`. Examples include `REL-OrderSvc-InventorySvc-Calls` and `REL-WebApp-AuthMod-DependsOn`.

2.3 Cross-Reference Notation

Cross-references to other documents, views, and artifacts use the following notation. References to other views appear as `[VIEW:<ViewName>]`, such as `[VIEW:Deployment]` or `[VIEW:ModuleDecomposition]`. References to requirements appear as `[REQ:<RequirementID>]`, such as `[REQ:FR-101]` or `[REQ:NFR-PERF-03]`. References to implementation artifacts appear as `[IMPL:<ArtifactPath>]`, such as `[IMPL:src/services/order/]`
References to external documentation appear as `[DOC:<DocumentID>]`, such as `[DOC:API-Spec-v2.1]`.

2.4 Property Value Conventions

Properties throughout the catalog use standardized value formats. Criticality levels use **Critical** (system failure if unavailable), **High** (significant degradation if unavailable), **Medium** (partial functionality affected), and **Low** (minimal impact if unavailable). Availability targets use percentage notation such as 99.99%, 99.9%, or 99%. Latency requirements use millisecond notation such as <50ms, <200ms, or <1s. Throughput requirements use rate notation such as 1000 req/s, 10K msg/min, or 1M records/day. Data volumes use standard units such as GB, TB, or records/day.

2.5 Reading Strategies

Different readers will approach this catalog with different goals. Developers implementing a specific element should start with that element's entry in Section A, review its interfaces in Section C, and understand its behavioral contracts in Section D. Integration engineers should focus on Section B (Relations) and Section C (Interfaces) to understand how elements connect. Architects performing analysis should review all sections, paying particular attention to criticality ratings, quality requirements, and dependency patterns. Operators should focus on criticality, availability targets, and behavioral aspects related to operational concerns.

3 Element Classification and Taxonomy

3.1 Element Type Hierarchy

Architectural elements can be classified into a taxonomy that aids understanding and enables consistent documentation. The following hierarchy provides a comprehensive classification scheme.

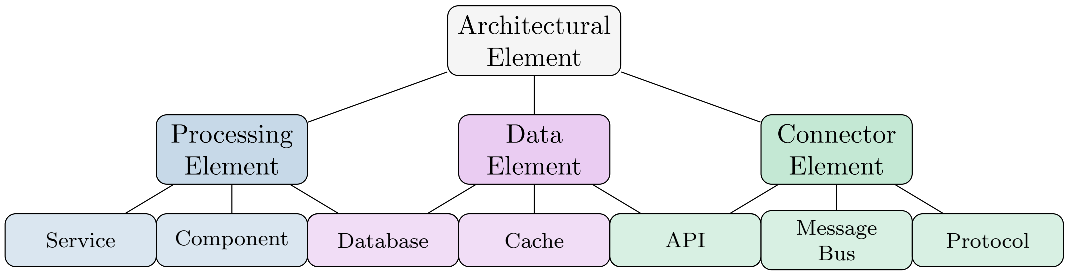


Figure 2: Element Type Taxonomy

3.1.1 Processing Elements

Processing elements perform computation, transformation, or coordination. **Services** are independently deployable units providing business capabilities through well-defined interfaces, typically in service-oriented or microservices architectures. **Components** are modular units of functionality with explicit interfaces, deployable as part of a larger application. **Modules** are logical groupings of code organized by responsibility or domain, representing compile-time structure. **Functions** are discrete units of computation, particularly relevant in serverless architectures. **Processes** are runtime execution units that may contain multiple threads or components.

3.1.2 Data Elements

Data elements store, manage, or transport information. **Databases** provide persistent storage with query capabilities, including relational, document, graph, and time-series variants. **Caches** provide high-speed temporary storage for frequently accessed data. **Message Queues** provide asynchronous message storage and delivery. **File Stores** provide unstructured or semi-structured data storage. **Data Streams** provide continuous data flow for real-time processing.

3.1.3 Connector Elements

Connector elements facilitate interaction between other elements. **APIs** define synchronous request-response interfaces. **Message Buses** provide publish-subscribe or point-to-point asynchronous messaging. **Event Brokers** manage event distribution and subscription. **Load Balancers** distribute requests across multiple instances. **Service Meshes** provide inter-service communication infrastructure.

3.2 View-Specific Element Types

Different architectural views emphasize different element types. The following table maps common views to their typical element types.

Table 1: View-Specific Element Types

Architectural View	**Primary Elements**	**Primary Relations**
Module Decomposition	Modules, Packages, Classes	Contains, Uses, Depends-on

Continued on next page

Architectural View	Primary Elements	Primary Relations
Component-Connector	Components, Connectors, Ports	Attaches, Connects, Flows-to
Client-Server	Clients, Servers, Tiers	Requests, Responds, Invokes
Service-Oriented	Services, Endpoints, Contracts	Calls, Publishes, Subscribes
Deployment	Nodes, Artifacts, Environments	Deploys-to, Hosts, Manifests
Data Flow	Processes, Data Stores, Flows	Reads, Writes, Transforms
Layered	Layers, Modules within Layers	Uses (constrained by layer rules)
Microservices	Services, APIs, Data Stores	Calls, Events, Sagas

3.3 Element Stereotypes

Within each element type, stereotypes provide additional classification. Stereotypes communicate patterns, roles, or implementation guidance.

Example

Service Stereotypes:

- `<<aggregate-root>>` – Service managing a DDD aggregate
- `<<gateway>>` – Entry point service handling cross-cutting concerns
- `<<orchestrator>>` – Service coordinating multi-step workflows
- `<<adapter>>` – Service adapting external system interfaces
- `<<facade>>` – Service simplifying complex subsystem access

Data Store Stereotypes:

- `<<system-of-record>>` – Authoritative source for specific data
- `<<read-replica>>` – Read-optimized copy of data
- `<<event-store>>` – Append-only event persistence
- `<<cache>>` – Temporary high-speed storage

4 Section A: Elements and Their Properties

4.1 Element Documentation Framework

Each element in the catalog receives comprehensive documentation following a consistent framework. This framework ensures that all information necessary for understanding, implementing, and maintaining the element is captured systematically.

Template

Element Specification Template

Element Identity
- **ID:** Unique identifier following naming conventions
- **Name:** Human-readable name
- **Type:** Classification from taxonomy
- **Stereotype:** Optional pattern or role indicator
- **Version:** Current version number

Description and Responsibility
- **Purpose:** Why this element exists
- **Responsibilities:** What this element does (and does not do)
- **Domain:** Business or technical domain

Properties
- **Criticality:** Impact of failure
- **Quality Attributes:** Performance, availability, security requirements
- **Technology Stack:** Implementation technologies
- **Constraints:** Limitations and restrictions

Traceability
- **Requirements:** Linked requirements
- **Implementation:** Source code or artifact references
- **Related Views:** Appearances in other views

4.2 Identity Properties

Identity properties uniquely identify and classify the element.

4.2.1 Element Identifier

The element identifier (ID) must be globally unique within the architecture and stable across versions. IDs should be meaningful (conveying type and purpose) yet concise. Once assigned, IDs should not change; if an element is fundamentally redesigned, it should receive a new ID.

4.2.2 Versioning

Elements evolve over time. The catalog should track version information including the current version number (following semantic versioning where applicable), version history summary, compatibility notes (backward/forward compatibility with previous versions), and deprecation status if applicable.

Table 2: Element Version History Example

Version	Date	Changes	Compatibility
1.0.0	2024-01-15	Initial release	N/A
1.1.0	2024-03-20	Added batch processing capability	Backward compatible

Version	Date	Changes	Compatibility
2.0.0	2024-06-01	Redesigned API; new data model	Breaking changes; migration required
2.0.1	2024-06-15	Bug fixes for edge cases	Fully compatible with 2.0.0

4.3 Description and Responsibilities

The description section captures the element's purpose and scope through several components.

4.3.1 Purpose Statement

The purpose statement explains why the element exists—what problem it solves or what value it provides. A good purpose statement is concise (1-3 sentences), business-oriented (explaining value, not just function), and unique (distinguishing this element from others).

Example

Good Purpose Statement: "The Order Processing Service validates customer orders, coordinates inventory reservation, and initiates payment processing to enable reliable order fulfillment within the e-commerce platform."
Poor Purpose Statement: "This service processes orders." (Too vague, doesn't explain value or distinguish from other services)

4.3.2 Responsibilities

Responsibilities define what the element does and, importantly, what it does not do. Following the Single Responsibility Principle, each element should have a cohesive set of responsibilities. Document responsibilities as a list of specific duties. Include explicit non-responsibilities to prevent scope creep. Explain the rationale for responsibility boundaries.

Example

Order Processing Service Responsibilities:

- Validate order requests against business rules
- Calculate order totals including taxes and discounts
- Coordinate with Inventory Service to reserve stock
- Initiate payment processing via Payment Service
- Persist order records to Order Database
- Publish order events for downstream processing

Non-Responsibilities (handled by other elements):

- Inventory management (Inventory Service)
- Payment processing logic (Payment Service)
- Shipping logistics (Fulfillment Service)
- Customer notifications (Notification Service)

4.4 Element Properties

Properties capture measurable or classifiable characteristics of the element.

4.4.1 Criticality Assessment

Criticality indicates the business impact of element failure. Use a consistent scale across all elements.

Table 3: Criticality Classification Scale

Level	**Business Impact**	**Example Consequences**	**Response**
criticalred!20**Critical**	Complete business stoppage	No orders processed; revenue loss; regulatory violation	Immediate escalation; 24/7 support; redundancy required
highyellow!30**High**	Significant degradation	Major features unavailable; customer-facing impact	Urgent response; business hours escalation
mediumblue!20**Medium**	Partial functionality loss	Some features degraded; workarounds available	Standard response; next business day
lowgray!20**Low**	Minimal impact	Internal tools affected; no customer impact	Scheduled maintenance acceptable

4.4.2 Quality Attribute Requirements

Each element may have specific quality attribute requirements that constrain its design and implementation.

Table 4: Element Quality Attribute Template

Attribute	**Requirement**	**Measure**	**Verification**
Performance	Response time	95th percentile < 200ms	Load testing
Availability	Uptime	99.9% monthly	Monitoring
Scalability	Throughput	10,000 req/sec at peak	Capacity testing
Security	Data protection	PII encrypted at rest/transit	Security audit
Reliability	Error rate	< 0.1% failed requests	Error tracking
Maintainability	Code coverage	> 80% unit test coverage	CI pipeline

4.4.3 Technology Stack

Document the technology choices for each element to support implementation and operational concerns.

Table 5: Technology Stack Documentation

Category	**Technology**	**Rationale/Notes**
Language	Java 17	Standard platform; team expertise
Framework	Spring Boot 3.x	Mature ecosystem; microservices support
Database	PostgreSQL 15	ACID compliance; complex queries
Cache	Redis 7.x	High-performance caching; pub/sub
Message Queue	Apache Kafka	Event streaming; high throughput
Container Runtime	Docker / Kubernetes	Standard deployment; orchestration
Observability	OpenTelemetry, Prometheus	Distributed tracing; metrics

4.4.4 Constraints

Constraints document limitations, restrictions, and assumptions that affect the element.

- **Technical Constraints:** Platform limitations, technology mandates, integration requirements
- **Business Constraints:** Regulatory requirements, licensing restrictions, budget limitations
- **Operational Constraints:** Deployment windows, geographic restrictions, support requirements
- **Design Constraints:** Architectural patterns required, interfaces that must be preserved

4.5 Comprehensive Element Registry

The element registry provides a summary table of all elements, enabling quick reference and navigation.

Table 6: Element Registry

Element ID	**Type**	**Critical.**	**Technology**	**Primary Responsibility**
SVC-API-Gateway	Service	Critical	Node.js, Kong	Route requests; authentication; rate limiting
SVC-ORDER-Process	Service	Critical	Java, Spring	Order validation, coordination, persistence
SVC-INV-Manager	Service	High	Go	Inventory tracking, reservation, alerts
SVC-PAY-Process	Service	Critical	Java, Spring	Payment processing, refunds, reconciliation
SVC-USER-Auth	Service	Critical	Go, OAuth2	Authentication, authorization, token management
SVC-NOTIFY-Send	Service	Medium	Python	Email, SMS, push notifications
DAT-ORDER-DB	Database	Critical	PostgreSQL	Order records, transaction history
DAT-INV-DB	Database	High	PostgreSQL	Inventory levels, locations, movements
DAT-USER-DB	Database	Critical	PostgreSQL	User accounts, profiles, credentials
DAT-CACHE-Redis	Cache	High	Redis	Session data, API responses, rate limits
MSG-EVENT-Bus	Queue	High	Kafka	Async event distribution, event sourcing

4.6 Detailed Element Specifications

Following the registry, provide detailed specifications for each element. The following example demonstrates the full documentation format.

Element: SVC-ORDER-Process

Identity

- **ID:** SVC-ORDER-Process
- **Name:** Order Processing Service
- **Type:** Service
- **Stereotype:** `<<orchestrator>>`
- **Version:** 2.3.1

Purpose

Orchestrates the order lifecycle from submission through fulfillment initiation, ensuring business rules are enforced, inventory is reserved, and payment is processed before committing orders.

Responsibilities

- Validate order requests against business rules and customer eligibility
- Calculate order totals including line items, taxes, discounts, and shipping
- Coordinate with Inventory Service to reserve stock (saga participant)
- Initiate payment authorization via Payment Service
- Persist order records with full audit trail
- Publish OrderCreated, OrderUpdated, OrderCancelled events
- Handle order modifications and cancellations within policy windows

Non-Responsibilities

- Inventory management and warehouse operations (SVC-INV-Manager)
- Payment gateway integration and PCI compliance (SVC-PAY-Process)
- Shipping carrier integration (SVC-FULFILL-Ship)
- Customer communication (SVC-NOTIFY-Send)

Properties

- **Criticality:** Critical—order processing is core revenue path
- **Availability Target:** 99.95% (4.4 hours downtime/year max)
- **Performance:** Order creation < 500ms (95th percentile)
- **Throughput:** 1,000 orders/minute sustained; 5,000/minute peak
- **Data Volume:** 50KB per order; 2M orders/month
- **Security:** PII handling; SOC 2 compliant

Technology Stack

- Runtime: Java 17 on Spring Boot 3.2
- Database: PostgreSQL 15 (primary); Redis (caching)
- Messaging: Apache Kafka (event publishing)
- Deployment: Kubernetes (3 replicas minimum)

Constraints

- Must maintain backward compatibility with mobile app v3.x
- Cannot store payment card numbers (PCI-DSS)
- Must support idempotent order creation for retry safety
- Geographic data residency: orders from EU stored in EU region

Traceability

- Requirements: [REQ:FR-ORD-001] through [REQ:FR-ORD-050]
- Implementation: [IMPL:services/order-service/]
- Deployment View: [VIEW:Deployment] Node: order-service-cluster
- API Documentation: [DOC:API-Order-v2.3]

5 Section B: Relations and Their Properties

5.1 Relationship Type Taxonomy

Relationships between elements capture how elements interact, depend on, or contain one another. Understanding relationship types is essential for dependency analysis, impact assessment, and implementation planning.

5.1.1 Structural Relationships

Structural relationships describe static organization.

Table 7: Structural Relationship Types

Relation	**Notation**	**Description**
Contains	A `contains` B	A is composed of B; B exists only within A's context
Uses	A `uses` B	A requires B's functionality; compile/build-time dependency
Depends-on	A `depends-on` B	A requires B at runtime; A cannot function without B
Extends	A `extends` B	A specializes or inherits from B
Implements	A `implements` B	A provides concrete realization of B (interface)
Aggregates	A `aggregates` B	A references B; B can exist independently

5.1.2 Behavioral Relationships

Behavioral relationships describe runtime interactions.

Table 8: Behavioral Relationship Types

Relation	**Notation**	**Description**
Calls	A `calls` B	A invokes B's interface synchronously
Sends	A `sends` B	A transmits message/data to B asynchronously
Publishes-to	A `publishes-to` B	A emits events that B distributes
Subscribes-to	A `subscribes-to` B	A receives events from B
Reads-from	A `reads-from` B	A retrieves data from B (data store)
Writes-to	A `writes-to` B	A persists data to B (data store)
Triggers	A `triggers` B	A causes B to execute (event-driven)

5.1.3 Deployment Relationships

Deployment relationships describe how elements map to infrastructure.

Table 9: Deployment Relationship Types

Relation	Notation	Description
Deploys-to	A `deploys-to` B	Artifact A is deployed on node B
Runs-on	A `runs-on` B	Process A executes on platform B
Hosts	A `hosts` B	Node A provides runtime for B
Connects-to	A `connects-to` B	Node A has network path to node B

5.2 Relationship Properties

Each relationship should be documented with properties that enable analysis and implementation.

Template

Relationship Specification Template

Identity

- **ID:** Unique relationship identifier
- **Source:** Originating element
- **Target:** Destination element
- **Type:** Relationship classification

Interaction Properties

- **Direction:** Unidirectional / Bidirectional
- **Cardinality:** 1:1, 1:N, N:M
- **Timing:** Synchronous / Asynchronous
- **Protocol:** Communication mechanism
- **Data Format:** Message/payload format

Quality Requirements

- **Latency:** Response time requirement
- **Throughput:** Volume requirement
- **Reliability:** Delivery guarantee
- **Security:** Authentication/encryption

Failure Handling

- **Timeout:** Maximum wait time
- **Retry Policy:** Retry strategy
- **Fallback:** Degraded behavior
- **Circuit Breaker:** Failure threshold

5.3 Relationship Registry

The relationship registry provides a comprehensive view of all element relationships.

Table 10: Relationship Registry

Source	Target	Type	Protocol	Key Properties
API-Gateway	ORDER-Svc	calls	HTTP/REST	Sync; <100ms; JWT auth; retry 3x
API-Gateway	USER-Auth	calls	HTTP/REST	Sync; <50ms; mTLS; critical path
ORDER-Svc	INV-Svc	calls	gRPC	Sync; <200ms; saga participant
ORDER-Svc	PAY-Svc	calls	HTTP/REST	Sync; <5s timeout; idempotent
ORDER-Svc	ORDER-DB	writes	PostgreSQL	Transactional; ACID; 10K TPS
ORDER-Svc	EVENT-Bus	publishes	Kafka	Async; at-least-once; ordered
INV-Svc	INV-DB	reads/writes	PostgreSQL	Read replicas; 50K queries/sec
INV-Svc	EVENT-Bus	subscribes	Kafka	Consumer group; offset tracking
NOTIFY-Svc	EVENT-Bus	subscribes	Kafka	Non-critical; best-effort delivery
CACHE-Redis	All Services	reads	Redis Protocol	<5ms; TTL-based expiry

5.4 Dependency Analysis

The relationship registry enables important dependency analyses.

5.4.1 Direct Dependencies

For each element, identify what it depends on (upstream) and what depends on it (downstream).

Table 11: Element Dependency Summary

Element	Depends On (Upstream)	Depended By (Downstream)
ORDER-Svc	INV-Svc, PAY-Svc, ORDER-DB, EVENT-Bus, CACHE	API-Gateway
INV-Svc	INV-DB, EVENT-Bus	ORDER-Svc, FULFILL-Svc
PAY-Svc	PAY-DB, External Payment Gateway	ORDER-Svc

Element	Depends On (Upstream)	Depended By (Downstream)
USER-Auth	USER-DB, CACHE	API-Gateway, All Services
EVENT-Bus	Kafka Cluster	ORDER-Svc, INV-Svc, NOTIFY-Svc

5.4.2 Transitive Dependencies

Identify chains of dependencies that create indirect coupling.

Key Point

Transitive dependencies can create unexpected failure modes. If A $\rightarrow$ B $\rightarrow$ C, then C's failure can impact A even though A has no direct relationship with C. Map transitive dependencies for critical paths and consider circuit breakers at trust boundaries.

5.4.3 Circular Dependencies

Identify and document any circular dependencies, which often indicate design issues.

Warning

Circular dependencies (A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A) create tight coupling, complicate deployment ordering, and can cause deadlocks or infinite loops. If circular dependencies exist, document them explicitly and plan for resolution.

6 Section C: Element Interfaces

6.1 Interface Classification

Interfaces define the contracts through which elements interact. Proper interface documentation is essential for integration, testing, and evolution.

6.1.1 Interface Types

Table 12: Interface Type Classification

Type	Direction	Description
Provided	Outward	Interface exposed by element for others to consume
Required	Inward	Interface that element needs from other elements
Event (Published)	Outward	Events emitted by element for subscribers

Type	Direction	Description
Event (Subscribed)	Inward	Events element listens for from publishers
Data (Export)	Outward	Data made available for external consumption
Data (Import)	Inward	Data consumed from external sources

6.1.2 Interface Styles

Different interface styles suit different interaction patterns.

Table 13: Interface Styles

Style	Pattern	Characteristics	Use Cases
REST API	Request-Response	Stateless; resource-oriented; HTTP verbs	CRUD operations; web APIs
GraphQL	Request-Response	Flexible queries; single endpoint	Complex data needs; mobile
gRPC	Request-Response	Binary protocol; strongly typed; streaming	High-performance; internal
Message Queue	Async Messaging	Decoupled; persistent; guaranteed delivery	Event-driven; batch processing
WebSocket	Bidirectional Stream	Real-time; persistent connection	Live updates; chat; gaming
File/Batch	Batch Transfer	Large volumes; scheduled	ETL; reporting; integration

6.2 Interface Summary Registry

The interface summary provides a quick reference to all element interfaces.

Table 14: Interface Registry

Element	Interface	Kind	Style	Purpose
ORDER-Svc	OrderAPI	Provided	REST	Create, read, update, cancel orders

Continued on next page

Element	**Interface**	**Kind**	**Style**	**Purpose**
ORDER-Svc	OrderEvents	Published	Kafka	Order lifecycle events
ORDER-Svc	InventoryClient	Required	gRPC	Reserve/release inventory
ORDER-Svc	PaymentClient	Required	REST	Process payments
INV-Svc	InventoryAPI	Provided	gRPC	Query/update inventory
INV-Svc	StockEvents	Published	Kafka	Stock level changes
PAY-Svc	PaymentAPI	Provided	REST	Payment processing
PAY-Svc	PaymentWebhook	Provided	HTTP	Payment status callbacks
USER-Auth	AuthAPI	Provided	REST	Login, token refresh
USER-Auth	TokenValidation	Provided	gRPC	Validate JWT tokens

6.3 Detailed Interface Specifications

For each significant interface, provide comprehensive documentation.

Interface: ORDER-Svc.OrderAPI

Identity

- **Owner Element:** SVC-ORDER-Process
- **Interface ID:** ORDER-Svc.OrderAPI
- **Type:** Provided (REST API)
- **Version:** v2.3
- **Base URL:** `/api/v2/orders`

Operations

Operation	Method	Path	Description
CreateOrder	POST	`/orders`	Create new order
GetOrder	GET	`/orders/{id}`	Retrieve order by ID
ListOrders	GET	`/orders`	List orders with filters
UpdateOrder	PATCH	`/orders/{id}`	Modify order details
CancelOrder	DELETE	`/orders/{id}`	Cancel pending order
GetOrderStatus	GET	`/orders/{id}/status`	Get current status

Operation Detail: CreateOrder

- **Method:** POST
- **Path:** `/orders`
- **Request Body:** OrderCreateRequest (JSON)
- **Response:** OrderResponse (JSON), 201 Created
- **Idempotency:** Supported via `X-Idempotency-Key` header
- **Rate Limit:** 100 requests/minute per user

Authentication & Authorization

- **Authentication:** Bearer JWT token (from USER-Auth)
- **Authorization:** Role-based; `orders:write` scope required for mutations
- **Rate Limiting:** Per-user and per-IP limits enforced

Error Responses

Code	Error	Description
400	ValidationError	Invalid request body or parameters
401	Unauthorized	Missing or invalid authentication
403	Forbidden	Insufficient permissions
404	NotFound	Order does not exist
409	Conflict	Order state prevents operation
422	BusinessRuleViolation	Business rule prevented operation
429	RateLimitExceeded	Too many requests
500	InternalError	Unexpected server error
503	ServiceUnavailable	Dependency unavailable

Quality Requirements

- **Latency:** CreateOrder $<$ 500ms (95th percentile)
- **Availability:** 99.95% uptime
- **Throughput:** 1,000 requests/second sustained

Documentation

6.4 Event Interface Specifications

Event interfaces require different documentation than request-response APIs.

Interface: ORDER-Svc.OrderEvents

Identity

- **Owner Element:** SVC-ORDER-Process
- **Interface ID:** ORDER-Svc.OrderEvents
- **Type:** Published Events (Kafka)
- **Topic:** `orders.events`
- **Schema Registry:** `schema-registry.internal/subjects/orders-value`

Event Types

Event Type	Trigger	Key Payload Fields
OrderCreated	New order submitted	orderId, customerId, items, total
OrderConfirmed	Payment successful	orderId, confirmedAt, paymentId
OrderShipped	Shipment created	orderId, trackingNumber, carrier
OrderDelivered	Delivery confirmed	orderId, deliveredAt, signature
OrderCancelled	Order cancelled	orderId, cancelledAt, reason
OrderRefunded	Refund processed	orderId, refundAmount, refundId

Event Schema (OrderCreated)

```
{
  "eventId": "uuid",
  "eventType": "OrderCreated",
  "eventTime": "ISO-8601 timestamp",
  "version": "2.0",
  "data": {
    "orderId": "uuid",
    "customerId": "uuid",
    "items": [...],
    "totalAmount": { "value": 99.99, "currency": "USD" },
    "createdAt": "ISO-8601 timestamp"
  }
}
```

Delivery Guarantees

- **Delivery:** At-least-once (consumers must be idempotent)
- **Ordering:** Per-partition ordering; keyed by orderId
- **Retention:** 7 days
- **Partitions:** 12 (allows 12 parallel consumers)

Consumers

- SVC-NOTIFY-Send (notifications)
- SVC-ANALYTICS-Ingest (analytics pipeline)
- SVC-FULFILL-Process (fulfillment initiation)

6.5 Interface Compatibility and Versioning

Document interface versioning strategy and compatibility guarantees.

Best Practice

Interface Versioning Guidelines:

- Use semantic versioning for interface versions
- Maintain backward compatibility within major versions
- Support N-1 version for minimum transition period
- Deprecate with minimum 6-month notice before removal
- Document breaking changes clearly in changelog
- Provide migration guides for major version upgrades

7 Section D: Element Behavior

7.1 Behavioral Documentation Overview

Element behavior captures the dynamic aspects of elements—how they respond to inputs, transition between states, and maintain invariants over time. This section complements the static structure documented elsewhere.

7.2 State and Lifecycle Models

Many elements have significant state that affects their behavior. Documenting states and transitions is essential for understanding element behavior.

7.2.1 State Machine Documentation

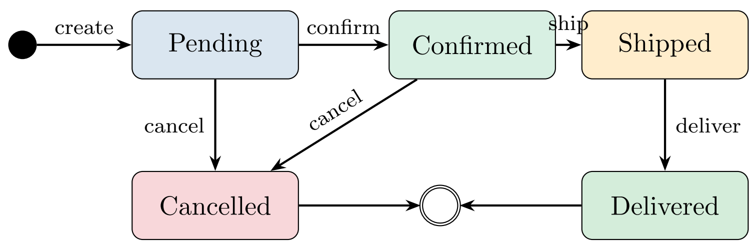


Figure 3: Order Lifecycle State Machine

Table 15: Order State Transition Table

From State	Event/Trigger	To State	Actions/Guards
(Initial)	CreateOrder	Pending	Validate order; reserve inventory
Pending	PaymentSucceeded	Confirmed	Record payment; notify customer

From State	**Event/Trigger**	**To State**	**Actions/Guards**
Pending	PaymentFailed	Pending	Retry payment; notify if max retries
Pending	CancelOrder	Cancelled	Guard: within cancel window; release inventory
Confirmed	ShipmentCreated	Shipped	Record tracking; notify customer
Confirmed	CancelOrder	Cancelled	Guard: not yet shipped; process refund
Shipped	DeliveryConfirmed	Delivered	Record delivery; close order
Delivered	(none)	(Final)	Order complete
Cancelled	(none)	(Final)	Order terminated

7.2.2 State Invariants

Document properties that must hold true in each state.

Table 16: Order State Invariants

State	**Invariants**
Pending	Inventory reserved; payment not processed; cancel allowed
Confirmed	Payment captured; inventory committed; shipment allowed
Shipped	Tracking number assigned; delivery pending; cancel not allowed
Delivered	Delivery timestamp recorded; all quantities accounted
Cancelled	Inventory released; payment refunded (if captured); reason recorded

7.3 Behavioral Responsibilities

Document key algorithms, processing rules, and behavioral contracts.

Behavioral Specification: Order Total Calculation

Element: SVC-ORDER-Process
Responsibility: Calculate accurate order totals

Algorithm:

1. Sum line item subtotals (quantity × unit price)
2. Apply item-level discounts (percentage or fixed)
3. Calculate subtotal after item discounts
4. Apply order-level promotions (coupon codes, loyalty points)
5. Calculate shipping cost based on destination and method
6. Calculate tax based on shipping destination and product categories
7. Sum components: subtotal + shipping + tax - order discounts

Business Rules:

- Maximum one coupon code per order
- Loyalty points valued at $0.01 per point
- Tax calculation delegated to tax service for accuracy
- Free shipping for orders over $50 (before tax)
- All monetary calculations use banker's rounding

Preconditions:

- All line items have valid product IDs and prices
- Shipping address is validated and complete
- Customer eligibility for promotions verified

Postconditions:

- Order total equals sum of components (auditable)
- All applied discounts recorded with reasons
- Tax breakdown recorded by jurisdiction

Performance: Calculation completes in < 50ms for orders up to 100 items

7.4 Concurrency and Synchronization

Document how elements handle concurrent access and maintain consistency.

Table 17: Concurrency Handling

Element	**Mechanism**	**Description**
ORDER-Svc	Optimistic Locking	Version field prevents lost updates; retry on conflict
INV-Svc	Database Locks	Row-level locks during reservation; timeout after 5s
PAY-Svc	Idempotency Keys	Client-provided keys prevent duplicate charges
EVENT-Bus	Partition Keys	Same order always goes to same partition

Element	**Mechanism**	**Description**
CACHE-Redis	Atomic Operations	SETNX for distributed locks; TTL prevents deadlocks

7.5 Failure Modes and Recovery

Document how elements behave when things go wrong.

Table 18: Failure Mode Analysis

Element	**Failure Mode**	**Detection**	**Recovery**
ORDER-Svc	Database unavailable	Connection timeout	Circuit breaker; queue requests; alert
ORDER-Svc	INV-Svc unavailable	gRPC timeout	Degrade to cached inventory; flag for reconciliation
ORDER-Svc	PAY-Svc timeout	HTTP 504	Retry with idempotency key; status: payment_pending
INV-Svc	Oversell detected	Negative stock count	Backorder; notify customer; alert operations
PAY-Svc	Gateway rejection	4xx response	Return error to caller; log for analysis
EVENT-Bus	Consumer lag	Lag metric threshold	Scale consumers; alert if lag exceeds SLA

7.6 Behavioral References

Link to detailed behavioral models maintained separately.

Table 19: Behavioral Model References

Model	**Elements Covered**	**Type**	**Reference**
Order Creation Flow	ORDER, INV, PAY	Sequence Diagram	[DOC:seq-order-create]
Order Cancellation	ORDER, INV, PAY	Sequence Diagram	[DOC:seq-order-cancel]

Model	Elements Covered	Type	Reference
Saga: Order Fulfillment	ORDER, INV, PAY, FULFILL	Saga Diagram	[DOC:saga-fulfillment]
Inventory Reservation	INV-Svc	State Machine	[DOC:state-inventory]
Payment Processing	PAY-Svc	Activity Diagram	[DOC:activity-payment]
Authentication Flow	USER-Auth	Sequence Diagram	[DOC:seq-auth-flow]

8 Traceability and Cross-References

8.1 Traceability Framework

Traceability connects catalog elements to other artifacts, enabling impact analysis, compliance verification, and knowledge navigation.

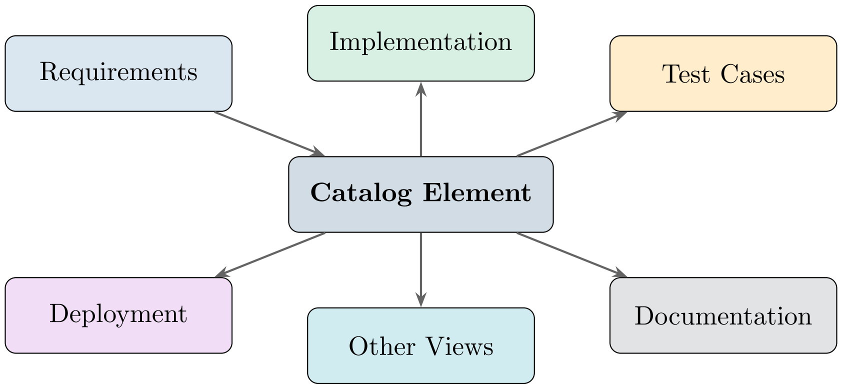


Figure 4: Element Traceability Relationships

8.2 Requirements Traceability

Map elements to the requirements they satisfy.

Table 20: Requirements Traceability Matrix

Element	Functional Requirements	Non-Functional Requirements
ORDER-Svc	FR-ORD-001 through FR-ORD-050	NFR-PERF-01, NFR-AVAIL-01, NFR-SEC-03
INV-Svc	FR-INV-001 through FR-INV-025	NFR-PERF-02, NFR-AVAIL-02

Element	Functional Requirements	Non-Functional Requirements
PAY-Svc	FR-PAY-001 through FR-PAY-030	NFR-SEC-01, NFR-SEC-02, NFR-COMPL-01
USER-Auth	FR-AUTH-001 through FR-AUTH-020	NFR-SEC-04, NFR-PERF-03
EVENT-Bus	FR-EVT-001 through FR-EVT-010	NFR-REL-01, NFR-SCAL-01

8.3 Implementation Mapping

Map elements to source code and deployment artifacts.

Table 21: Implementation Artifact Mapping

Element	Source Repository	Package/Module	Artifact
ORDER-Svc	github.com/co/order-service	com.company.order	order-service.jar
INV-Svc	github.com/co/inventory-svc	inventory	inventory-svc (Go binary)
PAY-Svc	github.com/co/payment-svc	com.company.payment	payment-service.jar
USER-Auth	github.com/co/auth-service	auth	auth-service (Go binary)
API-Gateway	github.com/co/api-gateway	api-gateway	Docker: api-gateway

8.4 Cross-View References

Map elements to their appearances in other architectural views.

Table 22: Cross-View Element Mapping

Element	This View	Other View	Representation
ORDER-Svc	Component-Connector	Module View	order-service module
ORDER-Svc	Component-Connector	Deployment View	order-svc-cluster pod
ORDER-Svc	Component-Connector	Data Flow View	Order Processing process

Element	This View	Other View	Representation
ORDER-DB	Component-Connector	Deployment View	orders-db-primary node
EVENT-Bus	Component-Connector	Deployment View	kafka-cluster

8.5 Decision Traceability

Link elements to architectural decisions that shaped them.

Table 23: Architectural Decision References

Element	ADR ID	Decision Summary
ORDER-Svc	ADR-005	Use saga pattern for distributed transactions
EVENT-Bus	ADR-008	Adopt Apache Kafka for event streaming
PAY-Svc	ADR-012	Isolate payment processing for PCI compliance
USER-Auth	ADR-003	Implement OAuth 2.0 with JWT tokens
All Services	ADR-001	Adopt microservices architecture
All Services	ADR-015	Use gRPC for internal service communication

9 Catalog Governance and Maintenance

9.1 Ownership and Accountability

Each element should have clear ownership for maintenance and evolution.

Table 24: Element Ownership Registry

Element	Owning Team	Technical Owner	Contact
ORDER-Svc	Order Team	J. Smith	order-team@company.com
INV-Svc	Inventory Team	A. Jones	inventory-team@company.com
PAY-Svc	Payments Team	B. Chen	payments-team@company.com
USER-Auth	Platform Team	M. Garcia	platform-team@company.com
API-Gateway	Platform Team	M. Garcia	platform-team@company.com

Element	Owning Team	Technical Owner	Contact
EVENT-Bus	Platform Team	K. Patel	platform-team@company.com

9.2 Change Management Process

Changes to catalog entries should follow a controlled process.

1. **Propose:** Submit change request with rationale and impact analysis
2. **Review:** Architecture review board evaluates change
3. **Approve:** Obtain approval from element owner and affected stakeholders
4. **Implement:** Update catalog entry with version increment
5. **Validate:** Verify catalog matches implementation
6. **Communicate:** Notify dependent teams of changes

9.3 Quality Assurance

Maintain catalog quality through regular validation.

Best Practice

Catalog Quality Checklist:

- All elements in diagrams have catalog entries
- All catalog entries correspond to actual elements
- No orphan elements (in catalog but not in any diagram)
- No phantom elements (in diagram but not in catalog)
- Interface specifications match implementation
- Cross-references are valid and current
- Version information is accurate
- Ownership information is current

9.4 Catalog Version History

Table 25: Catalog Version History

Version	Date	Author	Changes
1.0.0	2024-01-15	Architecture Team	Initial catalog creation
1.1.0	2024-02-20	J. Smith	Added ORDER-Svc behavioral specifications

Version	Date	Author	Changes
1.2.0	2024-03-15	A. Jones	Updated INV-Svc interface documentation
1.3.0	2024-04-10	B. Chen	Added PAY-Svc v2 interface specifications
2.0.0	2024-06-01	Architecture Team	Major revision; added EVENT-Bus; updated all interfaces

10 Common Pitfalls and Anti-patterns

Warning

Avoid These Catalog Anti-patterns:
Stale Catalog: Catalog that doesn't match current implementation. Establish automated validation where possible.
Over-Documentation: Excessive detail that becomes unmaintainable. Document what's needed, not everything possible.
Under-Documentation: Missing critical information like interface contracts or quality requirements. Use checklists to ensure completeness.
Inconsistent Abstraction: Mixing high-level and low-level elements. Maintain consistent abstraction within each view.
Missing Relationships: Documenting elements but not their relationships. Relationships are as important as elements.
Orphan Interfaces: Interface specifications without corresponding element documentation.
Copy-Paste Errors: Duplicated content that becomes inconsistent. Use single-source references where possible.
Unclear Ownership: No one responsible for keeping entries current. Assign and enforce ownership.

11 Appendix A: Element Specification Templates

11.1 Service Element Template

Service Element Specification

Identity

- ID: ____________________
- Name: ____________________
- Version: ________
- Stereotype: ____________

Purpose: ____

Responsibilities:

- ____
- ____
- ____

Properties

- Criticality: ____________
- Availability: ____________
- Performance: ____________________
- Technology: ____________________

Provided Interfaces: ________________________________

Required Interfaces: ________________________________

Traceability

- Requirements: ________________________
- Implementation: ________________________

11.2 Data Store Element Template

Data Store Element Specification

Identity

- ID: ______________________
- Name: ______________________
- Type: _____________ (Relational / Document / Key-Value / Graph)

Purpose: ____

Data Characteristics

- Primary Entities: __________________________
- Volume: __________________
- Growth Rate: __________________
- Retention: __________________

Access Patterns

- Read/Write Ratio: _____________
- Query Complexity: __________________
- Throughput: __________________

Properties

- Criticality: _____________
- Consistency: _____________ (Strong / Eventual)
- Backup/Recovery: ______________________
- Technology: ______________________

12 Appendix B: Glossary

Architectural Element	A fundamental building block of software architecture with defined interfaces and responsibilities
Catalog	A structured collection of element specifications within an architectural view
Component	A modular, deployable unit of software with explicit interfaces
Connector	An architectural element that mediates interaction between components
Interface	A defined boundary through which elements interact, specifying operations, data, and protocols
Module	A code unit implementing a coherent set of responsibilities
Property	A measurable or classifiable characteristic of an element
Relation	A connection or dependency between architectural elements
Service	An independently deployable unit providing capabilities through well-defined interfaces
Stereotype	A classification tag indicating an element's pattern or role

Traceability	The ability to relate architectural elements to other artifacts
View	A representation of a system from a particular perspective

13 Appendix C: References

1. Clements, P., et al. (2010). *Documenting Software Architectures: Views and Beyond* (2nd ed.). Addison-Wesley.
2. Bass, L., Clements, P., & Kazman, R. (2021). *Software Architecture in Practice* (4th ed.). Addison-Wesley.
3. IEEE. (2011). *ISO/IEC/IEEE 42010:2011 Systems and Software Engineering—Architecture Description*.
4. Brown, S. (2018). *The C4 Model for Visualising Software Architecture*. Retrieved from https://c4model.com
5. Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley.
6. Richardson, C. (2018). *Microservices Patterns*. Manning Publications.
7. Vernon, V. (2013). *Implementing Domain-Driven Design*. Addison-Wesley.
8. The Open Group. (2019). *ArchiMate 3.1 Specification*. Van Haren Publishing.