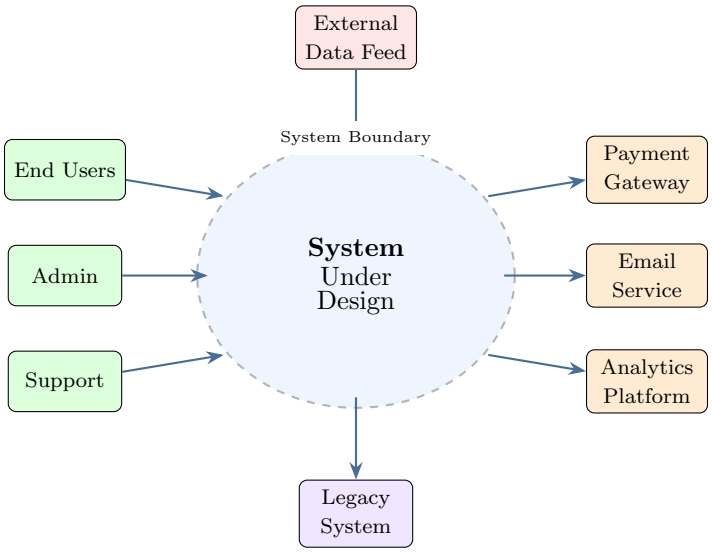
Context Viewpoint

Architecture Viewpoint Specification

System Environment, Boundaries & External Interactions



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:	Context Viewpoint
Synonyms:	System Context View, Environmental View, External Interface View, Boundary View, Ecosystem View, Integration Context View
Identifier:	VP-CTX-001
Version:	2.0

1.1 Viewpoint Classification

The Context Viewpoint defines the system's position within its broader environment, showing how the system interacts with external entities including users, systems, and services. While not explicitly defined as a style in the Views and Beyond approach, it represents essential architectural documentation that establishes scope and boundaries before detailed design. This viewpoint corresponds to the "context diagram" commonly used in system analysis and is foundational to IEEE 42010 architecture descriptions.

Table 1: Viewpoint Classification Taxonomy

Attribute	Value
Style Family	Foundational / Cross-Cutting
Primary Focus	System Boundaries and External Relationships
Abstraction Level	High-Level / Strategic
Temporal Perspective	Static Environment Structure
Related Styles	Component-and-Connector (external), Use Case
IEEE 42010 Category	Context, Stakeholder Viewpoint
C4 Model Equivalent	System Context Diagram (Level 1)

1.2 Viewpoint Scope

The Context Viewpoint encompasses the following aspects:

- **System Boundary:** Clear definition of what is inside versus outside the system, establishing scope for architectural decisions.
- **External Actors:** Users, roles, and personas that interact with the system, including their goals and interaction patterns.
- **External Systems:** Other software systems, services, and platforms that the system integrates with or depends upon.

- **External Data Sources:** Data feeds, APIs, and information sources consumed by the system.
- **External Data Consumers:** Systems or entities that consume data or services provided by the system.
- **Environmental Constraints:** Technical, regulatory, organizational, and operational constraints from the environment.
- **Interface Contracts:** High-level specifications of how the system interacts with external entities.
- **Dependencies and Assumptions:** External dependencies and assumptions about the environment.

2 Overview

The Context Viewpoint provides a comprehensive picture of the system's operating environment and its position within a larger ecosystem. It is typically the first architectural view developed and serves as the foundation for more detailed architectural work.

2.1 Purpose and Scope

The primary purpose of this viewpoint is to establish clear boundaries around what the system is and what it is not, to identify all external entities the system must interact with, and to define the nature of those interactions. This shared understanding is essential for scope management, integration planning, and stakeholder alignment.

Viewpoint Definition

The Context Viewpoint defines the system's environment by identifying its boundaries, external actors, external systems, and the interactions between the system and its environment. It establishes what is in scope for the architecture, what external dependencies exist, and what interfaces the system must provide or consume. This viewpoint provides essential context for all other architectural views.

2.2 Key Characteristics

The Context Viewpoint exhibits several distinctive characteristics:

Boundary Focus: The primary concern is establishing clear boundaries between the system and its environment, defining what is "in" versus "out" of scope.

High Abstraction: The system is treated as a single entity (black box) with focus on external relationships rather than internal structure.

Stakeholder Accessibility: Designed to be understandable by all stakeholders including non-technical audiences, using simple notation and business terminology.

Foundational Role: Serves as the starting point for architecture work, establishing context before detailed design begins.

Integration Emphasis: Highlights all integration points and external dependencies that will require architectural attention.

2.3 Relationship to Other Viewpoints

The Context Viewpoint connects to other architectural viewpoints as a foundational reference:

Table 2: Relationships to Other Viewpoints

Viewpoint	Relationship
Component-and-Connector	Context interfaces are realized by boundary components. External system connectors are detailed.
Deployment	External systems inform deployment integration. Network boundaries align with context boundaries.
Information/Data	External data sources feed data architecture. Data exchange formats are derived from context.
Development	External APIs drive module interfaces. Integration code modules are identified.
Security	Trust boundaries align with system boundaries. External actors drive access control design.
Operational	External dependencies affect monitoring. SLAs with external systems are identified.

2.4 Context Architecture Overview

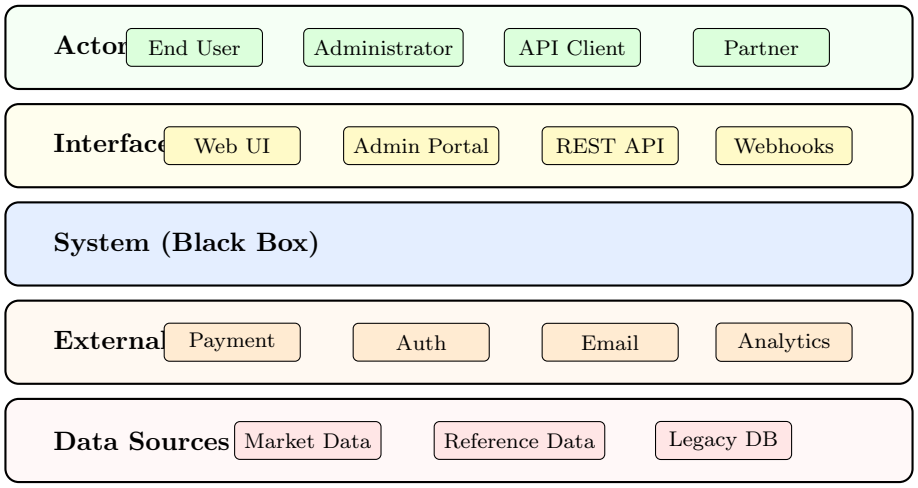


Figure 1: Context Architecture Layers

3 Concerns

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

3.1 Primary Concerns

C1: System Scope and Boundaries

- What functionality is included in the system?
- What is explicitly out of scope?
- Where are the system boundaries?
- What are the boundary interfaces?
- How are scope decisions documented and communicated?

C2: External Actors and Users

- Who are the human users of the system?
- What roles and personas exist?
- What are their goals and needs?
- How do they interact with the system?
- What are their technical capabilities?

C3: External System Dependencies

- What external systems does the system depend on?
- What services do they provide?
- What are the availability and reliability characteristics?
- What happens when dependencies are unavailable?

- What are the contractual relationships?

C4: Integration Points

- What integration interfaces exist?
- What protocols and formats are used?
- What are synchronous vs asynchronous integrations?
- What data is exchanged?
- What transformation or mapping is required?

C5: Data Exchange

- What data flows into the system?
- What data flows out of the system?
- What are the data volumes and frequencies?
- What data quality expectations exist?
- How is data ownership managed?

C6: Environmental Constraints

- What technical constraints exist (platforms, protocols)?
- What regulatory constraints apply?
- What organizational policies affect the system?
- What geographical or jurisdictional constraints exist?
- What time-based constraints apply?

C7: Security Boundaries

- What are the trust boundaries?
- What authentication/authorization exists at boundaries?
- What data protection applies at interfaces?
- What compliance requirements affect external interfaces?
- How are security incidents with external parties handled?

C8: Service Level Expectations

- What availability expectations exist for external interfaces?
- What performance expectations exist?
- What SLAs govern external dependencies?
- How are service levels monitored?
- What happens when SLAs are breached?

C9: Evolution and Change

- How stable are external interfaces?
- What versioning strategies are used?
- How are breaking changes communicated?
- What backward compatibility is required?

- How are new integrations added?

C10: Assumptions and Risks

- What assumptions are made about the environment?
- What risks arise from external dependencies?
- What contingency plans exist?
- How are assumptions validated?
- What external factors could disrupt the system?

3.2 Concern-Quality Attribute Mapping

Table 3: Concern to Quality Attribute Mapping

Concern	Interop.	Security	Availability	Modtic.	Perform.	Scalability	Usability	Compliance
Scope/Boundaries	○	●	○	●	–	–	○	○
External Actors	○	●	○	○	○	○	●	○
Dependencies	●	○	●	○	●	○	–	○
Integration	●	○	○	●	●	●	–	○
Data Exchange	●	●	○	○	●	●	–	●
Constraints	○	○	○	○	○	○	○	●
Security Bounds	○	●	○	○	○	–	–	●
Service Levels	○	–	●	–	●	●	○	○
Evolution	●	○	○	●	–	○	–	–
Assumptions	○	○	●	○	○	○	–	○

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

4 Anti-Concerns

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

4.1 Out of Scope Topics

AC1: Internal System Structure

- Component decomposition within the system
- Internal module dependencies
- Internal data flows
- Implementation technologies
- Internal algorithms and logic

AC2: Detailed Interface Specifications

- API endpoint definitions
- Message schema details
- Protocol specifications
- Error handling details
- Rate limiting configurations

AC3: Deployment Architecture

- Server and infrastructure topology
- Container orchestration
- Network configuration
- Cloud resource allocation
- Environment configurations

AC4: Process and Thread Design

- Concurrency architecture
- Thread pool configurations
- Process communication
- Synchronization mechanisms
- Runtime behavior

AC5: Operational Procedures

- Monitoring configurations
- Alerting rules
- Incident response procedures
- Deployment pipelines
- Runbook details

Common Misapplications

Avoid using the Context Viewpoint for:

- Documenting internal component structure (use C&C Viewpoint)
- Specifying API contracts in detail (use Interface Specifications)
- Defining deployment topology (use Deployment Viewpoint)
- Detailing data models (use Information Viewpoint)
- Specifying process architecture (use Process Viewpoint)

5 Typical Stakeholders

The Context Viewpoint serves a broad audience as it establishes foundational understanding of the system.

5.1 Primary Stakeholders

Table 4: Primary Stakeholder Analysis

Stakeholder	Role Description	Primary Interests
Project Sponsors	Fund and authorize project	Scope clarity, business alignment, risk understanding
Product Managers	Define product direction	Feature scope, user identification, market positioning
Software Architects	Design system structure	Integration requirements, constraints, dependencies
Business Analysts	Define requirements	User roles, business processes, external touchpoints
System Integrators	Connect systems together	Integration points, protocols, data formats
Enterprise Architects	Manage system portfolio	Ecosystem fit, standards compliance, redundancy

5.2 Secondary Stakeholders

Table 5: Secondary Stakeholder Analysis

Stakeholder	Role Description	Primary Interests
Development Teams	Build the system	External API understanding, integration work scope
Security Teams	Protect system assets	Trust boundaries, external attack surface, compliance
Operations Teams	Run production systems	External dependencies, monitoring boundaries, SLAs
Legal/Compliance	Ensure regulatory compliance	Data flows, jurisdictions, regulatory boundaries
Partner Organizations	External system owners	Integration requirements, responsibilities, timelines
End Users	Use the system	Understanding system capabilities and limitations

5.3 Stakeholder Concern Matrix

Table 6: Stakeholder-Concern Responsibility Matrix

	Scope	Actors	Depend.	Integr.	Data	Constr.	Security	SLAs	Evolut.	Risk
Sponsor	A	I	I	I	I	A	I	A	I	A
Product Mgr	R	R	C	C	C	C	I	C	R	C
Architect	R	C	R	R	R	R	R	R	R	R
Bus. Analyst	C	R	C	C	R	C	I	C	C	C
Integrator	C	I	A	A	A	C	C	C	A	C
Security	C	C	C	C	C	C	A	C	C	C

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

6 Model Types

The Context Viewpoint employs several complementary model types to capture different aspects of the system's environment.

6.1 Model Type Catalog

MT1: System Context Diagram

- *Purpose:* Show system as black box with all external entities
- *Primary Elements:* System, actors, external systems, data flows
- *Key Relationships:* Interacts-with, sends-to, receives-from
- *Typical Notation:* C4 Context, custom diagrams, DFD Level 0

MT2: Actor Catalog

- *Purpose:* Document all users and their characteristics
- *Primary Elements:* Actors, roles, goals, capabilities
- *Key Relationships:* Plays-role, has-goal, interacts-via
- *Typical Notation:* Tables, personas, user profiles

MT3: External System Inventory

- *Purpose:* Catalog all external system dependencies
- *Primary Elements:* Systems, owners, interfaces, SLAs
- *Key Relationships:* Depends-on, provides, consumes
- *Typical Notation:* Tables, dependency matrices

MT4: Interface Summary

- *Purpose:* Overview of all external interfaces
- *Primary Elements:* Interfaces, protocols, data types

- *Key Relationships:* Exposes, consumes, exchanges
- *Typical Notation:* Interface tables, API summaries

MT5: Data Flow Overview

- *Purpose:* Show high-level data movement across boundaries
- *Primary Elements:* Data sources, sinks, flow directions
- *Key Relationships:* Flows-from, flows-to, transforms
- *Typical Notation:* DFD Level 0, data flow diagrams

MT6: Trust Boundary Diagram

- *Purpose:* Show security boundaries and trust zones
- *Primary Elements:* Trust zones, boundaries, crossing points
- *Key Relationships:* Contains, crosses, authenticates
- *Typical Notation:* Security zone diagrams, threat models

MT7: Constraint Catalog

- *Purpose:* Document environmental constraints
- *Primary Elements:* Constraints, sources, impacts
- *Key Relationships:* Constrains, derives-from, affects
- *Typical Notation:* Tables, constraint matrices

MT8: Dependency Map

- *Purpose:* Visualize critical dependencies and risks
- *Primary Elements:* Dependencies, criticality, fallback
- *Key Relationships:* Depends-on, falls-back-to, critical-for
- *Typical Notation:* Dependency graphs, risk matrices

MT9: Ecosystem Map

- *Purpose:* Show system within broader organizational ecosystem
- *Primary Elements:* Systems, domains, relationships
- *Key Relationships:* Part-of, integrates-with, replaces
- *Typical Notation:* Enterprise architecture diagrams

6.2 Model Type Relationships

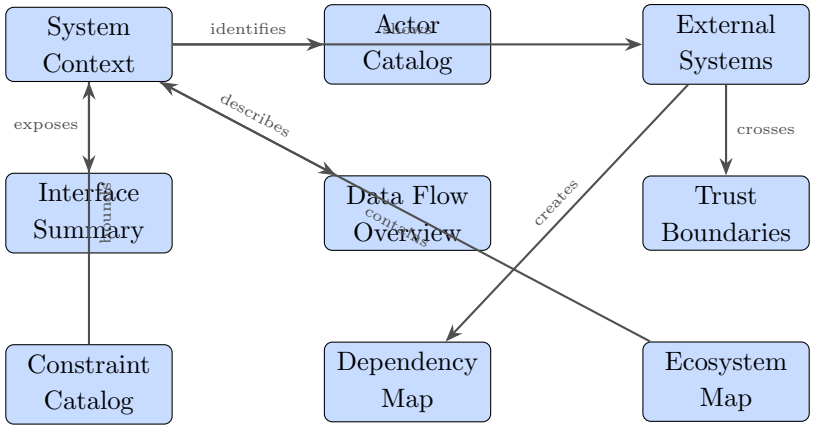


Figure 2: Model Type Dependency Relationships

7 Model Languages

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

7.1 Context Diagram Notation

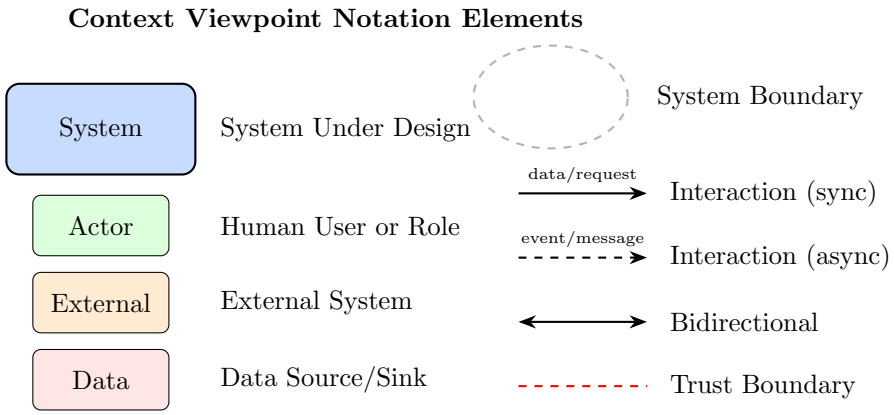


Figure 3: Context Viewpoint Notation Legend

7.2 Actor Classification

Table 7: Actor Type Classification

Actor Type	Description	Examples
End User	Direct users of system functionality	Customers, members, subscribers
Administrator	Users managing system configuration	System admins, tenant admins
Operator	Users monitoring and operating system	DevOps, support staff
API Consumer	Automated clients using APIs	Partner systems, mobile apps
Data Provider	Sources that feed data to system	Market data feeds, IoT devices
Auditor	Users reviewing system activity	Compliance officers, auditors

7.3 External System Classification

Table 8: External System Type Classification

System Type	Description	Examples
SaaS Service	Third-party cloud services	Stripe, SendGrid, Auth0
Enterprise System	Internal organizational systems	ERP, CRM, HR systems
Legacy System	Older systems being integrated	Mainframe, older databases
Partner System	External partner integrations	Supplier APIs, partner portals
Data Provider	External data sources	Bloomberg, weather services
Infrastructure	Platform infrastructure services	AWS, Azure, DNS, CDN

7.4 Interface Classification

Table 9: Interface Type Classification

Interface Type	Direction	Style	Use Cases
REST API	Bidirectional	Sync	CRUD operations, resource access
GraphQL	Bidirectional	Sync	Flexible queries, mobile apps
gRPC	Bidirectional	Sync	Service-to-service, high performance
Webhook	Outbound	Async	Event notifications, callbacks
Message Queue	Both	Async	Event streaming, decoupling
File Transfer	Both	Batch	Bulk data exchange, reports
Web UI	Inbound	Sync	Human user interaction
Database Link	Both	Sync	Legacy integration, replication

7.5 Tabular Specifications

7.5.1 Actor Specification Table

Table 10: Example Actor Specification Format

Actor	Type	Goals	Interface	Volume
Customer	End User	Browse, purchase, track orders	Web UI, Mobile	100K DAU
Store Admin	Administrator	Manage inventory, pricing	Admin Portal	500 users
Partner API	API Consumer	Sync inventory, orders	REST API	1M req/day
Warehouse	Operator	Fulfill orders, manage stock	Internal App	50 users

7.5.2 External System Specification Table

Table 11: Example External System Specification Format

System	Type	Purpose	Protocol	SLA	Owner
Stripe	SaaS	Payment processing	REST	99.99%	Stripe Inc
Auth0	SaaS	Authentication	OIDC	99.99%	Okta
SAP ERP	Enterprise	Order sync	RFC	99.5%	Finance IT
Kafka	Internal	Event streaming	Kafka	99.9%	Platform

7.5.3 Constraint Specification Table

Table 12: Example Constraint Specification Format

Constraint	Type	Description	Impact
GDPR	Regulatory	EU data protection rules	Data handling, consent, deletion
PCI-DSS	Regulatory	Payment card security	Payment interface isolation
Corporate SSO	Technical	Must use corporate identity	Auth0 integration required
AWS Only	Organizational	Cloud provider restriction	Limits service choices

8 Viewpoint Metamodels

This section defines the conceptual metamodel underlying the Context Viewpoint.

8.1 Core Metamodel

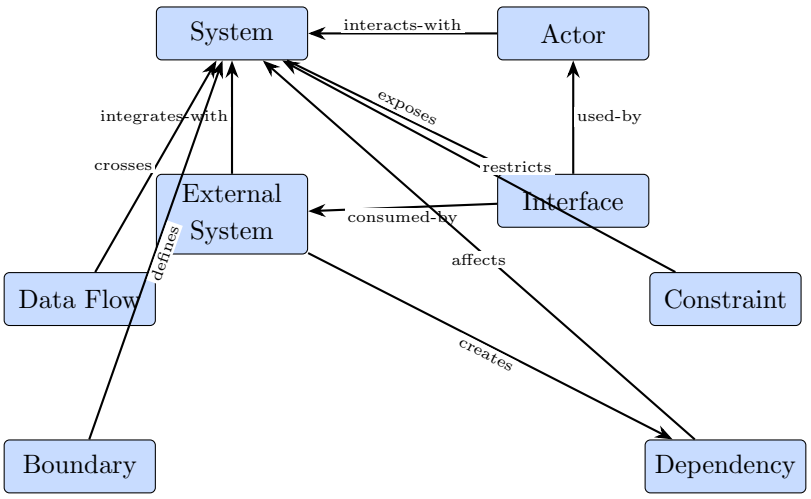


Figure 4: Context Viewpoint Core Metamodel

8.2 Entity Definitions

Entity: System

Definition: The system under design, treated as a black box in the context view, representing the scope of the architectural effort.

Attributes:

- `systemId`: Unique identifier
- `name`: System name
- `description`: System purpose and scope
- `version`: Current version
- `status`: Development status (planned, in-development, production)
- `owner`: Owning organization or team
- `domain`: Business domain
- `criticality`: Business criticality level
- `scopeStatement`: What is in/out of scope

Constraints:

- System scope must be clearly defined
- System must have identified owner
- System must have at least one external interaction

Entity: Actor

Definition: A human user, role, or persona that interacts with the system to achieve goals.

Attributes:

- `actorId`: Unique identifier
- `name`: Actor name or role title
- `type`: Actor type (end-user, admin, operator, auditor)
- `description`: Actor characteristics and responsibilities
- `goals`: What the actor wants to achieve
- `frequency`: How often they interact
- `volume`: Expected number of this actor type
- `technicalLevel`: Technical sophistication
- `accessLevel`: Authorization level
- `interfaces`: How they access the system

Constraints:

- Each actor must have defined goals
- Actor must have at least one interface for interaction
- Access levels must be defined for authorization

Entity: External System

Definition: A software system, service, or platform outside the system boundary that the system interacts with.

Attributes:

- `systemId`: Unique identifier
- `name`: System name
- `type`: System type (SaaS, enterprise, legacy, partner)
- `description`: System purpose and capabilities
- `owner`: Organization or vendor that owns it
- `contact`: Technical contact information
- `protocol`: Communication protocol(s)
- `dataExchanged`: Data types exchanged
- `direction`: Inbound, outbound, or bidirectional
- `sla`: Service level agreement
- `availability`: Expected availability
- `documentation`: Link to API/integration docs

Constraints:

- External system must have identified owner
- Integration protocol must be specified
- Critical dependencies must have SLAs
- Fallback behavior must be defined for critical dependencies

Entity: Interface

Definition: A point of interaction between the system and external entities, defining how communication occurs.

Attributes:

- `interfaceId`: Unique identifier
- `name`: Interface name
- `type`: Interface type (REST API, Web UI, Message Queue, etc.)
- `description`: Interface purpose
- `direction`: Provided or consumed
- `protocol`: Communication protocol
- `dataFormat`: Data format (JSON, XML, Protobuf)
- `authentication`: Authentication mechanism
- `authorization`: Authorization model
- `versioning`: Versioning strategy
- `documentation`: Link to detailed specification

Constraints:

- All external interfaces must have authentication
- Interfaces must have versioning strategy
- Public interfaces must have documentation

Entity: Data Flow

Definition: A movement of data across the system boundary, characterizing what data is exchanged with external entities.

Attributes:

- `flowId`: Unique identifier
- `name`: Flow name
- `description`: Flow purpose
- `source`: Origin of data
- `destination`: Destination of data
- `dataType`: Type of data being transferred
- `volume`: Expected data volume
- `frequency`: How often data flows
- `sensitivity`: Data sensitivity classification
- `transformation`: Any transformation required
- `latencyRequirement`: Maximum acceptable delay

Constraints:

- Sensitive data flows must be encrypted
- Data flows must have defined owners
- Cross-boundary flows must be documented

Entity: Constraint

Definition: A limitation or requirement imposed on the system by its environment that must be accommodated.

Attributes:

- `constraintId`: Unique identifier
- `name`: Constraint name
- `type`: Constraint type (regulatory, technical, organizational, business)
- `description`: Detailed description
- `source`: Where the constraint comes from
- `rationale`: Why this constraint exists
- `impact`: How it affects the system
- `compliance`: How compliance is verified
- `exceptions`: Any approved exceptions
- `reviewDate`: When constraint should be reviewed

Constraints:

- Regulatory constraints must have compliance verification
- Constraints must have documented rationale
- Exceptions must be formally approved

Entity: Boundary

Definition: A demarcation that separates the system from its environment, defining scope and responsibility limits.

Attributes:

- `boundaryId`: Unique identifier
- `name`: Boundary name
- `type`: Boundary type (system, trust, organizational, network)
- `description`: What the boundary separates
- `inside`: What is inside the boundary
- `outside`: What is outside the boundary
- `crossingPoints`: Interfaces that cross the boundary
- `controls`: Security or governance controls at boundary

Constraints:

- Trust boundaries must have security controls
- All boundary crossings must be through defined interfaces
- Boundary scope must be unambiguous

Entity: Dependency

Definition: A reliance on an external system or service that creates risk if the dependency is unavailable or changes.

Attributes:

- `dependencyId`: Unique identifier
- `name`: Dependency name
- `target`: External system depended upon
- `type`: Dependency type (runtime, build-time, data)
- `criticality`: How critical (critical, important, convenience)
- `impact`: Impact if dependency unavailable
- `fallback`: Fallback behavior or alternative
- `monitoring`: How dependency health is monitored
- `sla`: Required service level
- `riskMitigation`: Risk mitigation strategies

Constraints:

- Critical dependencies must have fallback strategies
- Dependencies must have monitoring
- High-risk dependencies must have mitigation plans

8.3 Relationship Definitions

Table 13: Metamodel Relationship Definitions

Relationship	Source	Target	Description
interacts-with	Actor	System	Actor uses the system
integrates-with	External	System	External system connects to system
exposes	System	Interface	System provides this interface
consumes	System	Interface	System uses external interface
used-by	Interface	Actor	Interface is accessed by actor
consumed-by	Interface	External	Interface is used by external system
crosses	Data Flow	Boundary	Data moves across boundary
restricts	Constraint	System	Constraint limits system
defines	Boundary	System	Boundary establishes system scope
creates	External	Dependency	External system creates dependency

9 Conforming Notations

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

9.1 C4 Model - Context Diagram

The C4 Model's System Context diagram is a primary conforming notation for this viewpoint.

Elements: Person (user), Software System (external), System Boundary.

Conformance Level: Full conformance for context modeling.

Tool Support: Structurizr, PlantUML-C4, draw.io C4 shapes.

9.2 UML Use Case Diagrams

UML Use Case diagrams show actors and their interactions with system use cases.

Elements: Actors, Use Cases, System Boundary, Associations.

Conformance Level: Partial - focuses on functionality rather than systems.

Tool Support: Enterprise Architect, Visual Paradigm, StarUML.

9.3 Data Flow Diagrams (Level 0)

Context-level DFDs show the system with external entities and data flows.

Elements: Process (system), External Entity, Data Flow.

Conformance Level: Good for data-centric systems.

Tool Support: Lucidchart, Visio, draw.io.

9.4 ArchiMate

ArchiMate provides comprehensive enterprise architecture notation including context elements.

Elements: Business Actors, Application Components, Relationships.

Conformance Level: Full - designed for enterprise context.

Tool Support: Archi, Sparx EA, BiZZdesign.

9.5 Notation Comparison

Table 14: Context Notation Comparison

Feature	C4	UML UC	DFD	ArchiMate	BPMN	Custom
System boundary	●	●	●	●	○	●
Human actors	●	●	○	●	●	●
External systems	●	○	●	●	○	●
Data flows	○	–	●	●	●	●
Trust boundaries	○	–	–	●	–	●
Constraints	–	–	–	●	–	●
Simplicity	●	●	●	○	○	●
Standardized	○	●	○	●	●	–

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

10 Model Correspondence Rules

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

10.1 Component-and-Connector View Correspondence

Correspondence Rule CR-01: Interface to Component Mapping

Rule: Every external interface in the context view must be realized by a boundary component in the C&C view.

Formal Expression:

$$\forall i \in Interfaces_{Context} : \exists c \in Components_{C\&C} : realizes(c, i)$$

Rationale: Ensures context interfaces are architecturally implemented.

Verification: Interface-to-component traceability matrix.

Correspondence Rule CR-02: External System to Connector Mapping

Rule: Every external system integration must have a corresponding connector in the C&C view.

Formal Expression:

$$\forall ext \in ExternalSystems : \exists conn \in Connectors : connects(conn, ext)$$

Rationale: Ensures integrations are architecturally visible.

Verification: Integration architecture review.

10.2 Deployment View Correspondence

Correspondence Rule CR-03: Boundary to Network Mapping

Rule: System boundaries should align with network security boundaries in deployment.

Formal Expression:

$$\forall b \in TrustBoundaries_{Context} : \exists n \in NetworkZones_{Deploy} : aligns(b, n)$$

Rationale: Ensures logical boundaries are physically enforced.

Verification: Network architecture review.

10.3 Security View Correspondence

Correspondence Rule CR-04: Actor to Access Control Mapping

Rule: Every actor type must have corresponding roles and permissions in security design.

Formal Expression:

$$\forall a \in Actors : \exists R \subseteq Roles : authorizes(R, a)$$

Rationale: Ensures actors have defined access controls.

Verification: Role mapping review.

11 Operations on Views

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

11.1 Creation Methods

11.1.1 View Development Process

Step 1: Define System Scope

1. Review project charter and business case
2. Identify system purpose and objectives
3. Document what is explicitly in scope
4. Document what is explicitly out of scope
5. Get stakeholder agreement on scope boundaries

Step 2: Identify External Actors

1. List all human user types
2. Define roles and personas
3. Document actor goals and needs
4. Estimate actor volumes
5. Identify actor technical capabilities

Step 3: Identify External Systems

1. List all systems the system must integrate with
2. Categorize by type (SaaS, enterprise, legacy, partner)
3. Identify system owners and contacts
4. Document integration purposes
5. Assess availability and reliability

Step 4: Define Interfaces

1. List all interfaces the system will expose
2. List all external interfaces the system will consume
3. Document protocols and data formats
4. Define authentication/authorization requirements
5. Plan versioning strategies

Step 5: Map Data Flows

1. Identify all data entering the system
2. Identify all data leaving the system
3. Document data types and volumes
4. Classify data sensitivity
5. Identify transformation requirements

Step 6: Document Constraints

1. Identify regulatory constraints
2. Document technical constraints
3. List organizational constraints
4. Assess business constraints
5. Document constraint impacts

Step 7: Analyze Dependencies and Risks

1. Assess dependency criticality
2. Identify single points of failure
3. Document fallback strategies
4. Plan dependency monitoring
5. Create risk mitigation plans

11.1.2 Integration Patterns

Pattern: API Gateway

Context: Multiple external consumers need to access system capabilities.
Solution: Single entry point that routes, authenticates, and rate-limits external requests.
Benefits:

- Centralized authentication and authorization
- Rate limiting and throttling
- Request/response transformation
- API versioning support
- Monitoring and analytics

Use When: Multiple external API consumers, need for centralized control.

Pattern: Anti-Corruption Layer

Context: Integrating with legacy or external system with incompatible model.
Solution: Translation layer that converts between system's domain model and external model.
Benefits:

- Protects domain model integrity
- Isolates external model changes
- Enables gradual migration
- Simplifies testing

Use When: Legacy integration, external systems with different domain models.

Pattern: Backend for Frontend (BFF)

Context: Different client types (web, mobile, partner) have different needs.
Solution: Dedicated backend service for each frontend type.
Benefits:

- Optimized API for each client
- Independent evolution
- Client-specific security
- Reduced over-fetching

Use When: Multiple client types with different requirements.

Table 15: Integration Patterns Summary

Pattern	Description	Use When
API Gateway	Centralized entry point	Multiple external consumers
Anti-Corruption Layer	Model translation layer	Legacy or incompatible systems
Backend for Frontend	Client-specific backends	Different client needs
Strangler Fig	Gradual replacement	Legacy modernization
Circuit Breaker	Failure isolation	Unreliable dependencies
Bulkhead	Resource isolation	Prevent cascade failures
Retry with Backoff	Transient failure handling	Intermittent failures
Event-Driven Integration	Async via events	Loose coupling needs

11.2 Analysis Methods

11.2.1 Dependency Risk Analysis

Dependency Risk Assessment

Purpose: Evaluate risks from external dependencies.

Factors:

- **Criticality:** How critical is the dependency to system function?
- **Availability:** What is the expected/historical availability?
- **Vendor Risk:** Risk of vendor failure, acquisition, or discontinuation
- **Lock-in:** Difficulty of switching to alternative
- **Security:** Security posture of the dependency

Risk Score: Combine factors into overall risk rating (High/Medium/Low).

Mitigation: Define fallbacks, alternatives, or acceptance for each risk level.

11.2.2 Interface Complexity Analysis

Interface Complexity Assessment

Purpose: Assess complexity of external interfaces for planning.

Complexity Factors:

- Number of operations/endpoints
- Data model complexity
- Authentication complexity
- Error handling requirements
- Versioning requirements
- Volume and performance requirements

Output: Complexity score guiding development effort estimates.

12 Examples

12.1 Example 1: E-Commerce System Context

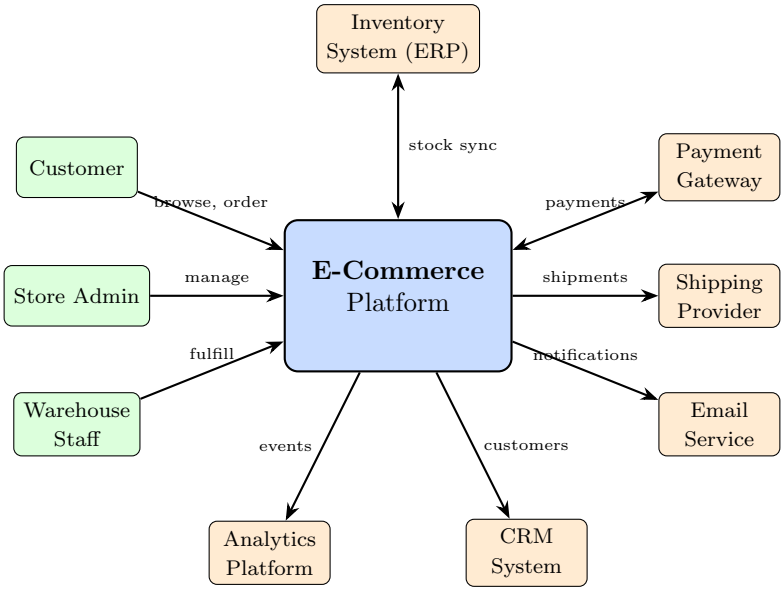


Figure 5: E-Commerce Platform System Context Diagram

Description: This context diagram shows an e-commerce platform with three actor types (Customer, Store Admin, Warehouse Staff) and six external system integrations. The diagram shows data flow directions - bidirectional for payment and inventory, outbound for shipping, email, analytics, and CRM.

12.2 Example 2: Trust Boundary Diagram

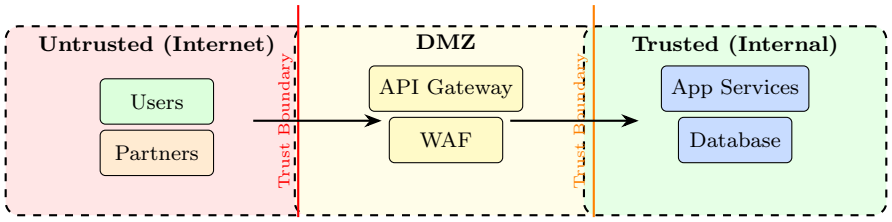


Figure 6: Trust Boundary Diagram

Description: This diagram shows three trust zones with different security postures. Traffic from the untrusted internet zone must pass through the DMZ (with API Gateway and WAF) before reaching internal trusted services. Each boundary crossing requires authentication and authorization.

12.3 Example 3: Actor Catalog

Table 16: Actor Catalog Example

Actor	Type	Primary Goals	Interface	Volume
Customer	End User	Browse products, place orders, track deliveries	Web, Mobile App	500K MAU
Guest	End User	Browse products, view prices	Web, Mobile App	2M visits/mo
Store Admin	Admin	Manage products, pricing, promotions	Admin Portal	50 users
Warehouse	Operator	Process orders, manage inventory	Warehouse App	200 users
Partner API	API Client	Sync inventory, retrieve orders	REST API	100K req/day
Support Agent	Operator	Handle customer issues, process refunds	Support Portal	100 users

13 Notes

13.1 Scope Management Best Practices

Scope Definition Guidelines

- Document scope decisions with rationale
- Get explicit stakeholder sign-off on boundaries
- Use "in/out of scope" lists for clarity
- Review scope at project milestones
- Maintain scope change control process
- Communicate scope to all team members
- Link scope to requirements traceability

13.2 External Dependency Management

Dependency Management Guidelines

- Maintain an up-to-date dependency inventory
- Monitor external system health proactively
- Establish communication channels with dependency owners
- Plan for dependency unavailability (graceful degradation)
- Version external interfaces explicitly
- Test integration points regularly
- Document escalation paths for dependency issues
- Review dependencies for security vulnerabilities

13.3 Common Pitfalls

Common Mistakes to Avoid

1. **Undefined Boundaries:** Ambiguous scope leading to feature creep
2. **Hidden Dependencies:** Undocumented external system dependencies
3. **Missing Actors:** Forgetting user types (auditors, support staff)
4. **One-Way Thinking:** Not considering data flowing out of the system
5. **Ignoring Constraints:** Not documenting regulatory or organizational limits
6. **Optimistic SLAs:** Assuming external systems are always available
7. **Static View:** Not planning for environment evolution
8. **Security Afterthought:** Not identifying trust boundaries early

14 Sources

14.1 Primary References

1. Clements, P., et al. (2010). *Documenting Software Architectures: Views and Beyond* (2nd ed.). Addison-Wesley Professional.

2. Brown, S. (2018). *The C4 Model for Visualizing Software Architecture*. Leanpub.
3. Rozanski, N., & Woods, E. (2011). *Software Systems Architecture* (2nd ed.). Addison-Wesley Professional.
4. Bass, L., Clements, P., & Kazman, R. (2021). *Software Architecture in Practice* (4th ed.). Addison-Wesley Professional.
5. ISO/IEC/IEEE 42010:2011. *Systems and software engineering — Architecture description*.

14.2 Supplementary References

6. Hohpe, G., & Woolf, B. (2003). *Enterprise Integration Patterns*. Addison-Wesley Professional.
7. Vernon, V. (2013). *Implementing Domain-Driven Design*. Addison-Wesley Professional.
8. Newman, S. (2021). *Building Microservices* (2nd ed.). O'Reilly Media.
9. Richardson, C. (2018). *Microservices Patterns*. Manning Publications.
10. The Open Group. (2018). *ArchiMate 3.1 Specification*.

14.3 Online Resources

- C4 Model: `https://c4model.com/`
- Arc42 Template: `https://arc42.org/`
- Structurizr: `https://structurizr.com/`
- ArchiMate: `https://www.opengroup.org/archimate-forum`
- Software Architecture Guide: `https://martinfowler.com/architecture/`

A Context View Checklist

Item	Complete?
Scope and Boundaries	
System scope clearly defined	☐
In-scope items documented	☐
Out-of-scope items documented	☐
Stakeholder agreement obtained	☐
External Actors	
All user types identified	☐
Actor goals documented	☐
Actor volumes estimated	☐
Interfaces for each actor defined	☐
External Systems	
All external systems identified	☐
System owners documented	☐
Protocols and formats specified	☐
SLAs documented	☐
Fallback strategies defined	☐
Data Flows	
Inbound data flows documented	☐
Outbound data flows documented	☐
Data sensitivity classified	☐
Volume estimates provided	☐
Constraints and Risks	
Regulatory constraints documented	☐
Technical constraints documented	☐
Dependencies assessed for risk	☐
Mitigation strategies defined	☐
Security	
Trust boundaries identified	☐
Authentication requirements defined	☐
Data protection requirements documented	☐

B Glossary

Actor A human user, role, or persona that interacts with the system.

API Gateway A server that acts as a single entry point for API requests.

Boundary A demarcation separating the system from its environment.

Constraint A limitation imposed on the system by its environment.

Context The environment in which the system operates.

Data Flow Movement of data across the system boundary.

Dependency A reliance on an external system or service.

External System
A software system outside the system boundary.

Interface A point of interaction with external entities.

Integration The connection between the system and external systems.

Scope The extent of what is included in the system.

SLA Service Level Agreement defining expected service quality.

Stakeholder Anyone with an interest in the system.

System Under Design
The system being architected and documented.

Trust Boundary
A boundary separating different security trust levels.