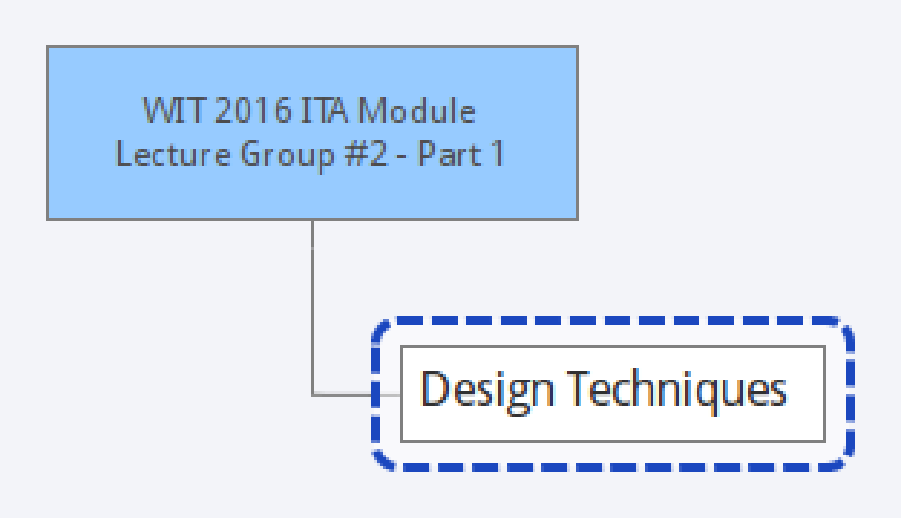
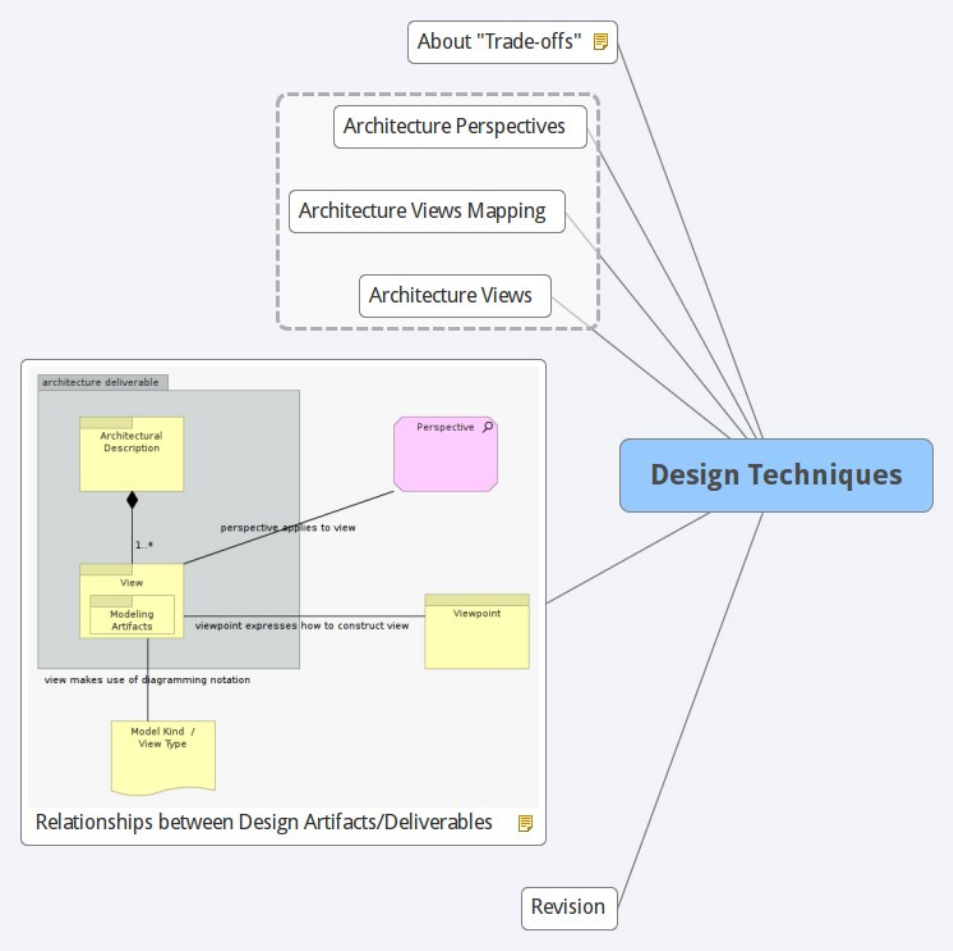
WIT 2016 ITA Module

Lecture Group #2 - Part 1a
Architecture Views

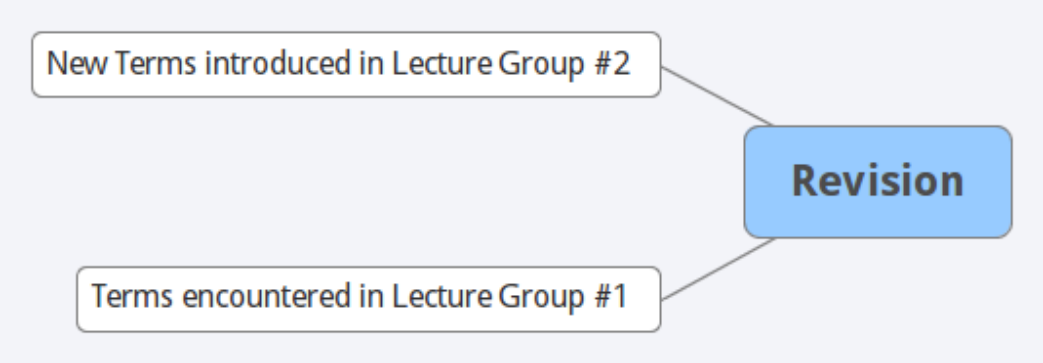
Lecture Group #2 - Part 1



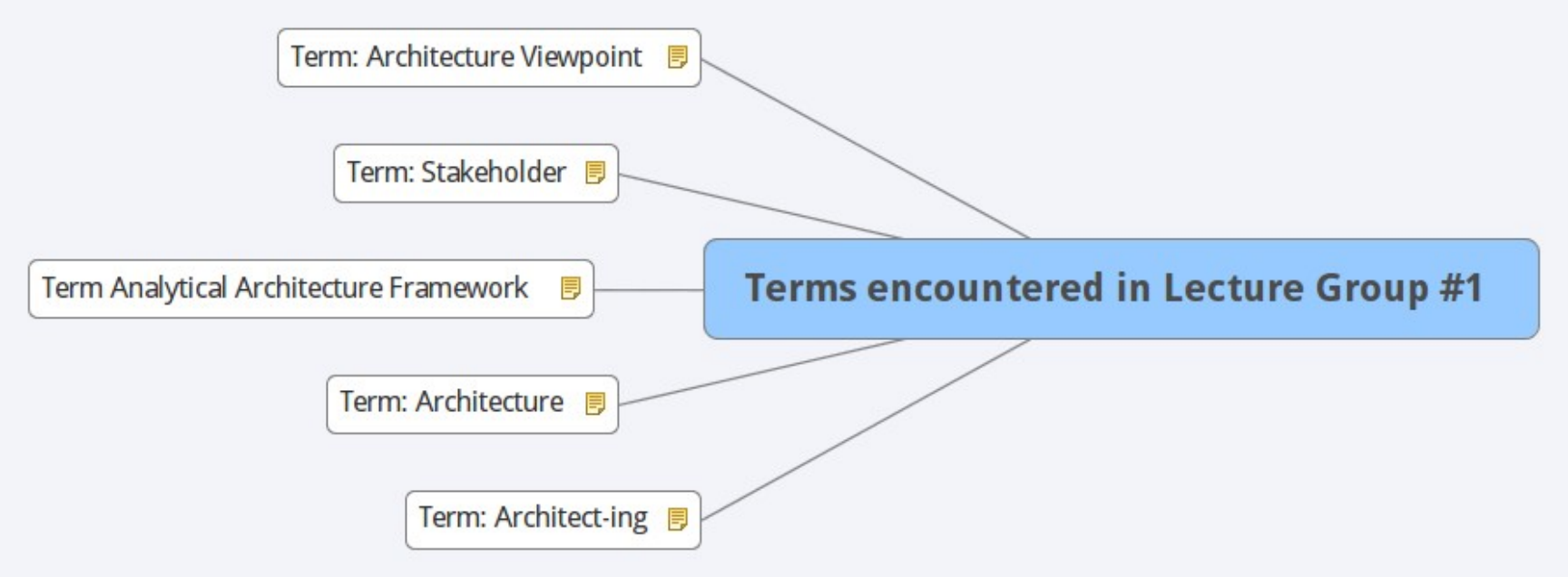
Design Techniques



Revision



Terms encountered in Lecture Group #1



Term: Architect-ing

Architecting: process of conceiving, defining, expressing, recording decisions, communicating, certifying proper implementation of, maintaining and improving an architecture throughout a system's life cycle.

Architecting takes place in the context of an organization ("person or a group of people and facilities with an arrangement of responsibilities, authorities and relationships") and/or a project ("endeavour with defined start and finish criteria undertaken to create a product or service in accordance with specified resources and requirements") [ISO/IEC 12207, ISO/IEC 15288].

Term: Architecture

Architecture: <system> fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.

Term Analytical Architecture Framework

Architecture Framework: conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders.

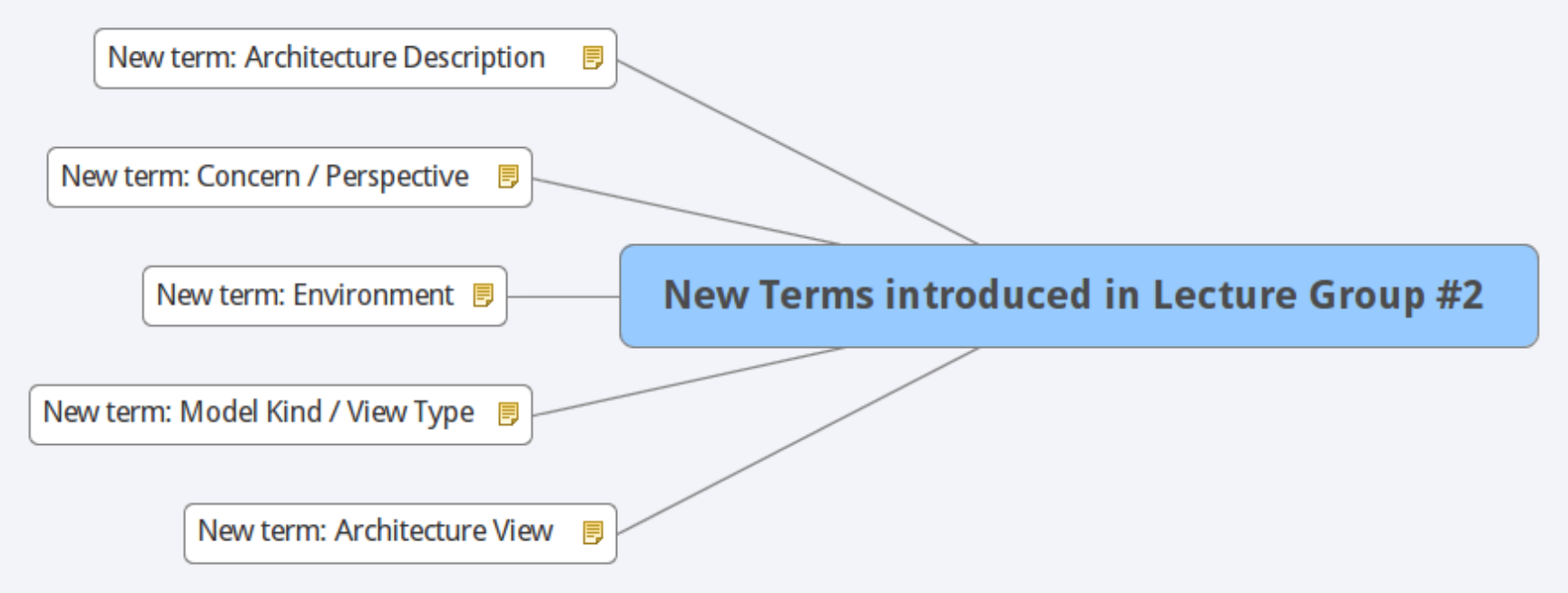
Term: Stakeholder

Stakeholder: individual, team, organization, or classes thereof, having an interest in a system.

Term: Architecture Viewpoint

Architecture Viewpoint: work product establishing the conventions for the construction, interpretation and use by architecture views to frame specific system concerns.

New Terms introduced in Lecture Group #2



New term: Architecture View

Architecture View: a work product expressing the architecture of a system from the perspective of specific set of system concerns.

New term: Model Kind / View Type

Model Kind (also known as View Type): conventions for a type of modelling. Examples of model kinds include: functional diagrams (ex. class or component diagrams), behavioral diagrams (ex. process diagram, state transition or data flow diagram), context diagrams (ex. organization charts, context diagram), server deployment diagrams (etc.)

New term: Environment

Environment: <system> context determining the setting and circumstances of all influences upon a system.

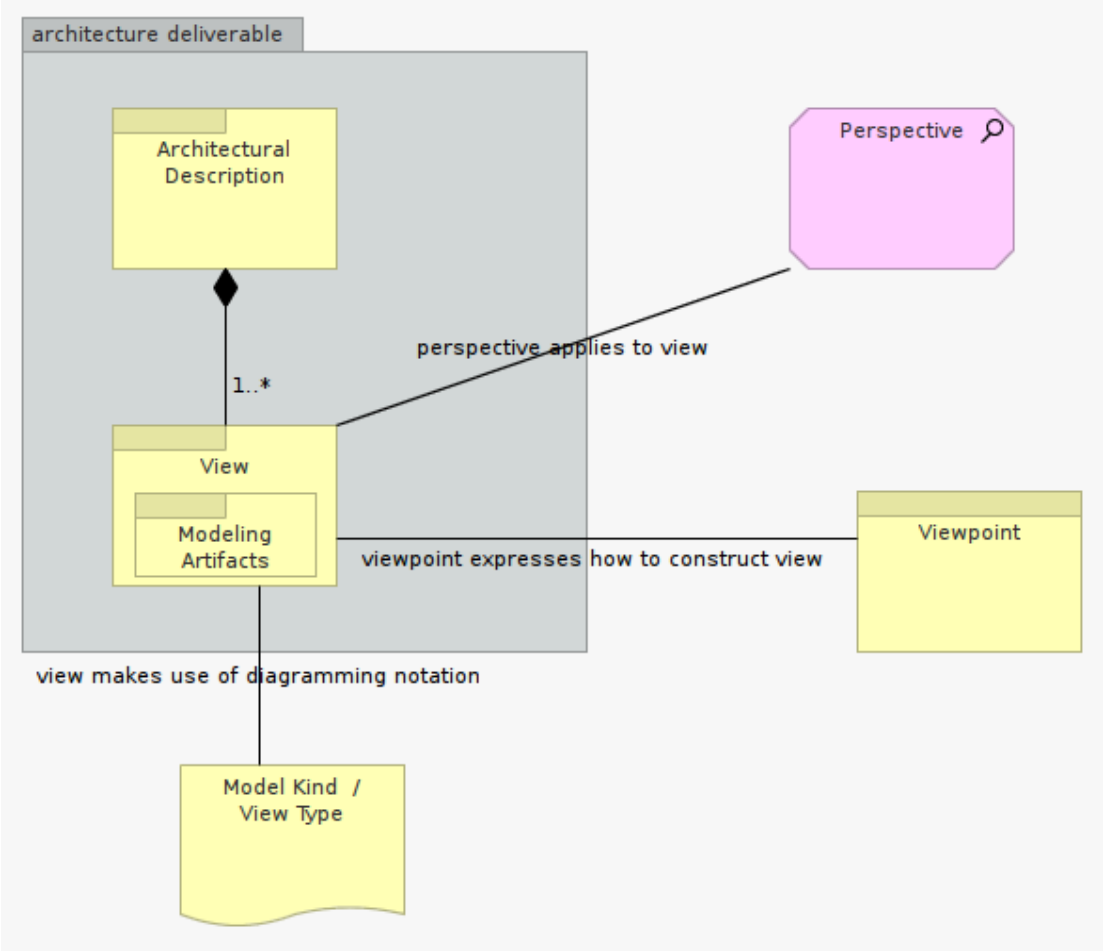
New term: Concern / Perspective

Concern (also known as Architecture Perspective): A concern pertains to any influence on a system from its environment, including developmental, technological, business, operational, organizational, political, economic, legal, regulatory, ecological and social influences.

New term: Architecture Description

Architecture Description (AD): a work product used to express an architecture solution.

Relationships between Design Artifacts/Deliverables



Relationships between Design Artifacts/Deliverables

Standardized Design Techniques are essential for Architects to:

- surface cross cutting concerns between a new systems solution and other existing applications

- avoid creating stovepipe application solutions that do not evolve easily

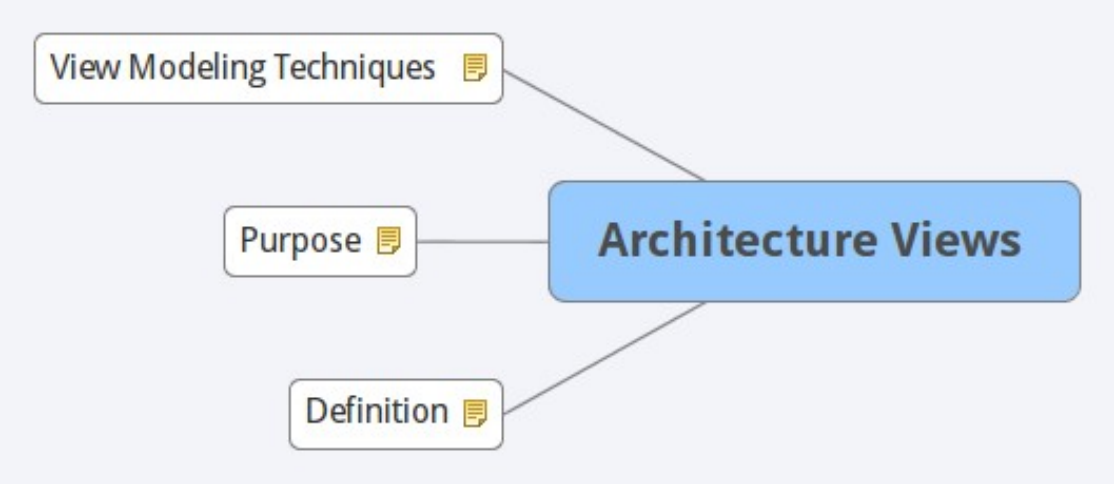
- align application features to required business capabilities

- help decision making and to record design decisions over time

- control and govern that execution conforms to intended design

- help stakeholders understanding risk / cost, and the possibilities of the proposed architecture

Architecture Views



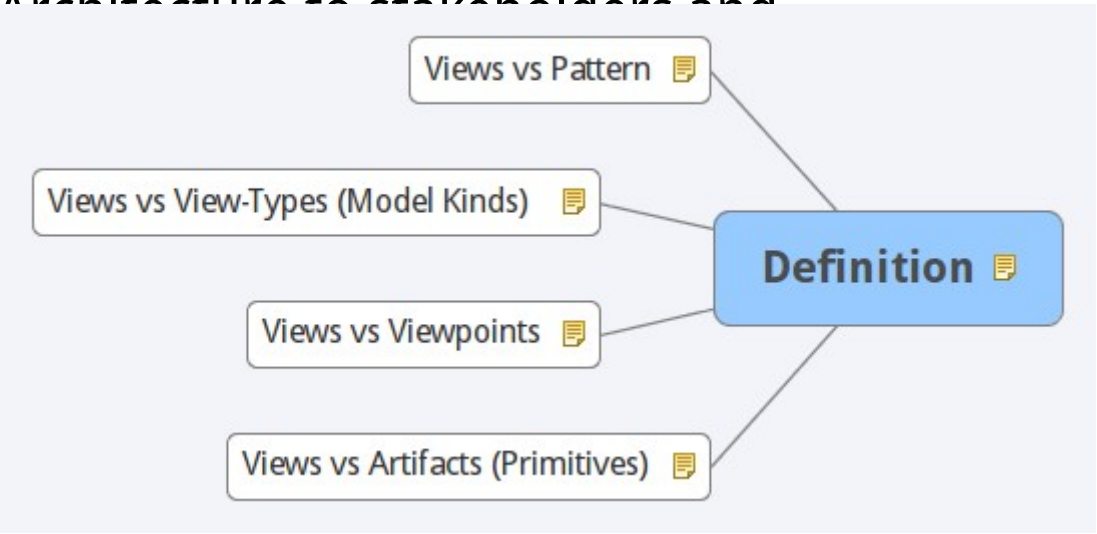
Definition

An Architectural View is an opinionated composition of (1,n) aspects of an architecture, illustrating how it addresses (1,n) concerns held by (1,n) stakeholders.

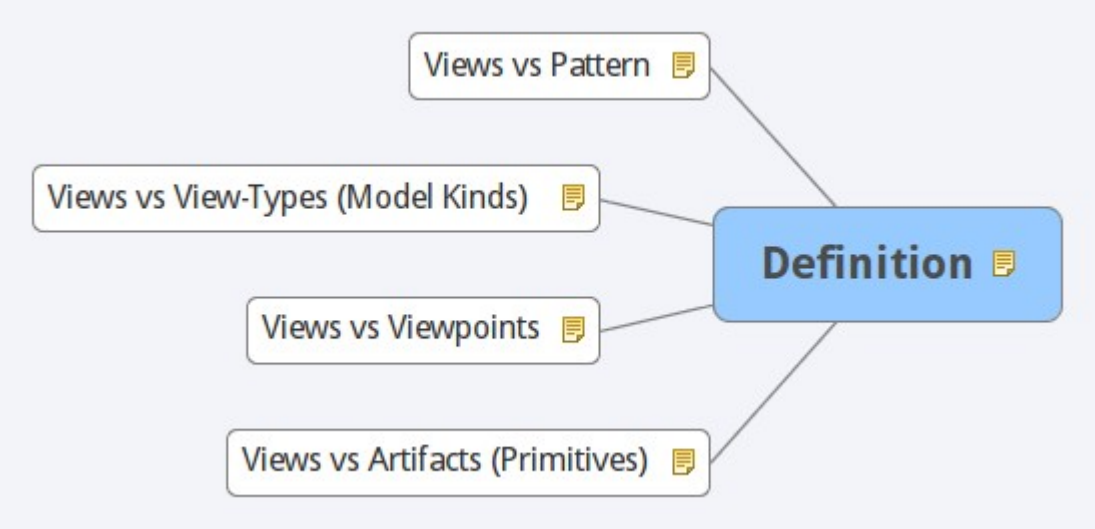
It is a way to portray all the elements of an architecture that are relevant to the concerns that the represented "View" intends to address,

...and by implication, relevant to the stakeholders for whom those concerns are important.

Views help to explain the Architecture to stakeholders and demonstrate that the goa



Definition



Views vs Artifacts (Primitives)

VIEWS are compositions of Work-Products Artifacts (i.e. Models)

...while ARTIFACTS are the "Primitives" capturing a specific aspect of the solution architecture.

Views vs Artifacts (Primitives)

VIEWS are compositions of Work-Products Artifacts (i.e. Models)

...while ARTIFACTS are the "Primitives" capturing a specific aspect of the solution architecture.

Views vs Viewpoints

A VIEW communicates the resolution of a class of Architectural concerns using VIEWPOINTS for frame of reference.

VIEWPOINTS express how to construct a VIEW, how to check the well-formedness of that VIEW, or how to analyze it to predict various properties of the system.

Views vs View-Types (Model Kinds)

A VIEW leverages A MODEL KIND / VIEW-TYPE, proposing a template for diagraming, and a modeling notation to articulate a solution design.

Examples:

- Functional View-Type (using UML or Archimate),
- Behavioral View-Type (using DFD, UML or Archimate),
- Data View-Type (using ERD, UML or Archimate)

(etc.)

Views vs Pattern

A VIEW leverages Architecture PATTERNS or/and STYLES for constructing a Solution Design.

An architecture VIEW is rooted in the business problem to solve for, and models a unique solution design (a software program).

An Architectural PATTERN or STYLE is an abstraction of multiple solution architectures that have been designed and successfully deployed to address the same types of business problems.

Purpose

The purpose of architecture views is to:

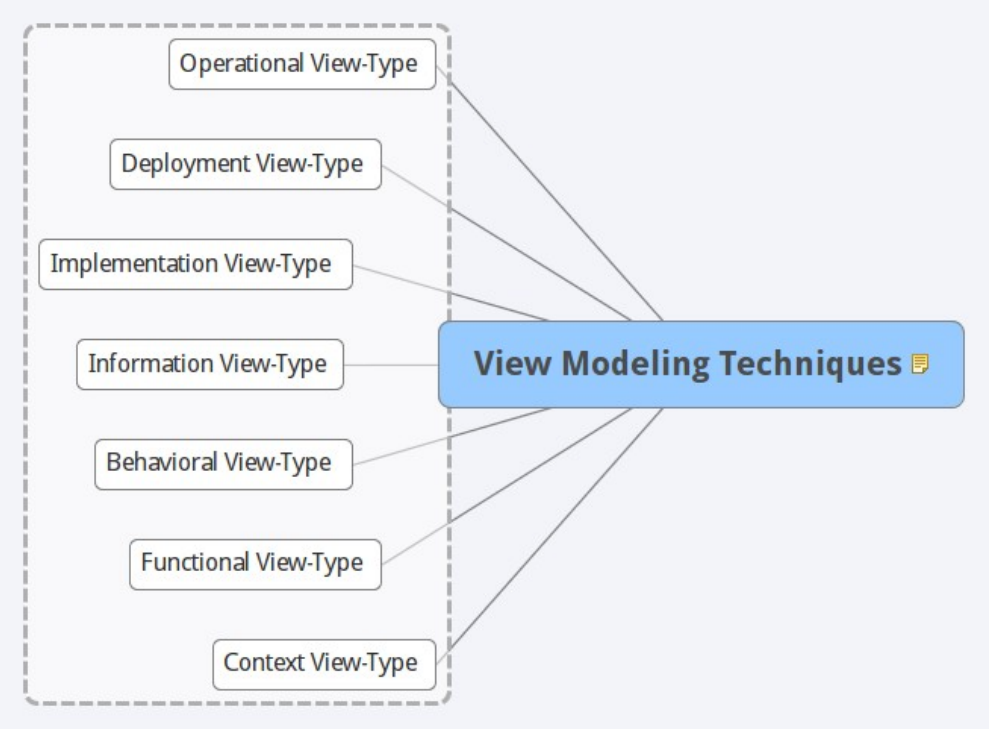
- illustrate how the solution architecture addresses (1,n) concerns held by (1,n) stakeholders.

- represent the structural and/or behavioral aspects of a proposed solution architecture.

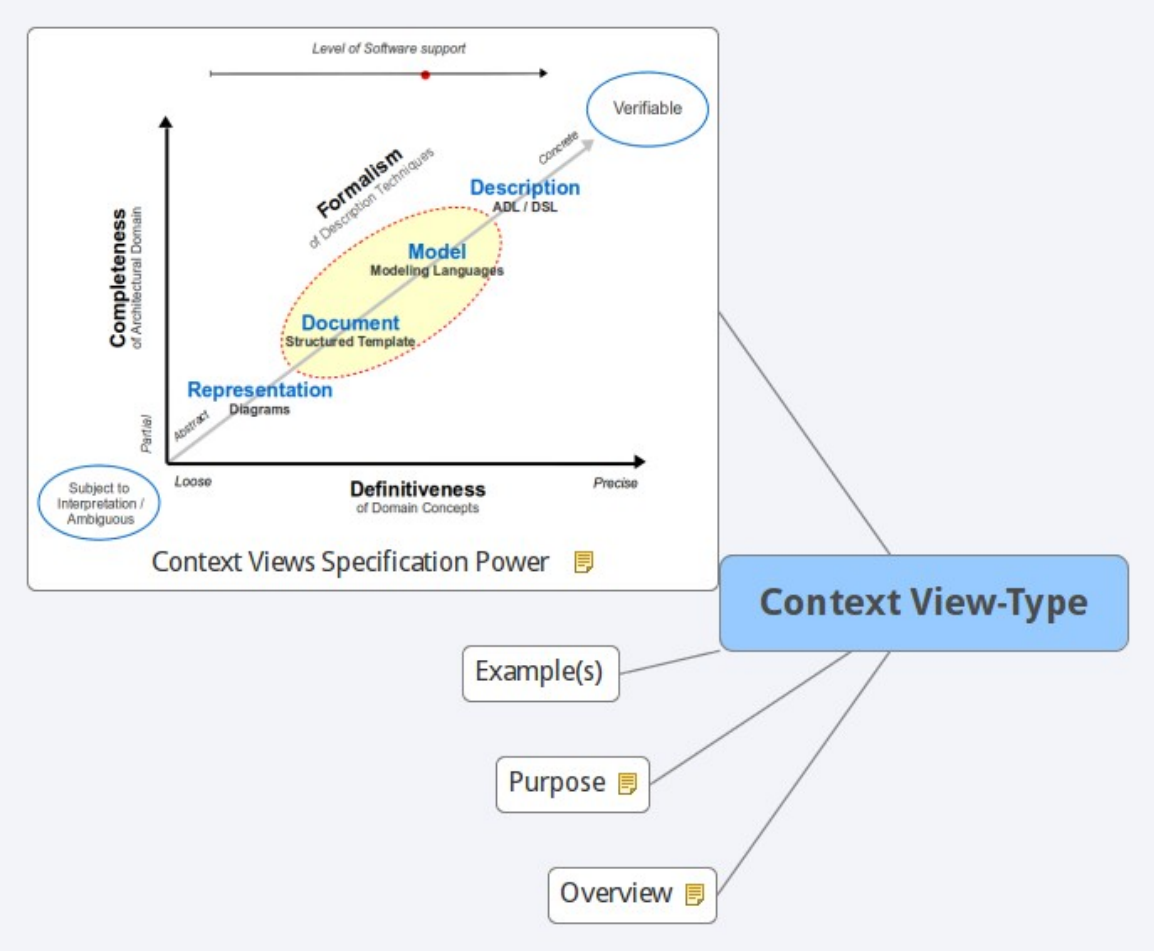
- achieve COMPLETENESS of the solution architecture.

View Modeling Techniques

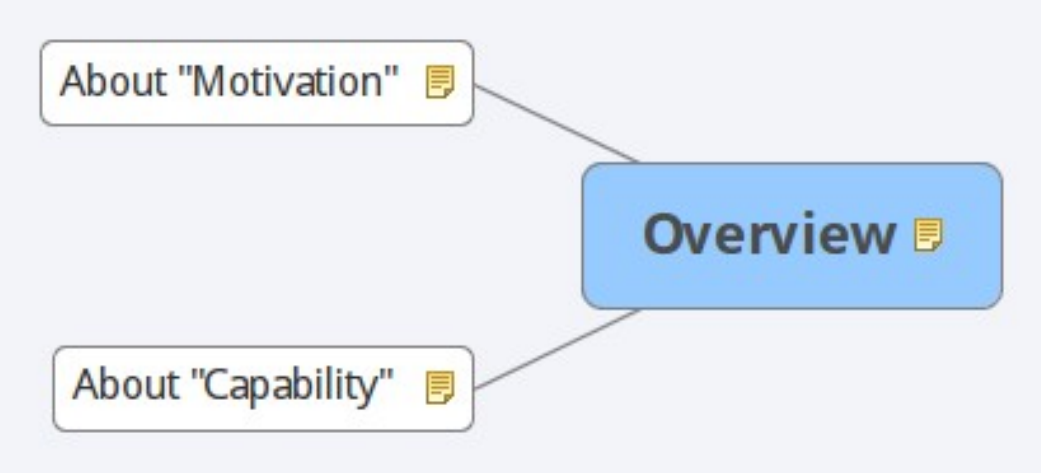
An architect settles on a collection of VIEWPOINTS and VIEW-TYPES to express the VIEWS of a solution architecture.



Context View-Type



Overview



Overview

The Context View-Type (often referred as Context View) describes the SCOPE of architecture solution and its FEATURES - ex. intent / requirements / constraints / capabilities.

Any Views adapted from this View-Type it is of interest of the many no-technical solution stakeholders, because of the simplicity of the model kinds available.

The Context View-Type plays an important role in helping stakeholders to understand the design and how it relates to their organization & objectives.

It describes the relationships between the solution and its environment (i.e. people, other systems, and external entities with which it interacts).

About "Capability"

A Capability is the "ability" that a Firm wants to possess to achieve a specific outcome.

Capabilities exclusively define WHAT an Organization does - not how, where nor why.

A Capability is described by a passive verb/noun construct vs. the active verb/noun construct used to describe a business function or process.

Concepts have only basic attributes: (1.) Identifier, (2.) Text description.

About "Motivation"

"Motivation" is driven by External pressure and as such is called "Extrinsic Motivation".

Competition is extrinsic because it forces a Firm to win and beat others, but reacting to CHANGE requires composure.

Composure means to DEFINE "why" (that is, what DESIRED results) the reaction or approach is meant to achieve.

It can be about "Growth" (i.e. Market Penetration, Market Development, Service/Product Development, Diversification), "Optimization" or "Rationalization".

It DEFINES a Course of Action (i.e. a specific behaviour or "Tactic").

It also refers to the measured performance of an activity in order to attain an outcome, leading to the DEFINITION of an orientation, and establishment of its directional components.

Purpose

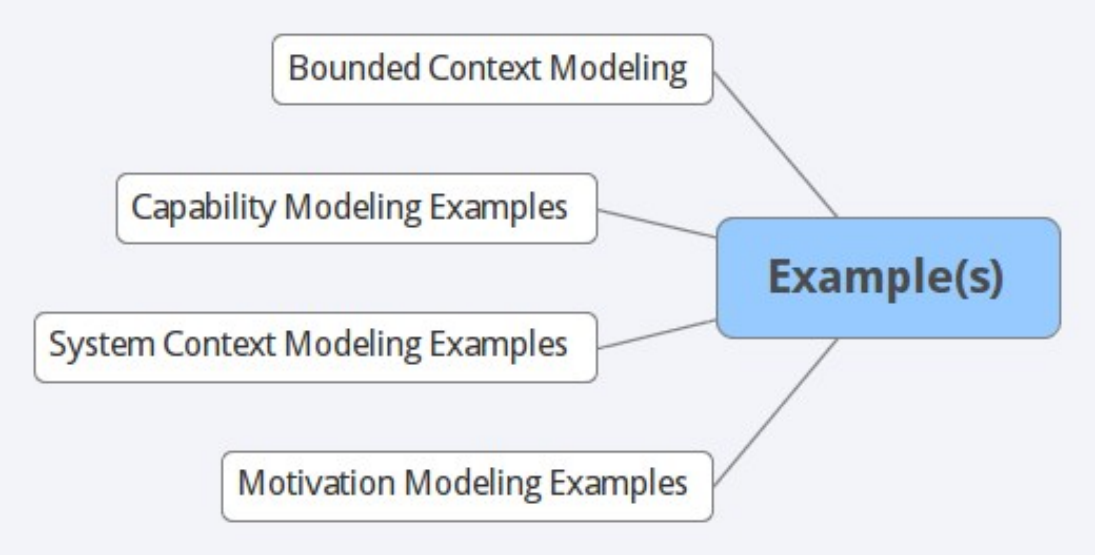
The Context View-type specifies a "scope of responsibility" for an Architecture, in terms that align with Business Motivations.

Work-Products in this View describe the responsibilities of a System with its Environment.

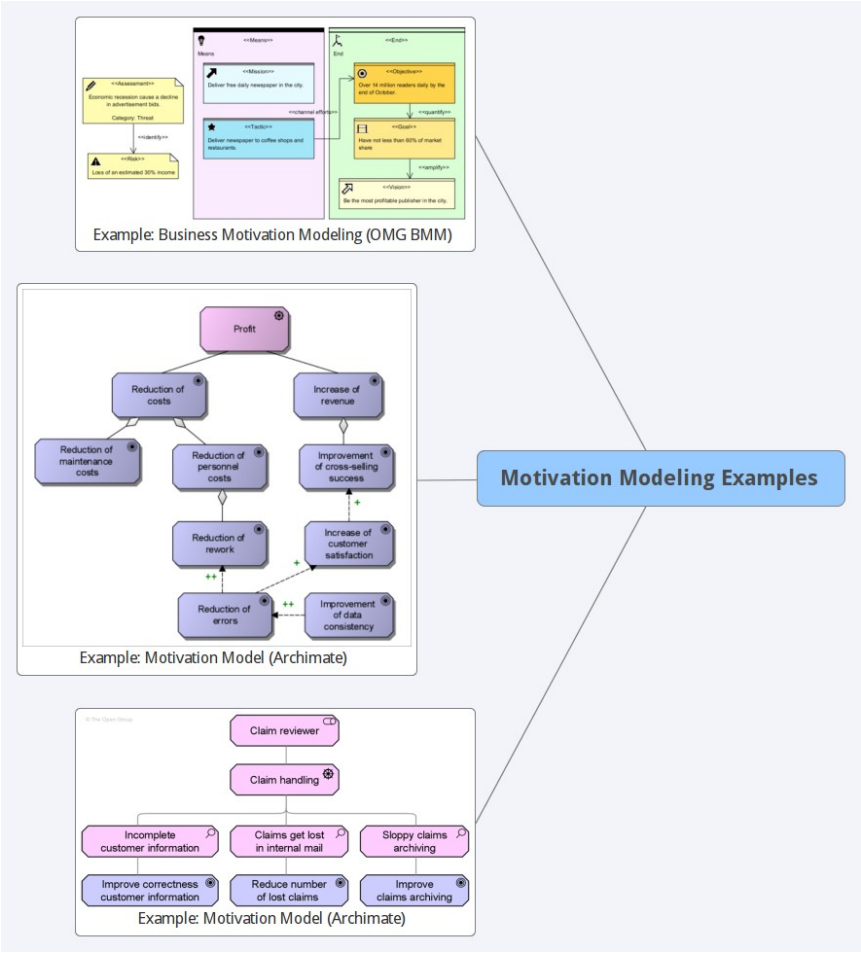
They specify the nature of internal & external entities in the Environment i.e. Actors (People, Systems).

It sets the CONTEXT for (1.) Associations, (2.) Dependencies, (3.) Interactions between Entities.

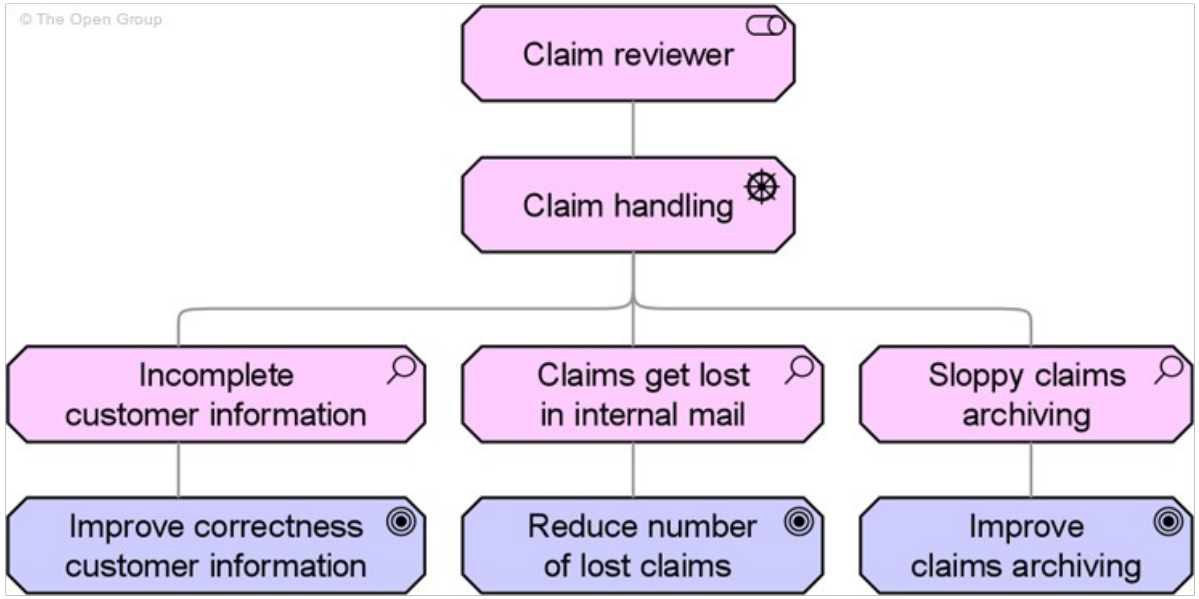
Example(s)



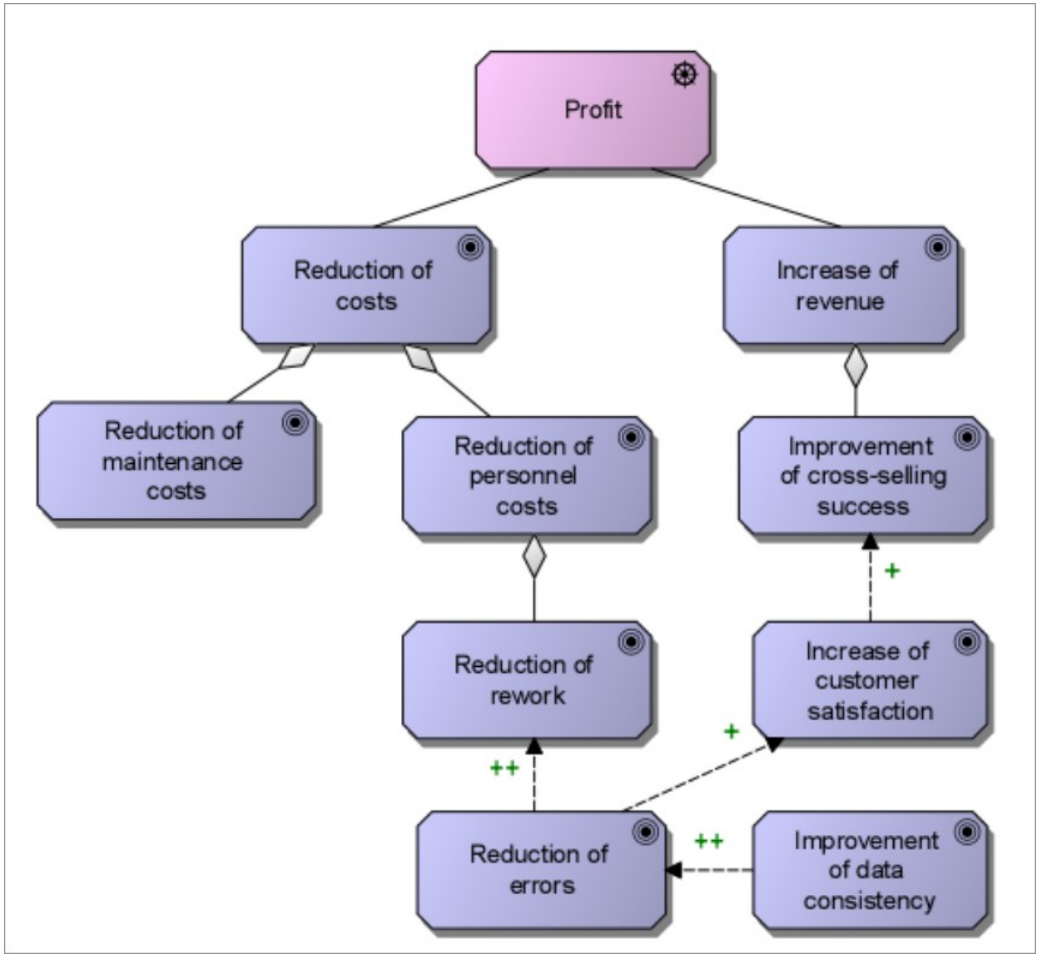
Motivation Modeling Examples



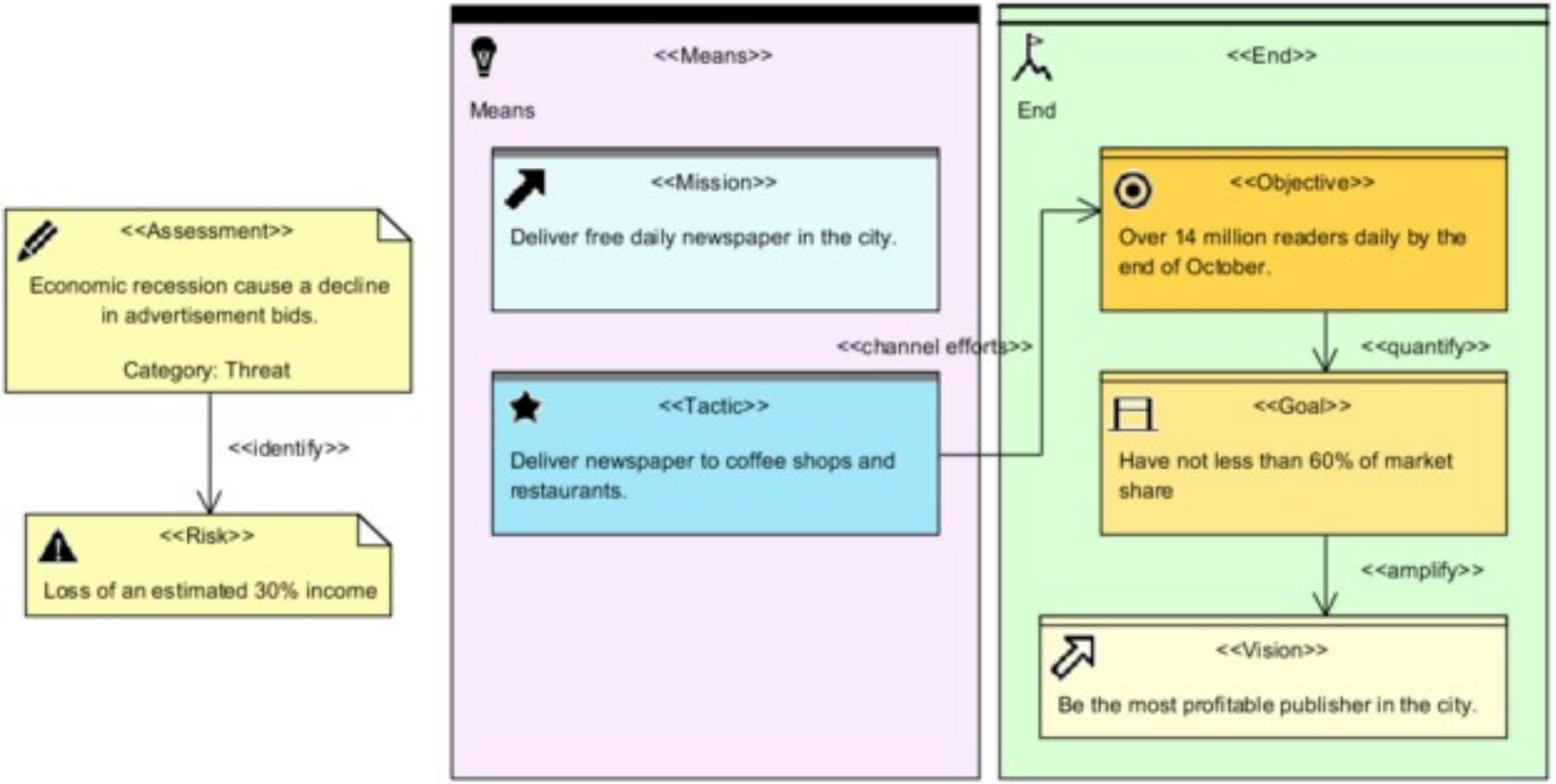
Example: Motivation Model (Archimate)



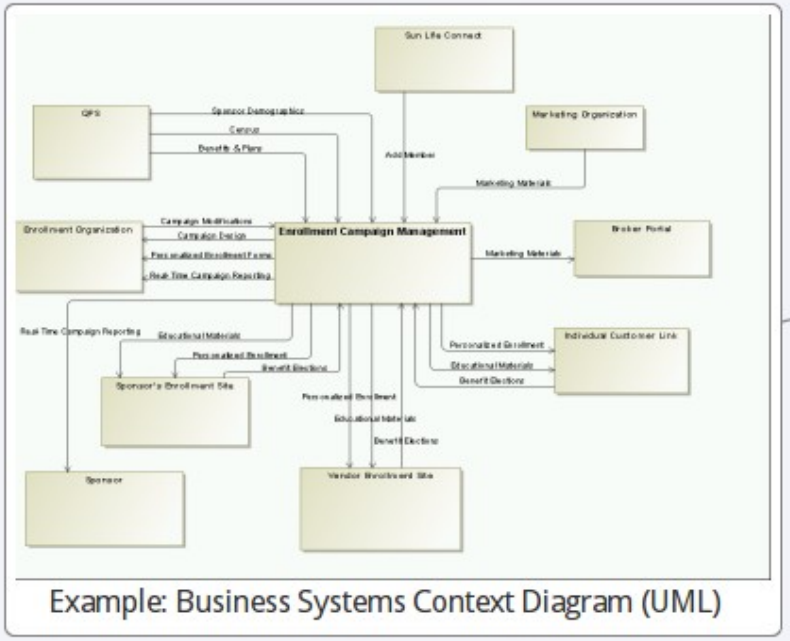
Example: Motivation Model (Archimate)



Example: Business Motivation Modeling (OMG BMM)



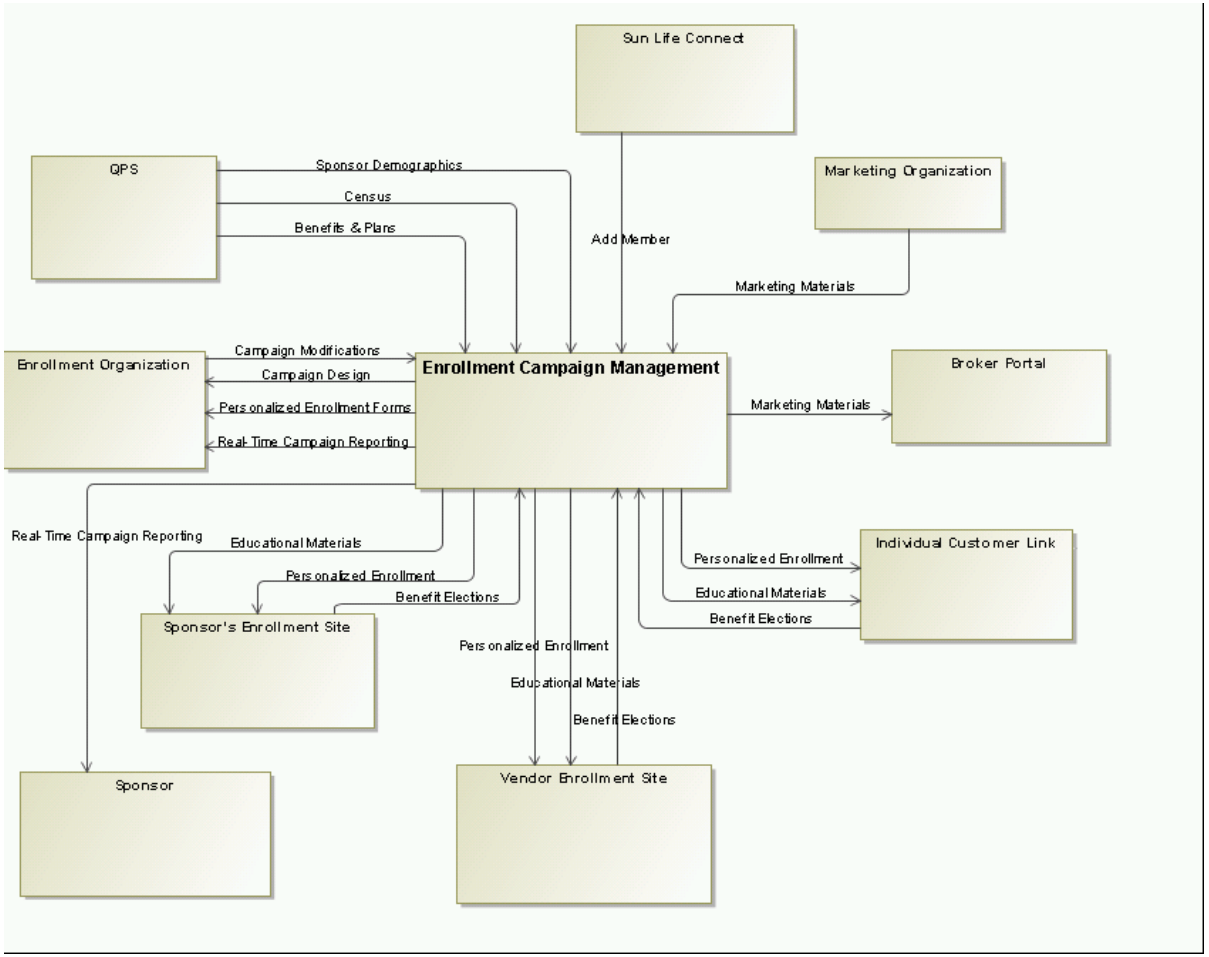
System Context Modeling Examples



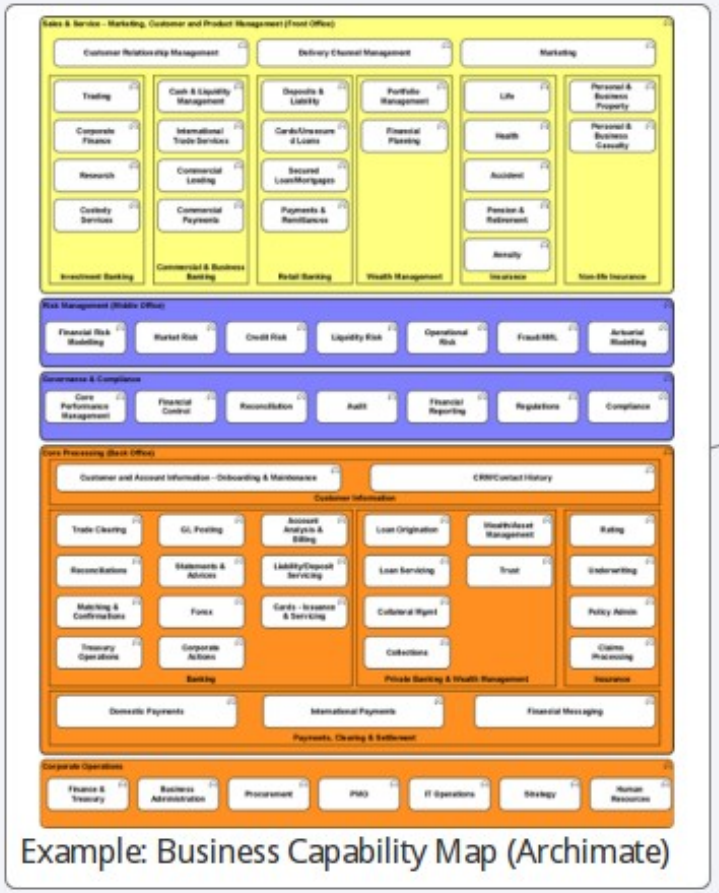
Example: Business Systems Context Diagram (UML)

System Context Modeling Examples

Example: Business Systems Context Diagram (UML)

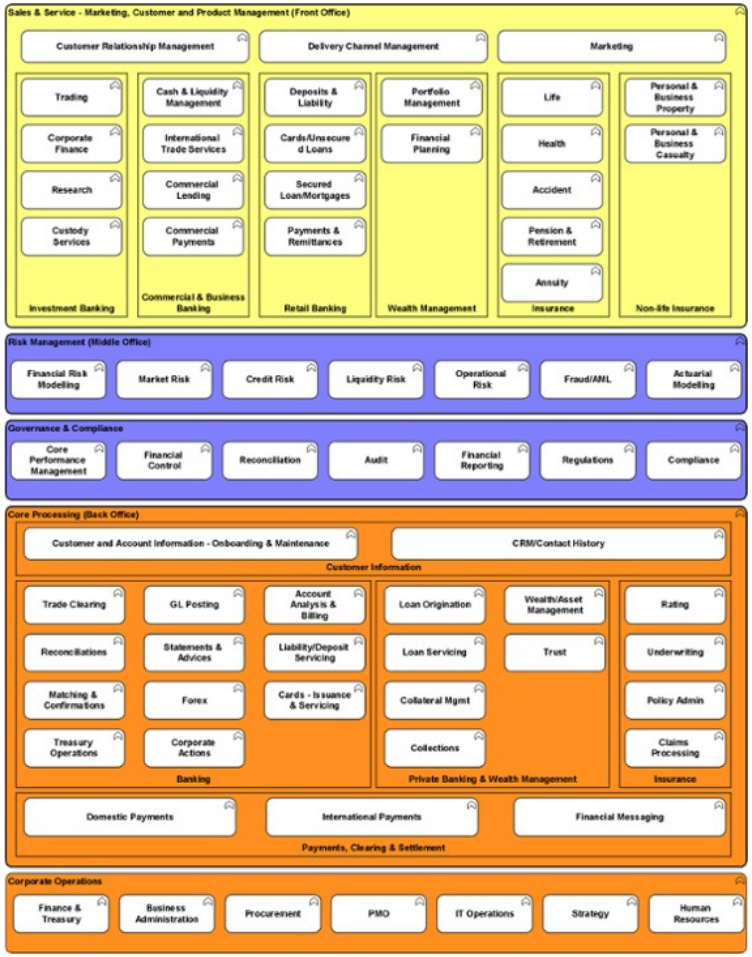


Capability Modeling Examples

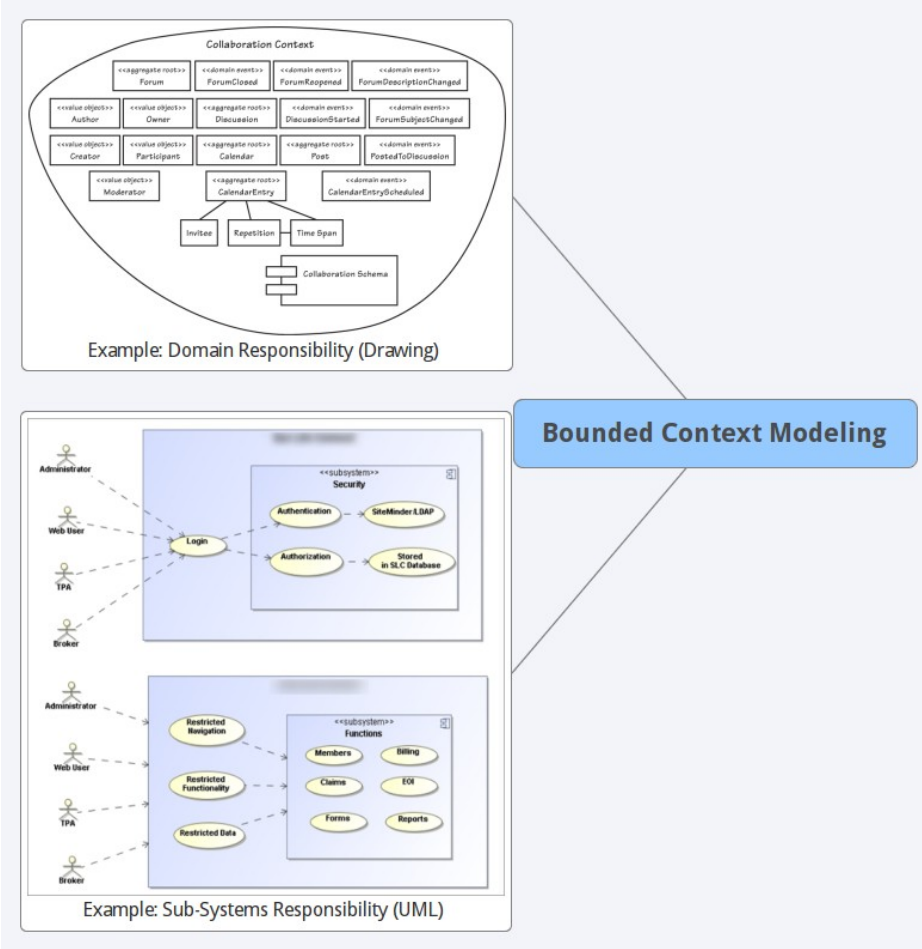


Capability Modeling Examples

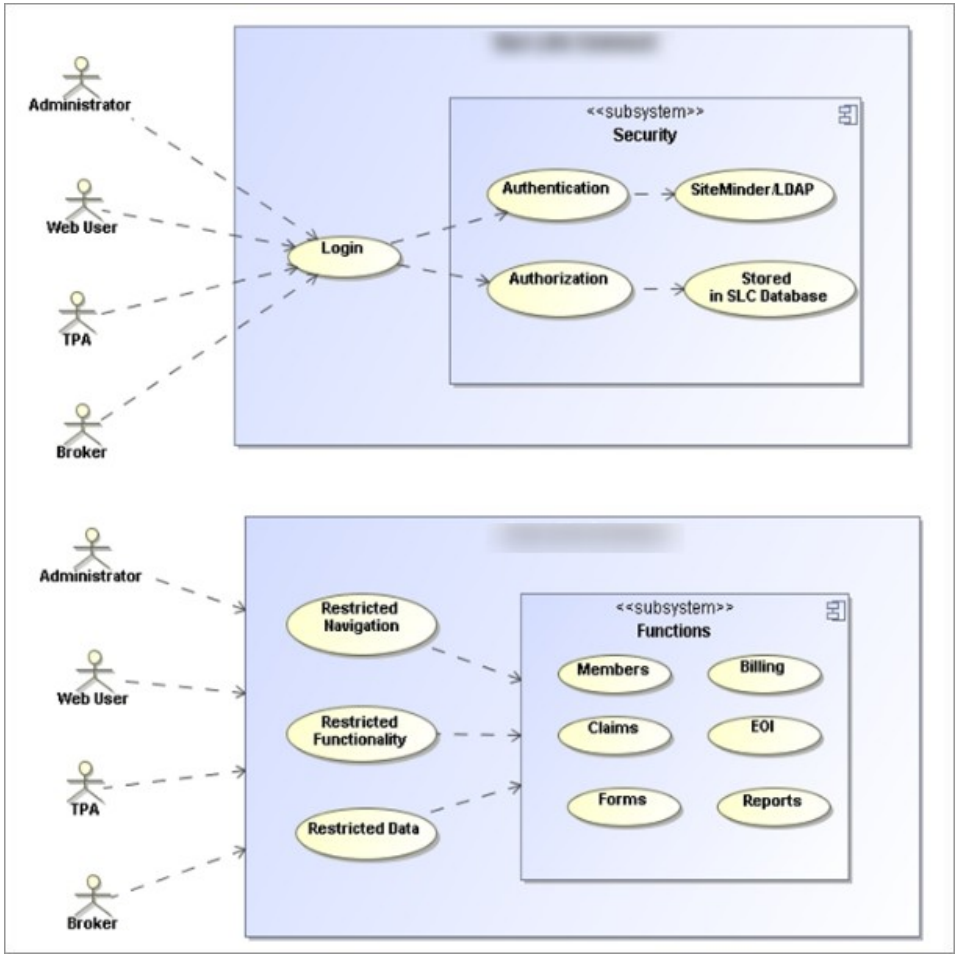
Example: Business Capability Map (Archimate)



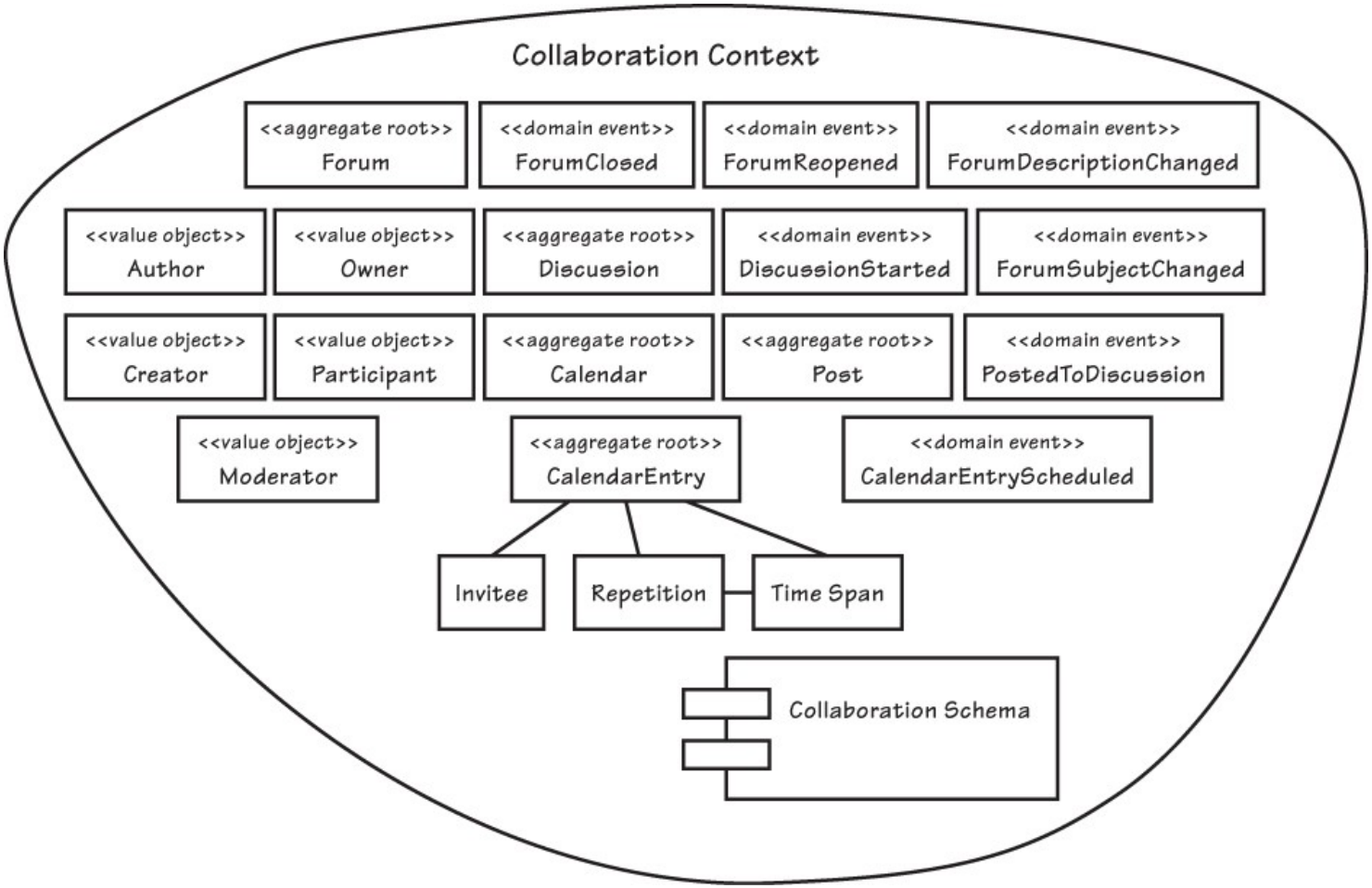
Bounded Context Modeling



Example: Sub-Systems Responsibility (UML)



Example: Domain Responsibility (Drawing)



Context Views Specification Power

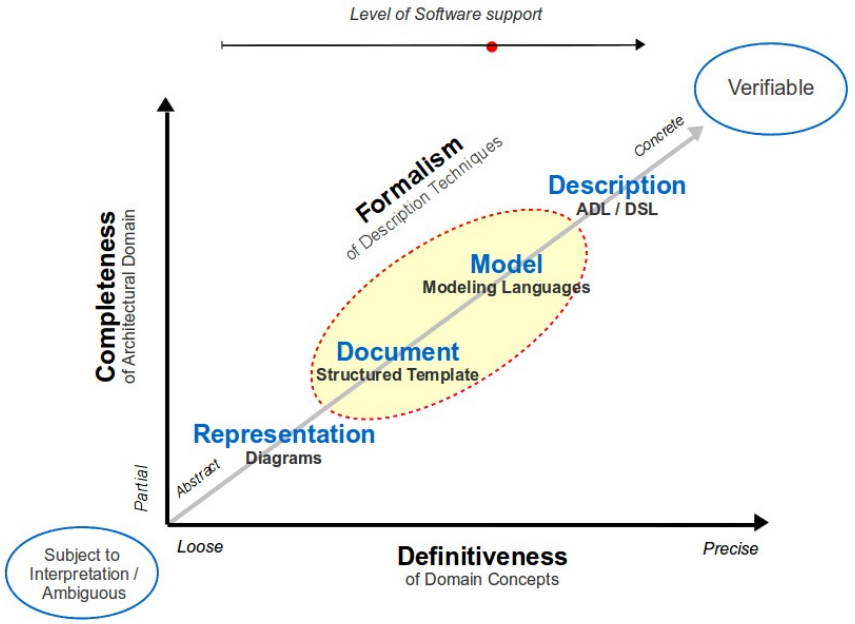
Typically a:

- motivation model (Archimate, BMM)

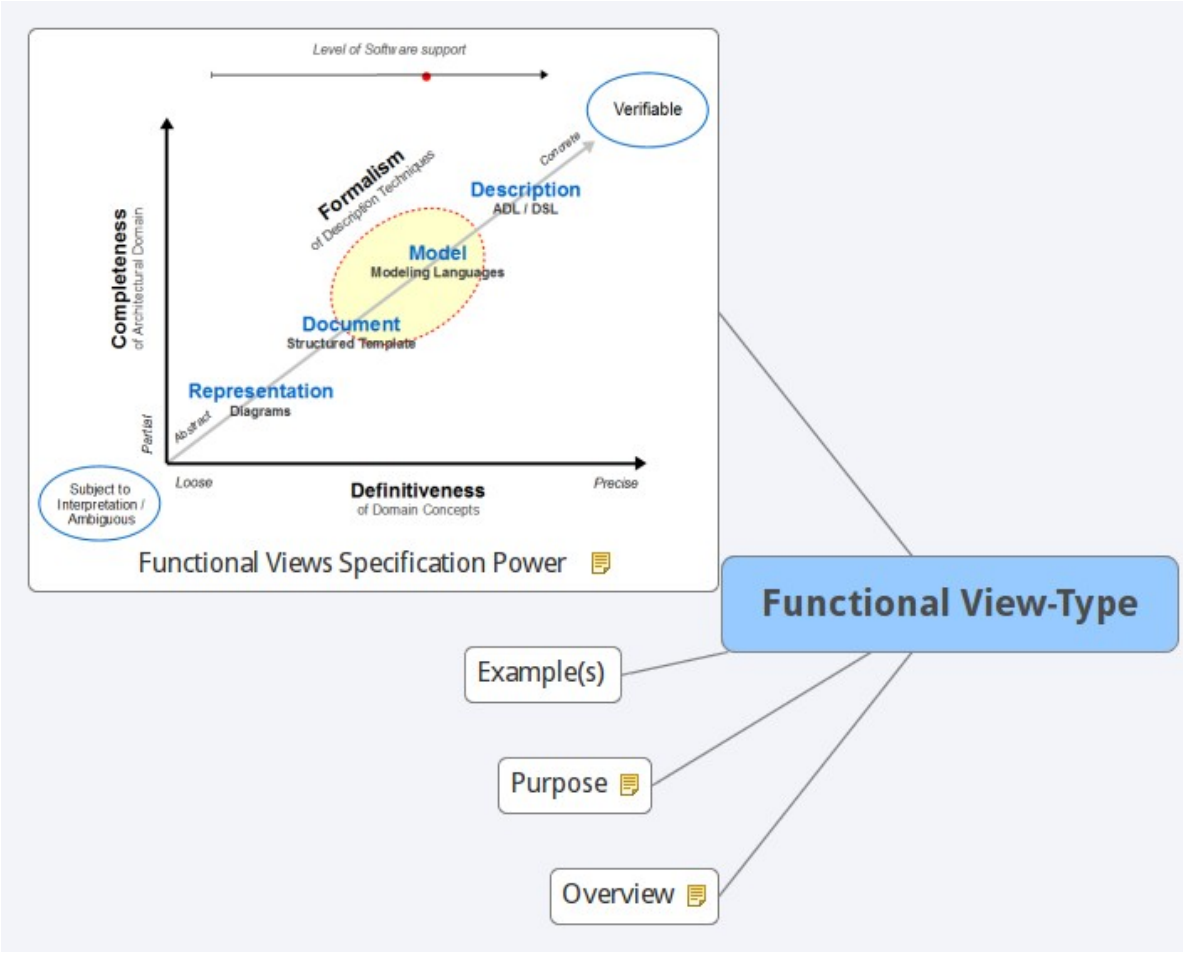
- system context diagram (UML, Archimate)

- capability map (Archimate)

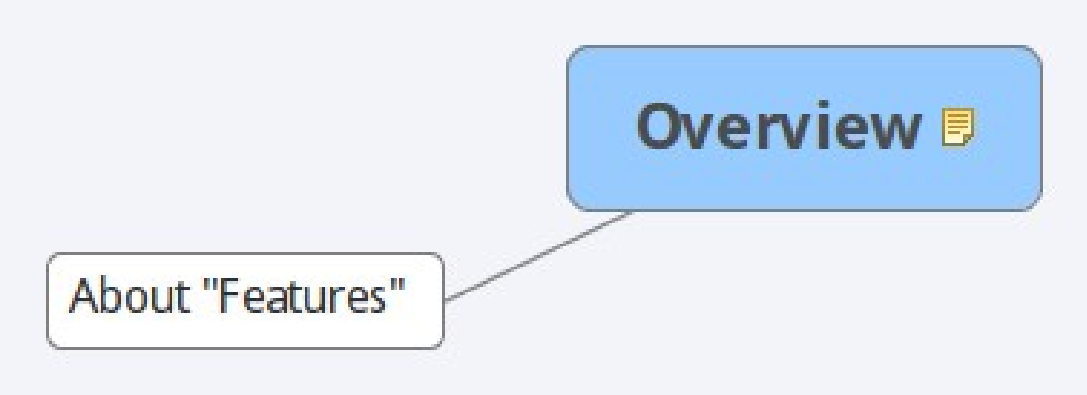
- sub-systems responsibility diagram (UML, Archimate)



Functional View-Type



Overview



Feature: Definition

Kang et al. [70]: “a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems”

Kang et al. [71]: “a distinctively identifiable functional abstraction that must be implemented, tested, delivered, and maintained”

Czarnecki and Eisenecker [47]: “a distinguishable characteristic of a concept (e.g., system, component) that is relevant to some stakeholder of the concept”

Bosh [35]: “a logical unit of behaviour specified by a set of functional and non-functional requirements”

Chen et al. [40]: “a product characteristic from user or customer views, which essentially consists of a cohesive set of individual requirements”

Classen et al. [42]: “a triplet, f = (R,W, S), where R represents the requirements the feature satisfies, W the assumptions the feature takes about its environment and S its specification”

Zave [129]: “an optional or incremental unit of functionality”

Apel et al. [21]: “a structure that extends and modifies the structure

Feature: Purpose

The purpose of an Application or Component Feature is to satisfy a requirement, represent a design decision, and provide a potential configuration option.

Any software system can be decomposed in the features it provides.

The goal of the decomposition is to construct well-structured software that can be tailored to the needs of the user and the application scenario.

From a set of features, many different software systems can be generated that share common features and differ in other features.

Purpose

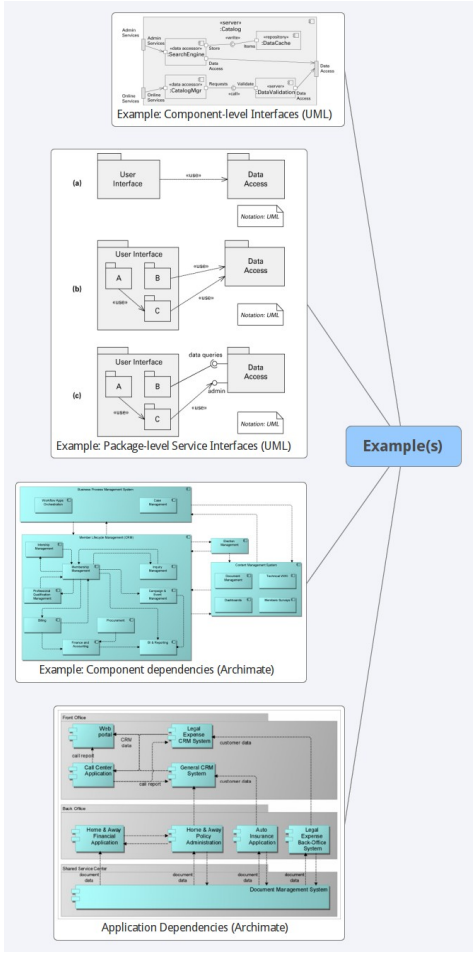
The purpose of the Functional View-Type is:

- to clearly identify the concerns addressed by key architectural components

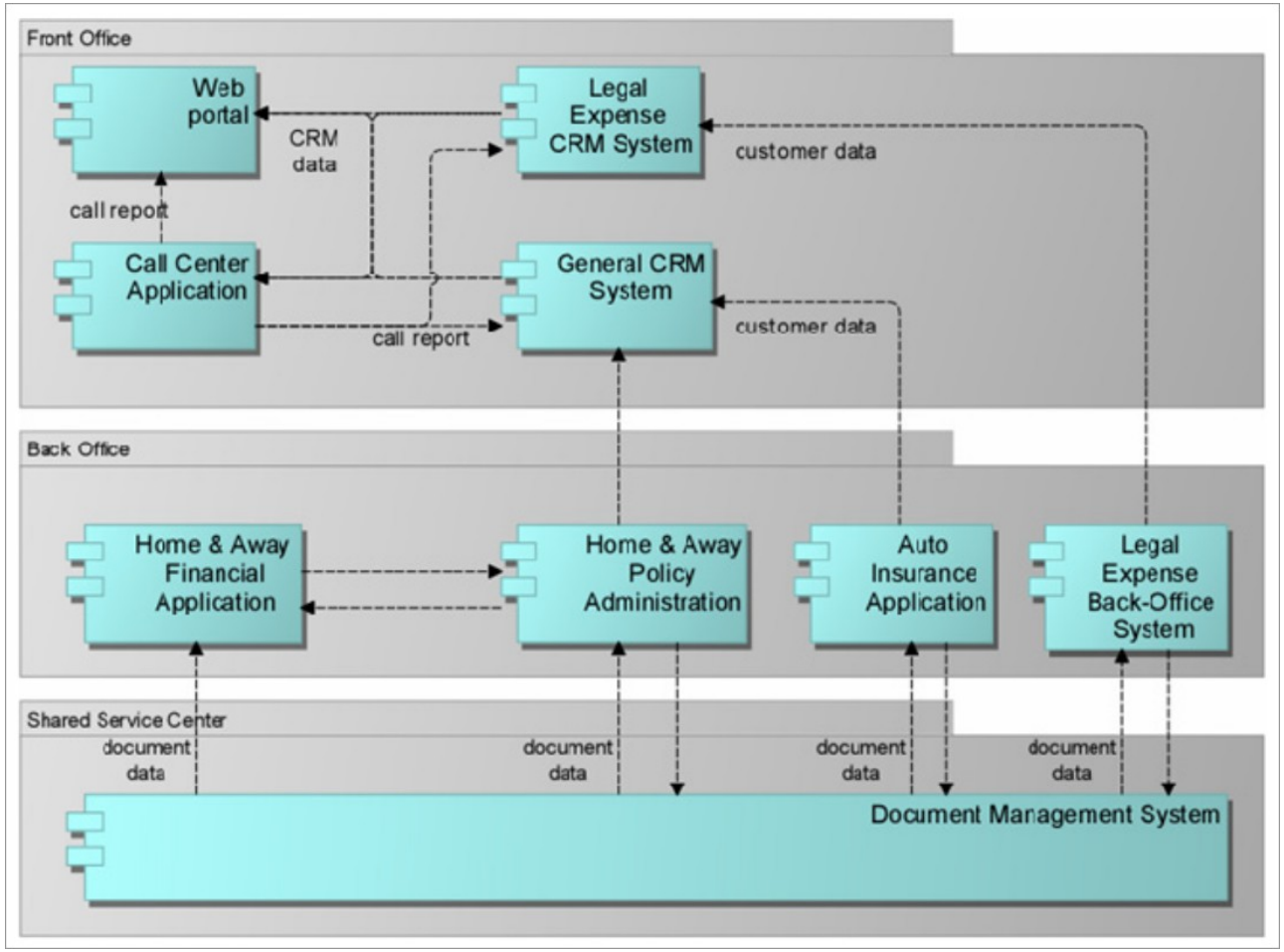
- to outline and describe the key features of an application or component answering concerns

- to decide on what levels of indirection are applicable to a given situation (patterns, styles).

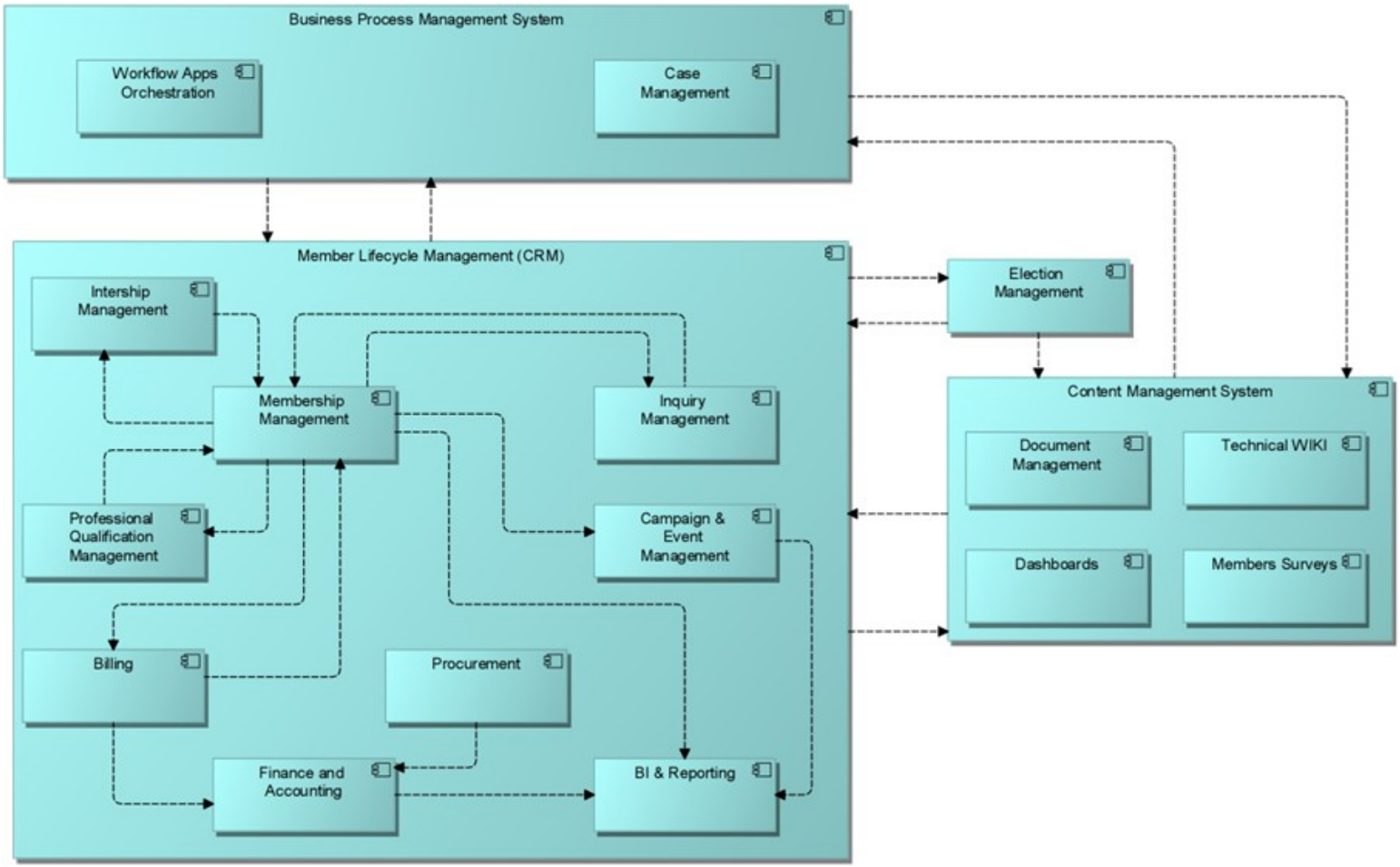
Example(s)



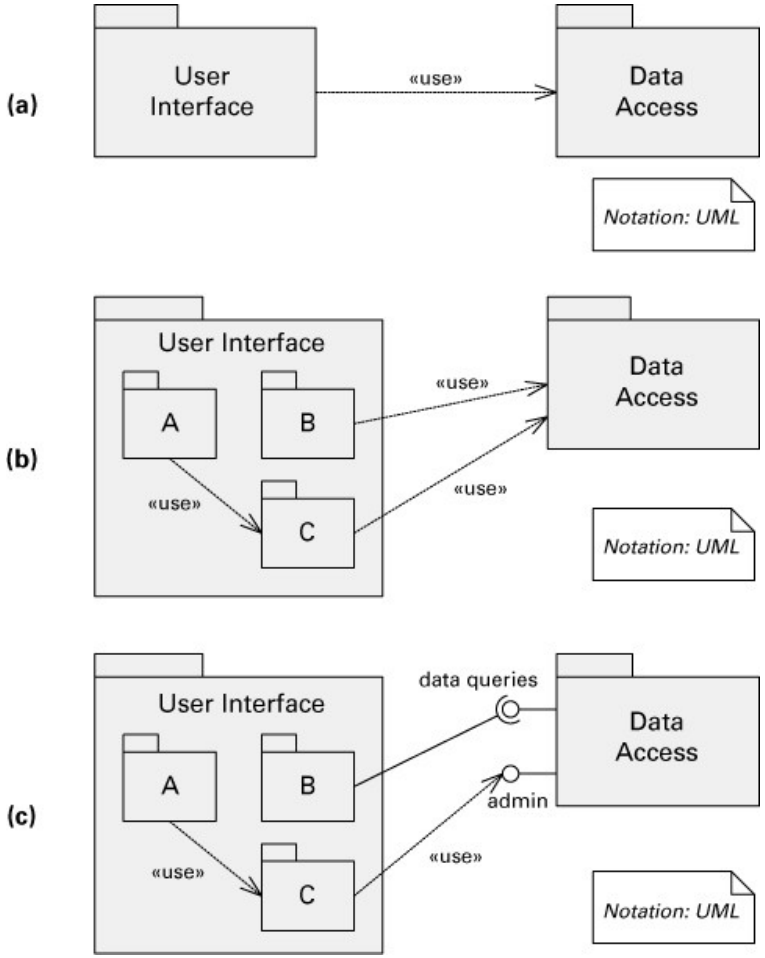
Application Dependencies (Archimate)



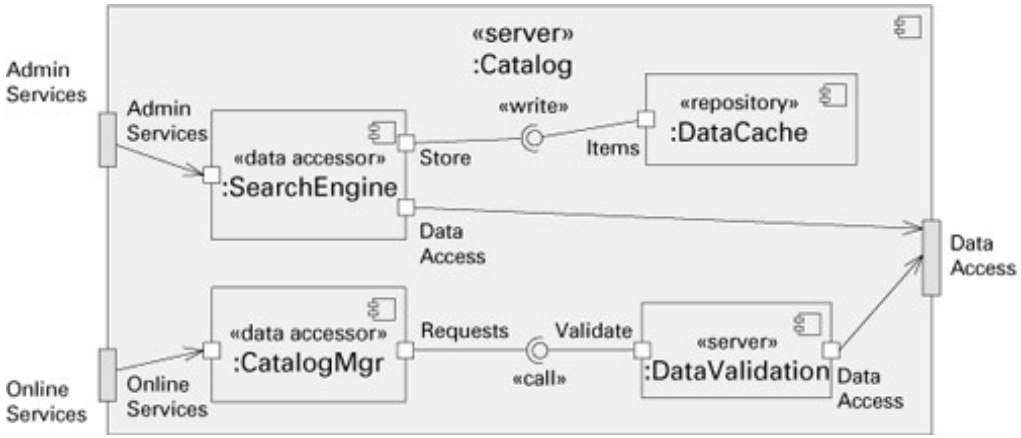
Example: Component dependencies (Archimate)



Example: Package-level Service Interfaces (UML)



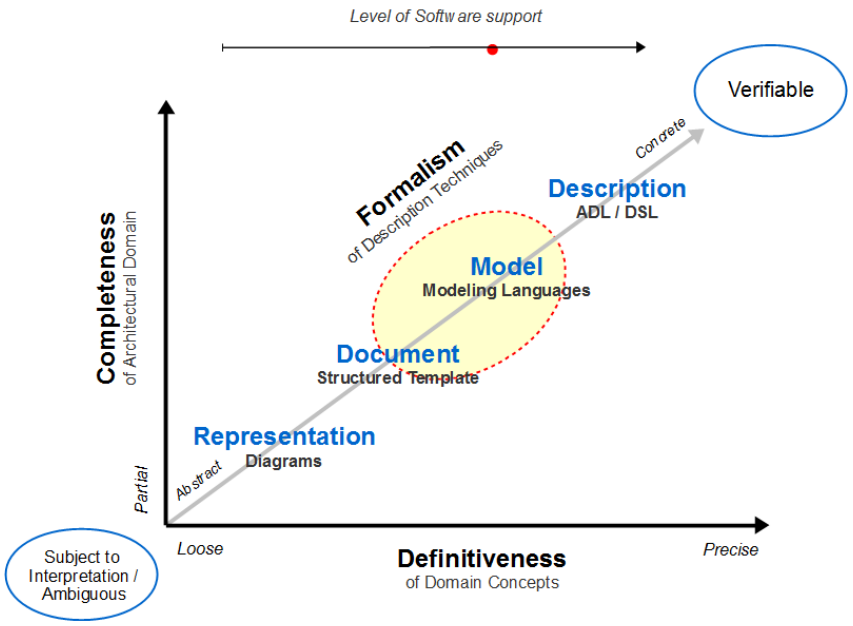
Example: Component-level Interfaces (UML)



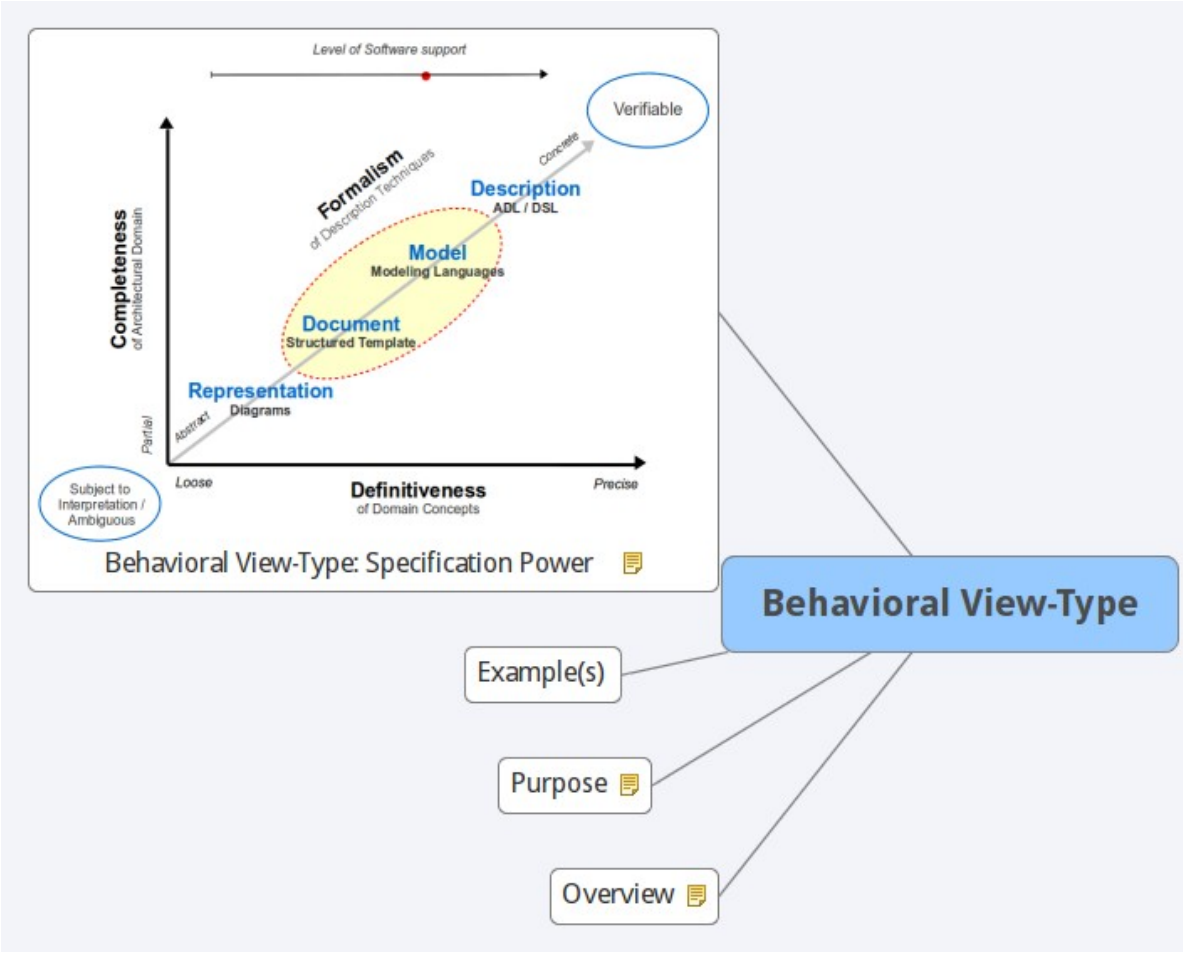
Functional Views Specification Power

Typically a:

- package diagram (UML)
- a component diagram (UML)
- an application structure diagram (Archimate)
- an application co-operation diagram (Archimate)



Behavioral View-Type



Overview

The "Behavioral View" describes the decision/collaboration logic between sub-systems/components in terms of their:

(1.) Triggers & Events,

(2.) Flow-Logic execution tree,

(3.) Flow-Synchronization.

Work-Products in this View identify when system tasks can execute (sequentially or in parallel) and who coordinates, controls and integrates their input/execution/output.

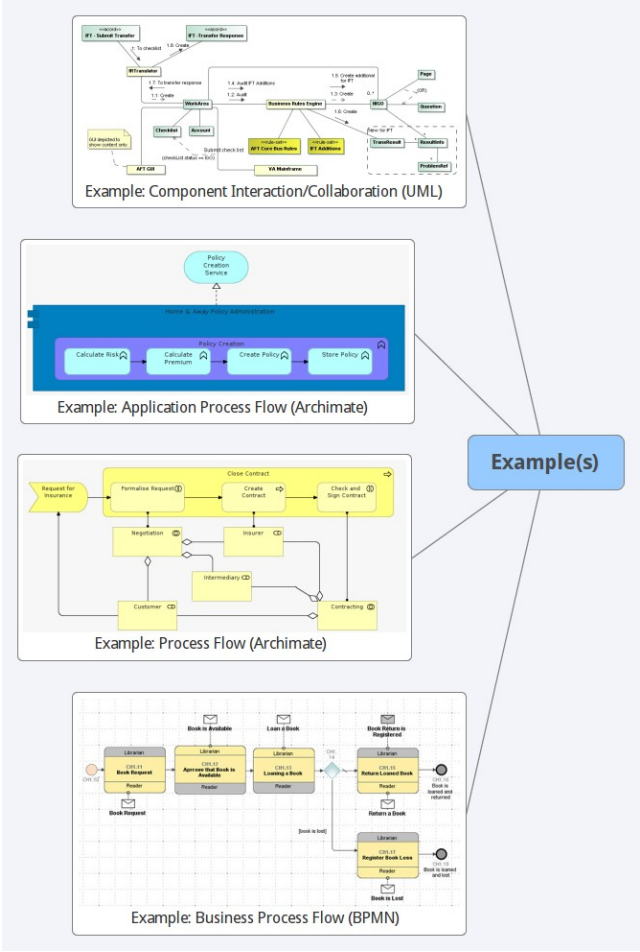
Purpose

The purpose of the Behavioral View-Type is to:

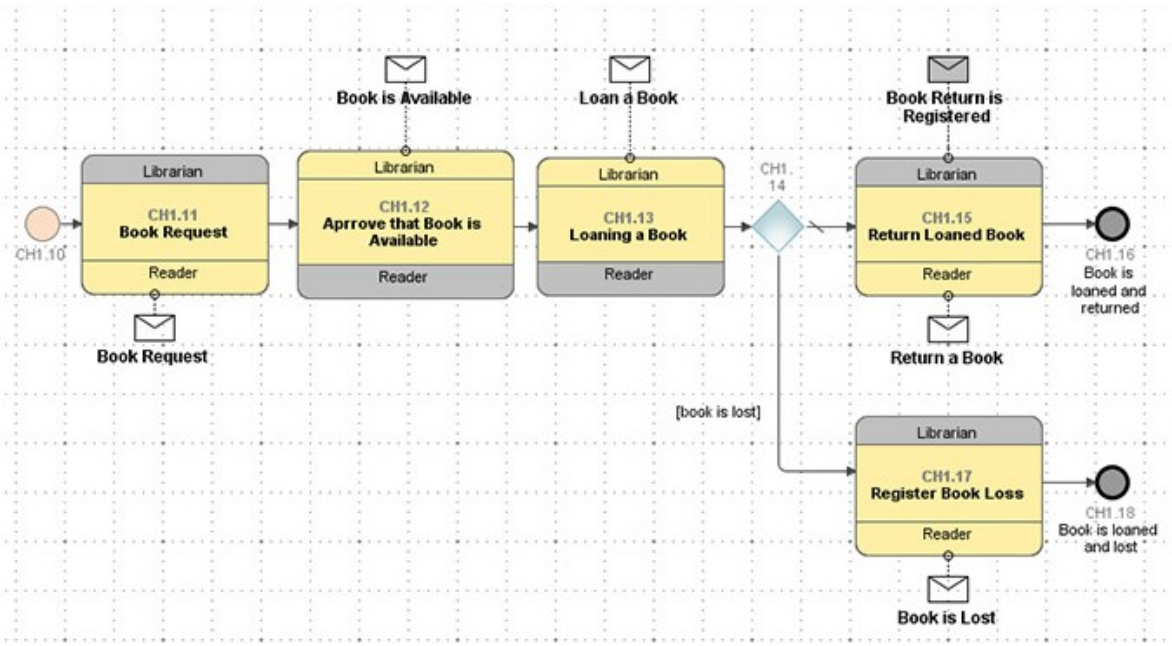
- describe the logical process flow of the system logic by sequencing functional elements to clearly identify the parts of the system that can execute concurrently and how this is coordinated and controlled

- show the communication mechanisms used to coordinate operations between functional elements.

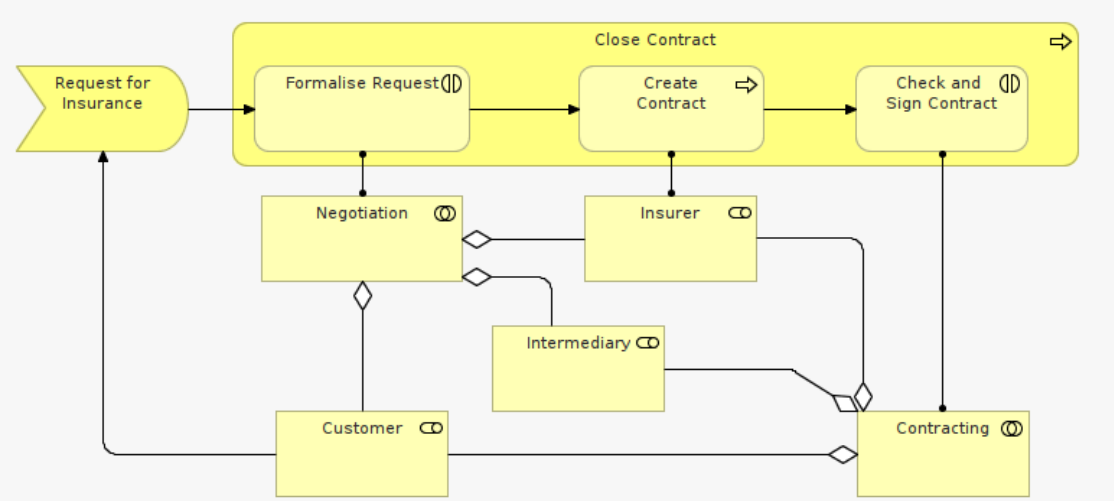
Example(s)



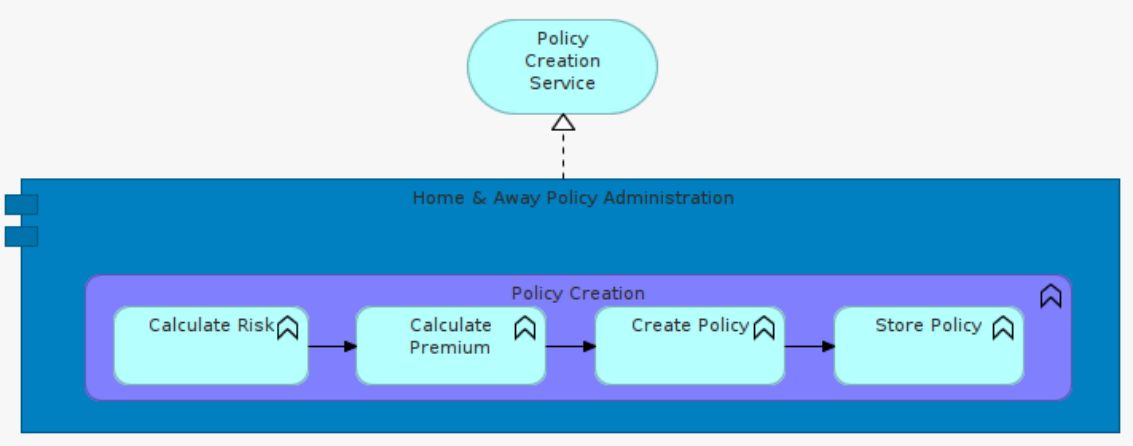
Example: Business Process Flow (BPMN)



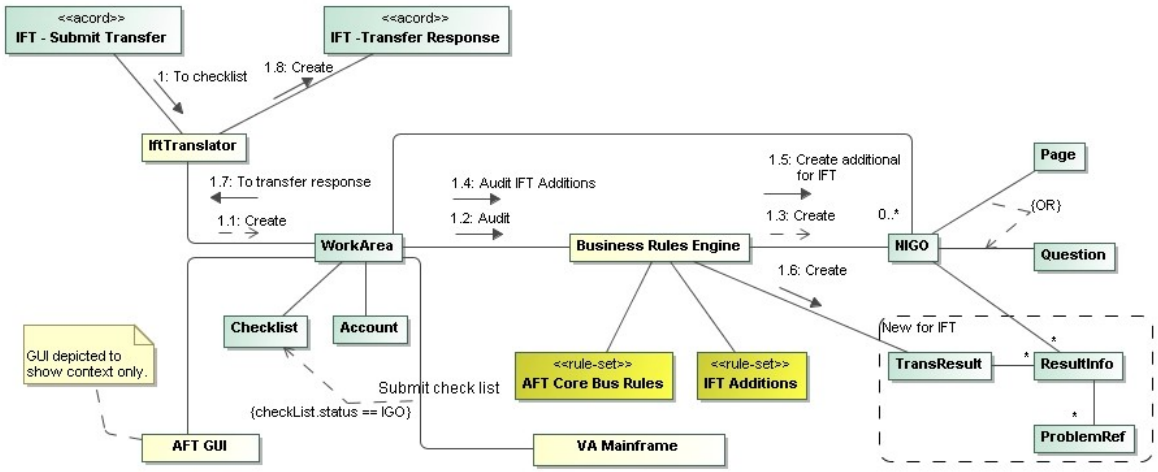
Example: Process Flow (Archimate)



Example: Application Process Flow (Archimate)



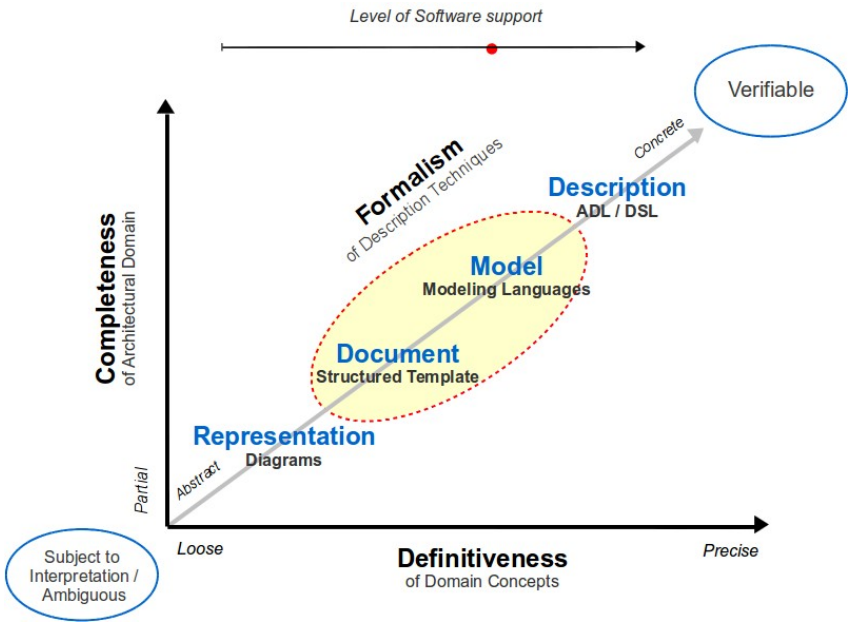
Example: Component Interaction/Collaboration (UML)



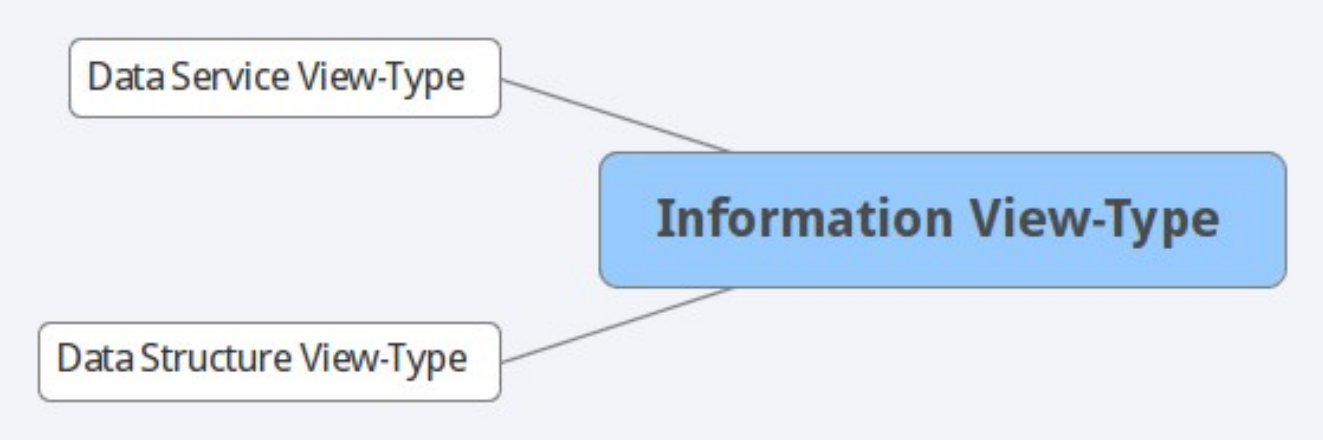
Behavioral View-Type: Specification Power

Typically a:

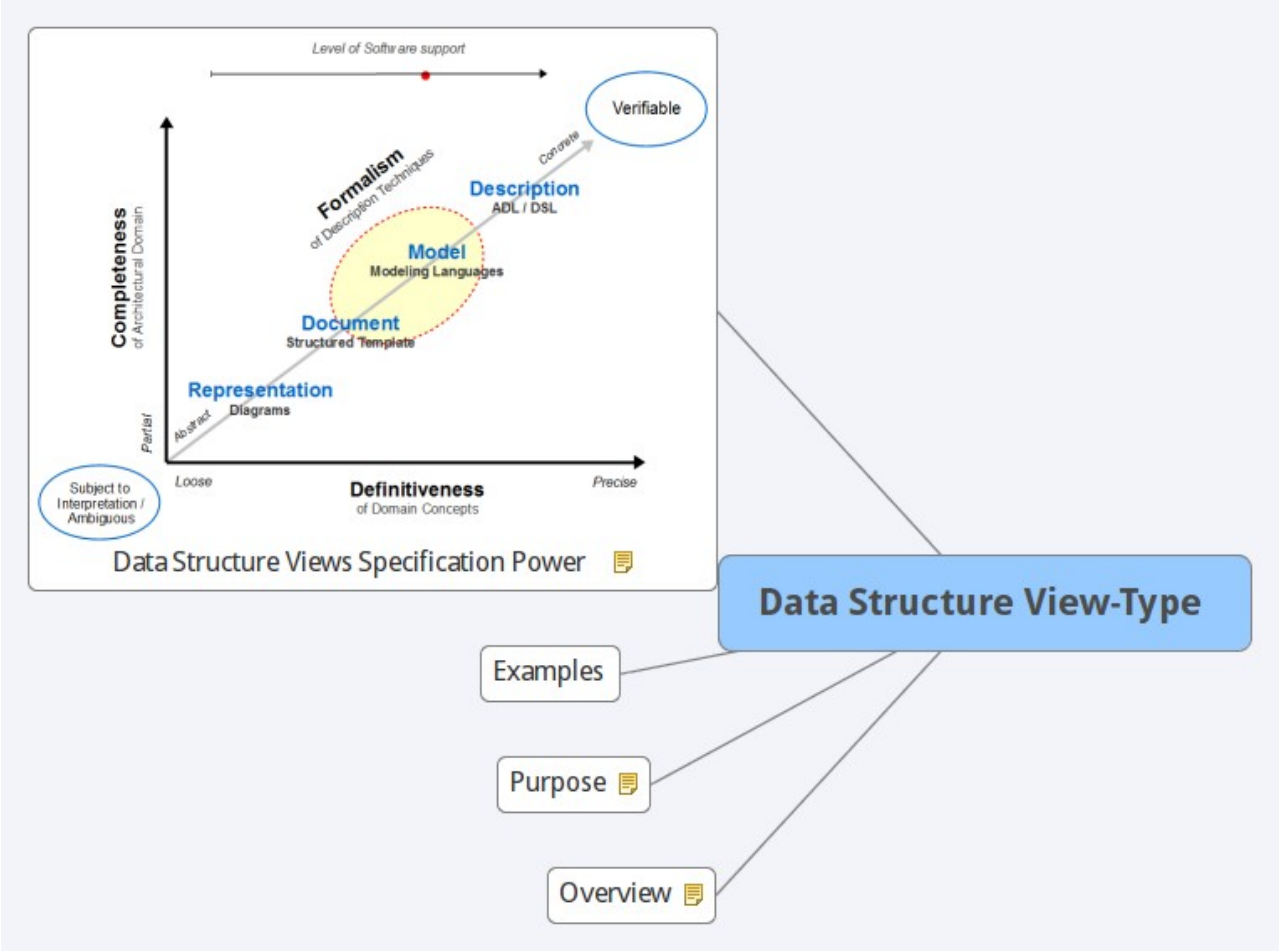
- business process flow (BPMN, Archimate)
- application process plow (Archimate)
- component interaction/collaboration flow (UML)
- component sequence diagram (UML)
- activity diagram (UML)



Information View-Type



Data Structure View-Type

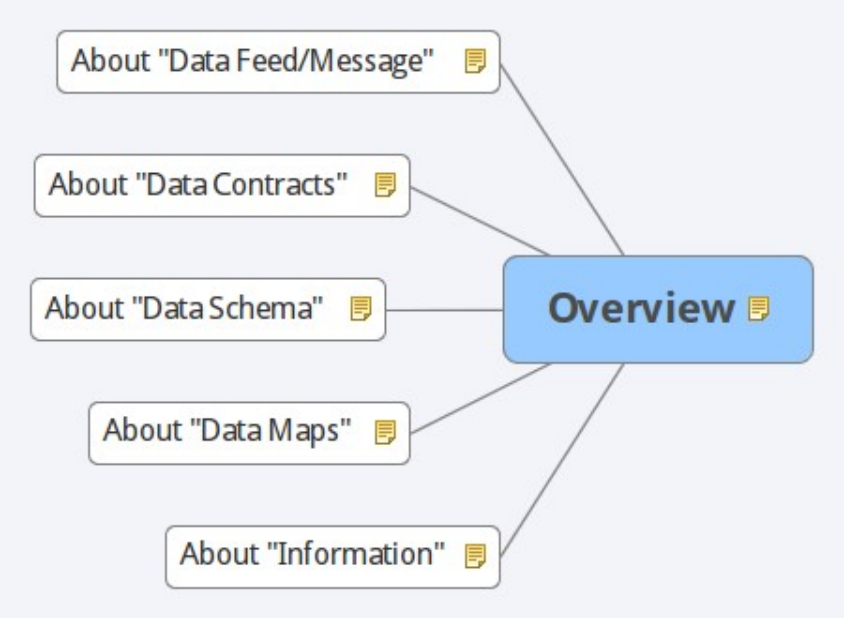


Overview

The "Data Structure View-Type" describes the way that the architecture stores information (referred as data at rest).

This view-type develops a complete but high-level view of static data structures used by the application solution.

It answers important questions around authoritative sources of master data and data format.



About "Information"

An important objective of an architecture is to clarify and obtain consensus on the definition of data concepts (i.e. data entities) at play in an application design.

"Information" is built on top of factual data collected, generated, updated by users interacting with applications.

"Information" is (1.) the composition of data points into a business concept (2.) valid for a given context of interpretation.

About "Data Maps"

A "Data Map" maps data concepts to real world Business concepts, terms and definitions.

It provides an inventory and classification for the types of data entities at play in an application.

It show the primary relationships between data entities.

It transforms business concept into data assets for the company, qualifying the relevant, timeliness, accuracy of the data managed by the Firm.

About "Data Schema"

A data schema is the skeleton structure that represents the logical view of the data source underpinning an application design.

It defines how the data is organized and what relations between data entities exist, to associate and create other composite data entities.

It formulates all the constraints that are to be applied on the data to assert its integrity (format, reference keys, etc).

About "Data Contracts"

A data contract is a formal agreement between a service and a client that abstractly describes the data to be exchanged.

To communicate, the client and the service do not have to share the same types, only the same data contracts.

It defines how the data entities are organized in a data message, and optionally how to access the data.

About "Data Feed/Message"

A data feed / data message can be loosely described as an object that contains the data being transferred between system actors.

Depending of the technology choices in architecture, it can take diverse physical forms, and be distributed in many diverse ways.

Purpose

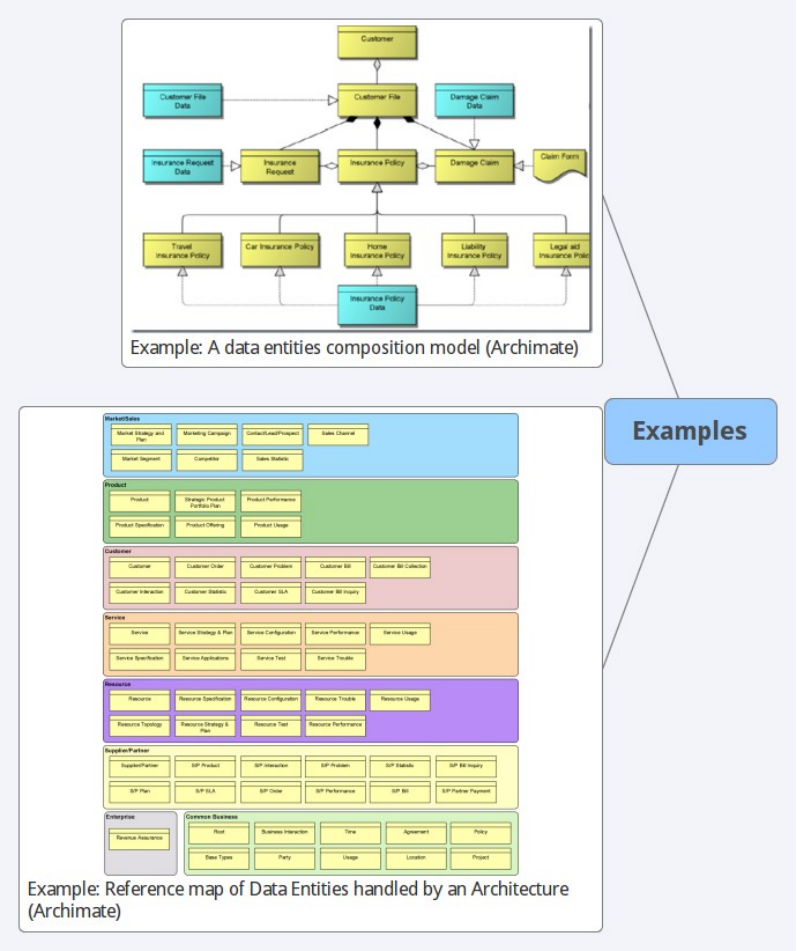
The purpose of the Data-at-Rest View-Type is to:

- to define and reference any architecturally significant data structures for stored and transient data (ex. data models)

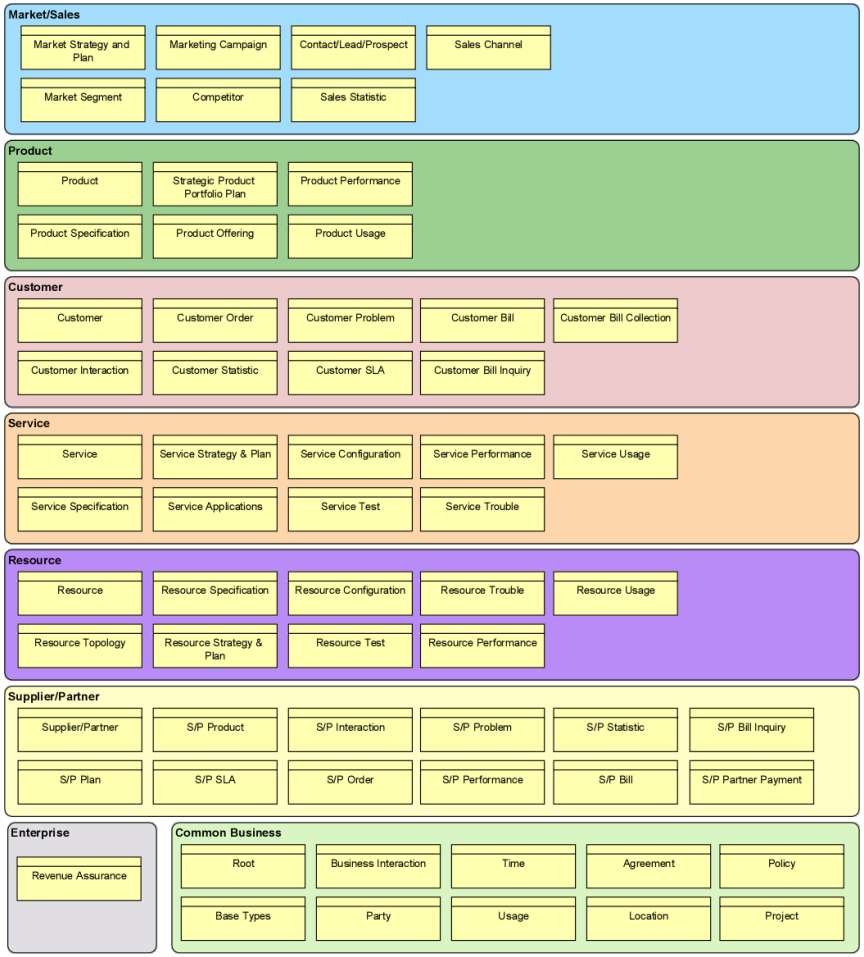
- show the data exchanged between functional elements

- to answer the key questions around data structure, ownership, currency, (etc.)

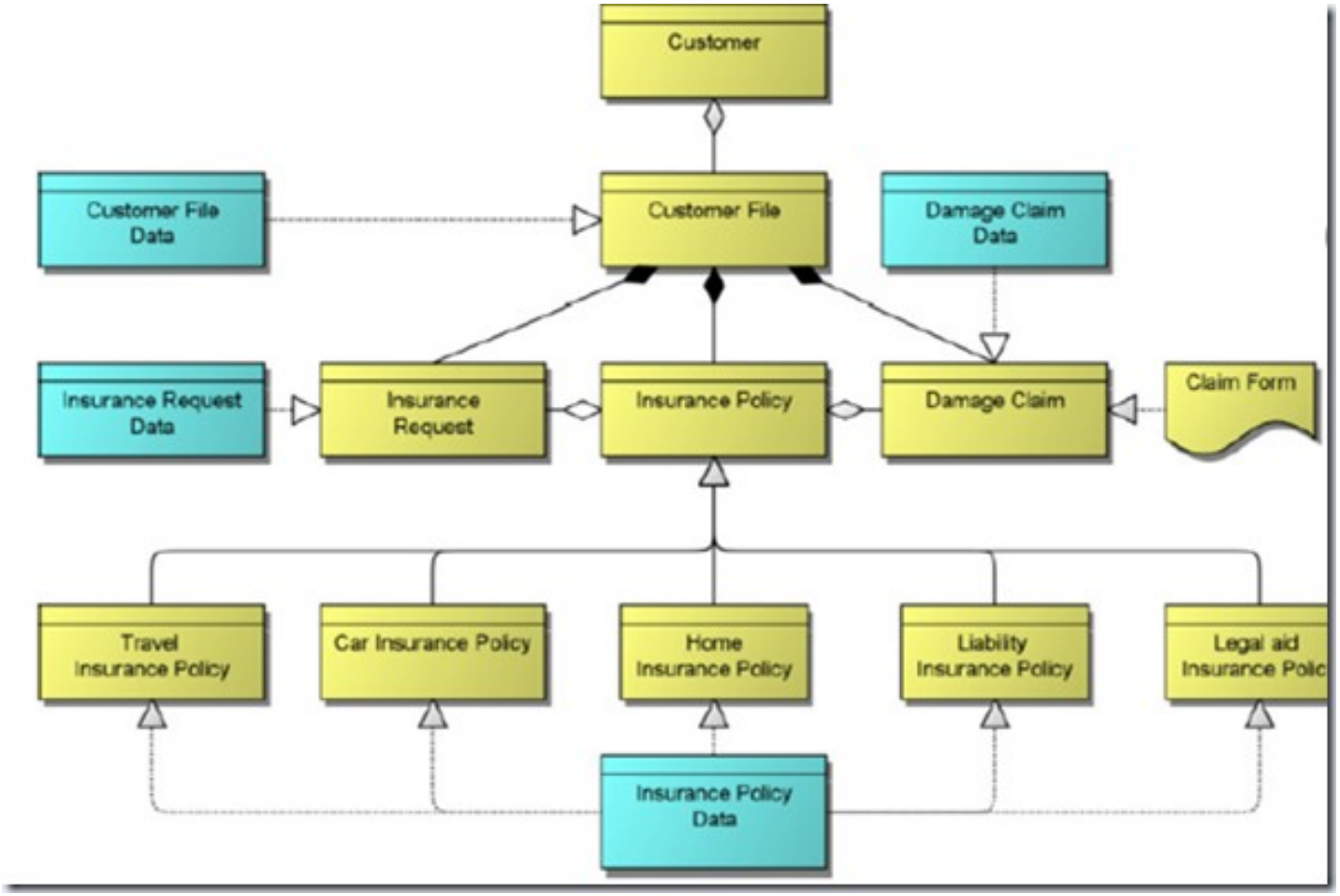
Examples



Example: Reference map of Data Entities handled by an Architecture (Archimate)

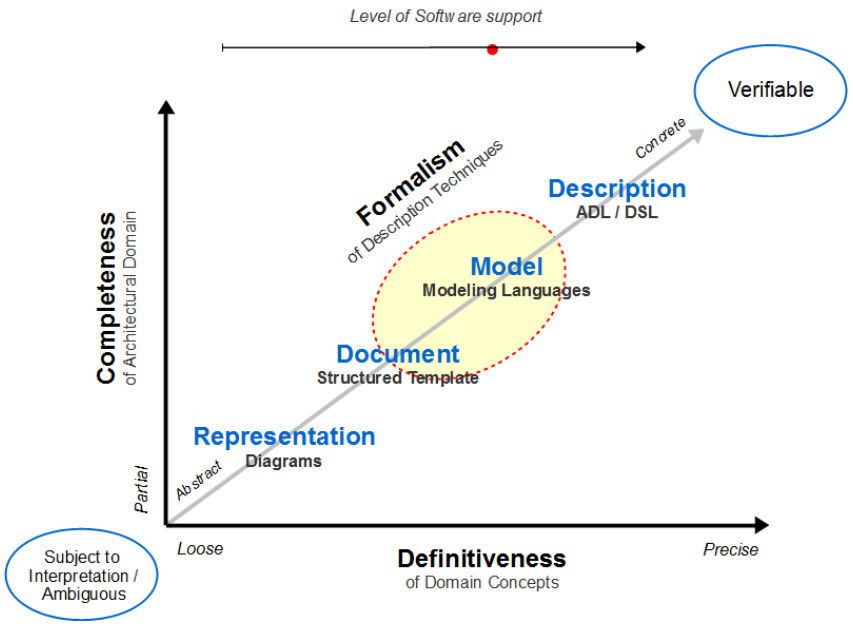


Example: A data entities composition model (Archimate)

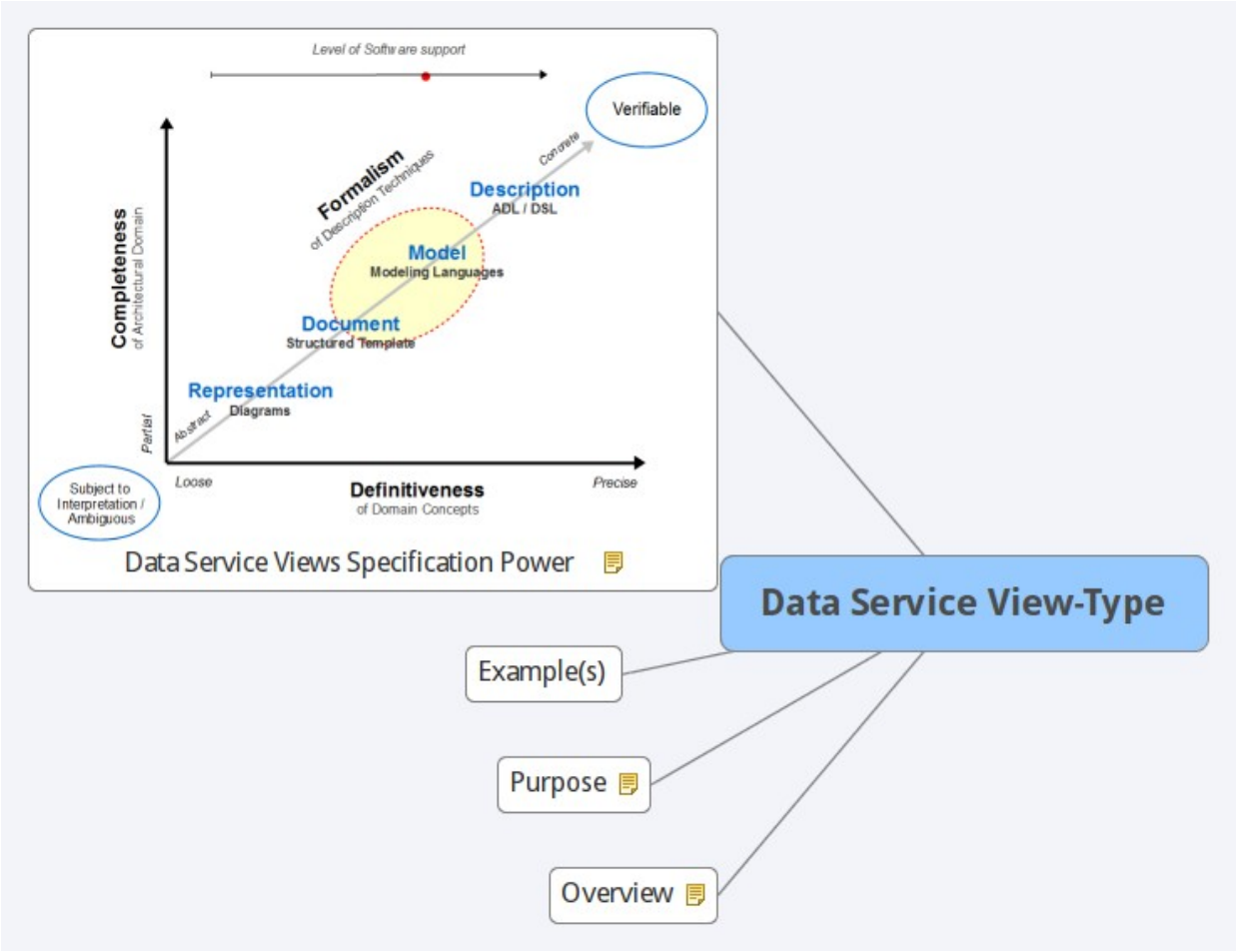


Data Structure Views Specification Power

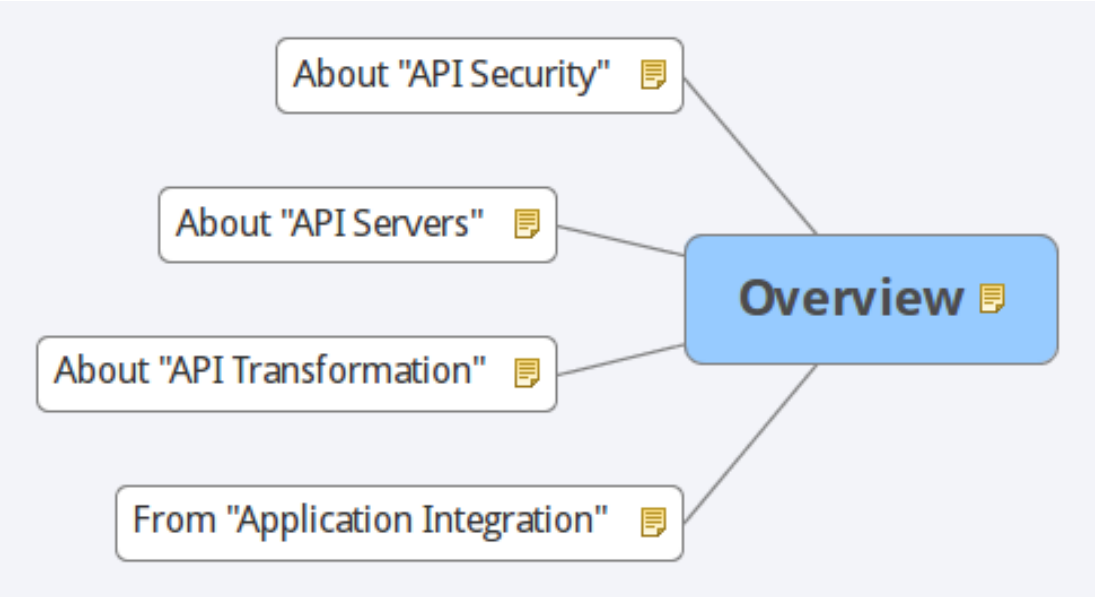
Typically a:
- Conceptual or Logical Relational model (ERD)
- Conceptual Data Entity model (UML)
- Data entities structure (Archimate)



Data Service View-Type



Overview



Overview

The "Data Service View-Type", often referred as data-in-motion, describes how the data services of an application manipulate, manage, and ultimately distribute information.

It exclusively focuses on data exchanges (ex. via data messages, or via data feeds) by specifying the data interfaces by which information transits.

It specifies when, where and how data is: (1.) collected (2.) transformed, and (3.) delivered.

It answers important questions around information format, timeliness/latency, and transactional integrity transiting between application components.

About "Application Integration"

“Application Integration” moves away from binary point-to-point connections running vendor-dependent processing logic on single Server-nodes (ETL, Stored Procedures/SQL).

Distributed Application Programming Interfaces (API) are the modern pipes and pumps of an Information Architecture.

Message-Oriented middleware has evolved into the new “Data Integration” Hub, standardized into:

(a.) Enterprise Service Bus (ESB): orchestrating messaging internally, and

(b.) API Gateways: orchestrating messaging externally.

From the standpoint of external data consumers, APIs are the application.

About "API Transformation"

Protocol Transformation: Protocol and message format transformation to enable API interoperability (HTTP, FTP, SMTP, POP, TCP, SOAP, REST, XML, Other).

Flow Orchestration: Orchestration of transformation, integration, control, and security tasks to be applied to an API, Template Flows

Message Brokering: Aggregation (mash-up) of API into one Format.

Service Registry Management: Version control of API Specification, Format revisions, Impact Analysis.

About "API Servers"

Service Level Monitoring, Alerting, Notifications, Analytics.

Adaptive Configuration (Resilience, Routing Optimization).

About "API Security"

Interface Security: (1.) Scan headers, messages, and attachments for viruses, (2.) Detect and block attacks such as denial-of-service, cross-site scripting.

Access Control: Pre-built integration for Authentication, Authorization with LDAP/AD/RDBMS, Resource access, Registry Contract access, Device Identity Management, Other.

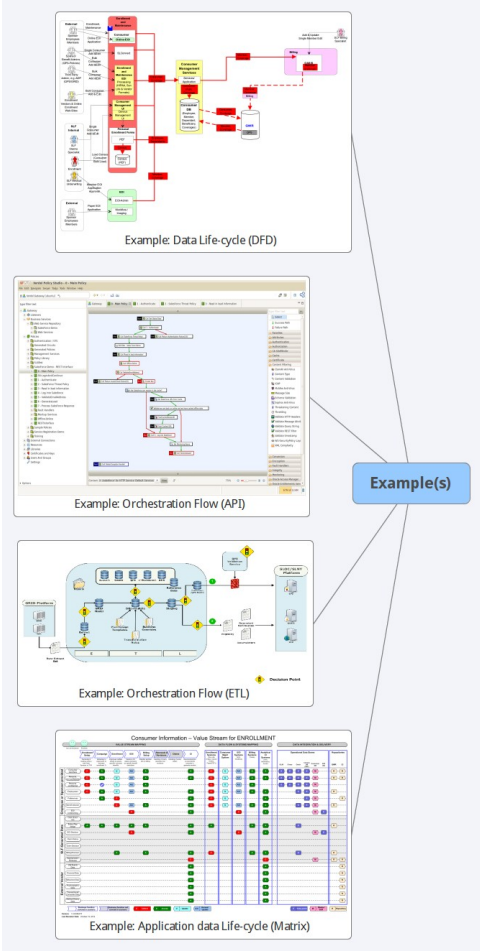
Protocol and Message Encryption.

Purpose

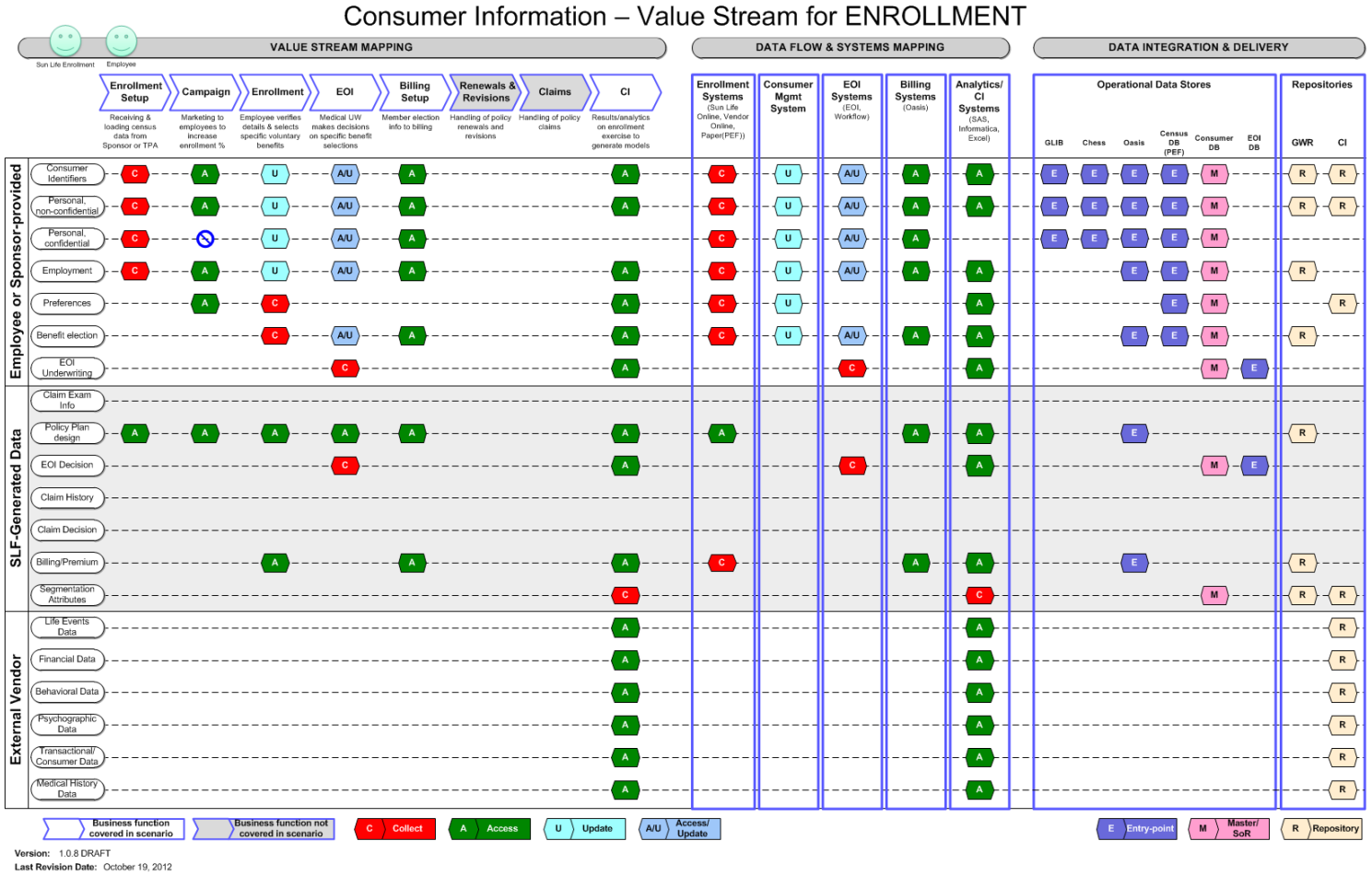
The purpose of the Data-in-Motion View-Type is to:

- define and reference any architecturally significant message schemas, data contract structures transiting between system actors.

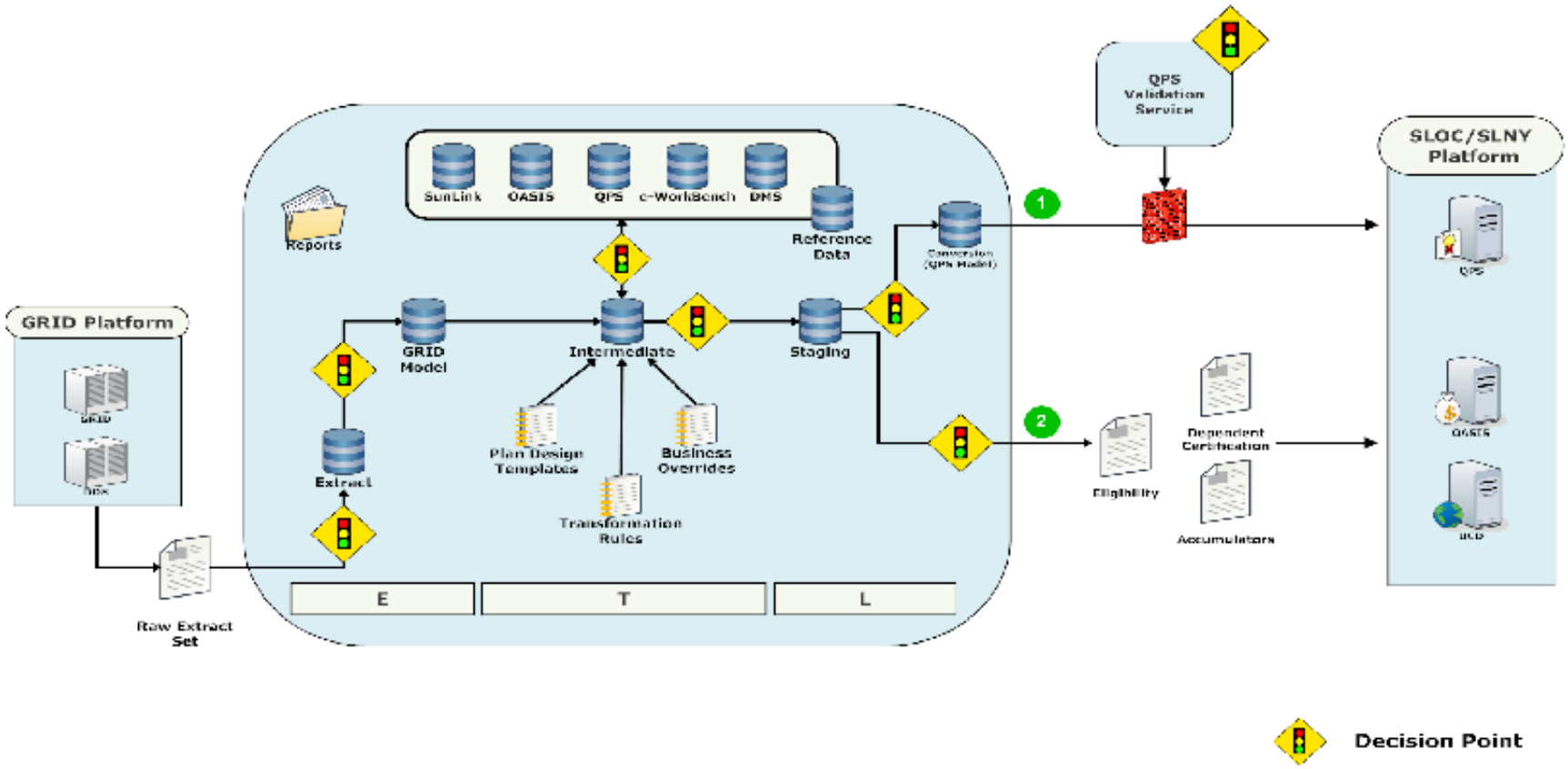
Example(s)



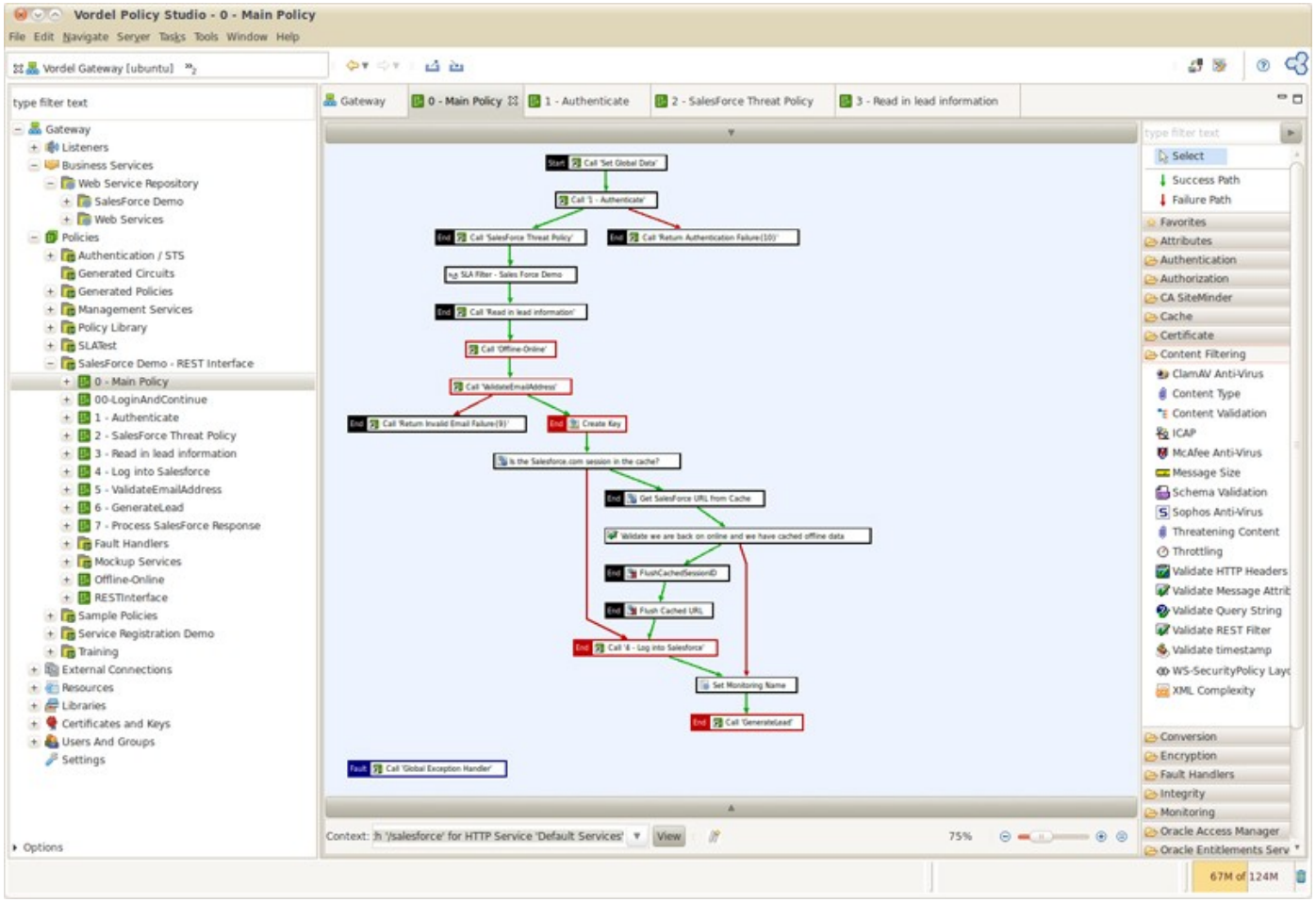
Example: Application data Life-cycle (Matrix)



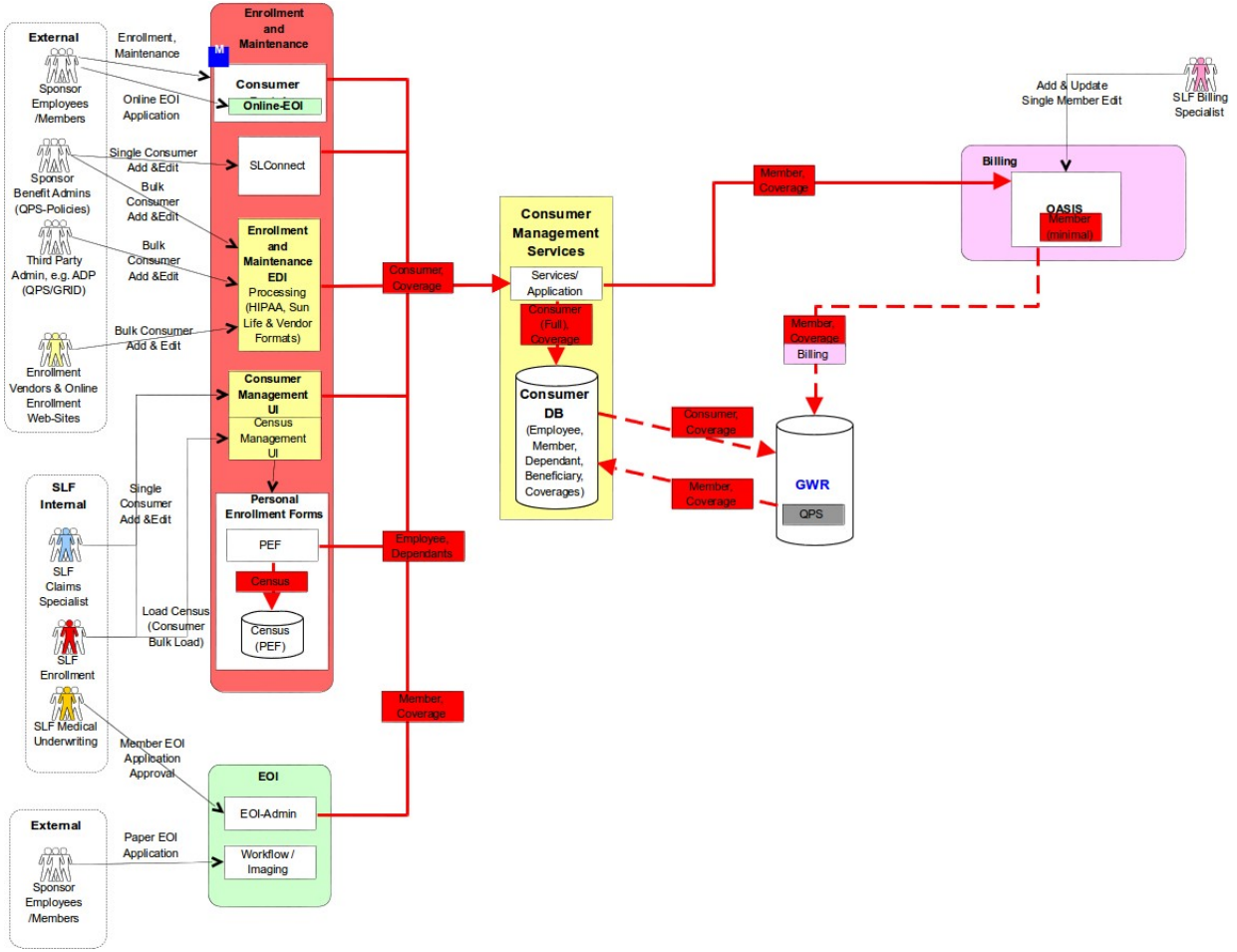
Example: Orchestration Flow (ETL)



Example: Orchestration Flow (API)



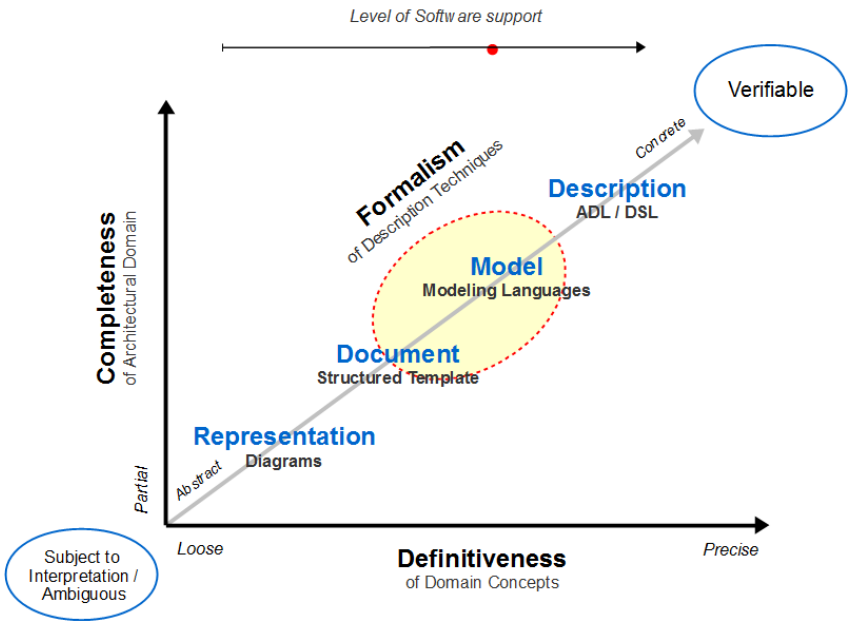
Example: Data Life-cycle (DFD)



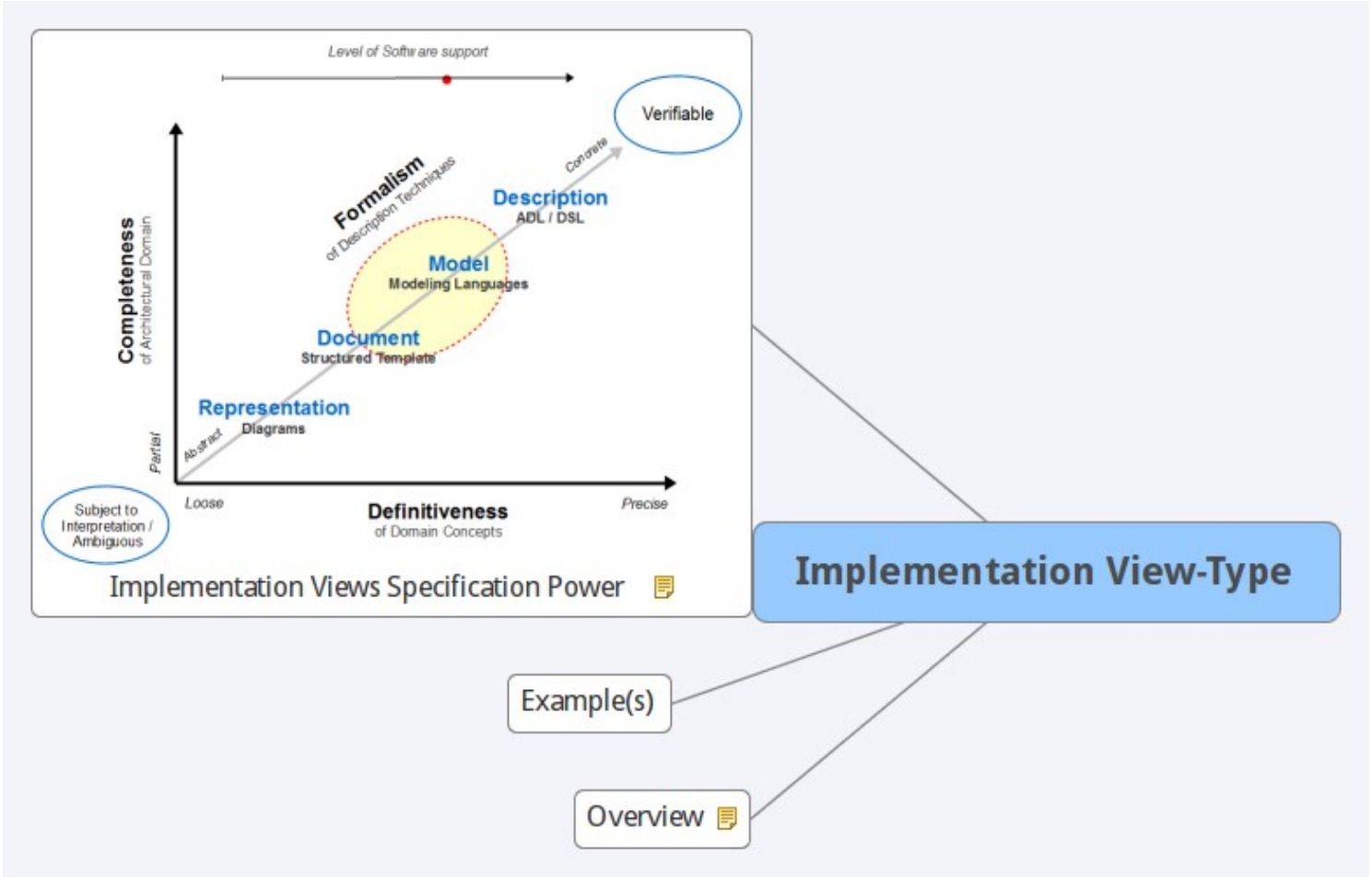
Data Service Views Specification Power

Typically a:

- Data Flow diagram (DFD)
- Data Feeds diagram (ETL)
- State Transition diagram (UML)
- API Orchestration Flow (UML, Archimate)



Implementation View-Type

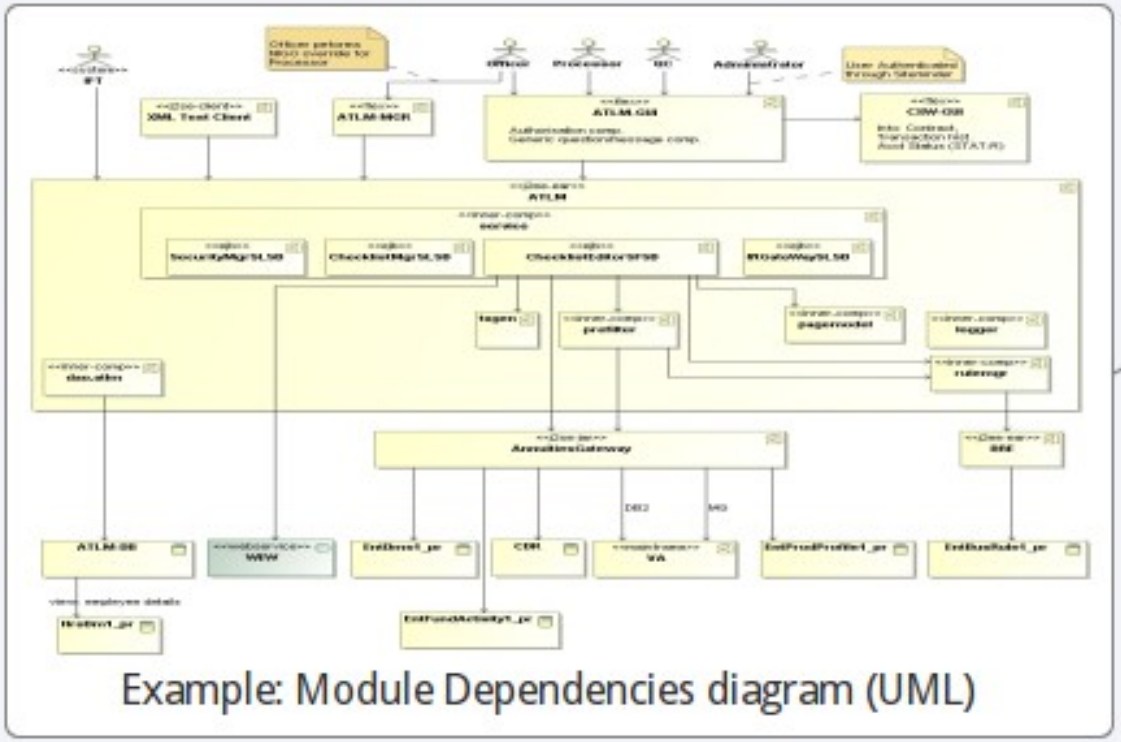


Overview

The "Implementation View-Type" describes all required component dependencies for successful build of the architecture.

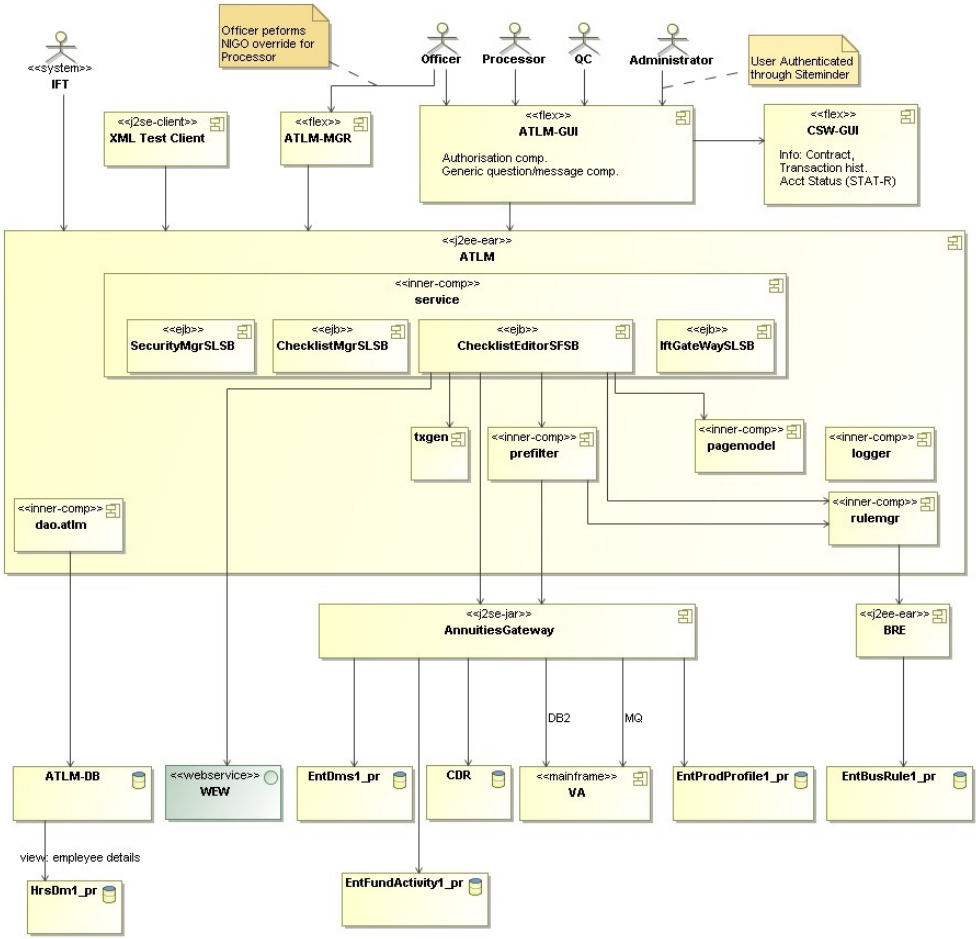
It specifies how to the (1.) build, (2.) integration test, (3.) configure/tune, the solution architecture.

Example(s)



Example(s)

Example: Module Dependencies diagram (UML)

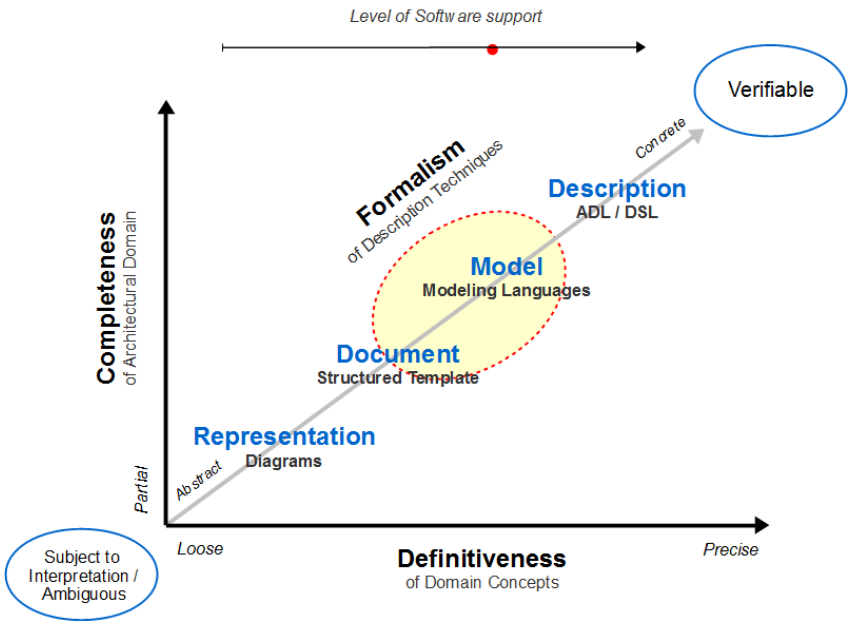


Implementation Views Specification Power

Typically a:

Package dependencies diagram (UML)

Module dependencies diagram (UML)



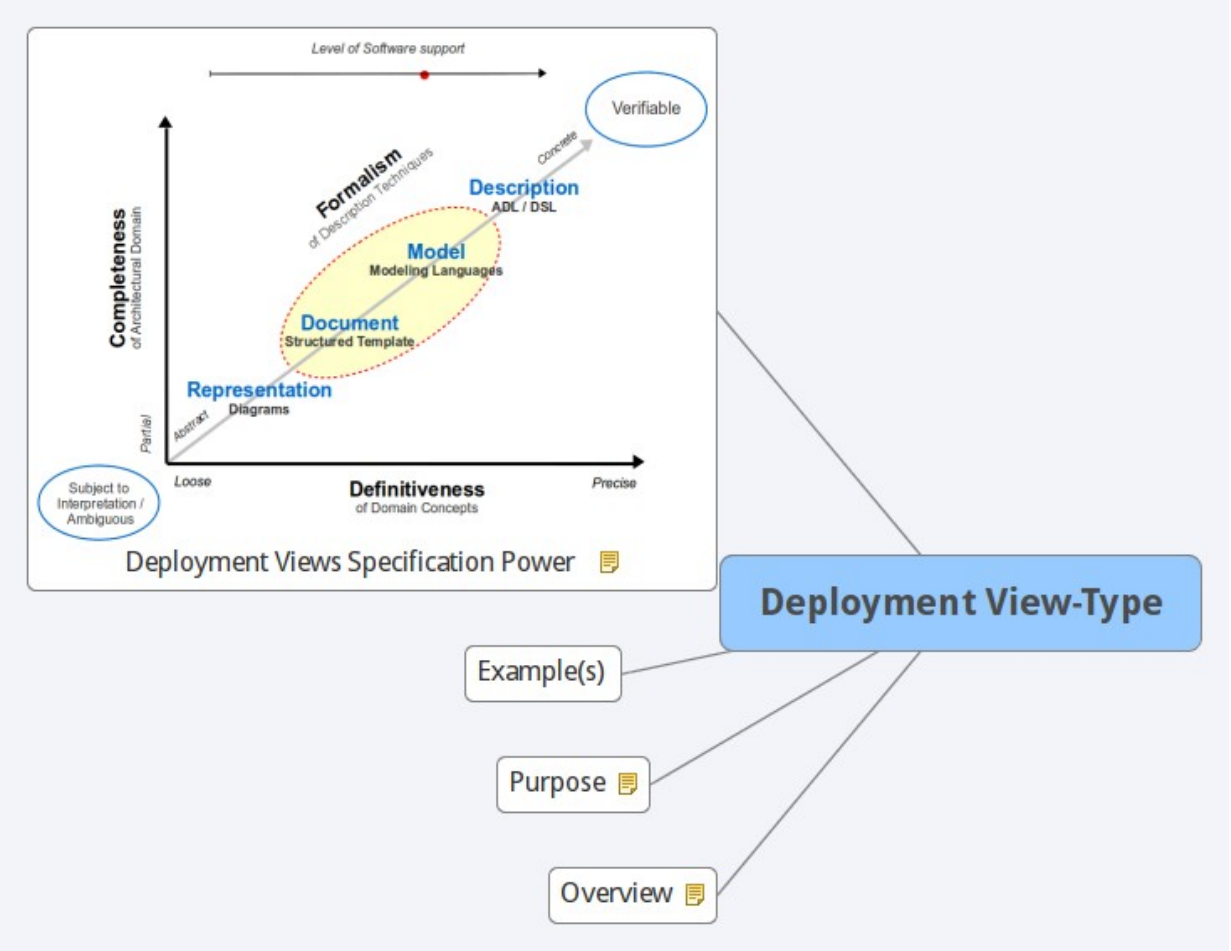
Implementation Views Specification Power

Typically a:

Package dependencies diagram (UML)

Module dependencies diagram (UML)

Deployment View-Type



Overview

The "Deployment View-Type" describes the infrastructure of the environment into which a Architectural Solution is be deployed.

This view-type captures the hardware environment that your system needs (primarily the processing nodes, network interconnections, and disk storage facilities required), the technical environment requirements for each element, and the mapping of the software elements to the runtime environment that will execute them.

Purpose

The purpose of the Deployment View-Type is to:

- decide how applications (that have lots to do together) should be grouped (ex. into a same virtual instance, or a same logical, or physical server instance)

- describe the environment into which the system will be deployed, including capturing the dependencies the system has on its runtime environment.

And to capture:

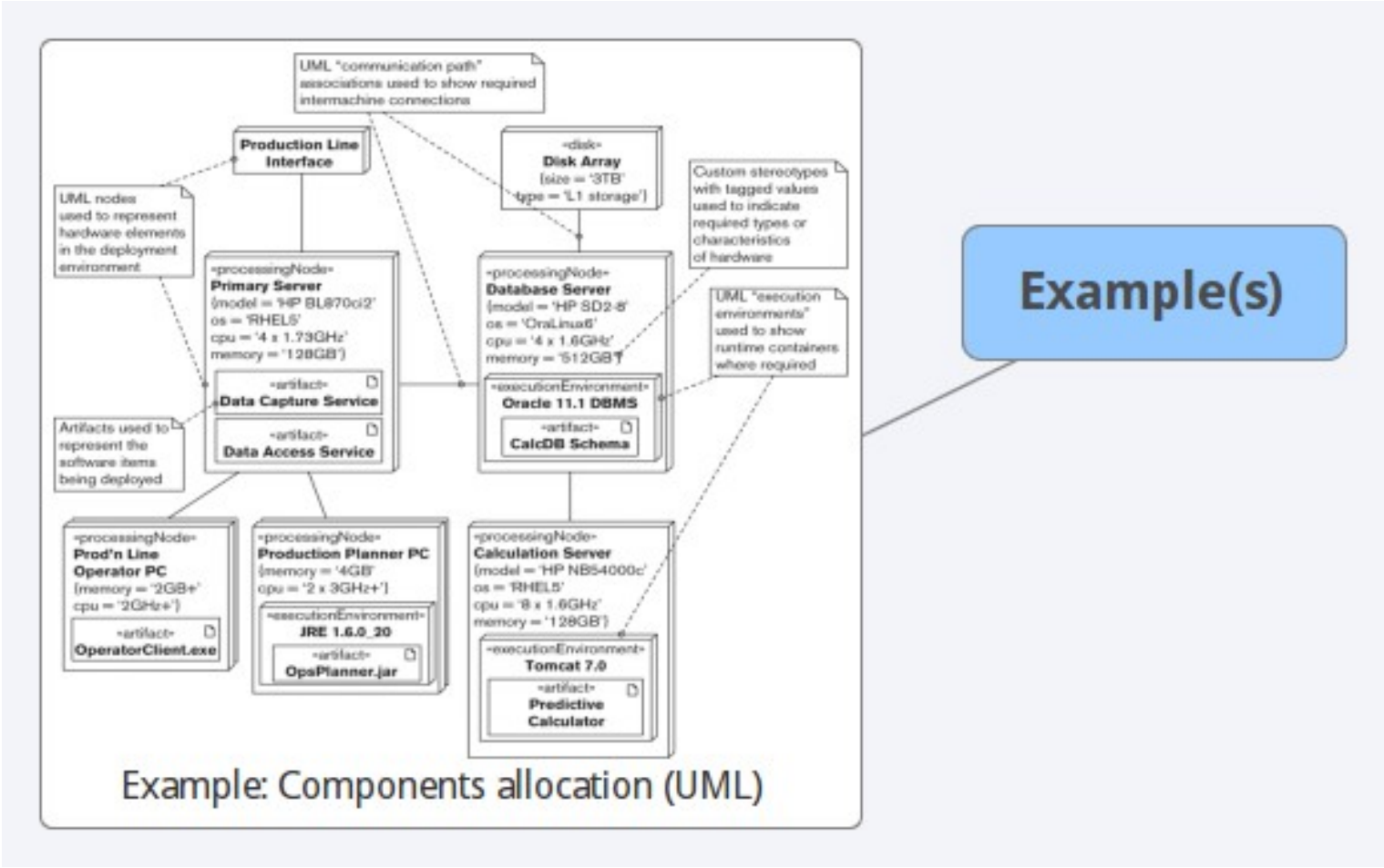
- the dependencies with the Solution runtime environment,

- the hardware infrastructure required,

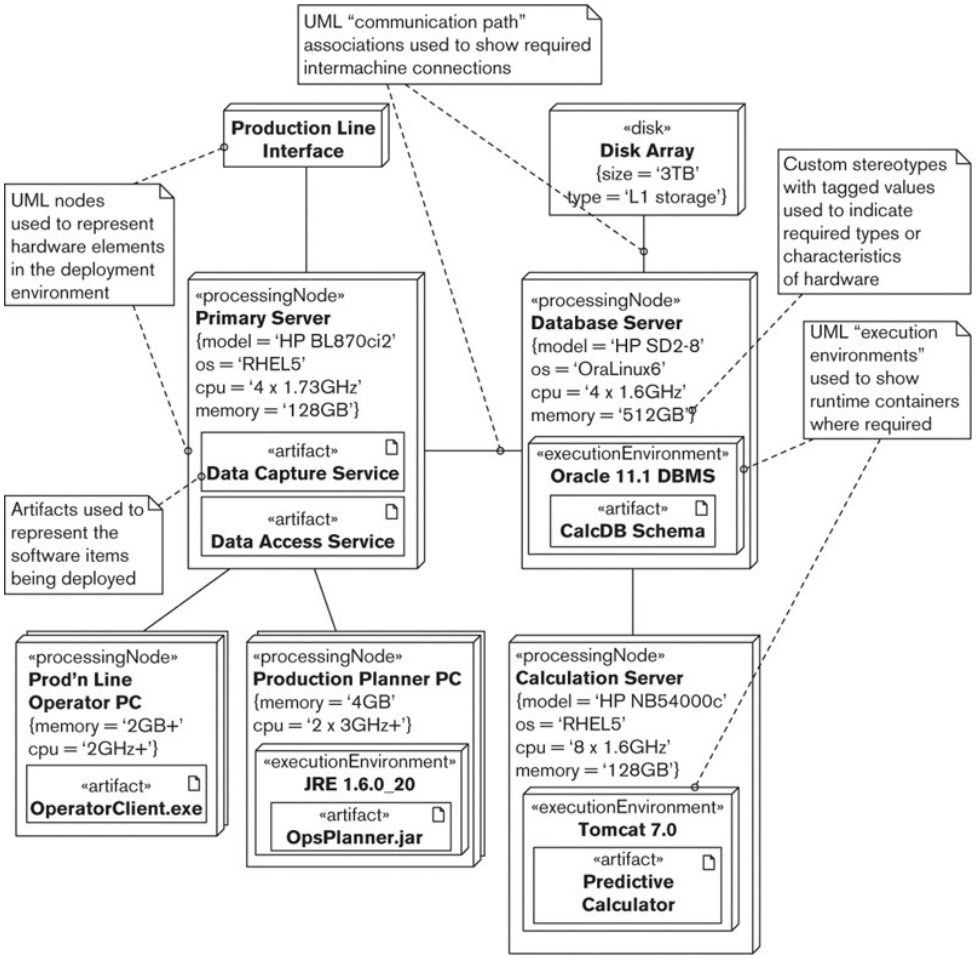
- the technical requirements for each, and

- the mapping of the software elements to the runtime environment that will execute them (release and configuration management).

Example(s)



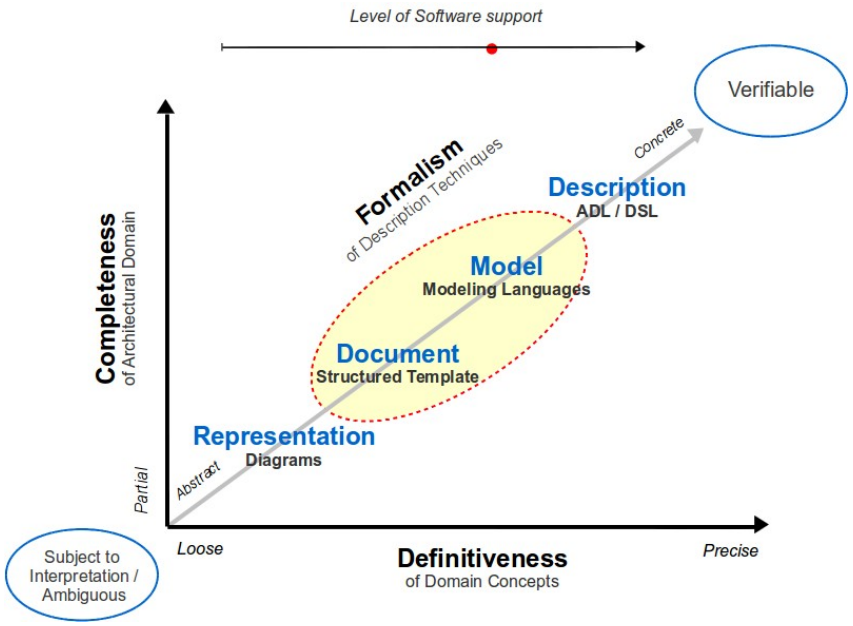
Example: Components allocation (UML)



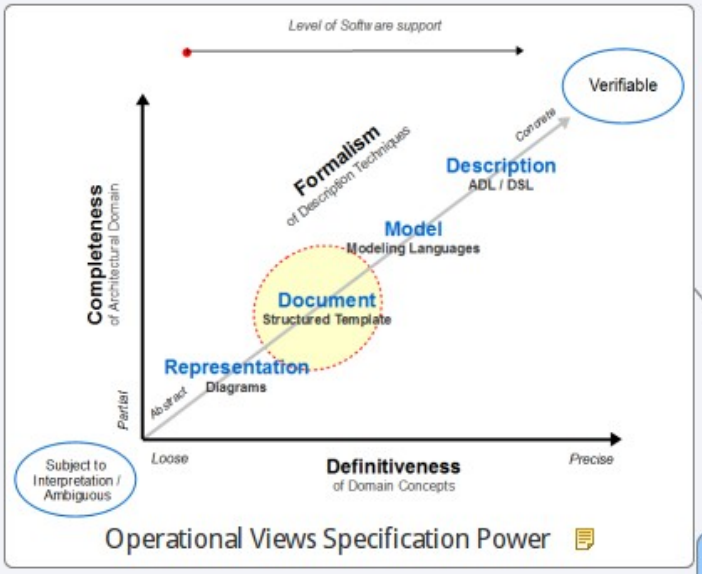
Deployment Views Specification Power

Typically a:

- Deployment diagram (UML, Archimate)



Operational View-Type



Operational View-Type

- Examples
- Purpose
- Overview

Overview

The "Operational View-Type" describes, for each IT functional area, the strategy supporting the operational concerns of the Architecture.

Application architectures often assume the solution can be allocated onto one server, one Runtime... until NON-functional requirements start posing questions about distribution of system resources.

For all but the simplest systems, installing, managing, and operating the system is a significant task that must be considered and planned at design time.

Purpose

The purpose of the "Operational View-Type" is to:

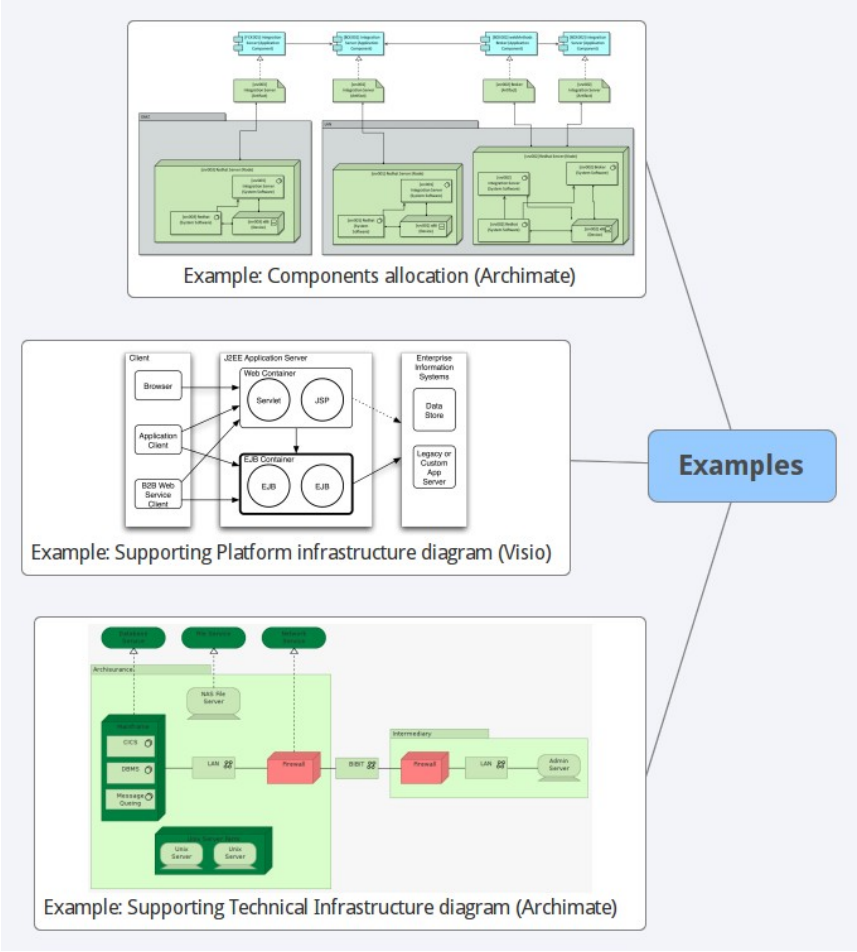
- define the characteristics by which Architecture will be operated, administered & supported (e.g. once deployed in its production environment).

- help to understand how to change the underlying infrastructure of the application architecture without breaking it.

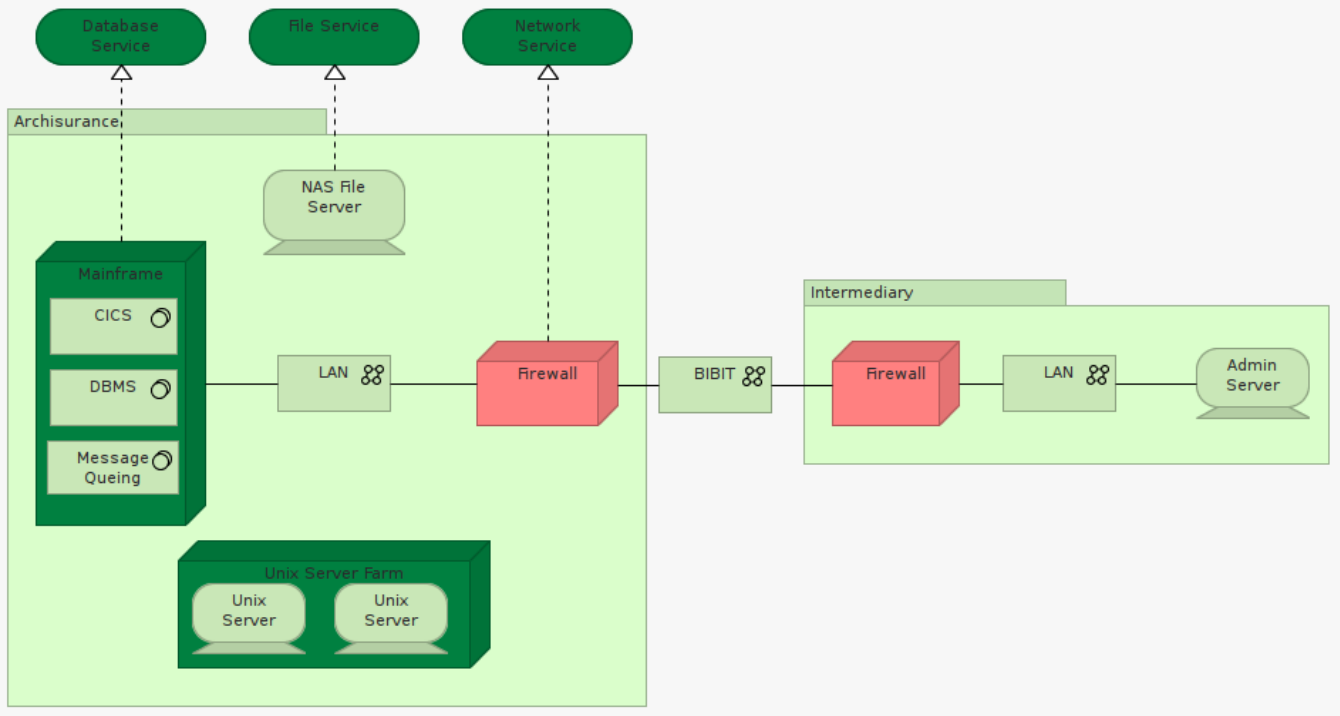
- specify the means of configuration and tuning of the solution architecture, without having to re-deploy the application.

- specify what components can be physically distributed or should be centralized.

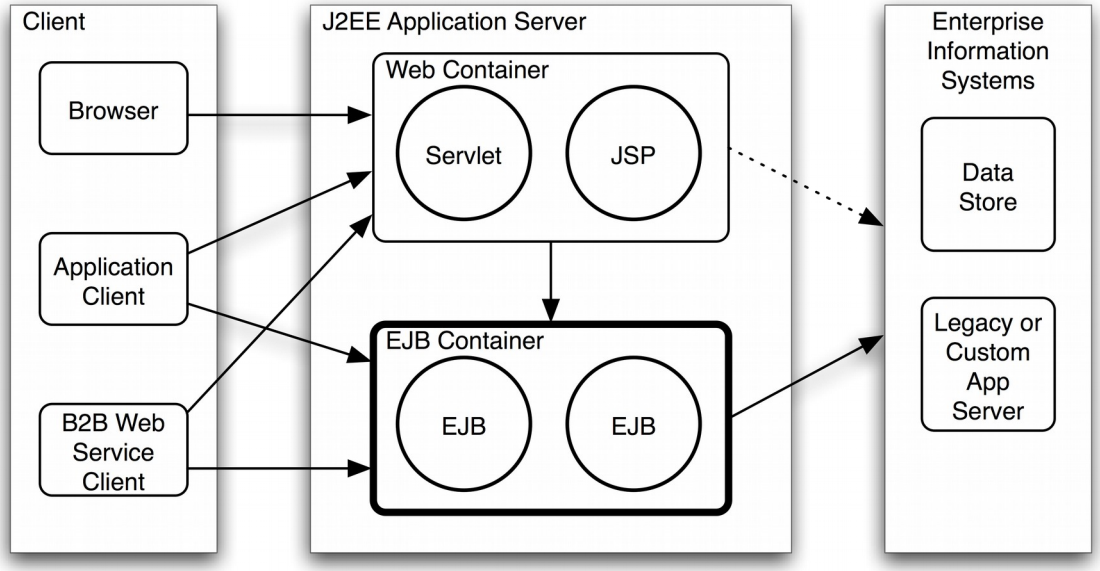
Examples



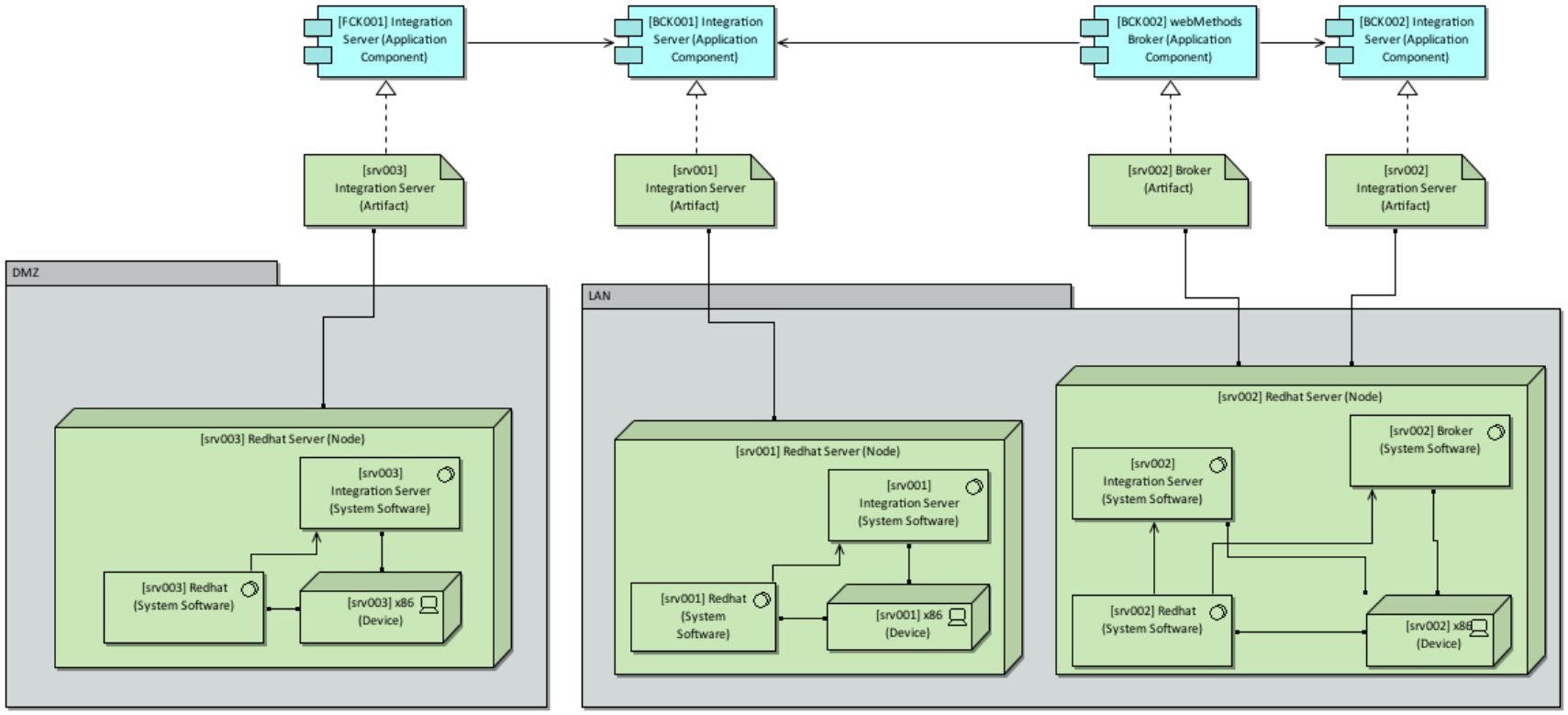
Example: Supporting Technical Infrastructure diagram (Archimate)



Example: Supporting Platform infrastructure diagram (Visio)



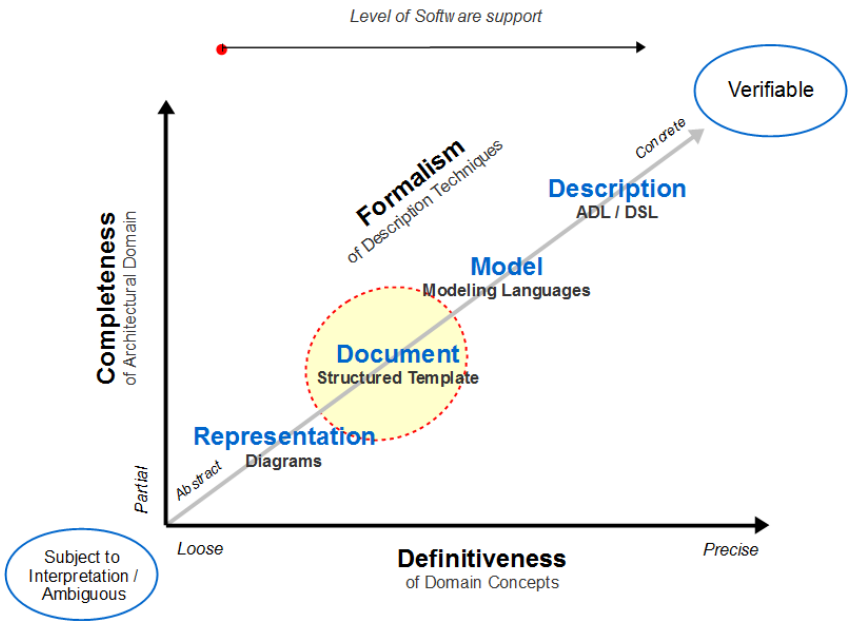
Example: Components allocation (Archimate)



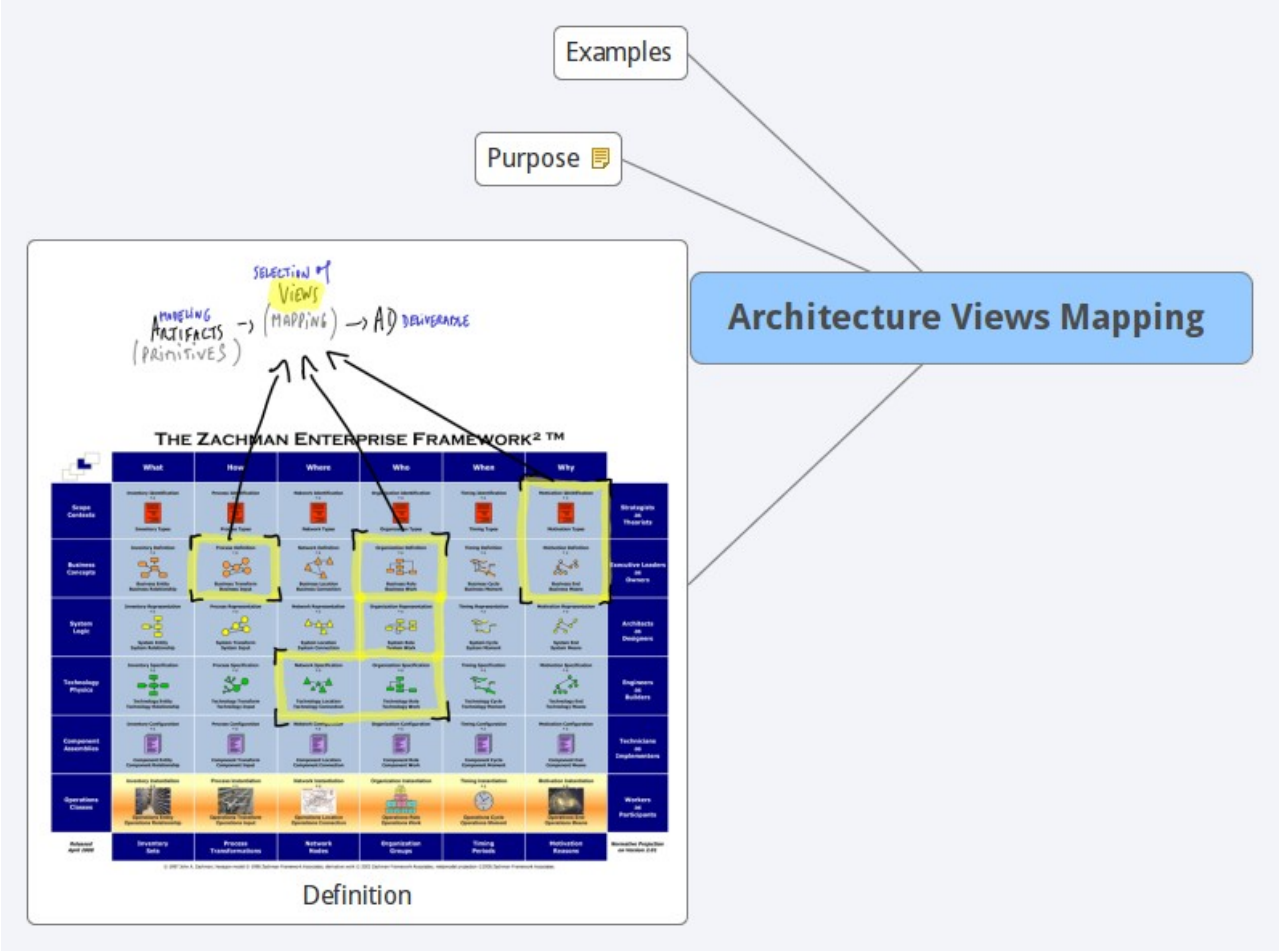
Operational Views Specification Power

Typically a:

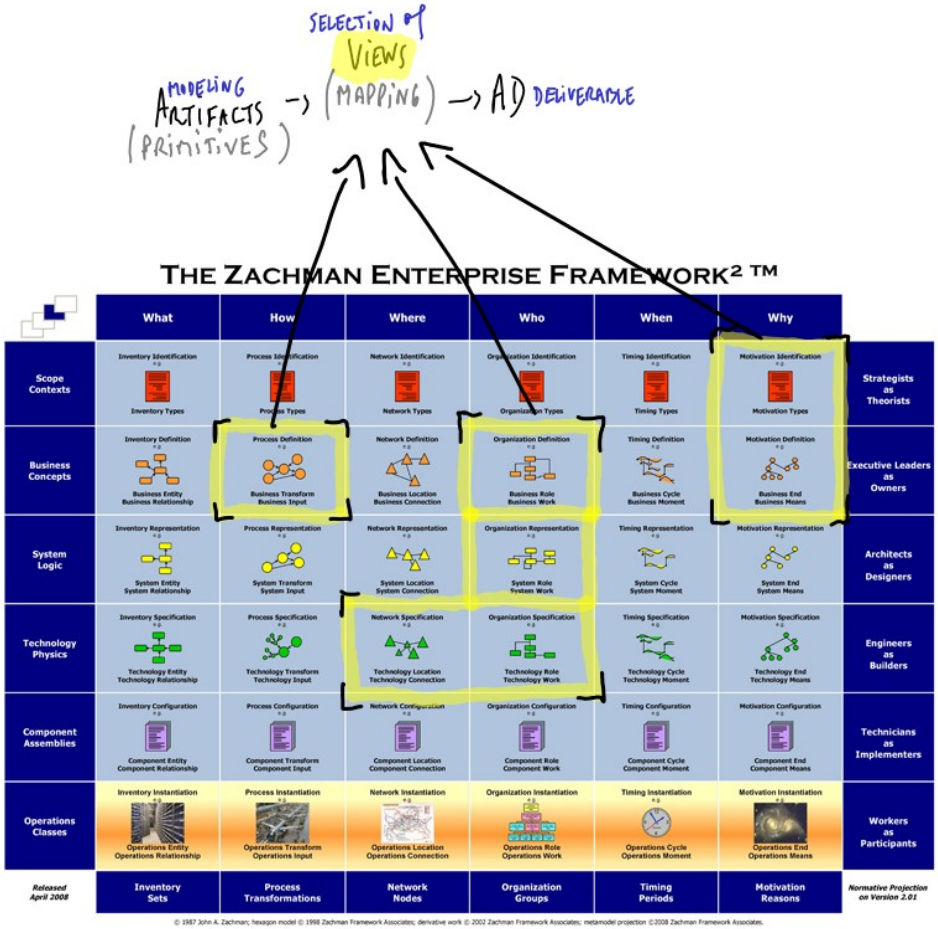
- Technical infrastructure diagram (Visio, Archimate)



Architecture Views Mapping



Definition



Purpose

Mapping Architecture Views helps to achieve TRACEABILITY of the solution architecture.

It helps architect to detects architectural GAPS in the solution design.

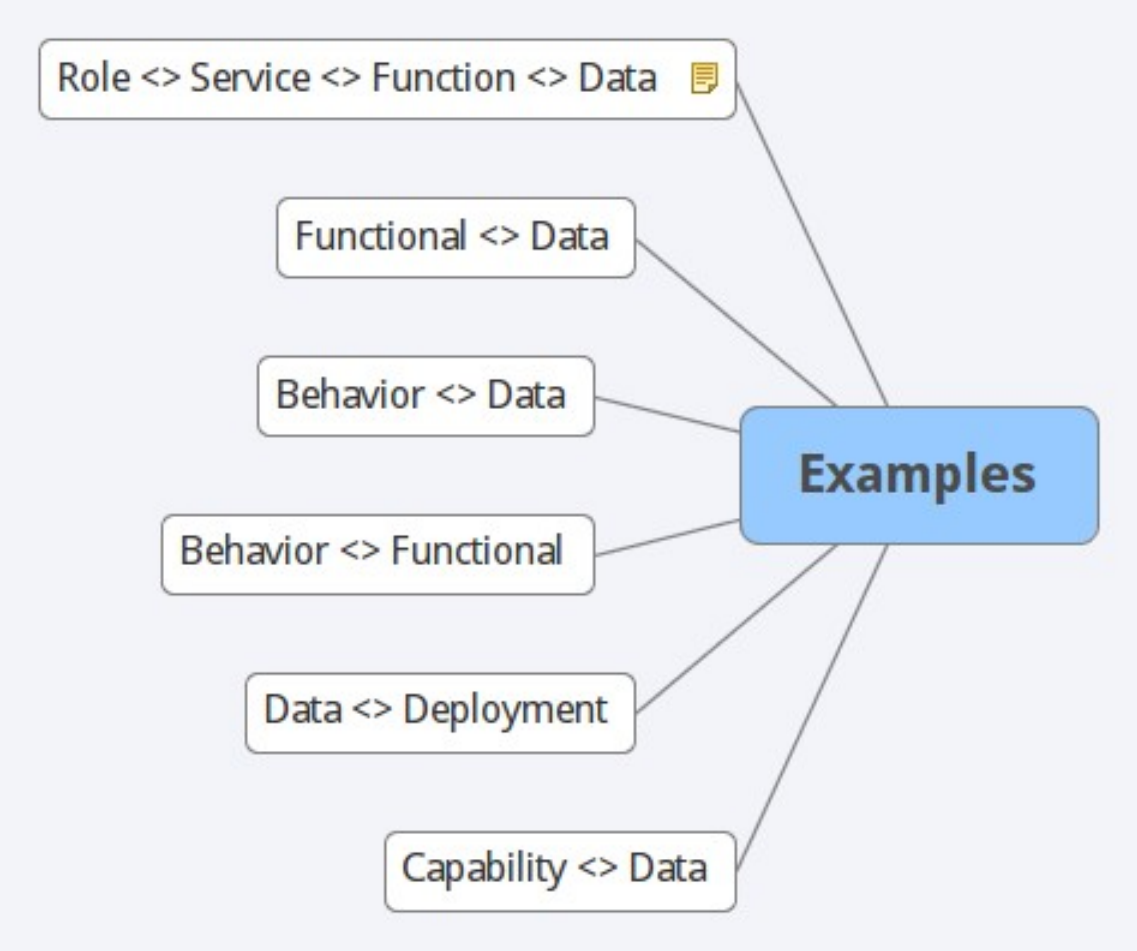
To understand what key aspects of the solution MUST be modeled: Aiming to map Views of an architecture helps architects to iteratively decide what models are strictly required to define the design.

To avoid inconsistency: What doesn't "map" (i.e. doesn't have a correspondence or cannot be related to other modeling elements), is potentially pointing a gap in the design.

To prevent fragmentation: To reduce the number of models to a minimum, keep only what modeling artifacts are strictly required to explain the architecture

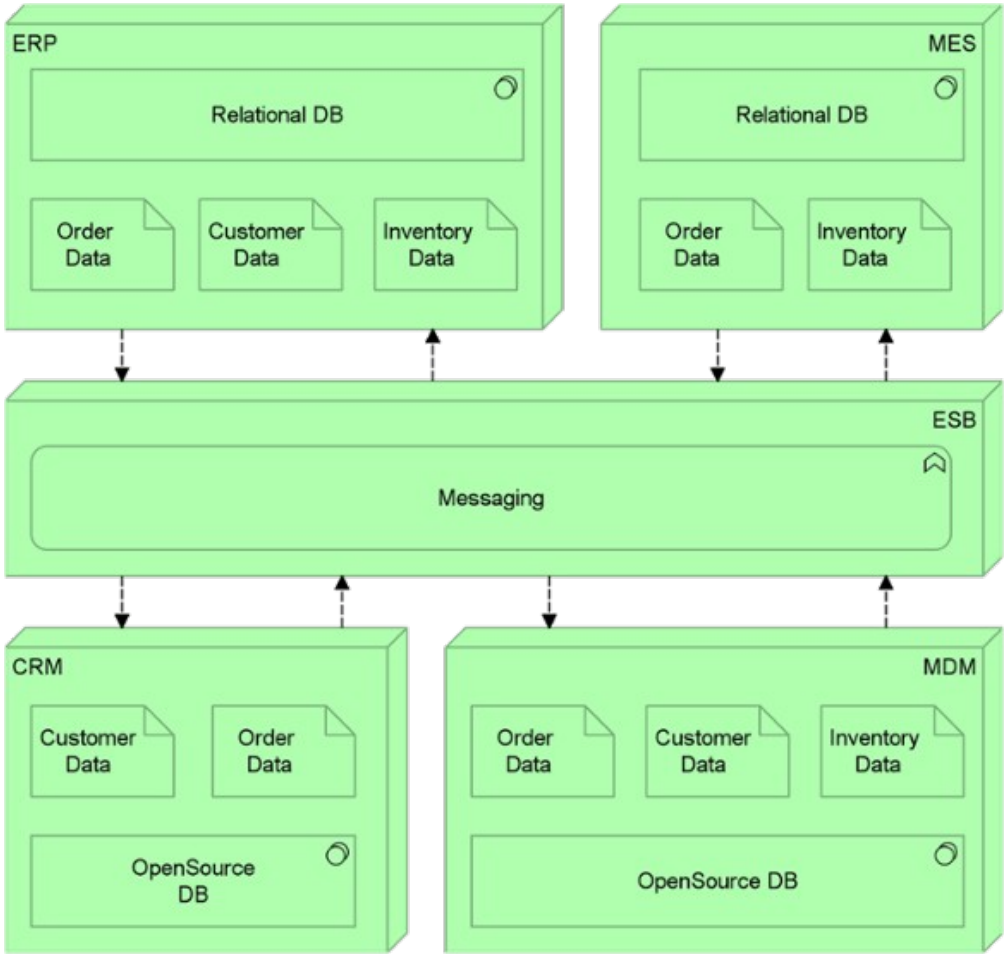
...hence increasing the conciseness and expressiveness of the architectural description.

Examples

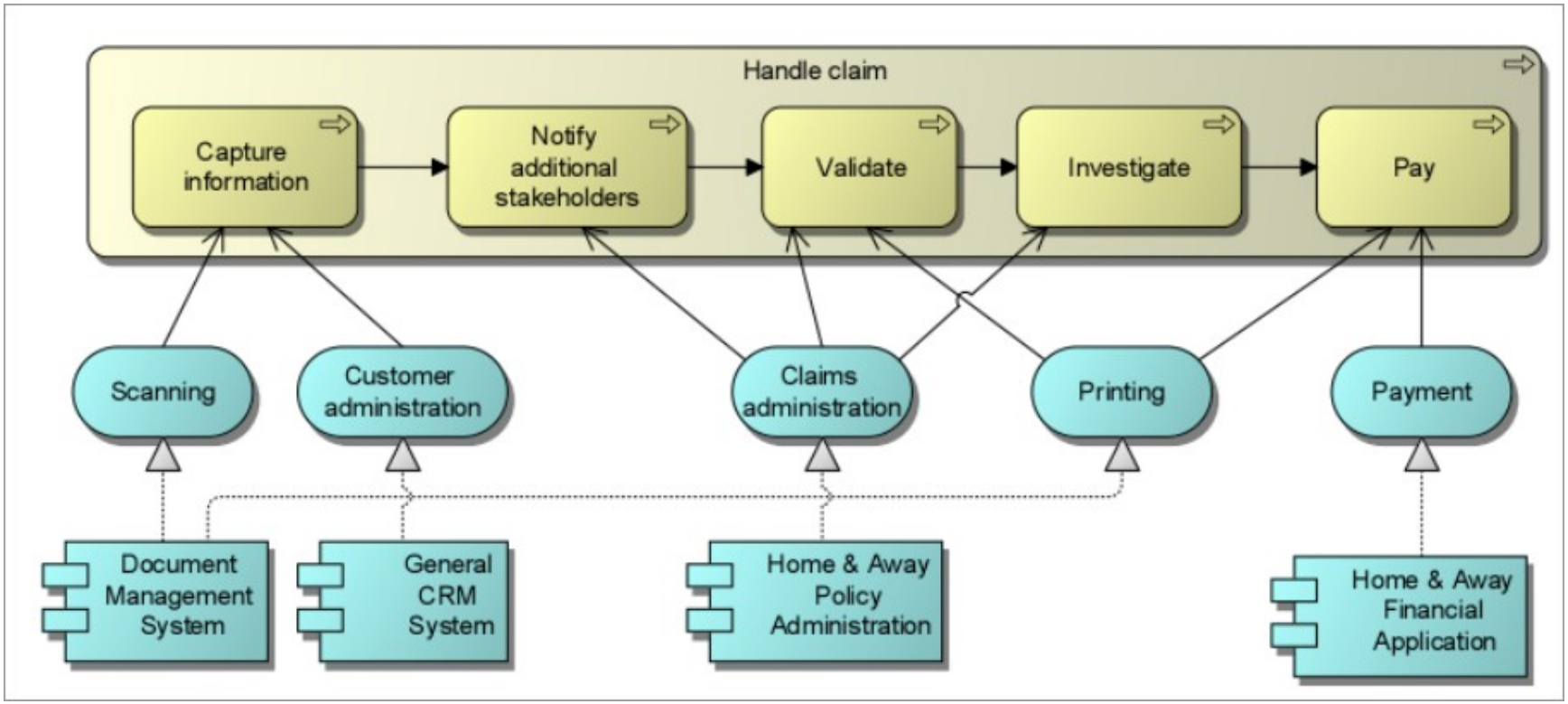


Capability <> Data

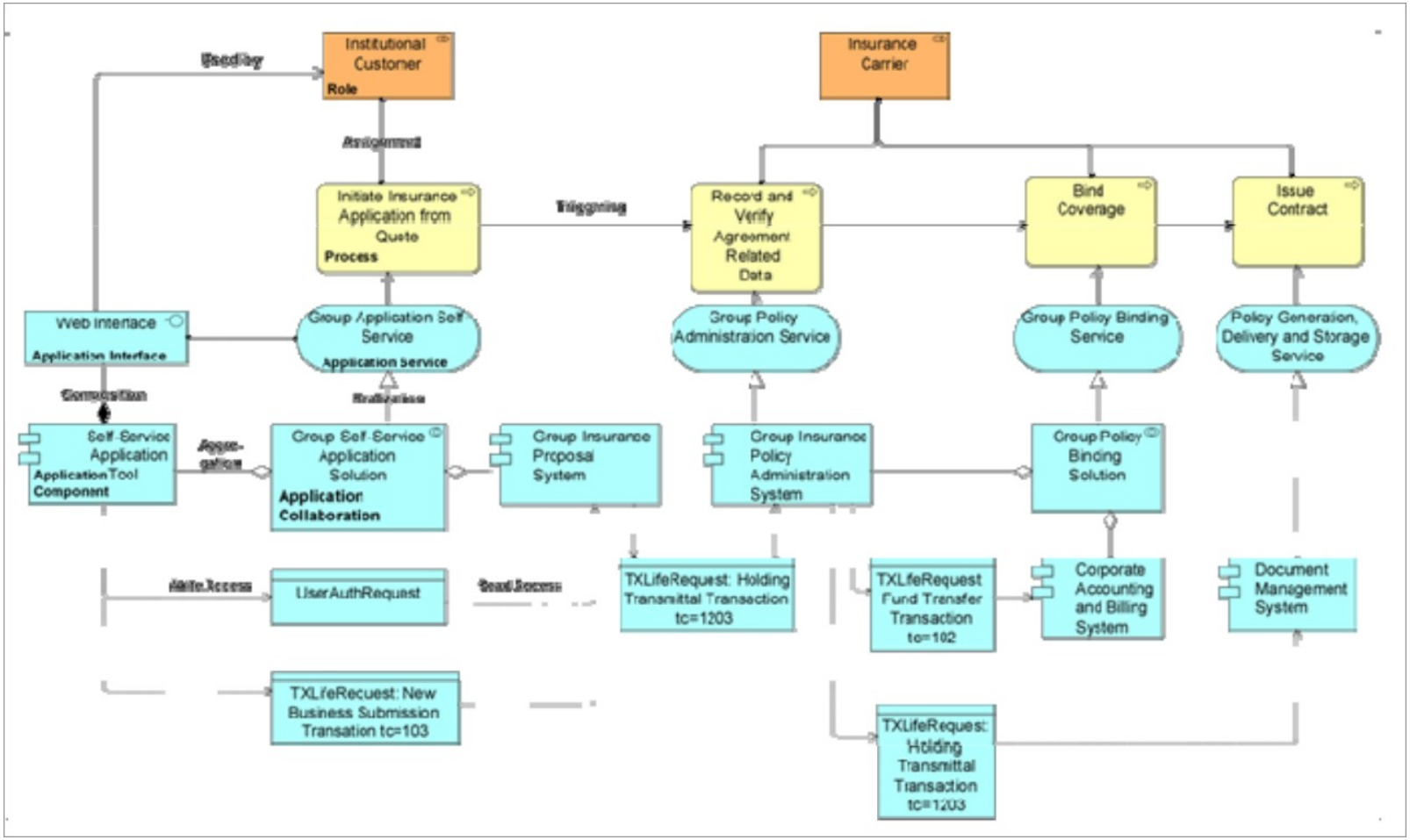
Data <> Deployment



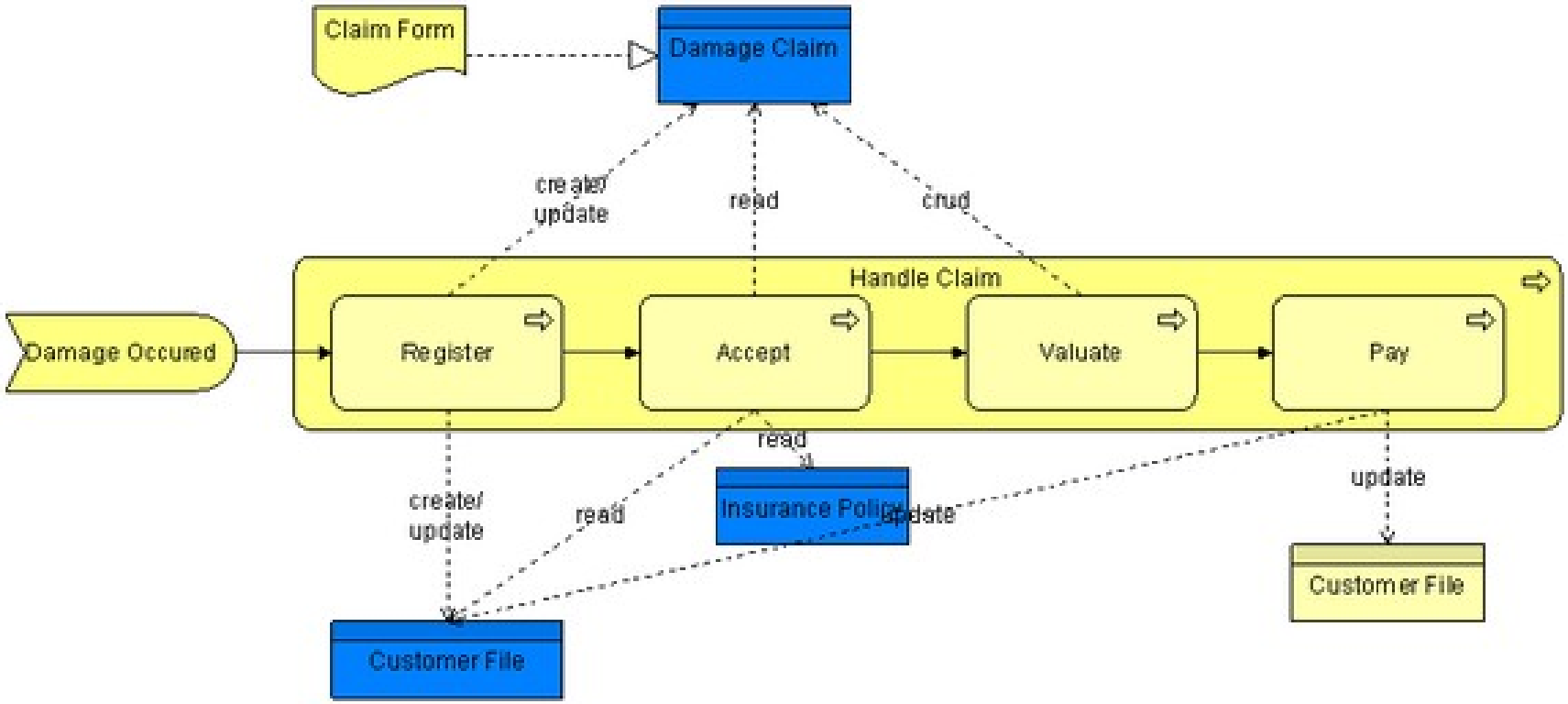
Behavior <> Functional



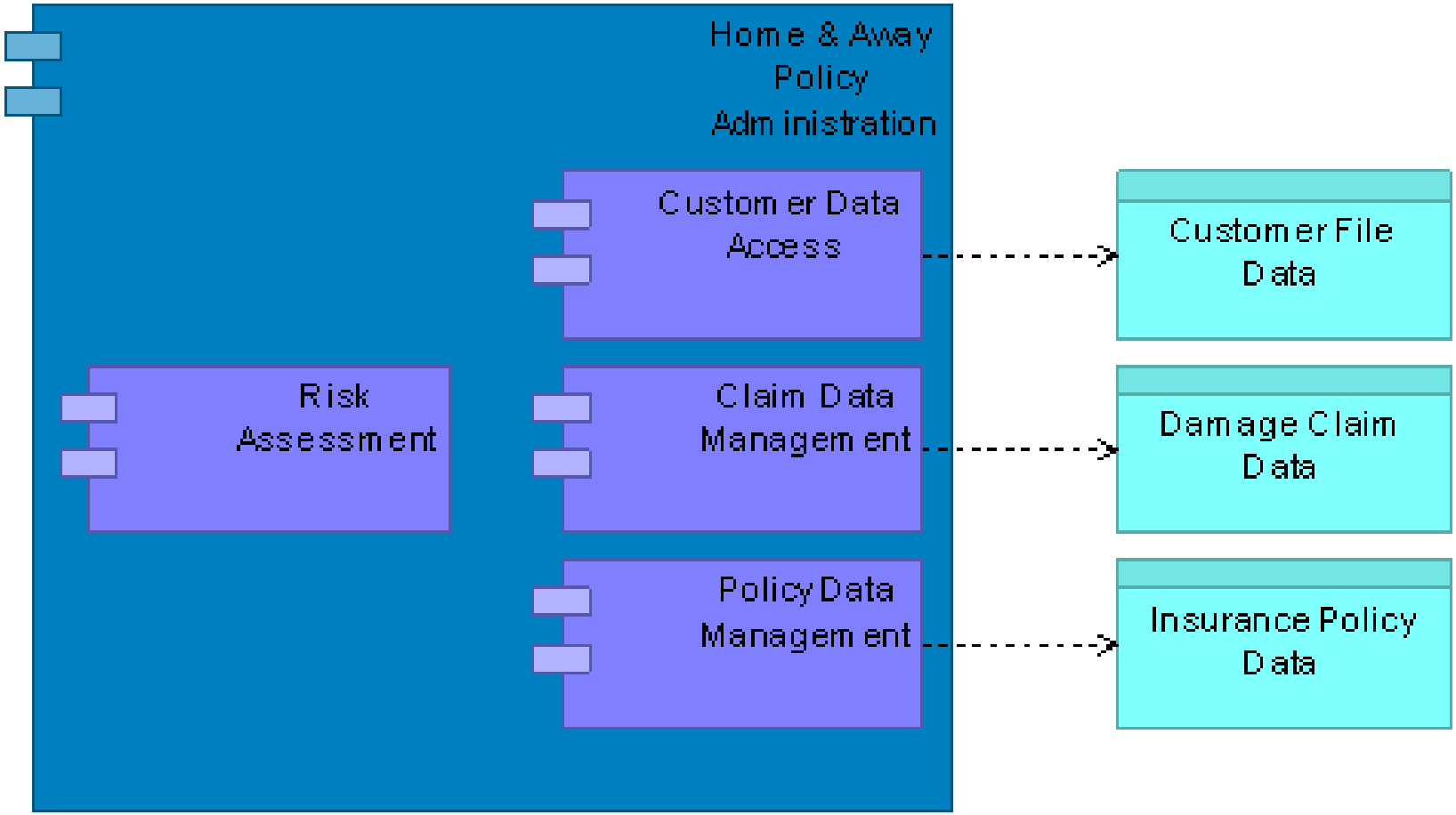
Behavior <> Functional



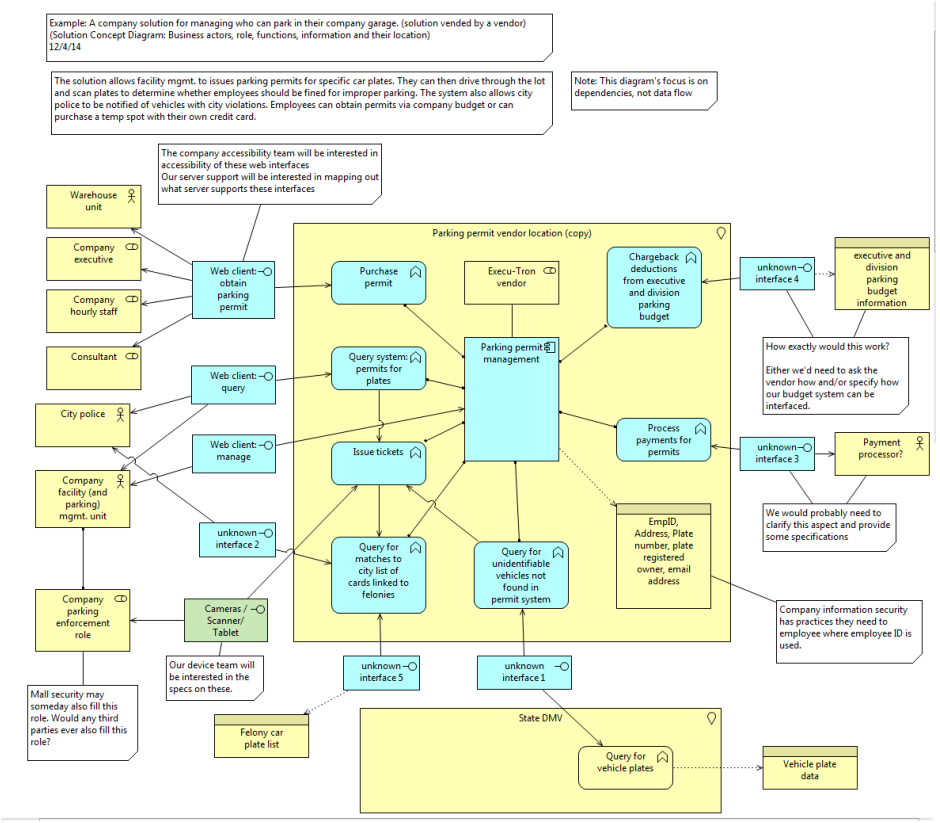
Behavior <> Data



Functional <> Data



Role <> Service <> Function <> Data

- 

Role <> Service <> Function <> Data

-