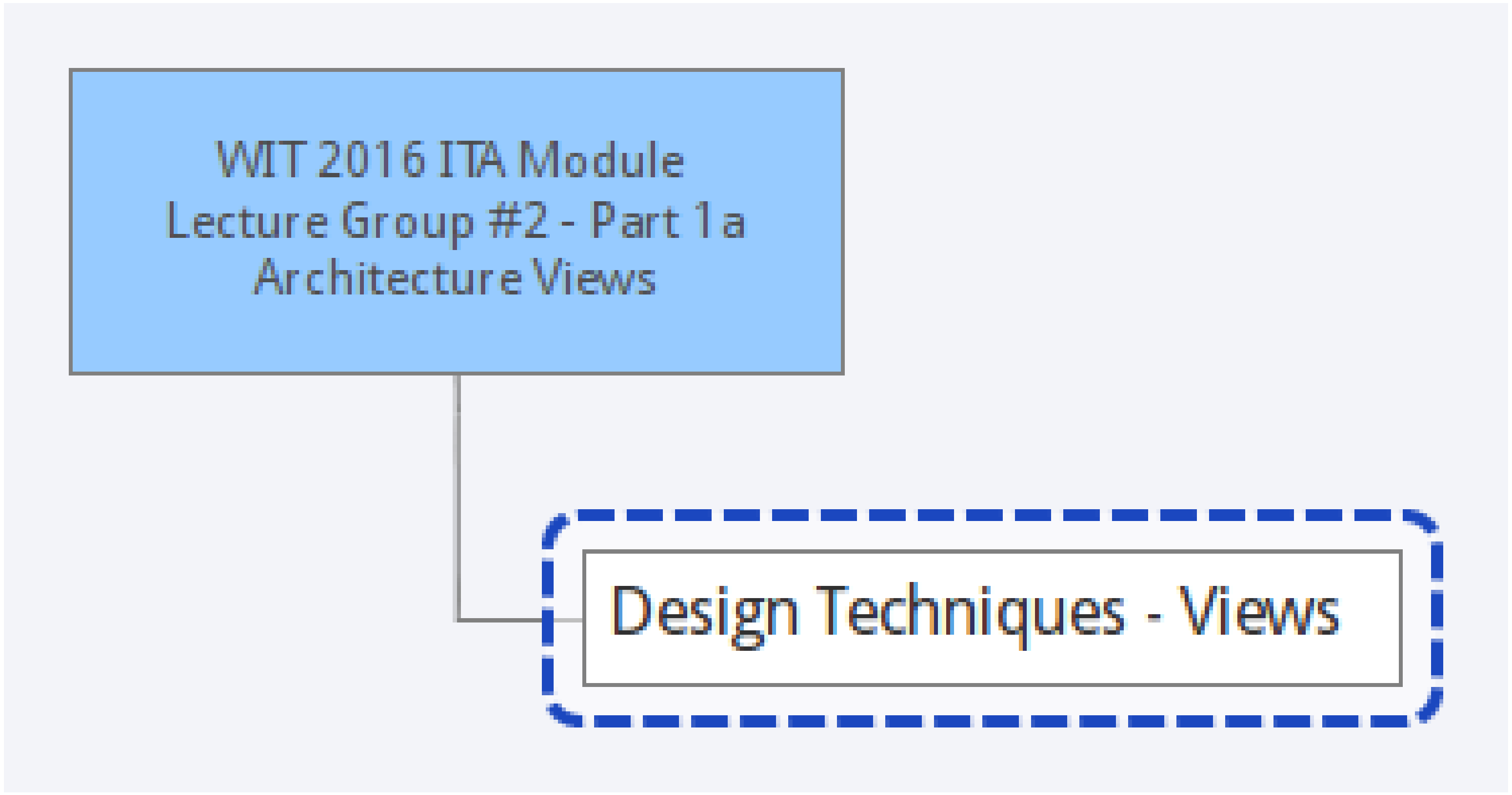
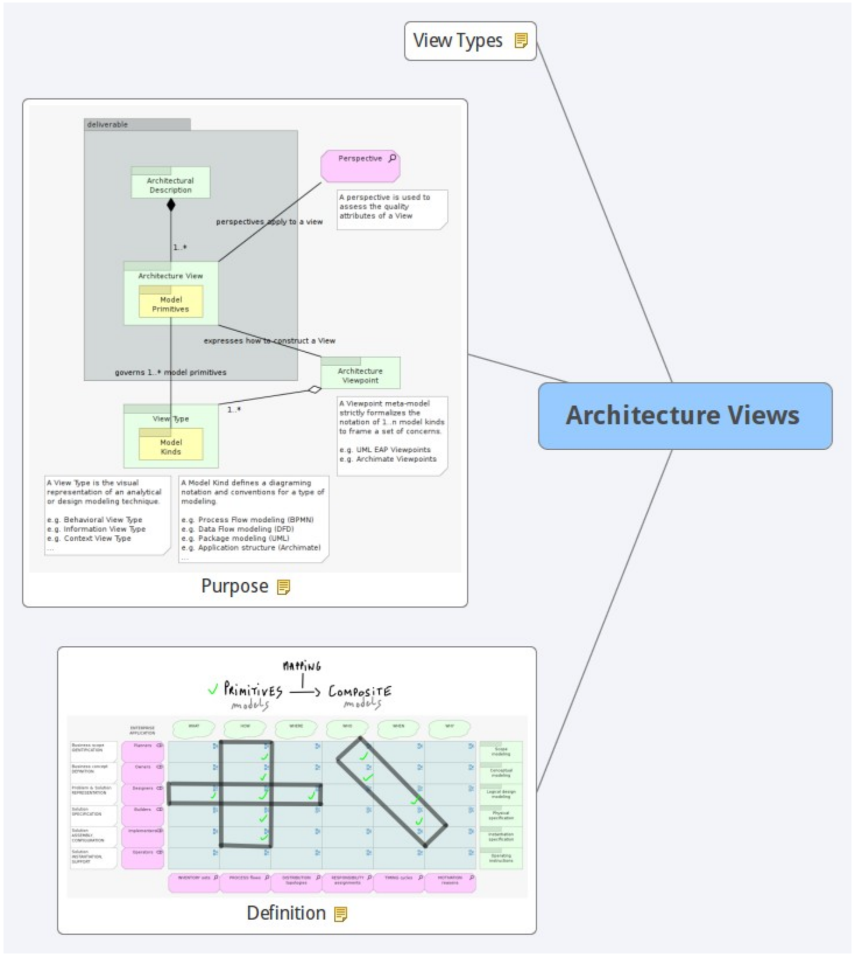
WIT 2016 ITA Module

Architecture Views
Lecture Group #2 - Part 1a

Lecture Group #2 - Part 1a
Architecture Views



Architecture Views



A perspective is used to assess the quality attributes of a View

A Viewpoint meta-model strictly formalizes the notation of 1..n model kinds to frame a set of concerns.

e.g. UML EAP Viewpoints
e.g. Archimate Viewpoints

A View Type is the visual representation of an analytical or design modeling technique.

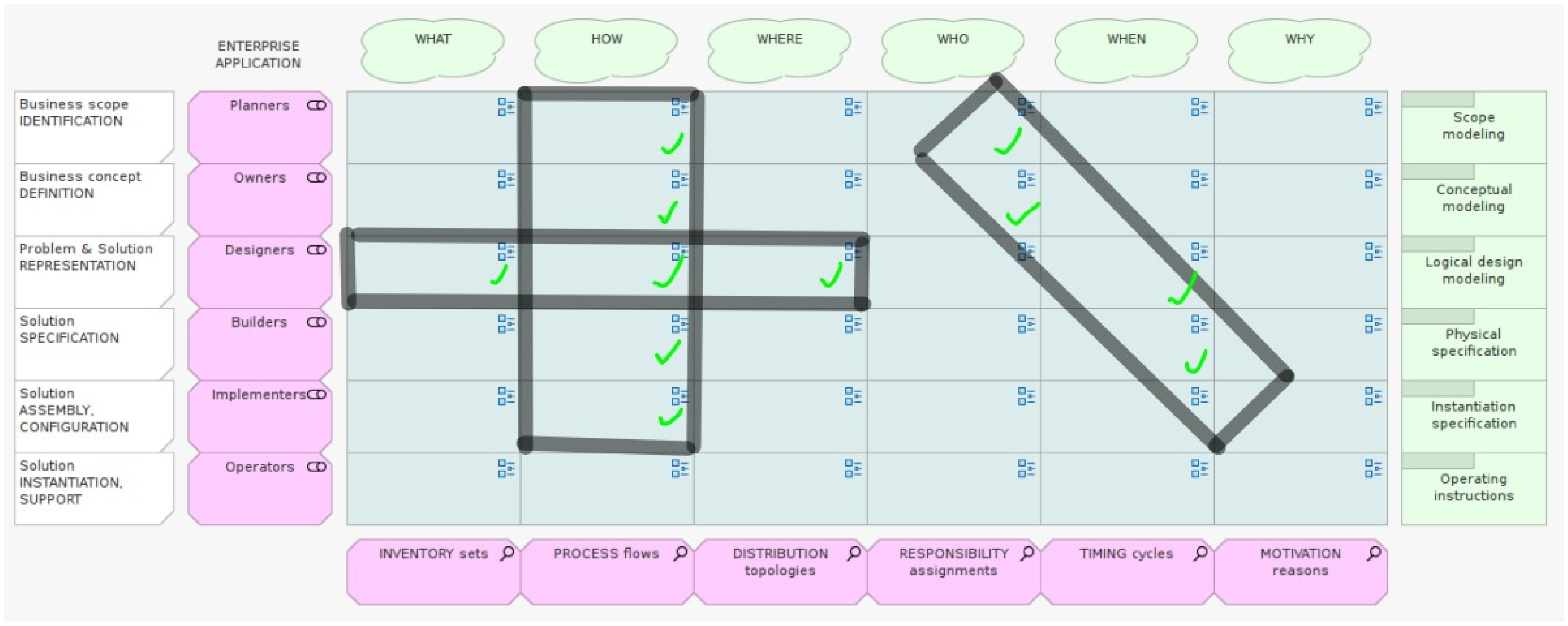
e.g. Behavioral View Type
e.g. Information View Type
e.g. Context View Type
...

A Model Kind defines a diagraming notation and conventions for a type of modeling.

e.g. Process Flow modeling (BPMN)
e.g. Data Flow modeling (DFD)
e.g. Package modeling (UML)
e.g. Application structure (Archimate)
...

Definition

$$\begin{tikzcd} & \text{MAPPING} \arrow[d, dash] & \\ \checkmark \text{ PRIMITIVES models} \arrow[rr] & {} & \text{COMPOSITE models} \end{tikzcd}$$



	ENTERPRISE APPLICATION	WHAT	HOW	WHERE	WHO	WHEN	WHY	
Business scope IDENTIFICATION	Planners		✓		✓			Scope modeling
Business concept DEFINITION	Owners		✓		✓			Conceptual modeling
Problem & Solution REPRESENTATION	Designers	✓	✓	✓		✓		Logical design modeling
Solution SPECIFICATION	Builders		✓			✓		Physical specification
Solution ASSEMBLY, CONFIGURATION	Implementers		✓					Instantiation specification
Solution INSTANTIATION, SUPPORT	Operators							Operating instructions
		INVENTORY sets	PROCESS flows	DISTRIBUTION topologies	RESPONSIBILITY assignments	TIMING cycles	MOTIVATION reasons	

Definition

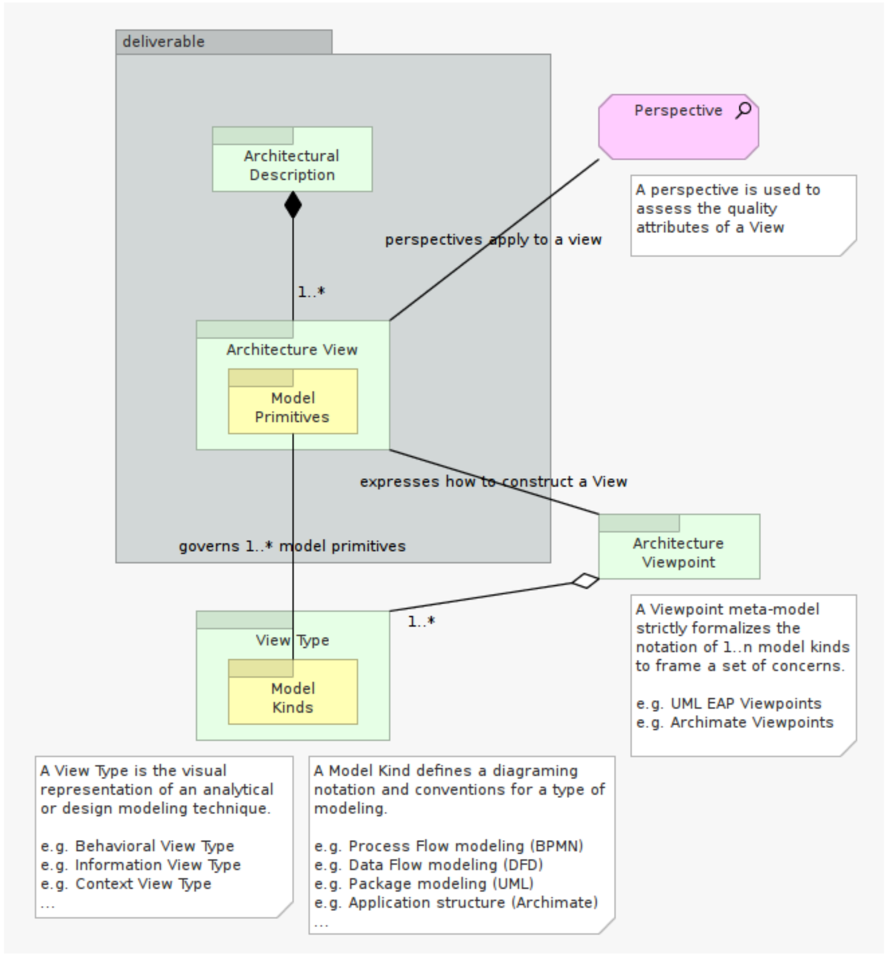
An ARCHITECTURE VIEW is an opinionated composition of (1,n) aspects of an architecture, illustrating how it addresses (1,n) concerns held by (1,n) stakeholders.

A View is COMPOSITE MODEL, made of a selection of (1,n) modeling elements belonging to Primitive Models.

It is a way to portray all the elements of a proposed solution 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.

Purpose


deliverable
Architectural Description
perspectives apply to a view
1..*
Architecture View
Model Primitives
expresses how to construct a View
governs 1..* model primitives
Perspective
Architecture Viewpoint
1..*
View Type
Model Kinds

A perspective is used to assess the quality attributes of a View

A Viewpoint meta-model strictly formalizes the notation of 1..n model kinds to frame a set of concerns.

e.g. UML EAP Viewpoints
e.g. Archimate Viewpoints

A View Type is the visual representation of an analytical or design modeling technique.

e.g. Behavioral View Type
e.g. Information View Type
e.g. Context View Type
...

A Model Kind defines a diagraming notation and conventions for a type of modeling.

e.g. Process Flow modeling (BPMN)
e.g. Data Flow modeling (DFD)
e.g. Package modeling (UML)
e.g. Application structure (Archimate)
...

Purpose

Architecture Views aim COVER as much of the problem space as possible,

... by illustrating the COMPLETENESS of the proposed solution (using Viewpoints),

... by answering for the desired quality attributes of the application (using Perspectives).

A VIEW communicates the resolution of a class of Architectural concerns using a VIEWPOINT as frame of reference.

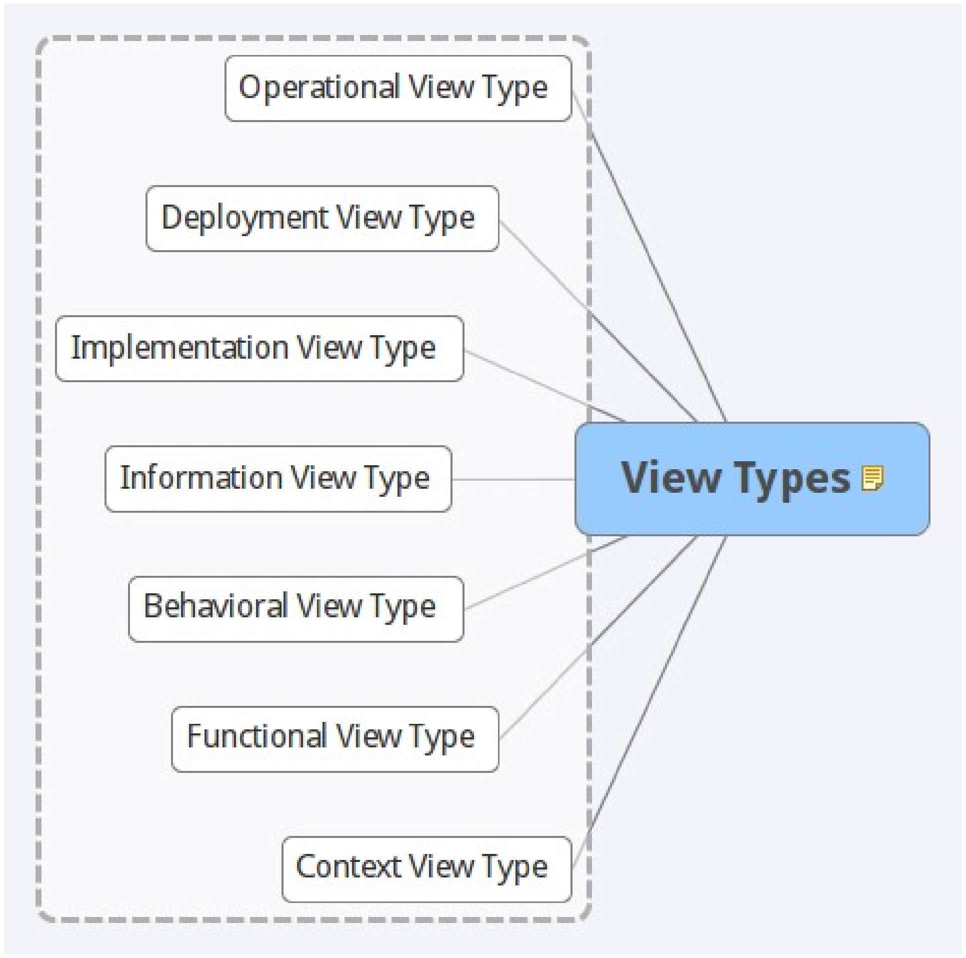
A VIEW leverages a VIEWPOINT, proposing rules to construct a model.

VIEWPOINTS express how to check the well formedness of that VIEW.

VIEWPOINTS provide rules for sharing details between views and for the use of multiple notations within a model.

Each View is modelled in accordance with the conventions established by one Viewpoint.

View Types


Operational View Type
Deployment View Type
Implementation View Type
Information View Type
Behavioral View Type
Functional View Type
Context View Type
View Types

View Types

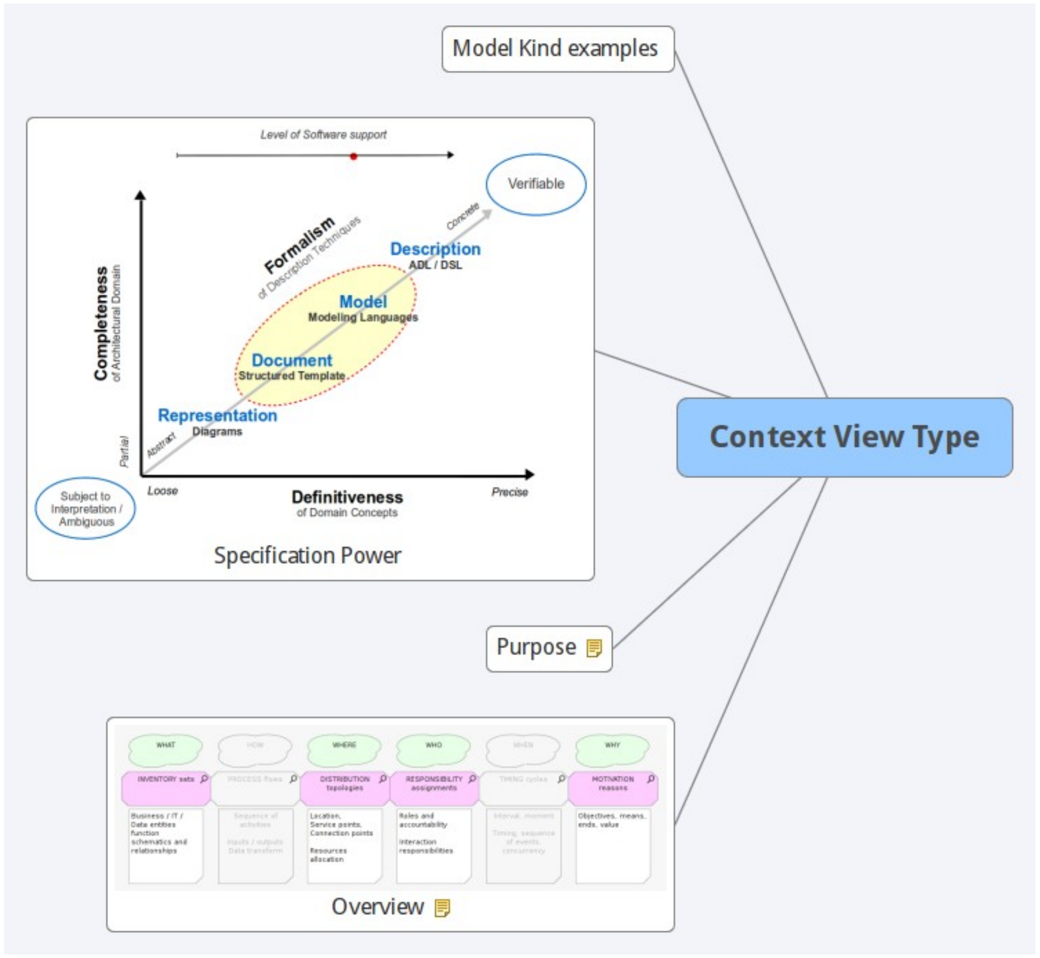
Each View describing the architecture is constructed within the guidelines of a model type, broadly speaking a template.

For example:

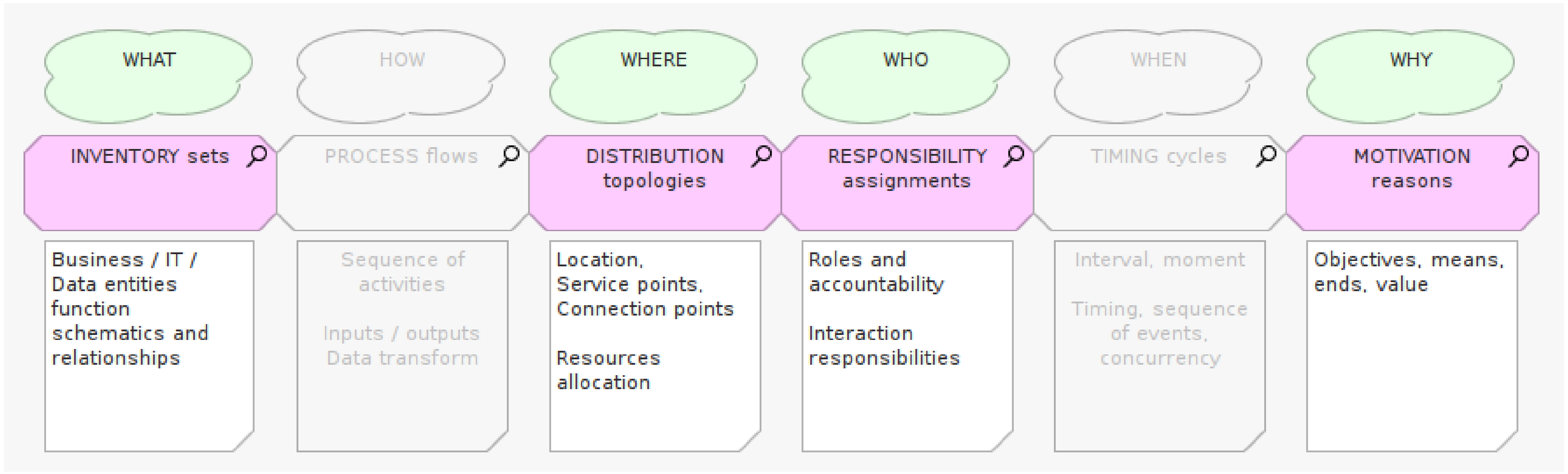
- Some ideas are best expressed by showing the "static" structure of an application, using a component diagram illustrating what function is realized by a set of components.

- Some ideas are best expressed by showing the flow of data within an application using a "dynamic" model type, using a process flow diagram illustrating the behavior of a set of components.

Context View Type



Overview


WHAT
HOW
WHERE
WHO
WHEN
WHY
INVENTORY sets
PROCESS flows
DISTRIBUTION topologies
RESPONSIBILITY assignments
TIMING cycles
MOTIVATION reasons
Business / IT / Data entities function schematics and relationships
Sequence of activities
Inputs / outputs Data transform
Location, Service points, Connection points
Resources allocation
Roles and accountability
Interaction responsibilities
Interval, moment
Timing, sequence of events, concurrency
Objectives, means, ends, value

Overview

The Context View Type describes the SCOPE of architecture solution - ex. objectives / capabilities / features / stakeholder concerns.

Views adapted from this Type are of interest to non-technical Stakeholders, because of the simplicity of the Model Kinds available.

This 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).

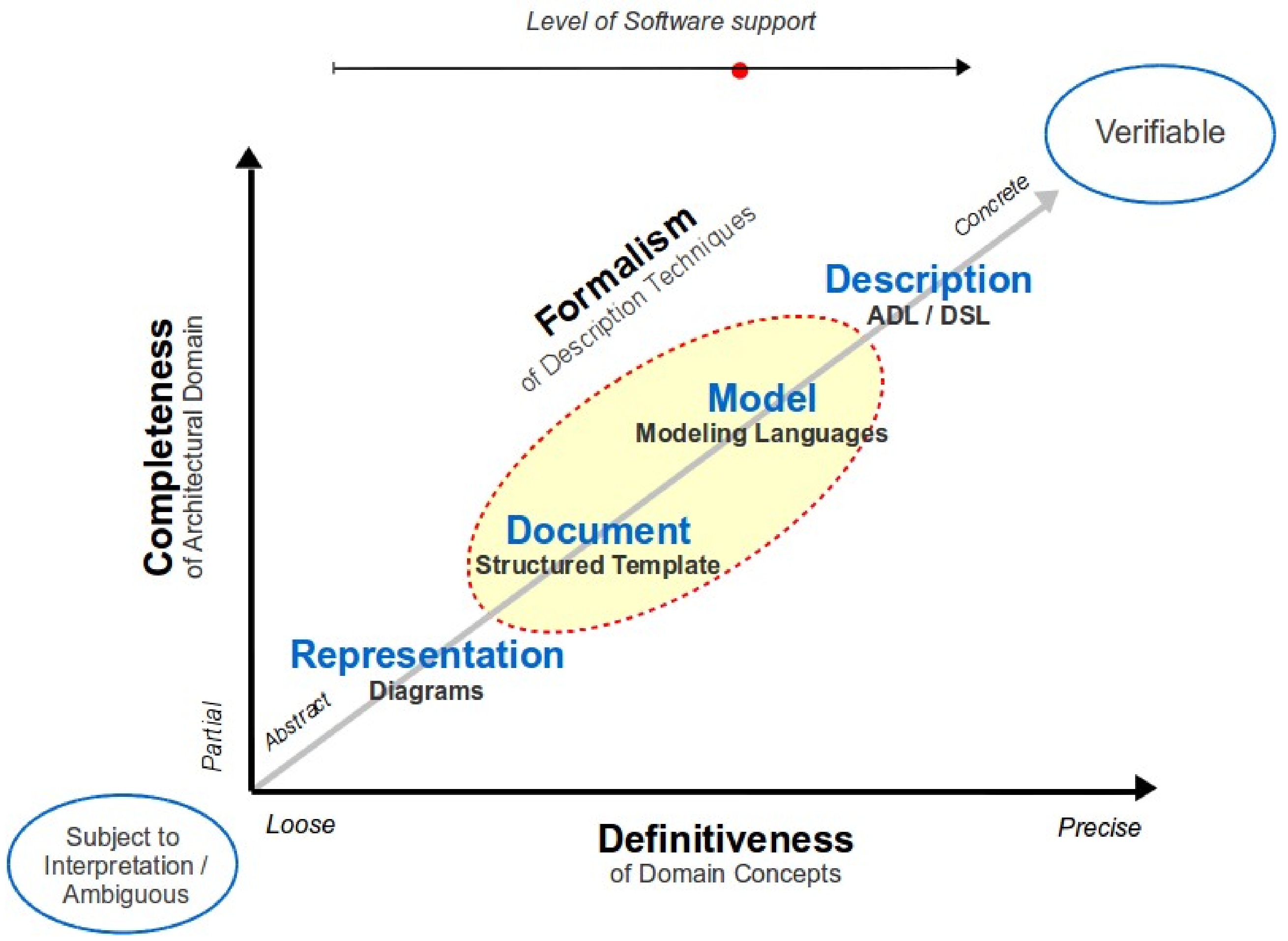
Purpose

The purpose of the Context View Type is to:

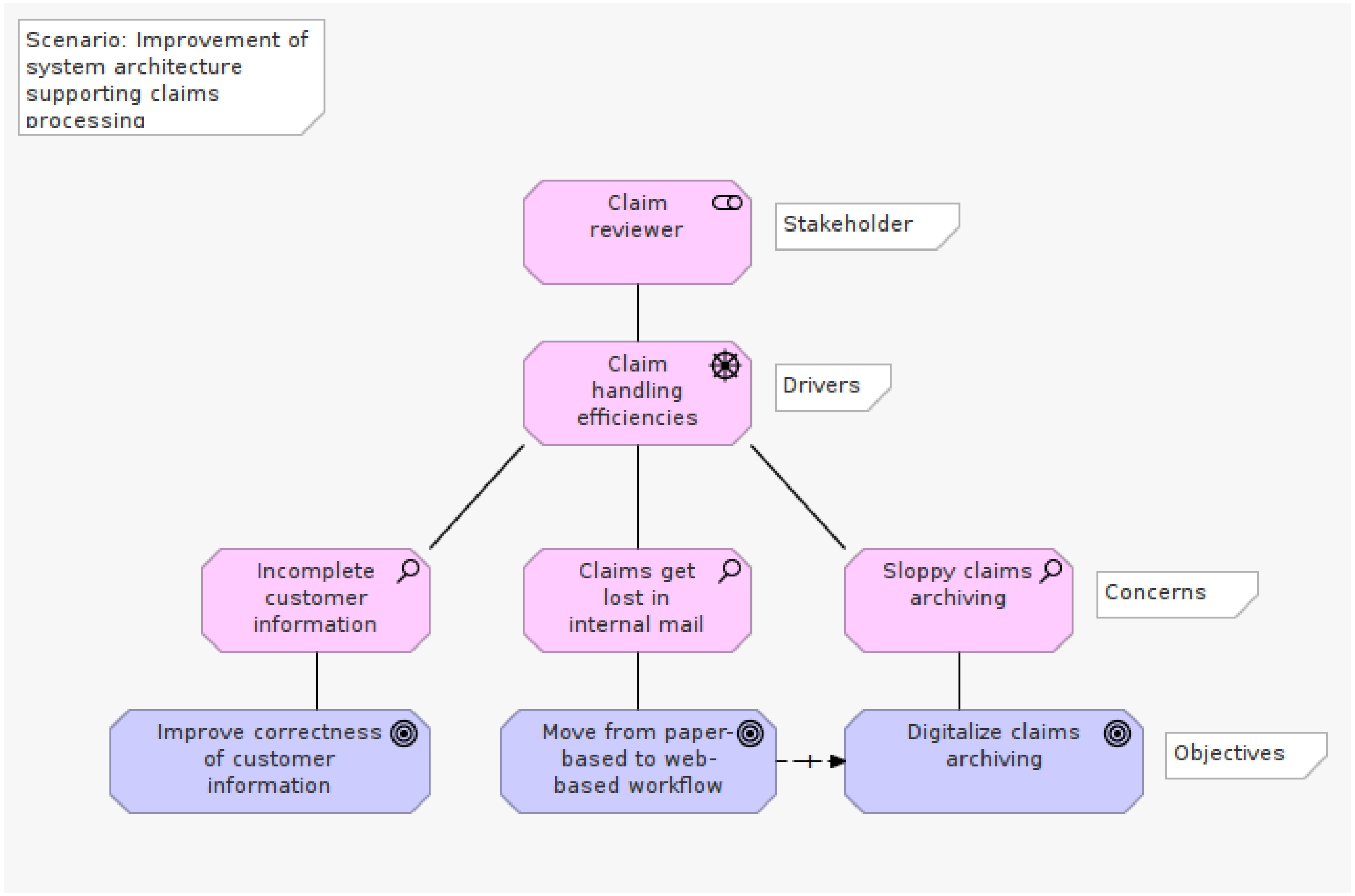
- specify a "scope of responsibility" for an Architecture, in terms that align with Business Motivations

- to illustrate the nature of internal & external entities in make up the environment of the Architecture i.e. Actors (People, Systems).

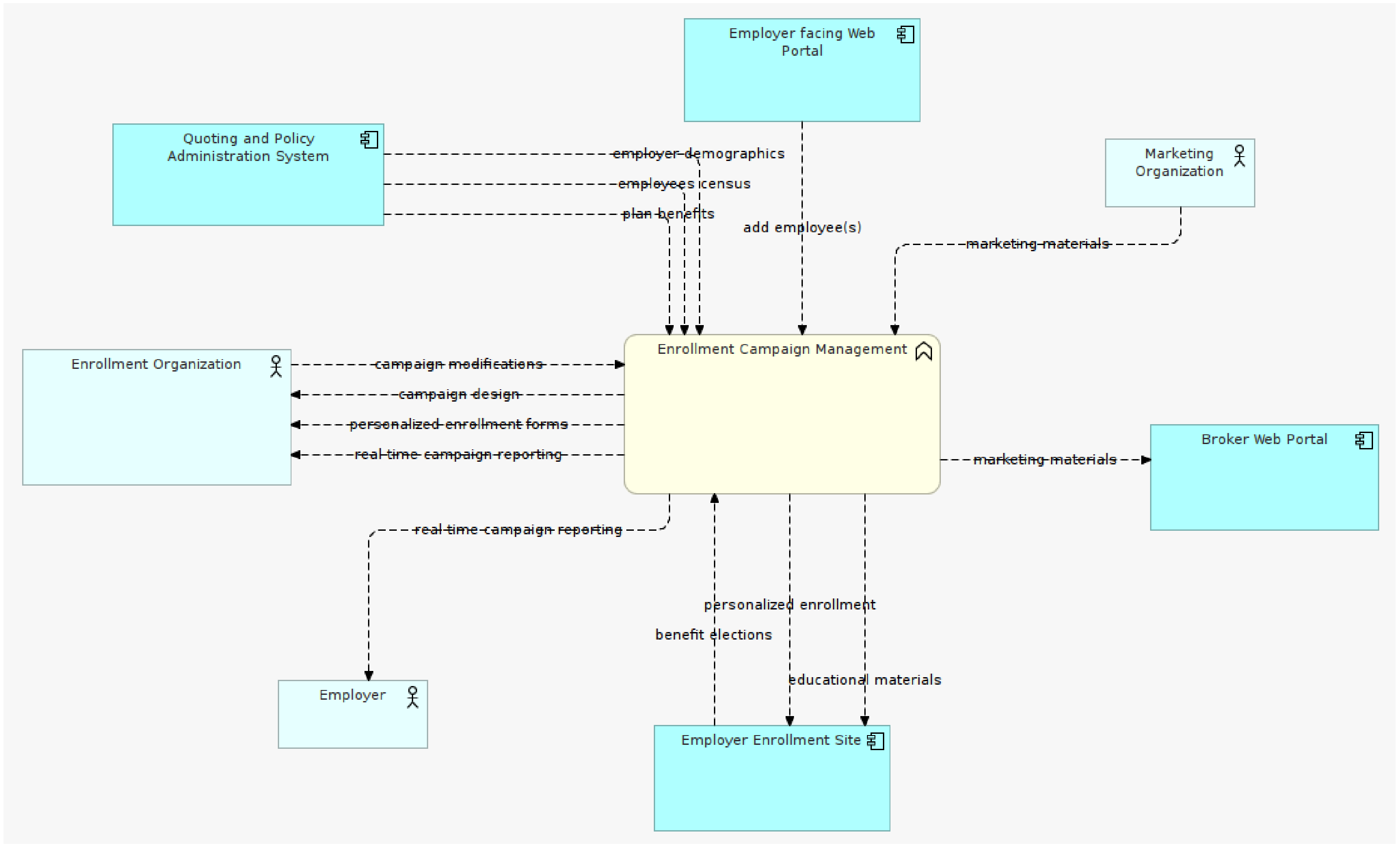
Specification Power



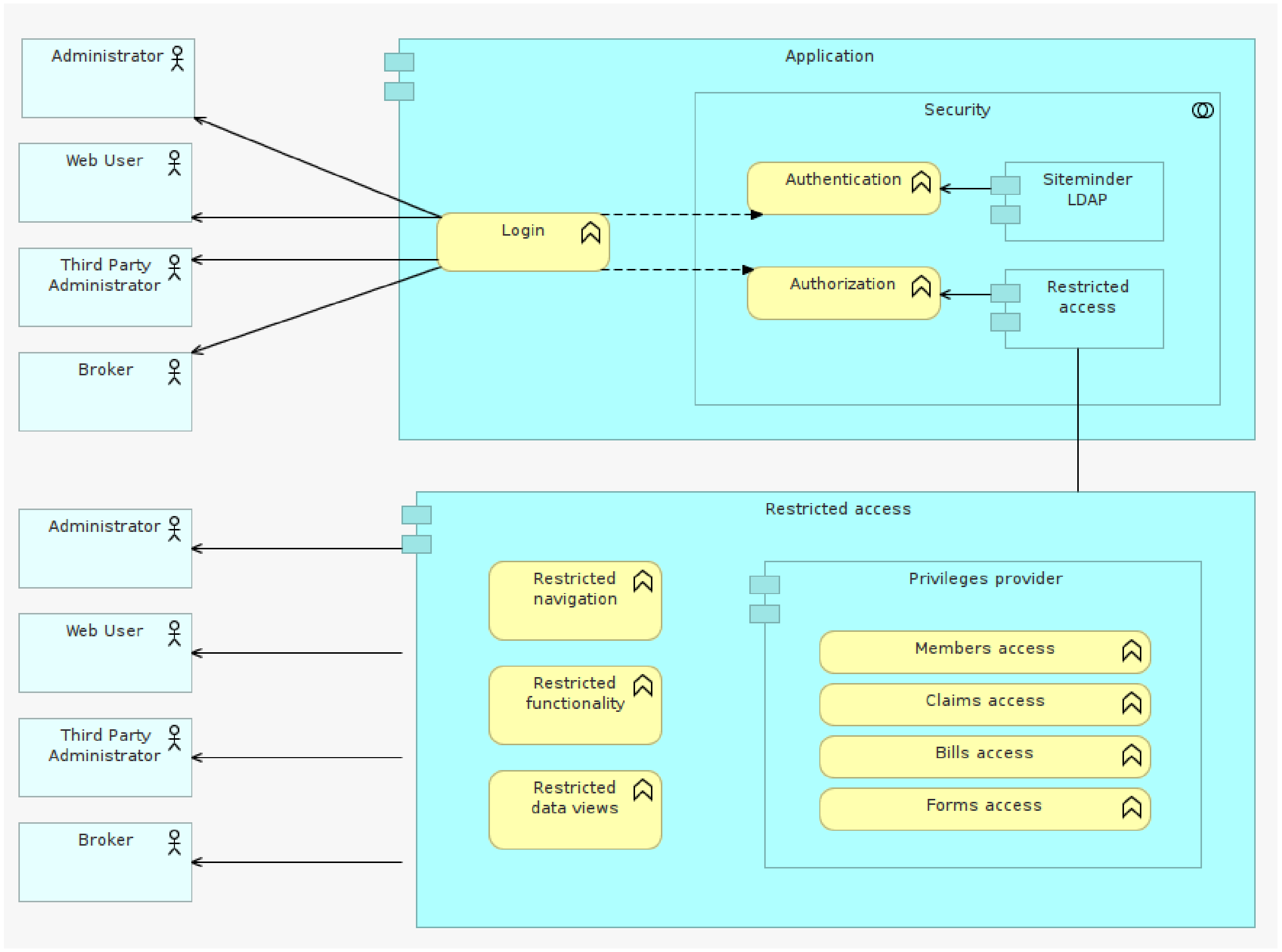
Example: Motivation modeling


Scenario: Improvement of system architecture supporting claims processing
Claim reviewer
Stakeholder
Claim handling efficiencies
Drivers
Incomplete customer information
Claims get lost in internal mail
Sloppy claims archiving
Concerns
Improve correctness of customer information
Move from paper-based to web-based workflow
Digitalize claims archiving
Objectives

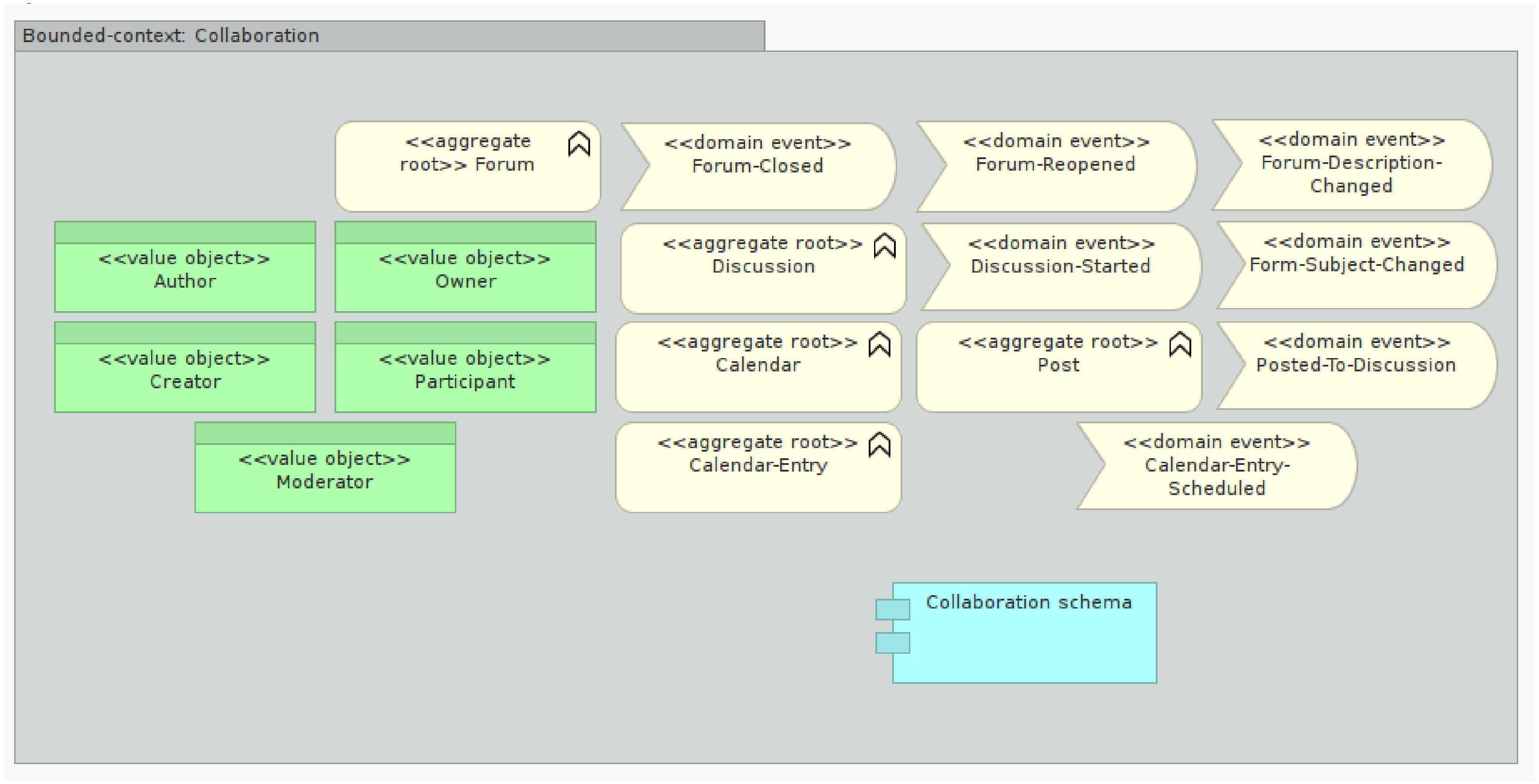
Example: System Context modeling



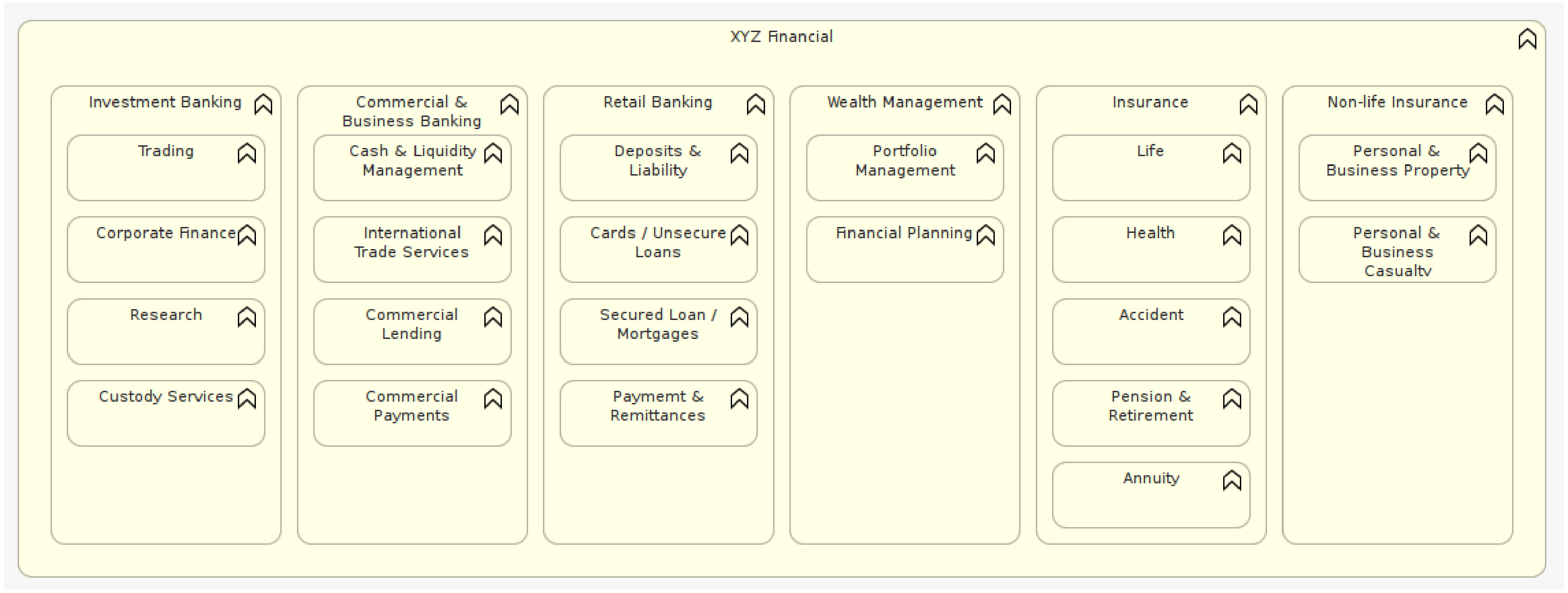
Example: Application Features modeling



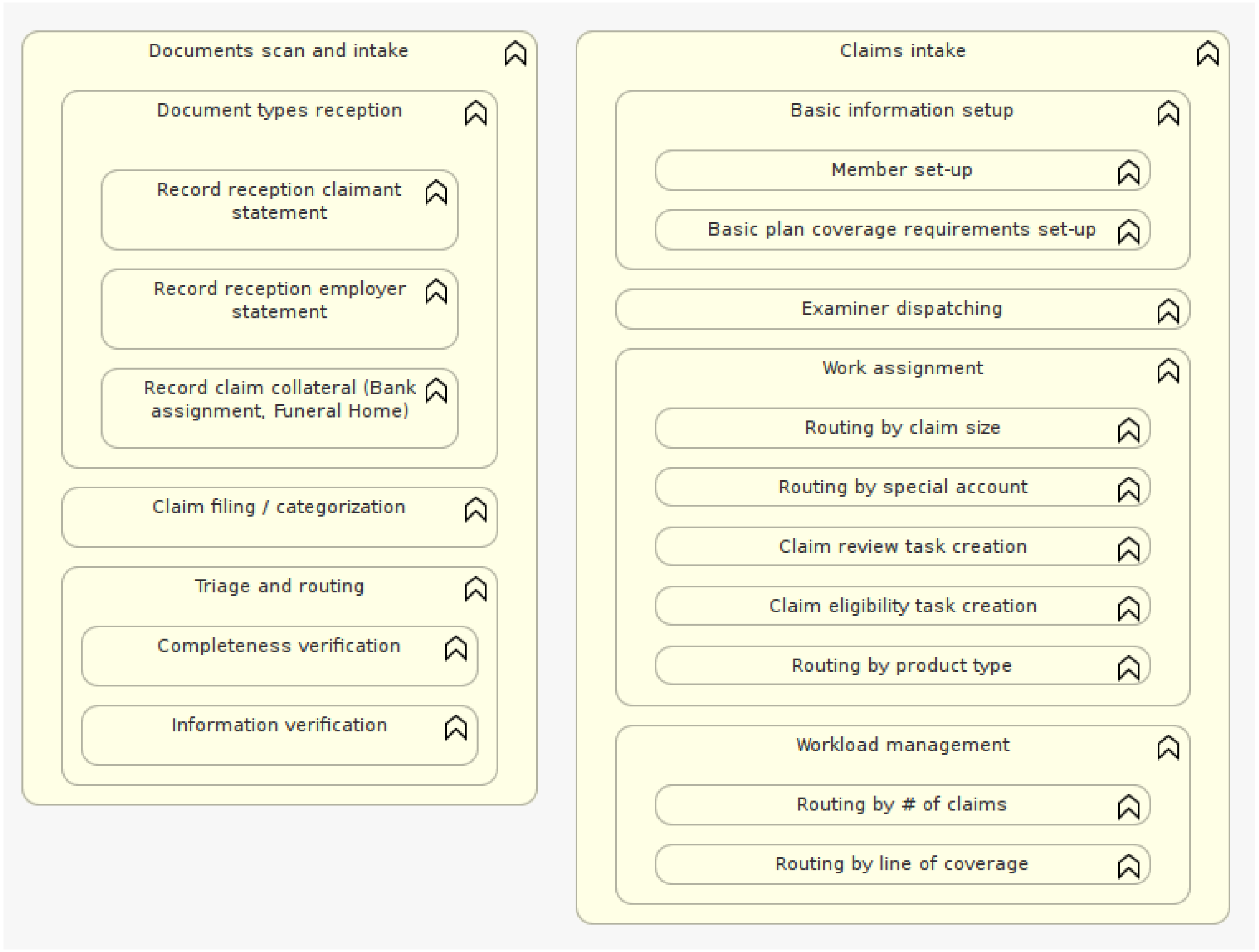
Example: Domain-driven modeling


Bounded-context: Collaboration
<<aggregate root>> Forum
<<domain event>> Forum-Closed
<<domain event>> Forum-Reopened
<<domain event>> Forum-Description-Changed
<<value object>> Author
<<value object>> Owner
<<aggregate root>> Discussion
<<domain event>> Discussion-Started
<<domain event>> Form-Subject-Changed
<<value object>> Creator
<<value object>> Participant
<<aggregate root>> Calendar
<<aggregate root>> Post
<<domain event>> Posted-To-Discussion
<<value object>> Moderator
<<aggregate root>> Calendar-Entry
<<domain event>> Calendar-Entry-Scheduled
Collaboration schema

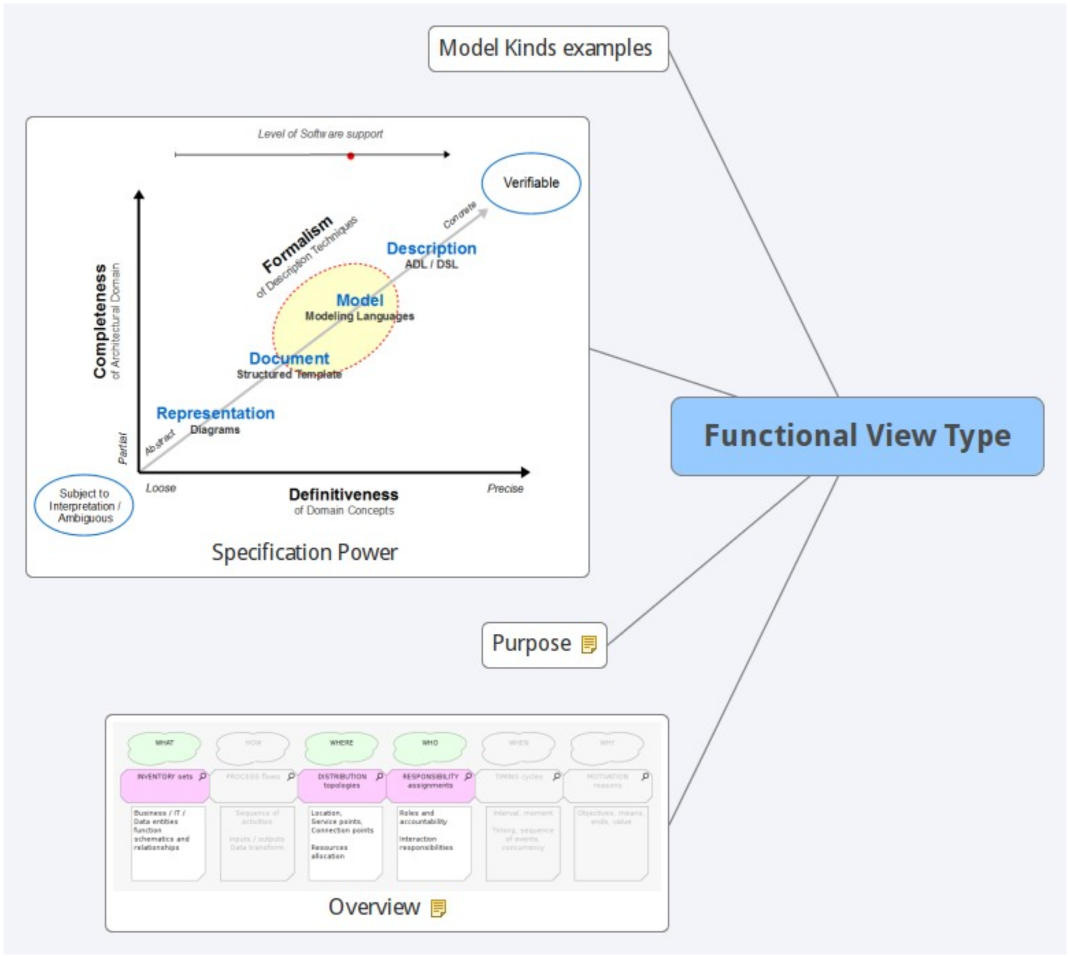
Example: Business Function modeling



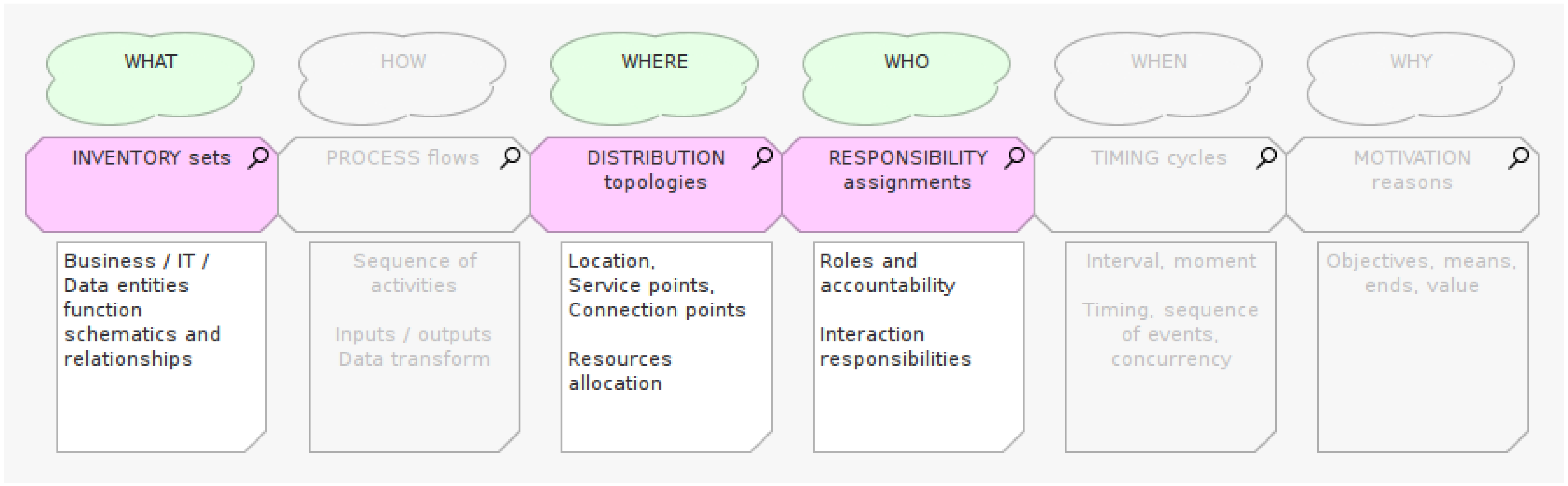
Example: Business Capability modeling


Documents scan and intake
Document types reception
Record reception claimant statement
Record reception employer statement
Record claim collateral (Bank assignment, Funeral Home)
Claim filing / categorization
Triage and routing
Completeness verification
Information verification
Claims intake
Basic information setup
Member set-up
Basic plan coverage requirements set-up
Examiner dispatching
Work assignment
Routing by claim size
Routing by special account
Claim review task creation
Claim eligibility task creation
Routing by product type
Workload management
Routing by # of claims
Routing by line of coverage

Functional View Type



Overview


WHAT
HOW
WHERE
WHO
WHEN
WHY
INVENTORY sets
PROCESS flows
DISTRIBUTION topologies
RESPONSIBILITY assignments
TIMING cycles
MOTIVATION reasons
Business / IT / Data entities function schematics and relationships
Sequence of activities
Inputs / outputs Data transform
Location, Service points, Connection points
Resources allocation
Roles and accountability
Interaction responsibilities
Interval, moment
Timing, sequence of events, concurrency
Objectives, means, ends, value

Overview

The "Functional View Type" describes the functional elements of a System in terms of their static functional STRUCTURE.

It is the cornerstone of most architecture documents and is often the first part of the documentation technical stakeholders read.

It specifies component elements, describes primary interactions for groups of components, as well as describing their RELATIONSHIP.

Purpose

The purpose of the Functional View Type is:

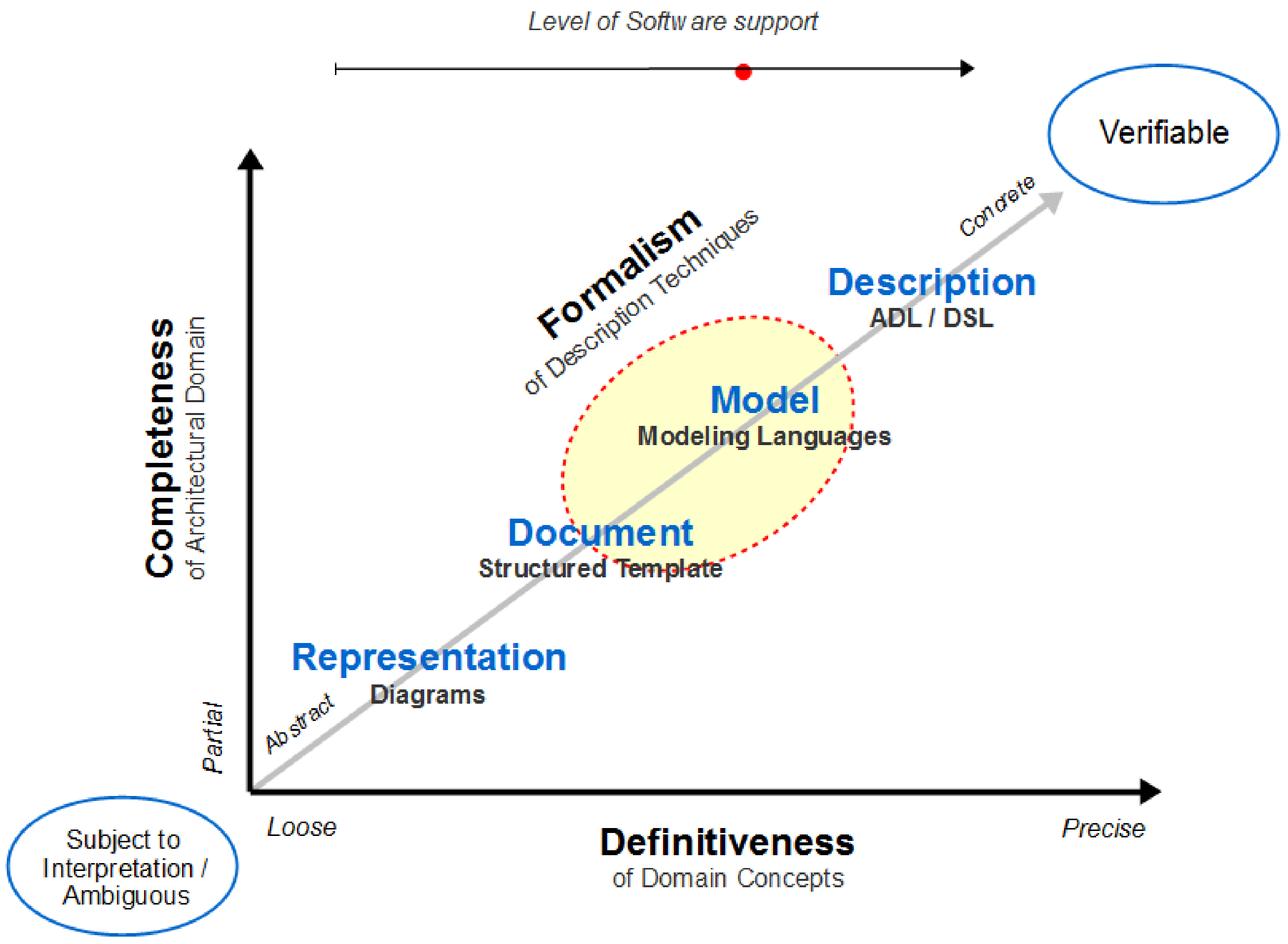
- to outline the key architectural components of an application or component answering concerns

- to describe component responsibilities, interfaces, and primary interactions

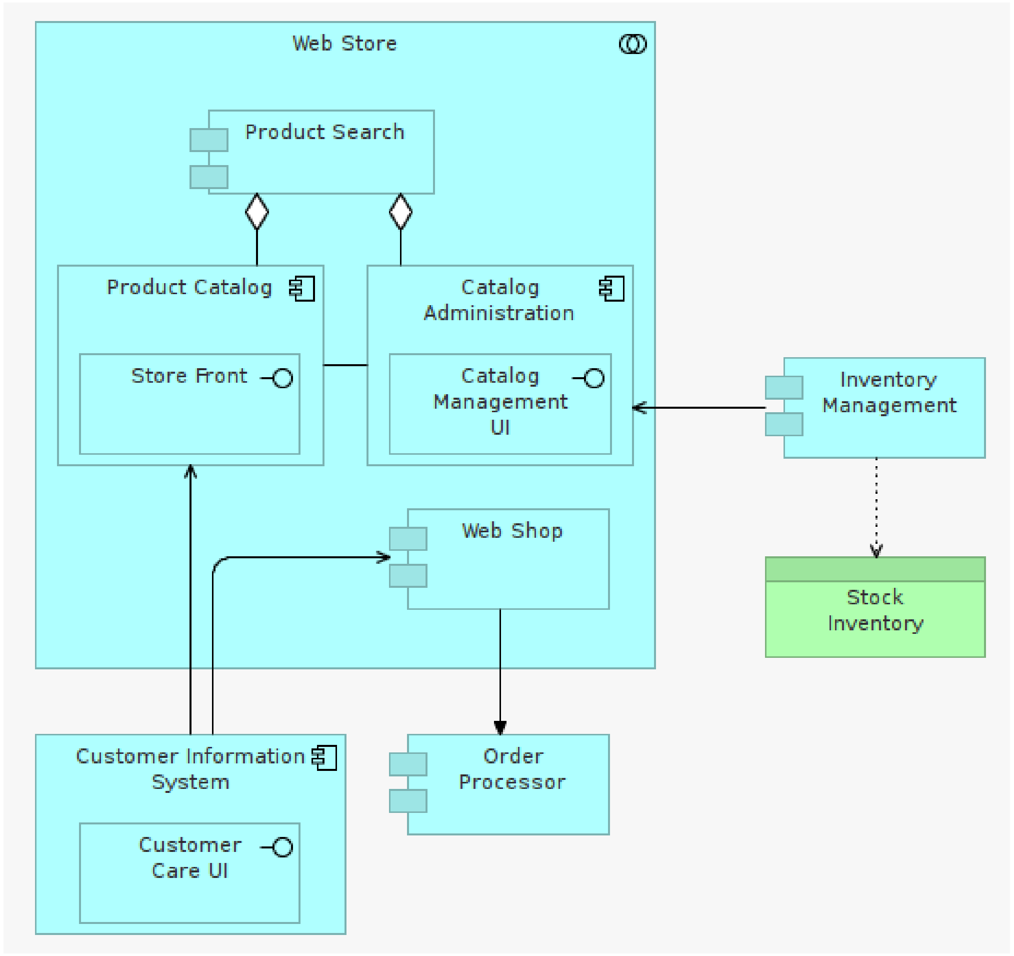
- to decide on what levels of indirection are applicable to a given situation (patterns, styles)

An architect chooses among design options in order to create an architecture that meets the requirements, exhibits the required quality properties, and is fit for purpose.

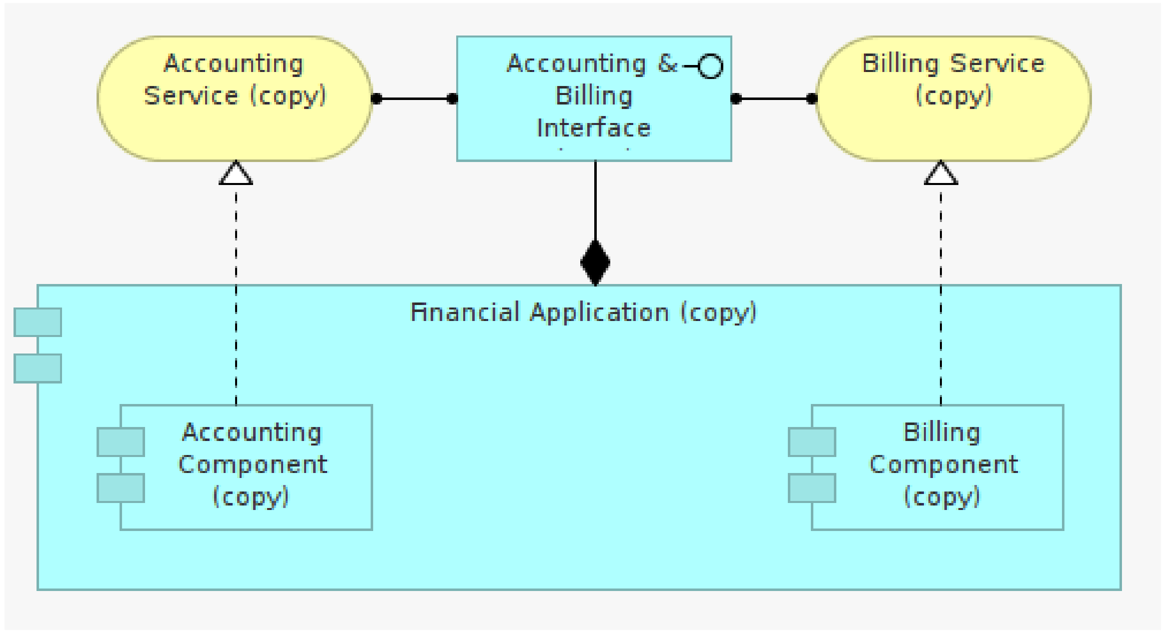
Specification Power



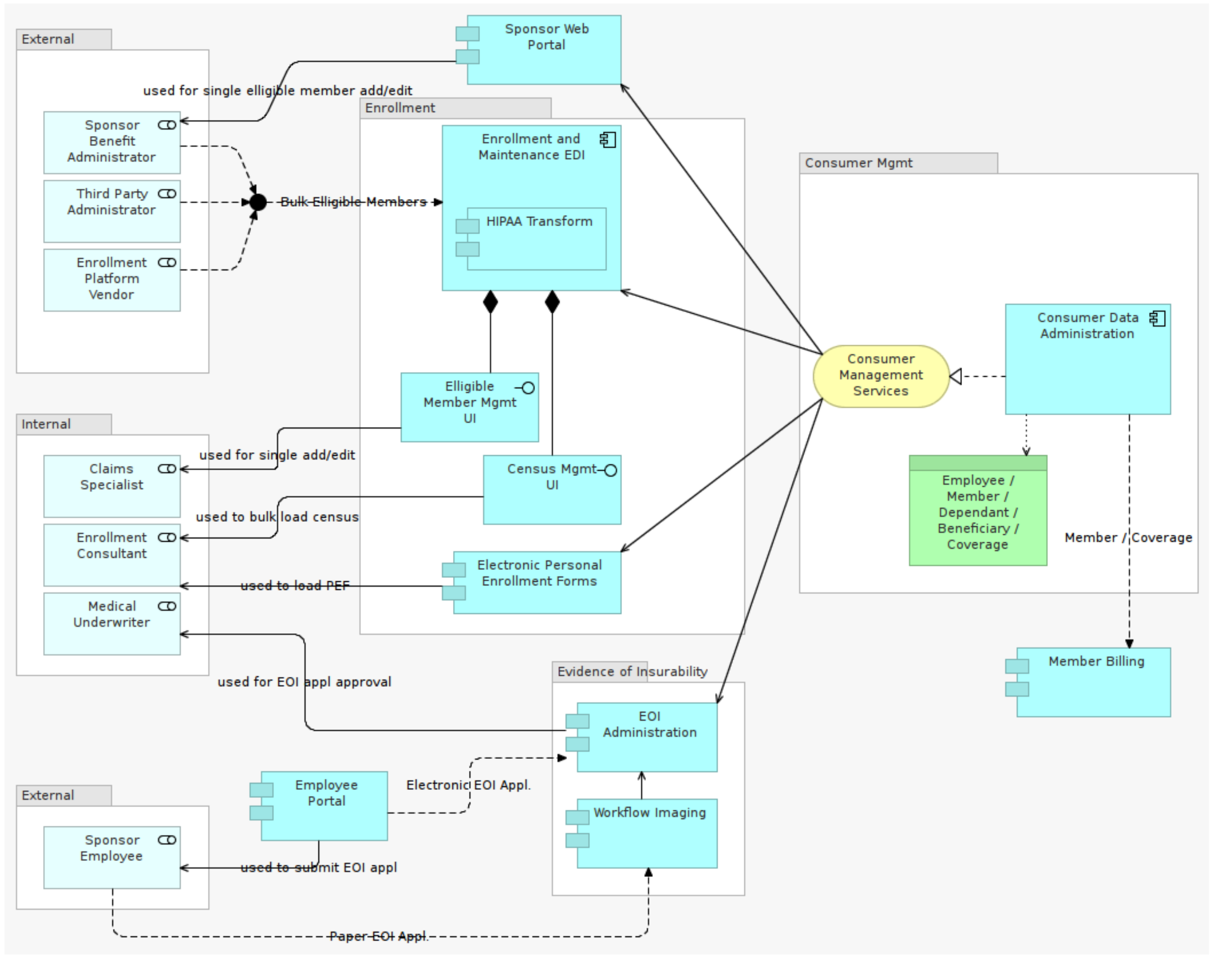
Example: Application Structure modeling



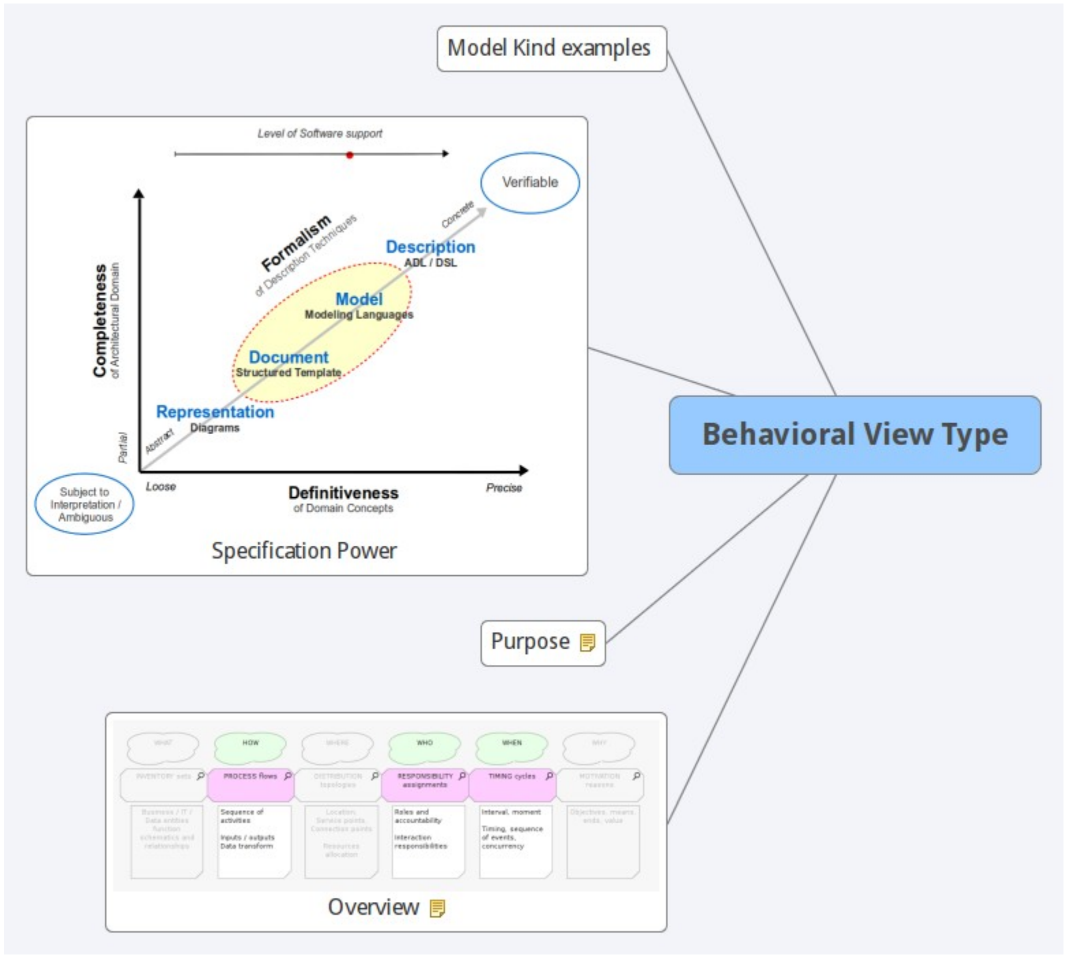
Example: Application Co-operation modeling


Accounting Service (copy)
Accounting & Billing Interface
Billing Service (copy)
Financial Application (copy)
Accounting Component (copy)
Billing Component (copy)

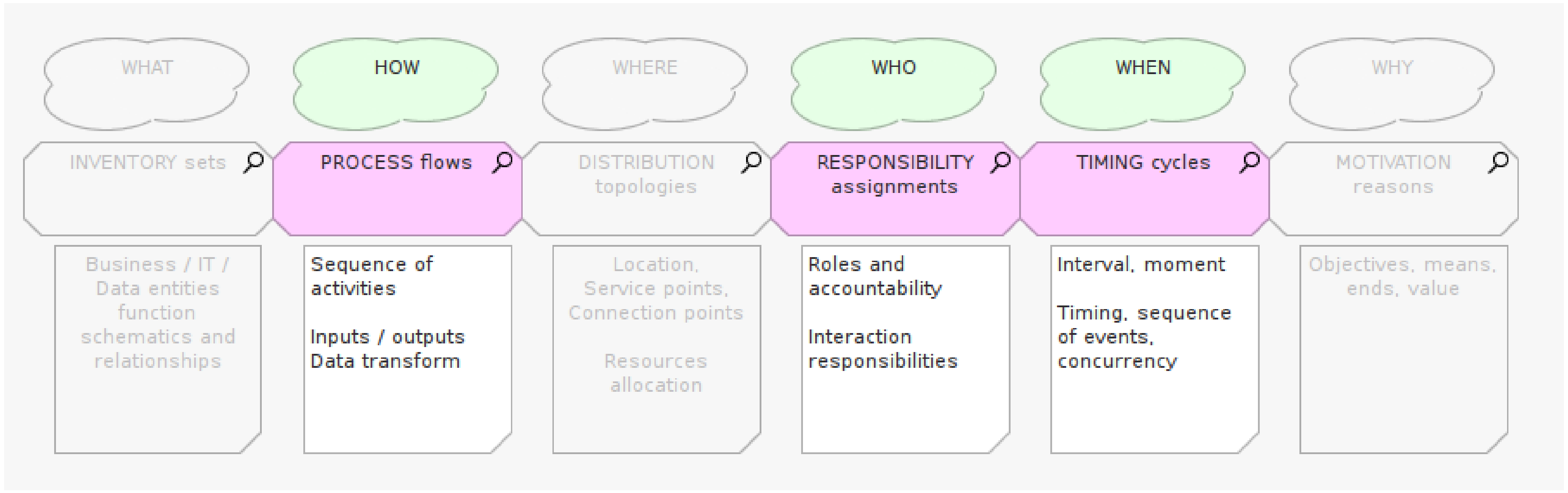
Example: Application Usage modeling



Behavioral View Type



Overview


WHAT
HOW
WHERE
WHO
WHEN
WHY
INVENTORY sets
PROCESS flows
DISTRIBUTION topologies
RESPONSIBILITY assignments
TIMING cycles
MOTIVATION reasons
Business / IT / Data entities function schematics and relationships
Sequence of activities
Inputs / outputs Data transform
Location, Service points, Connection points
Resources allocation
Roles and accountability
Interaction responsibilities
Interval, moment
Timing, sequence of events, concurrency
Objectives, means, ends, value

Overview

The "Behavioral View Type" describes the logical process flow of the system logic by sequencing functional elements to clearly identify how the parts of the system are coordinated and controlled.

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.

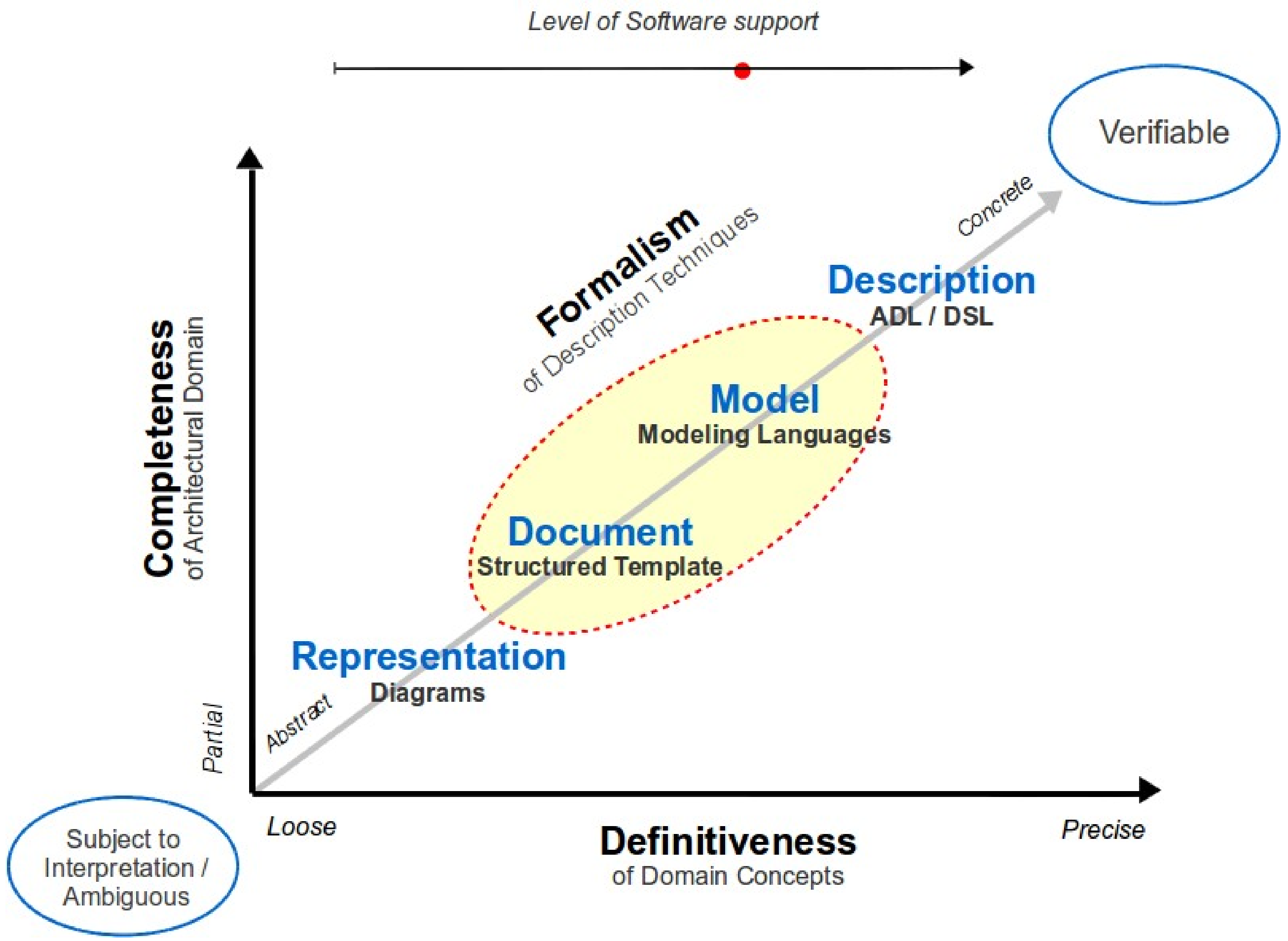
Purpose

The purpose of the Behavioral View Type is to:

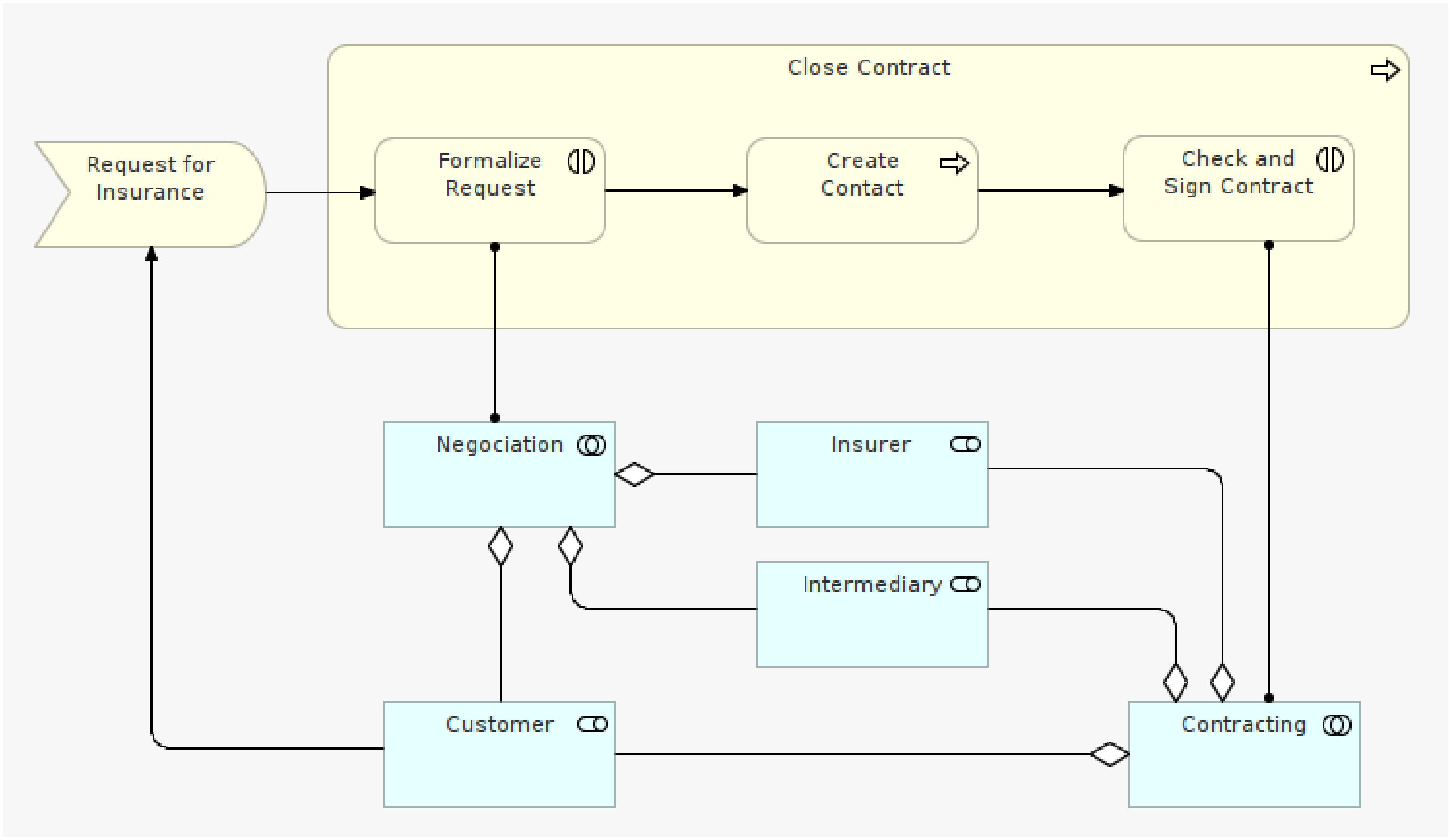
- outline Triggers and Events, Flow-Logic execution tree, Flow-Synchronization

- show the communication mechanisms required to coordinate operations between functional elements

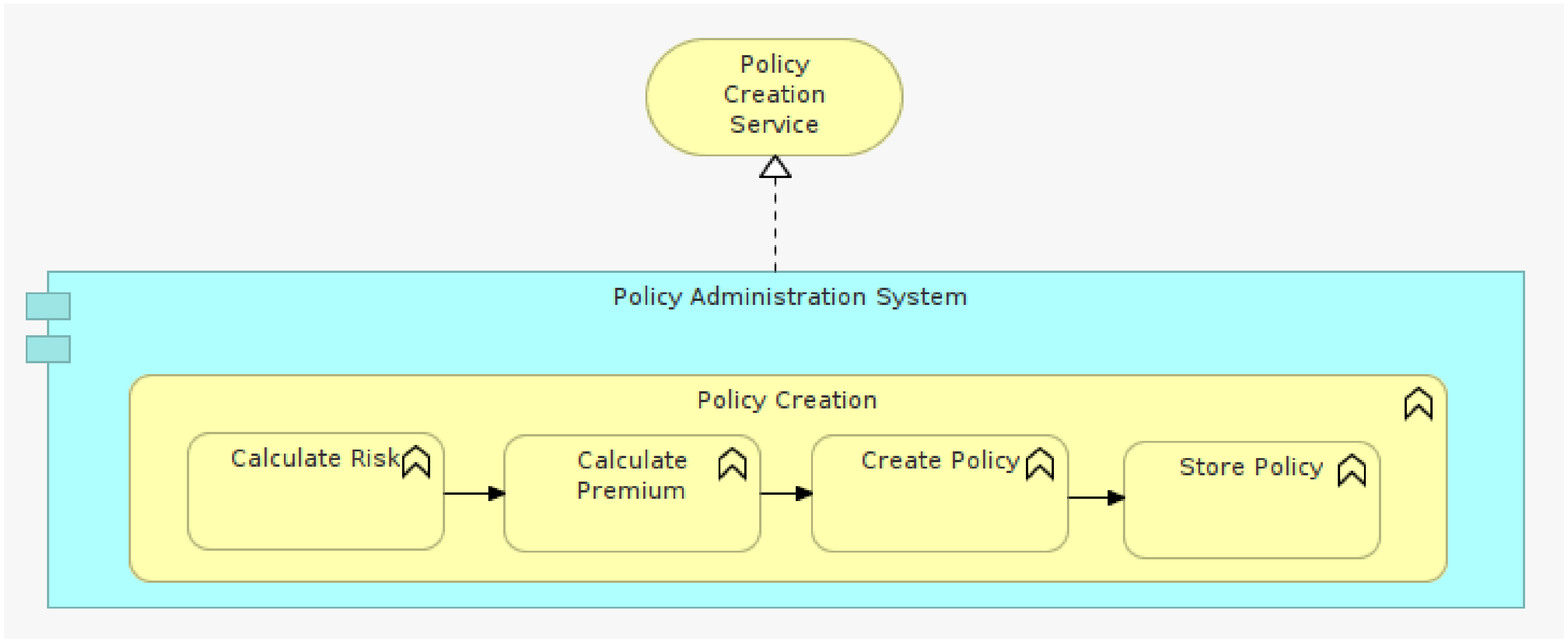
Specification Power



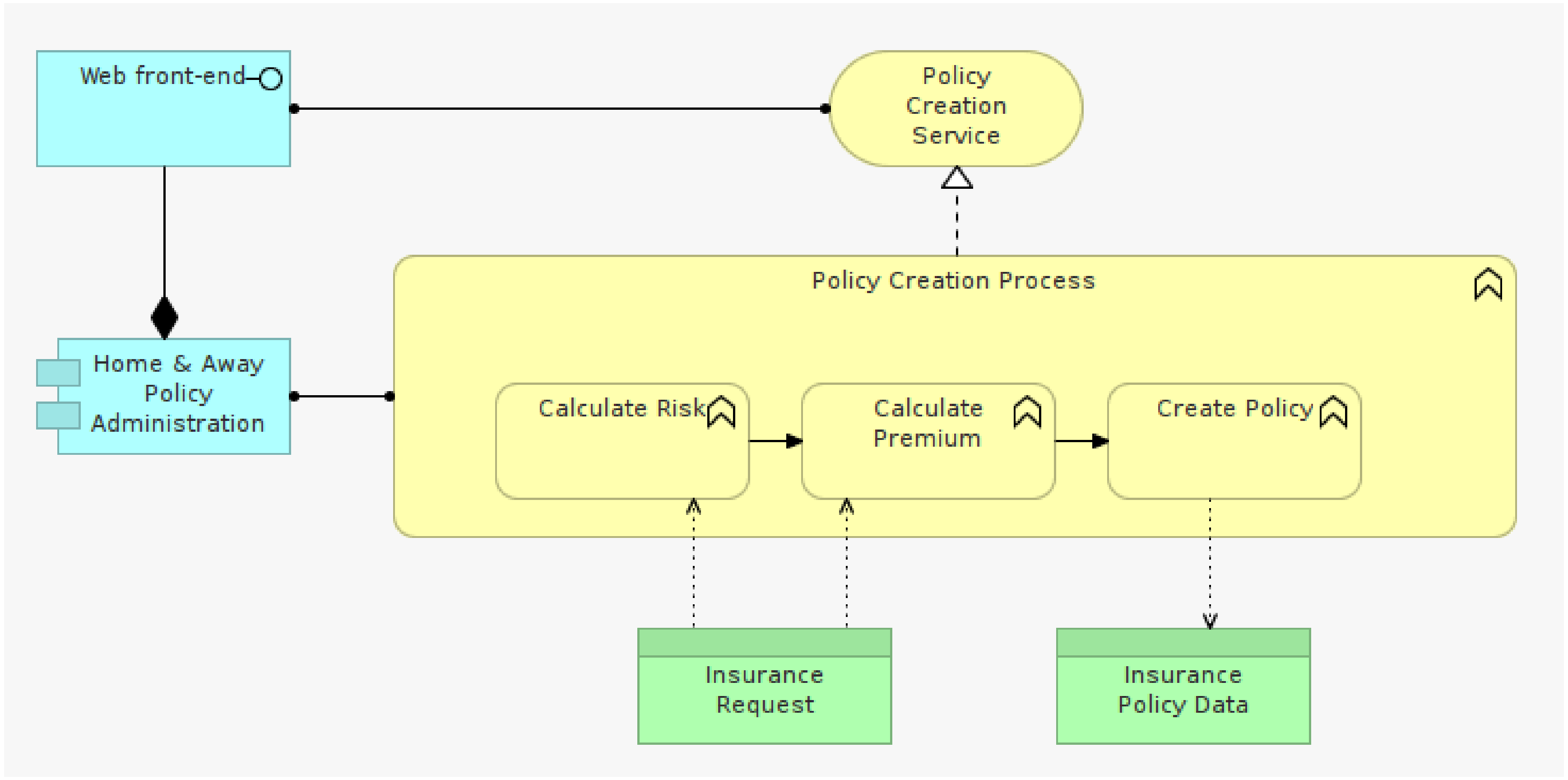
Example: Process Flow modeling



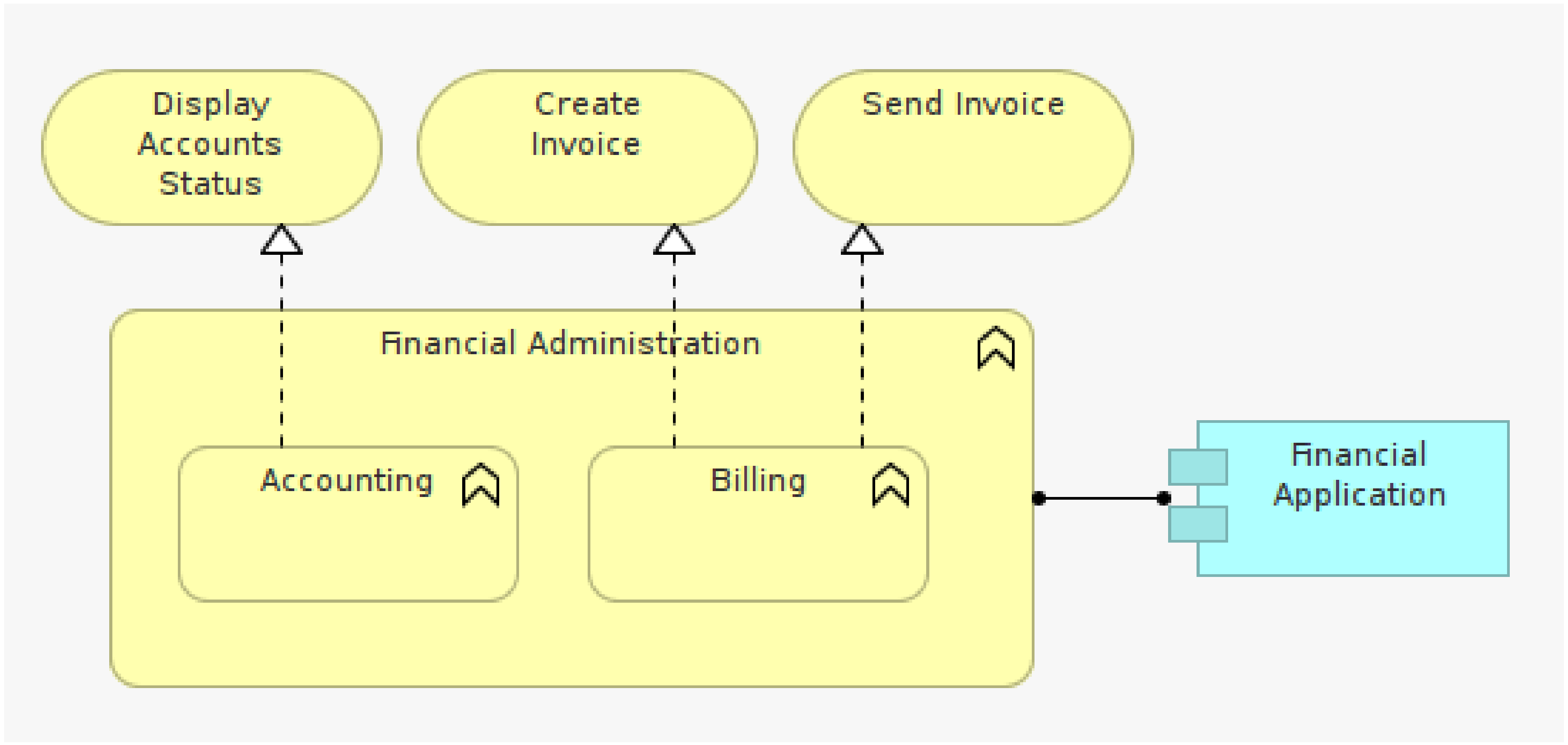
Example: Application Behavior modeling



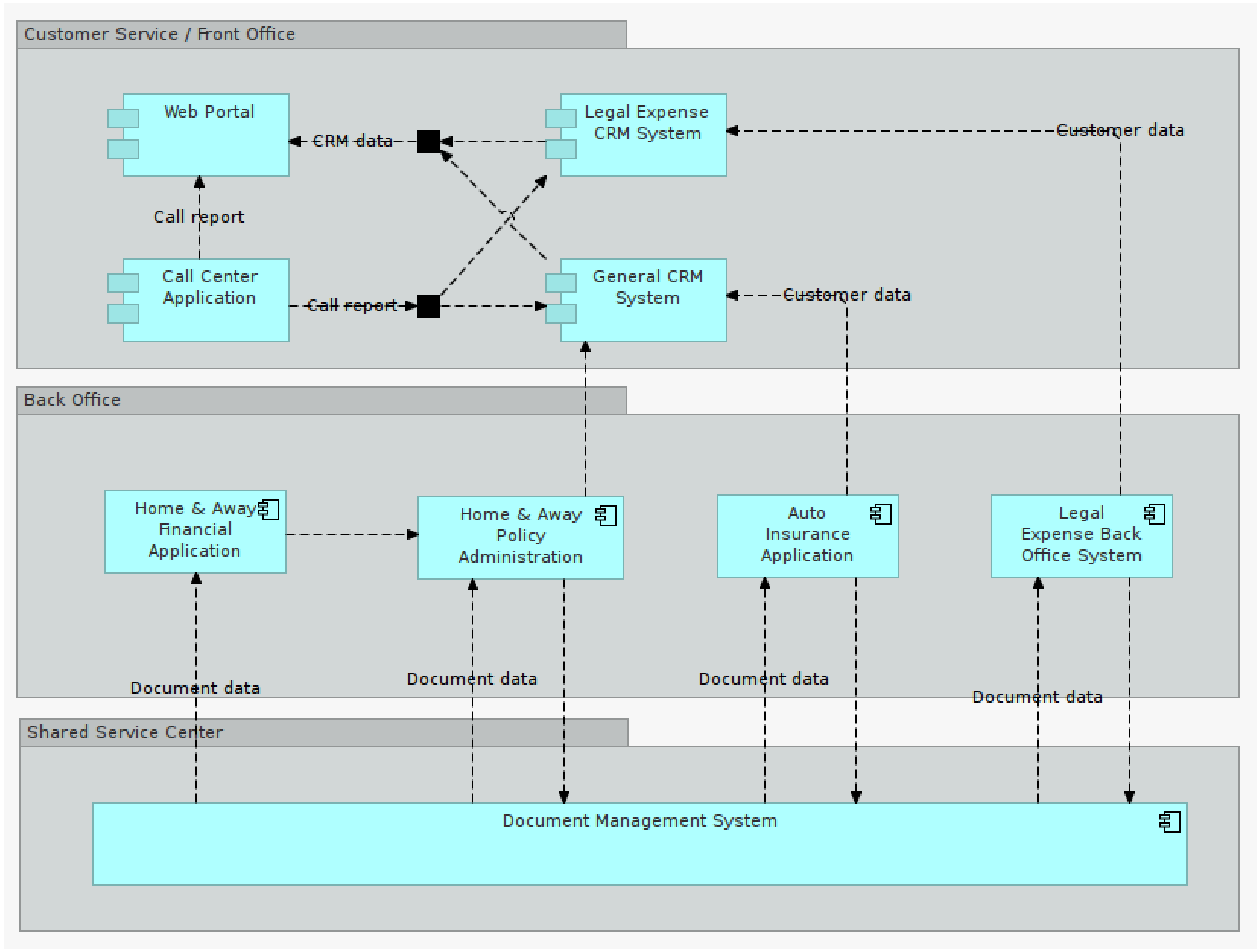
Example: Application Behavior modeling



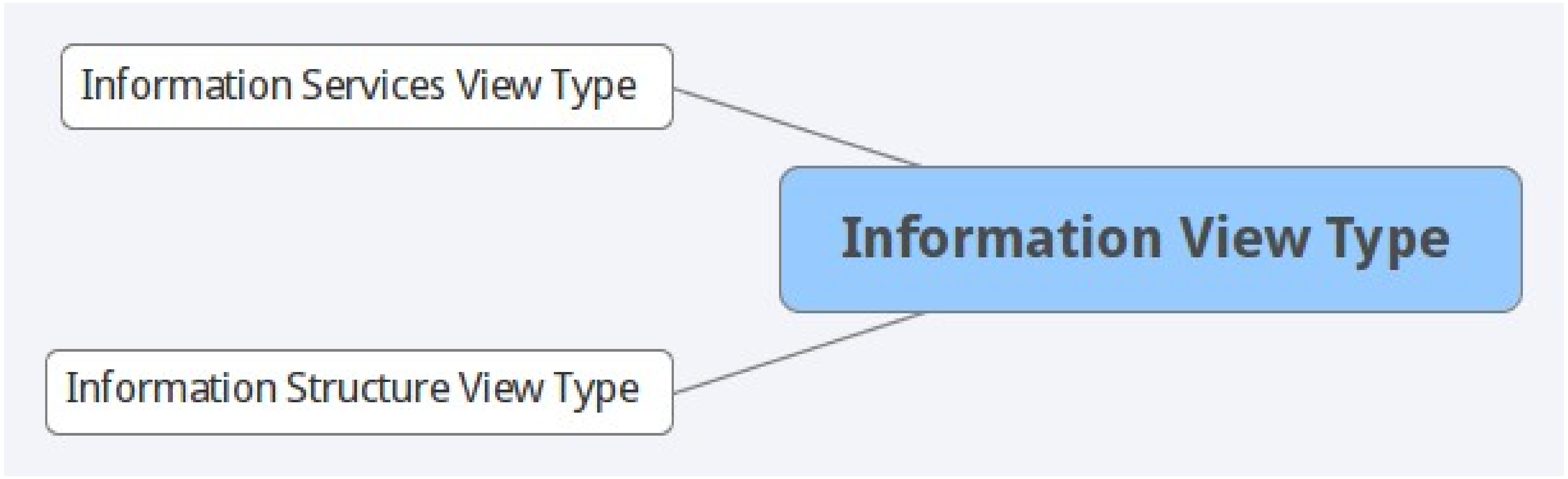
Example: Application Behavior modeling



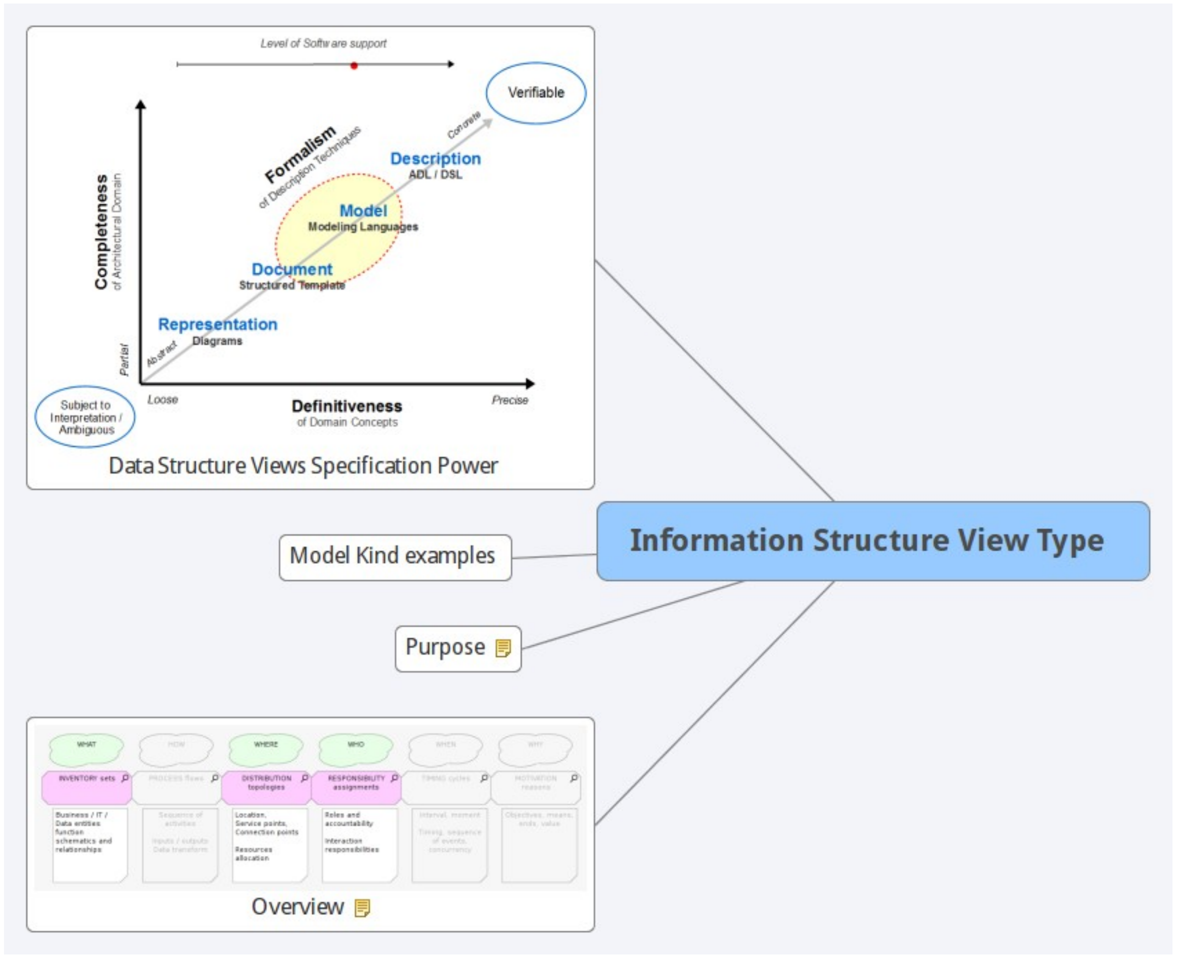
Example: Application Behavior modeling



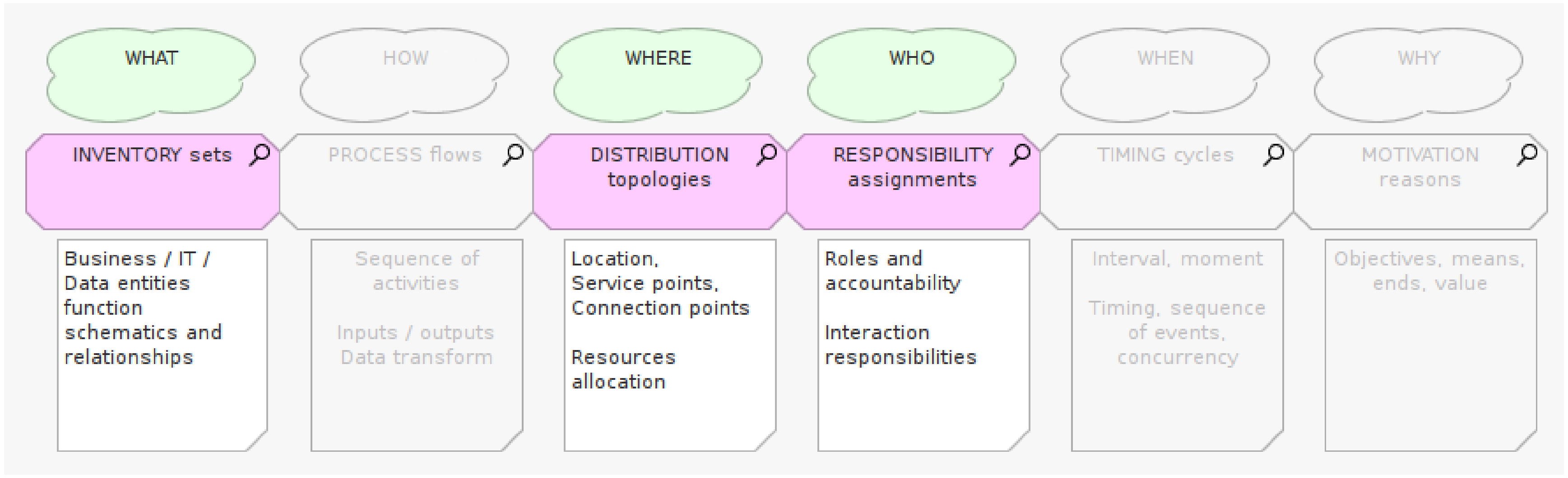
Information View Type


Information Services View Type
Information View Type
Information Structure View Type

Information Structure View Type



Overview


WHAT
HOW
WHERE
WHO
WHEN
WHY
INVENTORY sets
PROCESS flows
DISTRIBUTION topologies
RESPONSIBILITY assignments
TIMING cycles
MOTIVATION reasons
Business / IT / Data entities function schematics and relationships
Sequence of activities
Inputs / outputs Data transform
Location, Service points, Connection points
Resources allocation
Roles and accountability
Interaction responsibilities
Interval, moment
Timing, sequence of events, concurrency
Objectives, means, ends, value

Overview

The "Information Structure View-Type" often referred as referred as DATA-AT-REST or DATA PERSISTENCE, describes the way that the architecture stores information.

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 message format.

Purpose

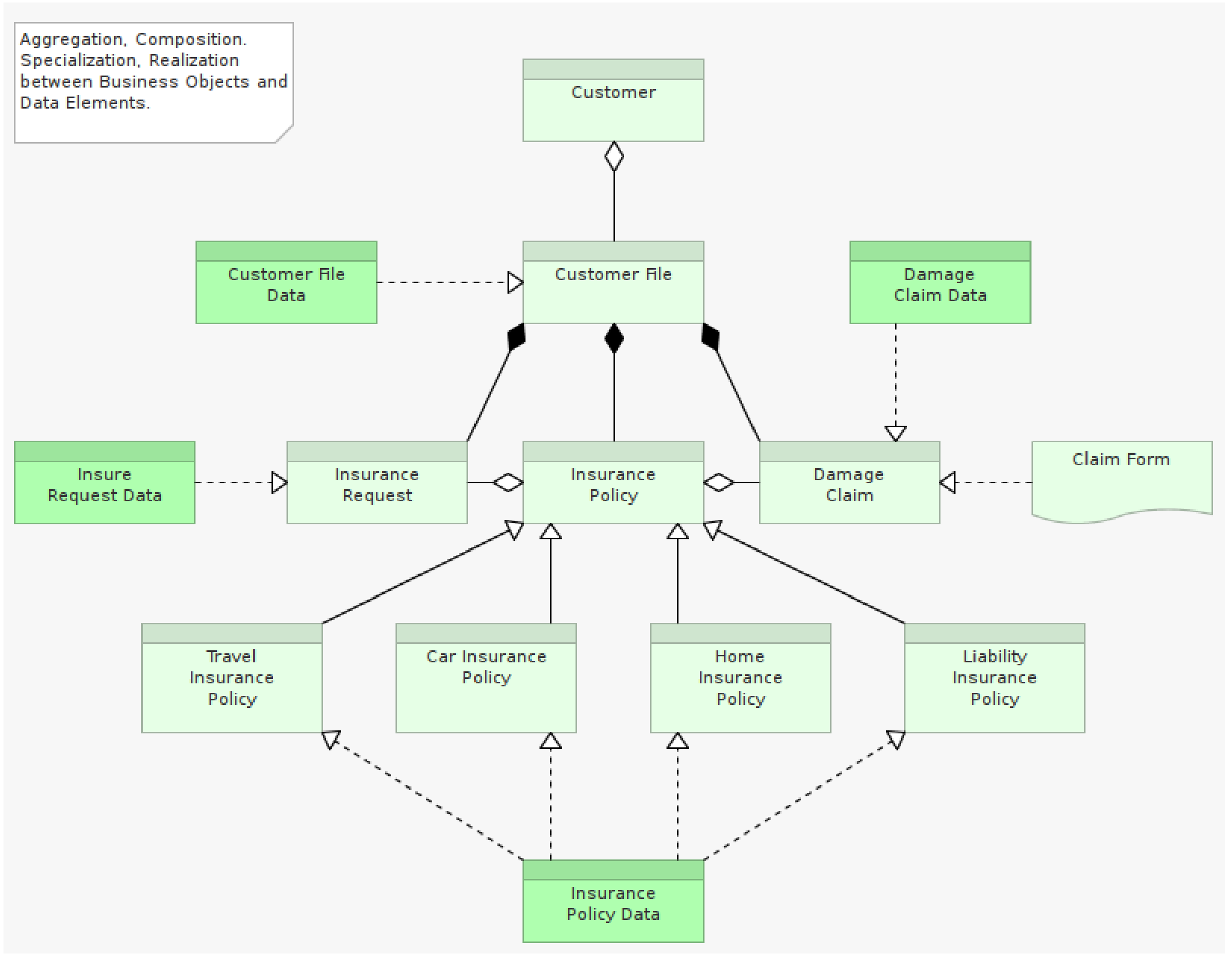
The purpose of this View-Type is to:

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

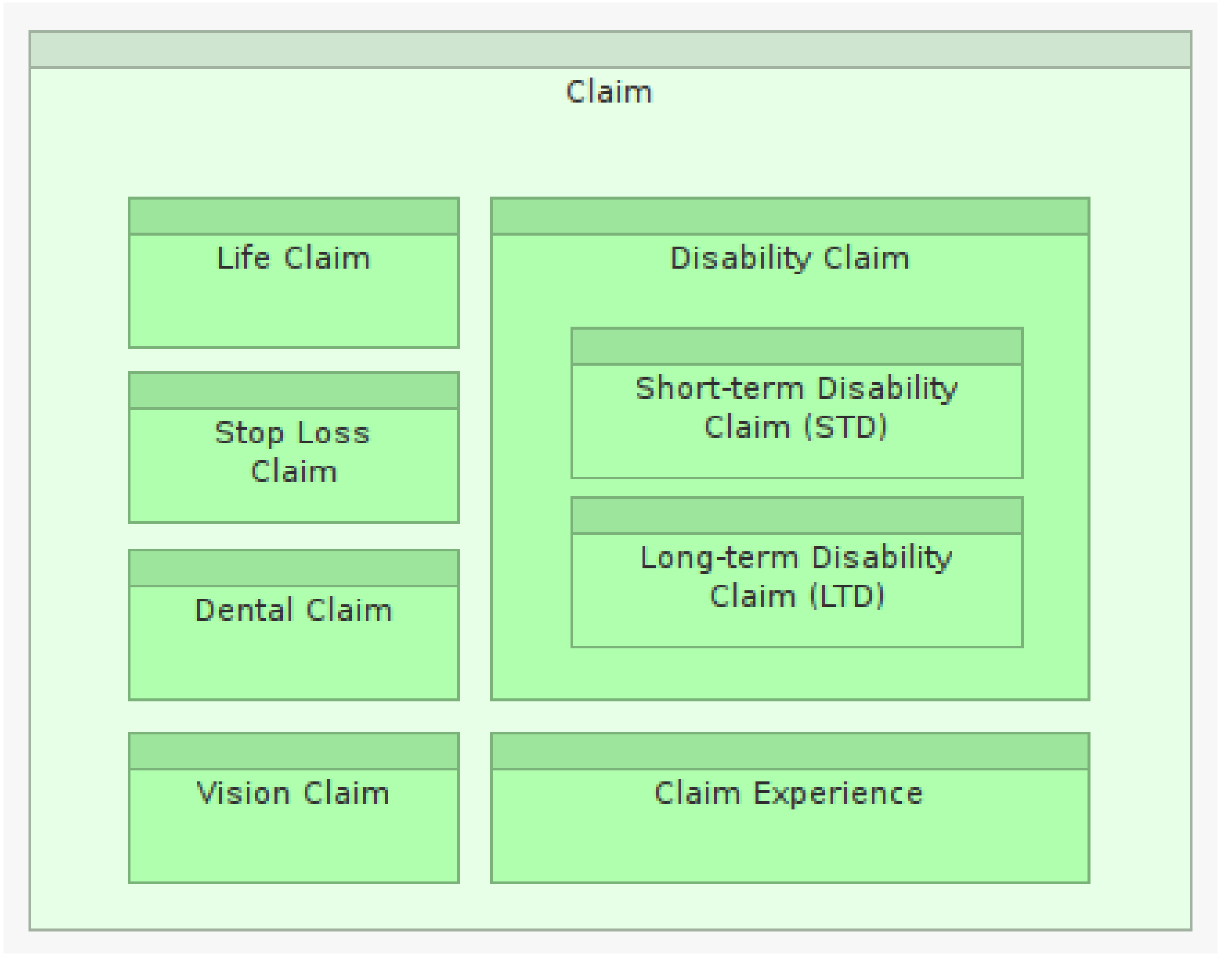
- show the data exchanged between functional elements

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

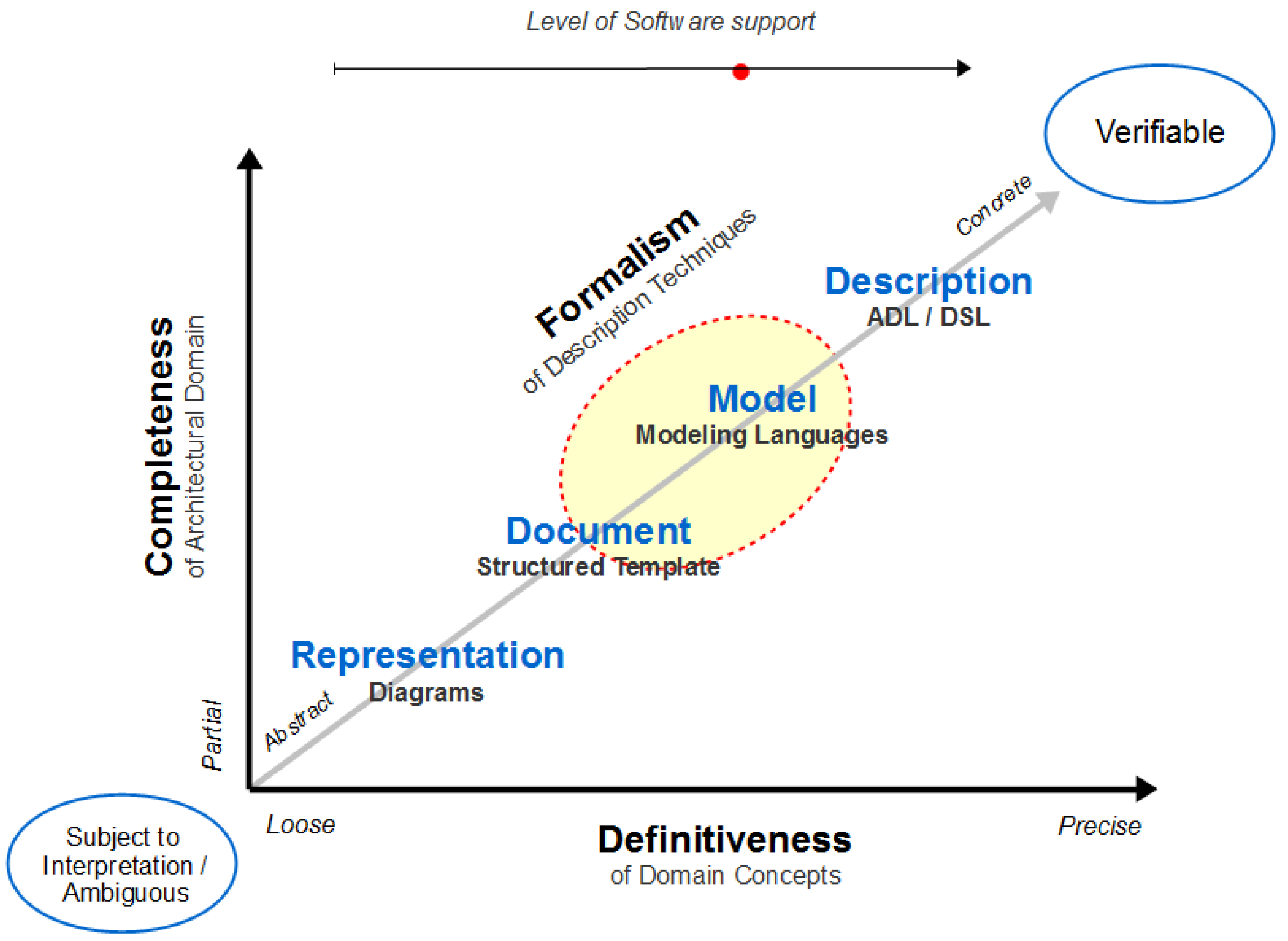
Example: Data Structure modeling



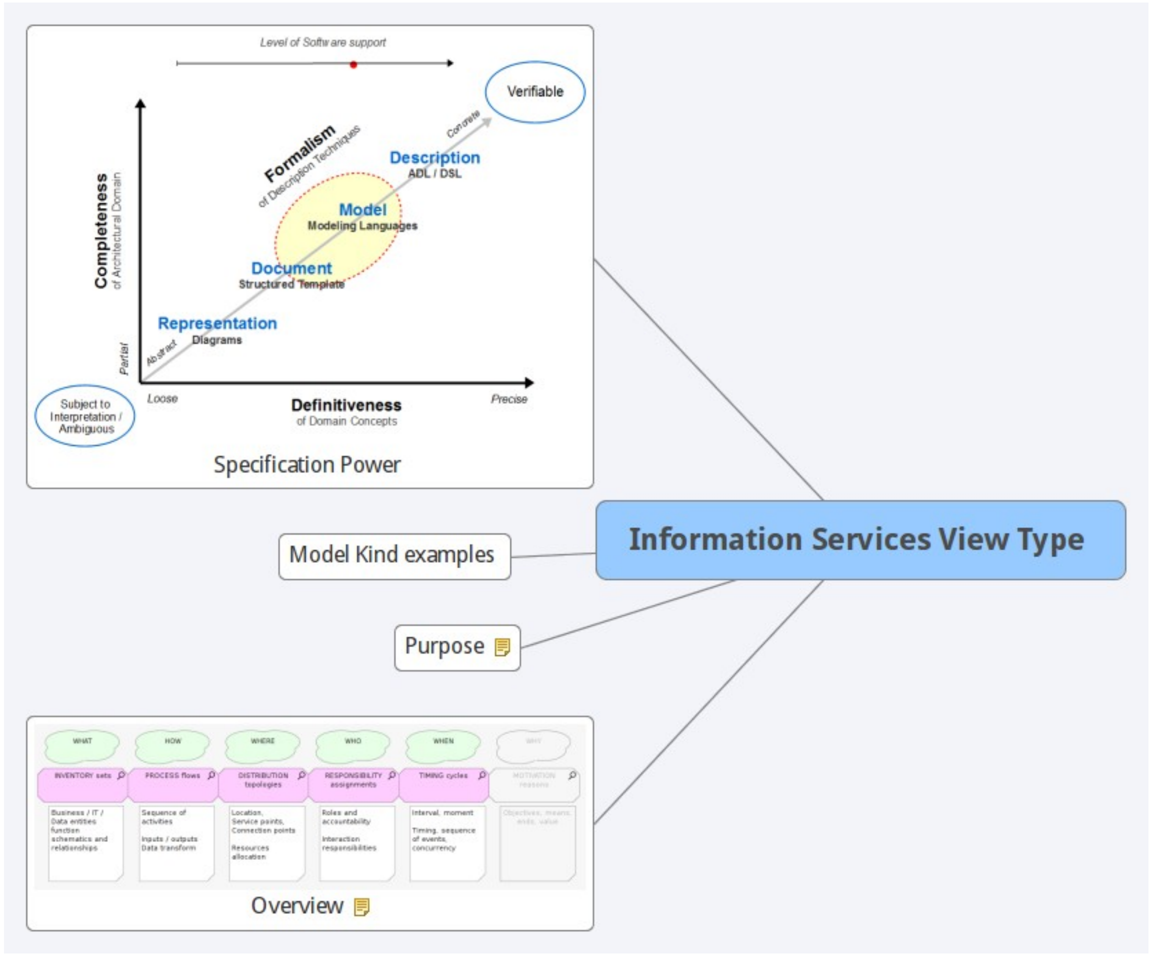
Example: Information Asset modeling



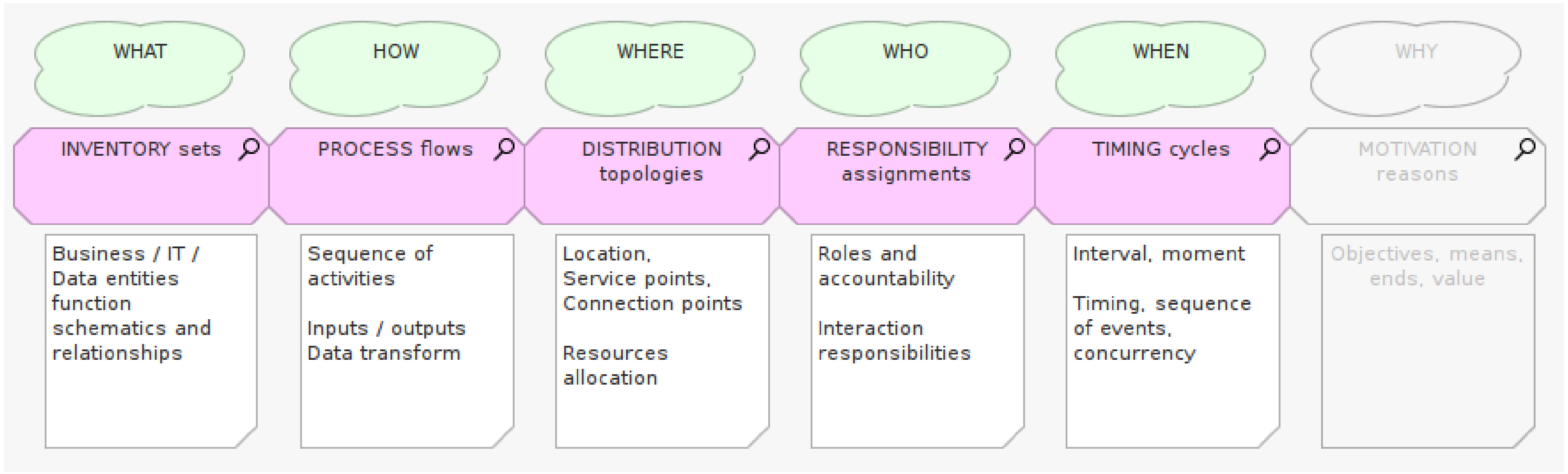
Data Structure Views Specification Power



Information Services View Type



Overview


WHAT
HOW
WHERE
WHO
WHEN
WHY
INVENTORY sets
PROCESS flows
DISTRIBUTION topologies
RESPONSIBILITY assignments
TIMING cycles
MOTIVATION reasons
Business / IT / Data entities function schematics and relationships
Sequence of activities
Inputs / outputs Data transform
Location, Service points, Connection points
Resources allocation
Roles and accountability
Interaction responsibilities
Interval, moment
Timing, sequence of events, concurrency
Objectives, means, ends, value

Overview

The "Information Services 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/how, where and what data is: (1.) collected (2.) transformed, and (3.) delivered (and optionally who interacts with it).

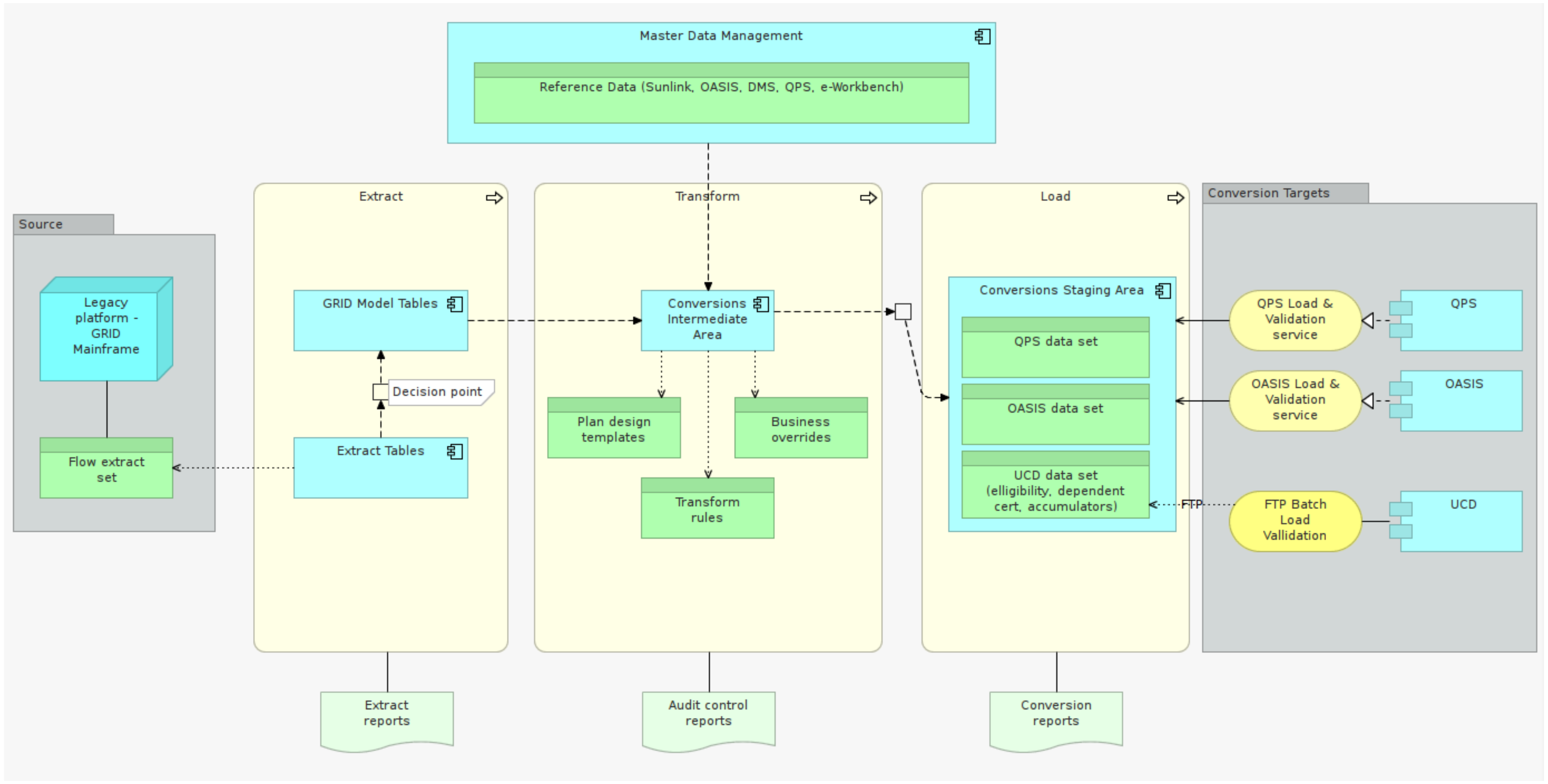
In its most detailed representations, it answers important questions around information format, timeliness/latency, and transactional integrity transiting between application components.

Purpose

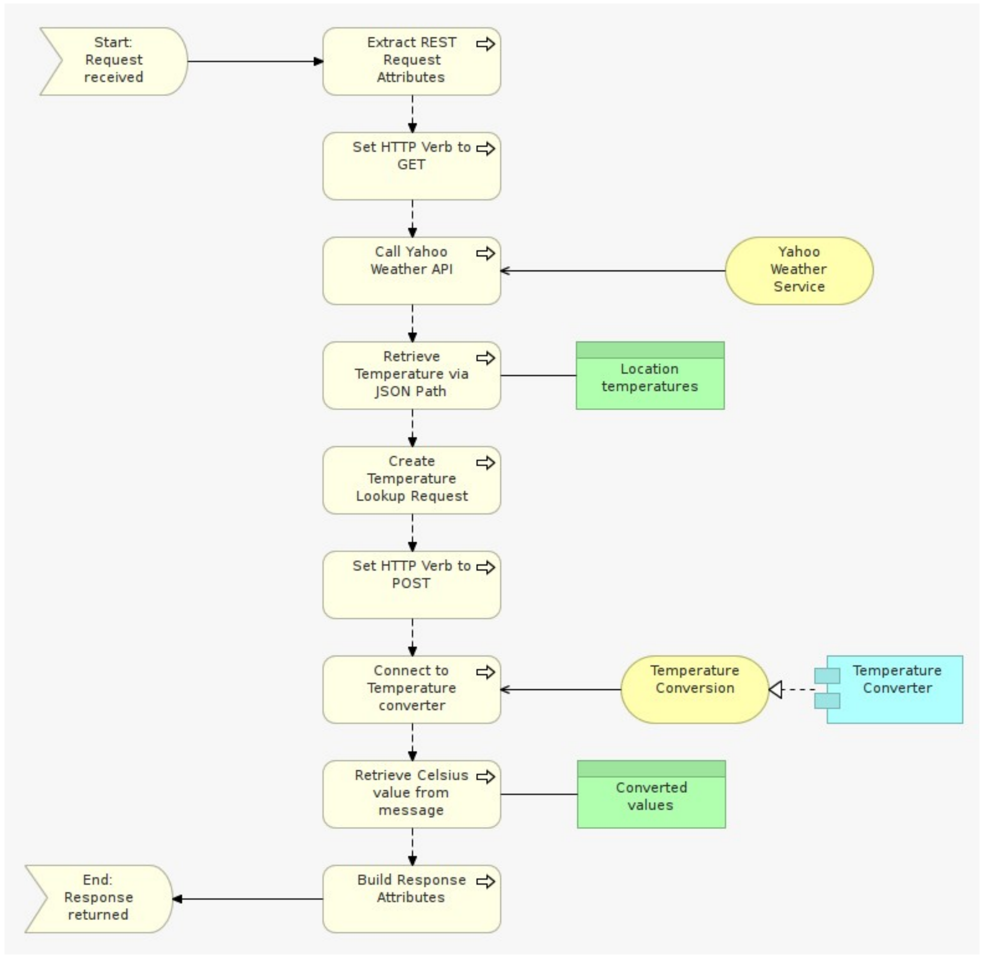
The purpose of this View-Type is to:

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

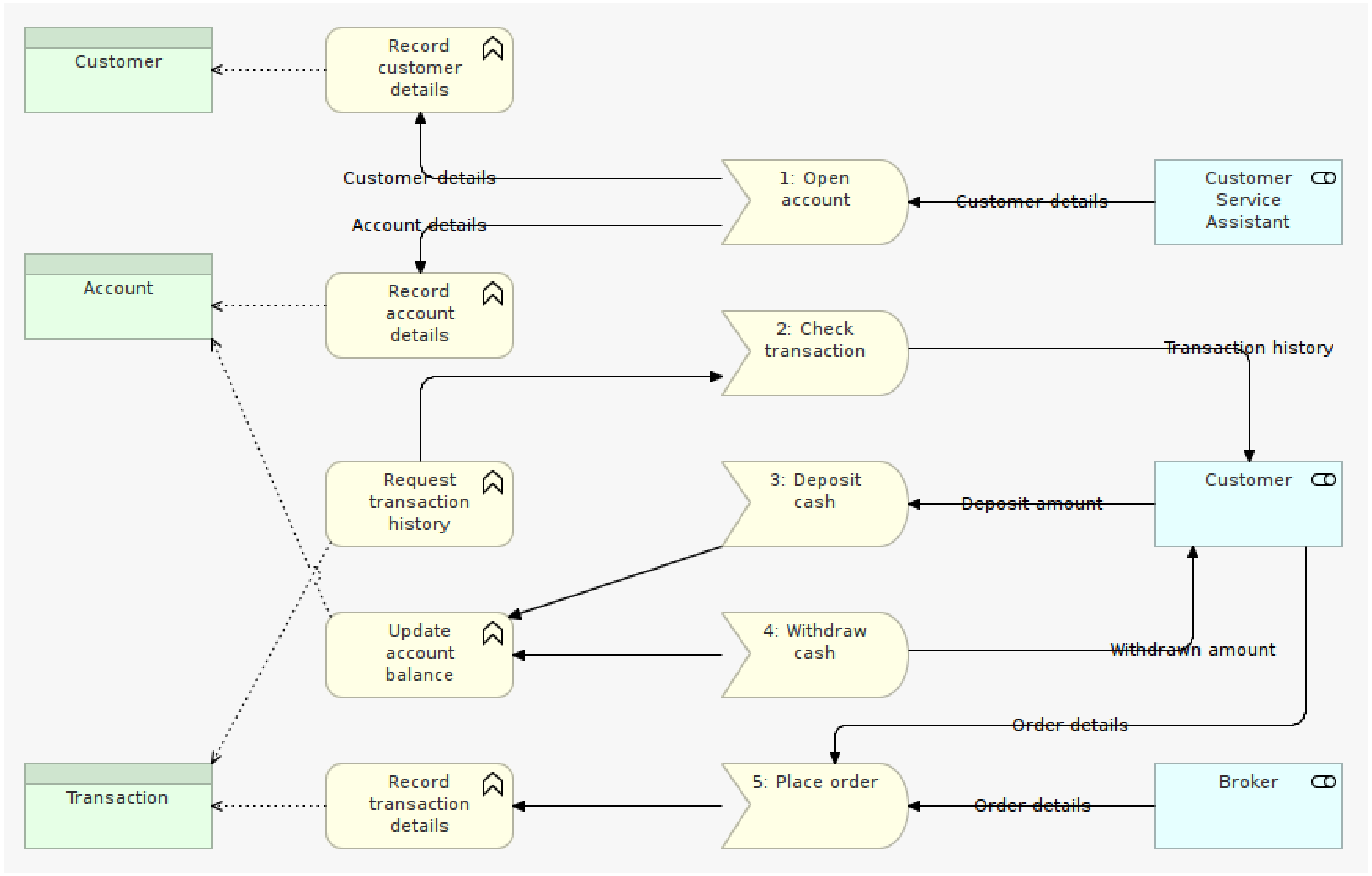
Example: Data Flow modeling (ETL orchestration)



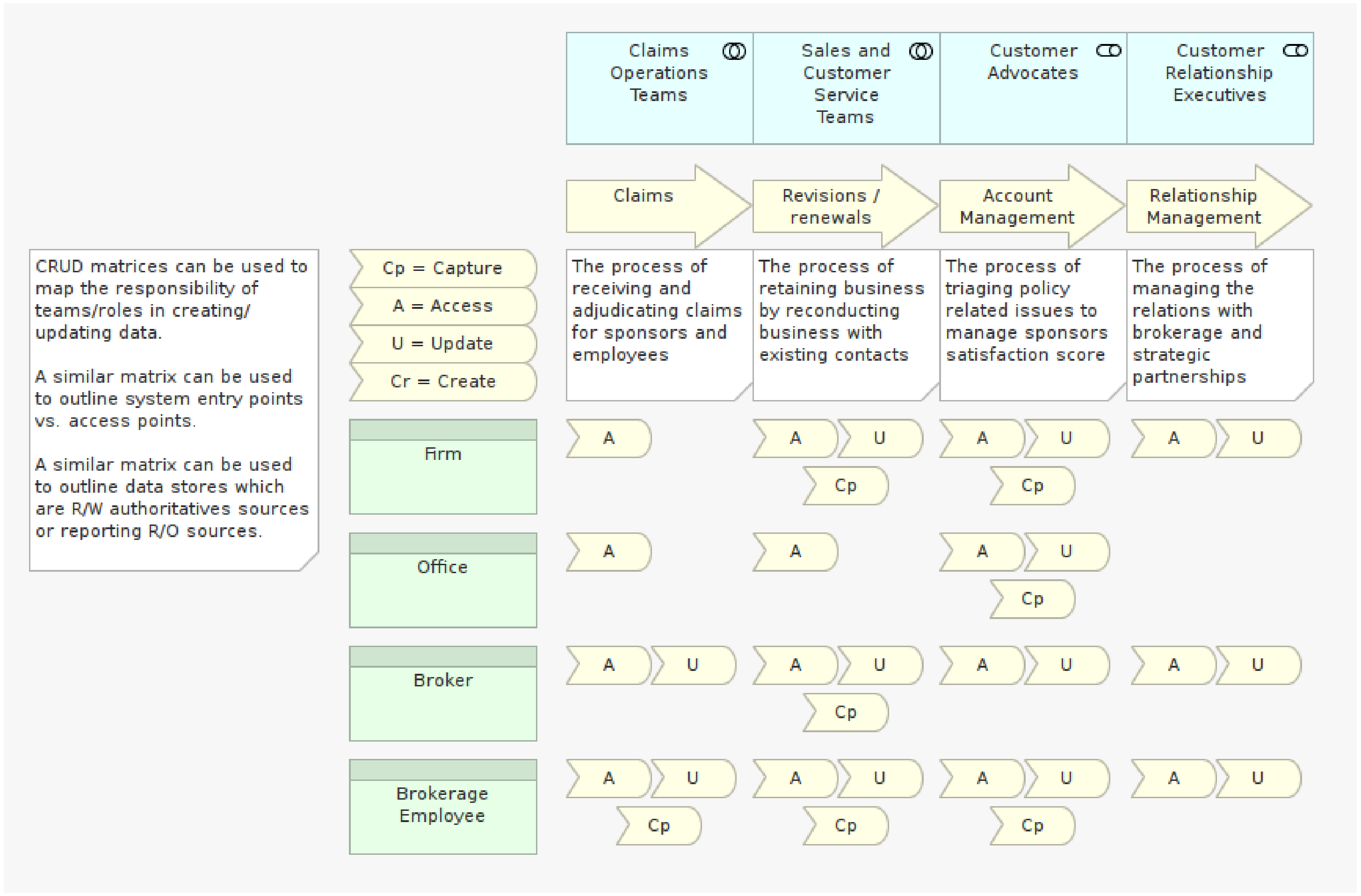
Example: Data Flow modeling (API orchestration)



Example: Data Flow modeling (DFD)



Example: Data Life-cycle (TOGAF Matrix)



CRUD matrices can be used to map the responsibility of teams/roles in creating/ updating data.

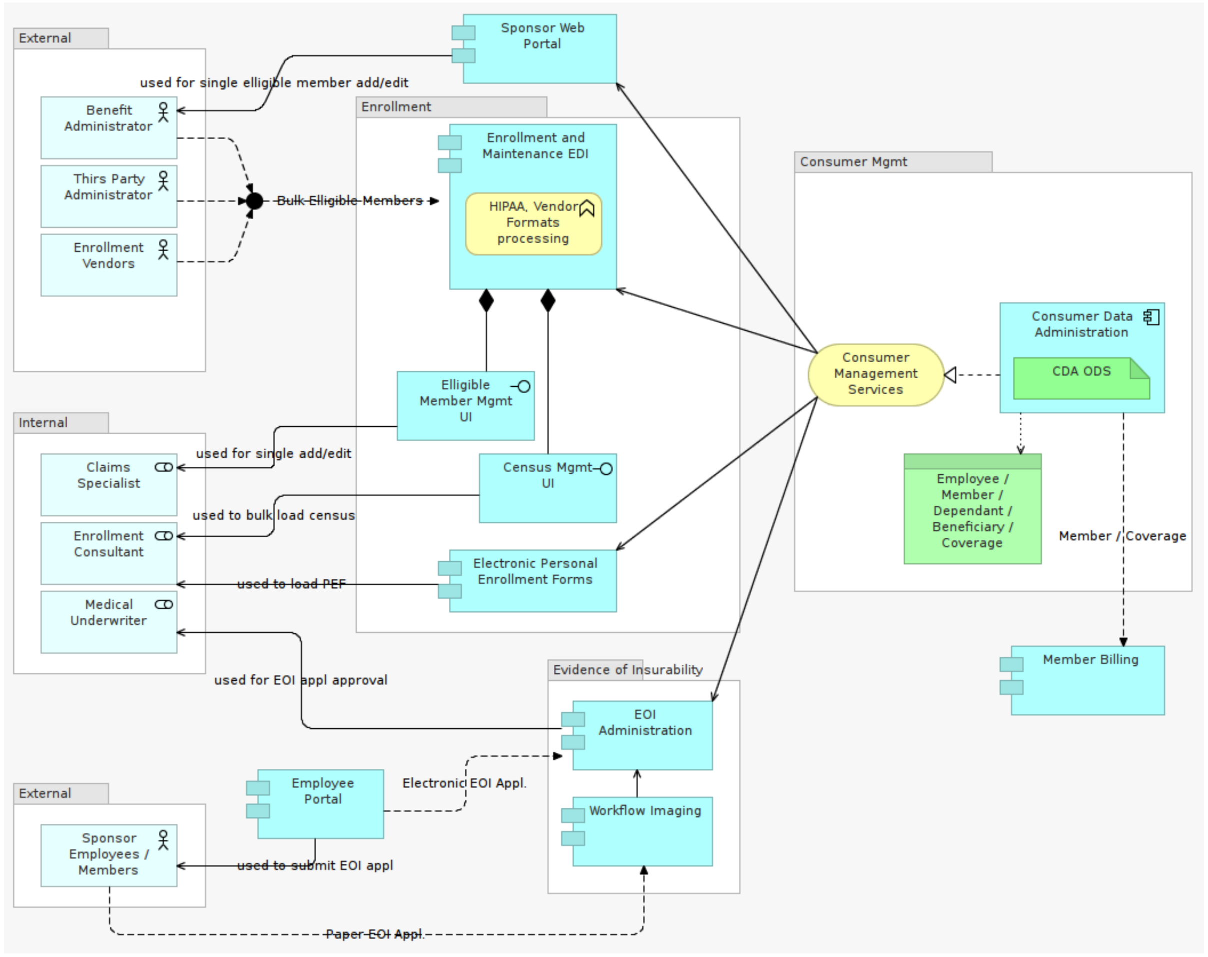
A similar matrix can be used to outline system entry points vs. access points.

A similar matrix can be used to outline data stores which are R/W authoritatives sources or reporting R/O sources.

Cp = Capture
A = Access
U = Update
Cr = Create

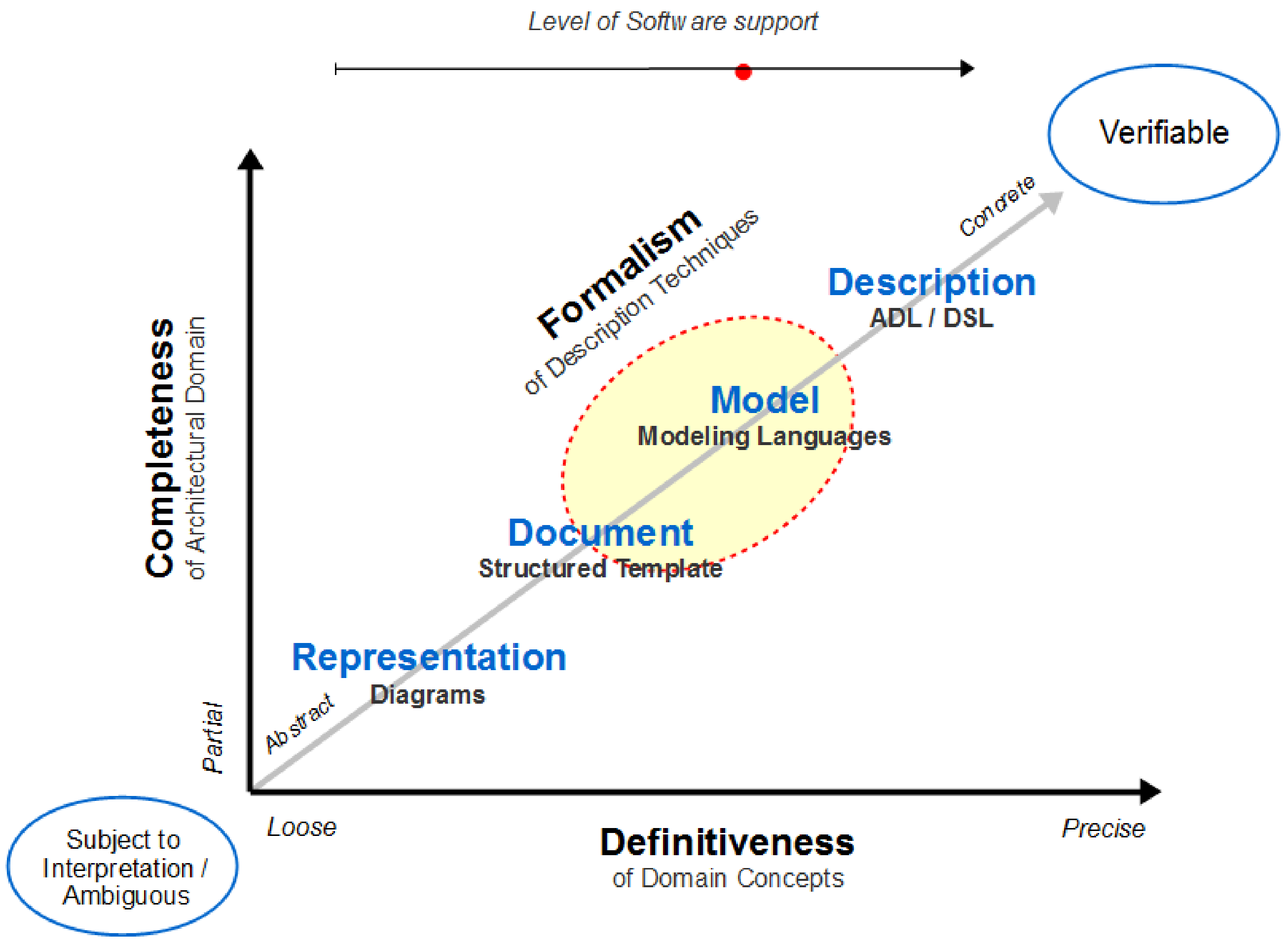
	Claims Operations Teams	Sales and Customer Service Teams	Customer Advocates	Customer Relationship Executives
	Claims	Revisions / renewals	Account Management	Relationship Management
	The process of receiving and adjudicating claims for sponsors and employees	The process of retaining business by reconducting business with existing contacts	The process of triaging policy related issues to manage sponsors satisfaction score	The process of managing the relations with brokerage and strategic partnerships
Firm	A	A U Cp	A U Cp	A U
Office	A	A	A U Cp	
Broker	A U	A U Cp	A U	A U
Brokerage Employee	A U Cp	A U Cp	A U Cp	A U

Example: Data Flow modeling

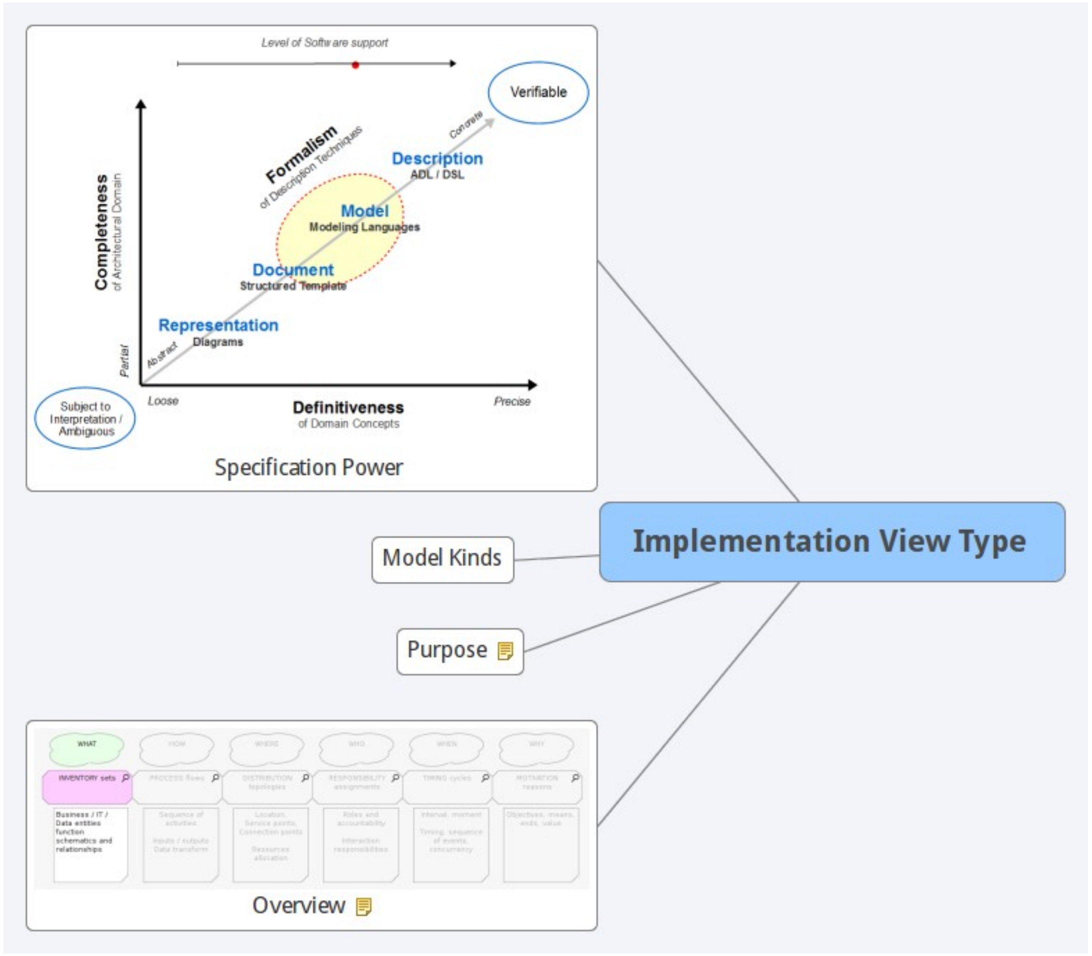


Specification Power

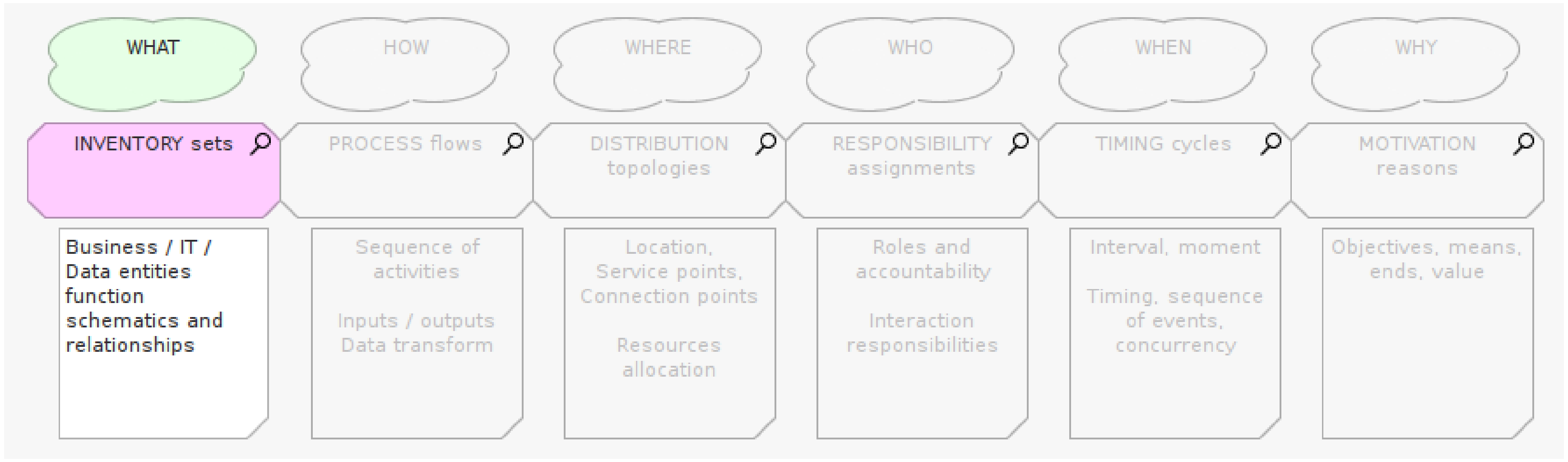
-



Implementation View Type



Overview


WHAT
HOW
WHERE
WHO
WHEN
WHY
INVENTORY sets
PROCESS flows
DISTRIBUTION topologies
RESPONSIBILITY assignments
TIMING cycles
MOTIVATION reasons
Business / IT / Data entities function schematics and relationships
Sequence of activities
Inputs / outputs Data transform
Location, Service points, Connection points
Resources allocation
Roles and accountability
Interaction responsibilities
Interval, moment
Timing, sequence of events, concurrency
Objectives, means, ends, value

Overview

The "Implementation View Type" describes all required component dependencies to successful build the application.

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

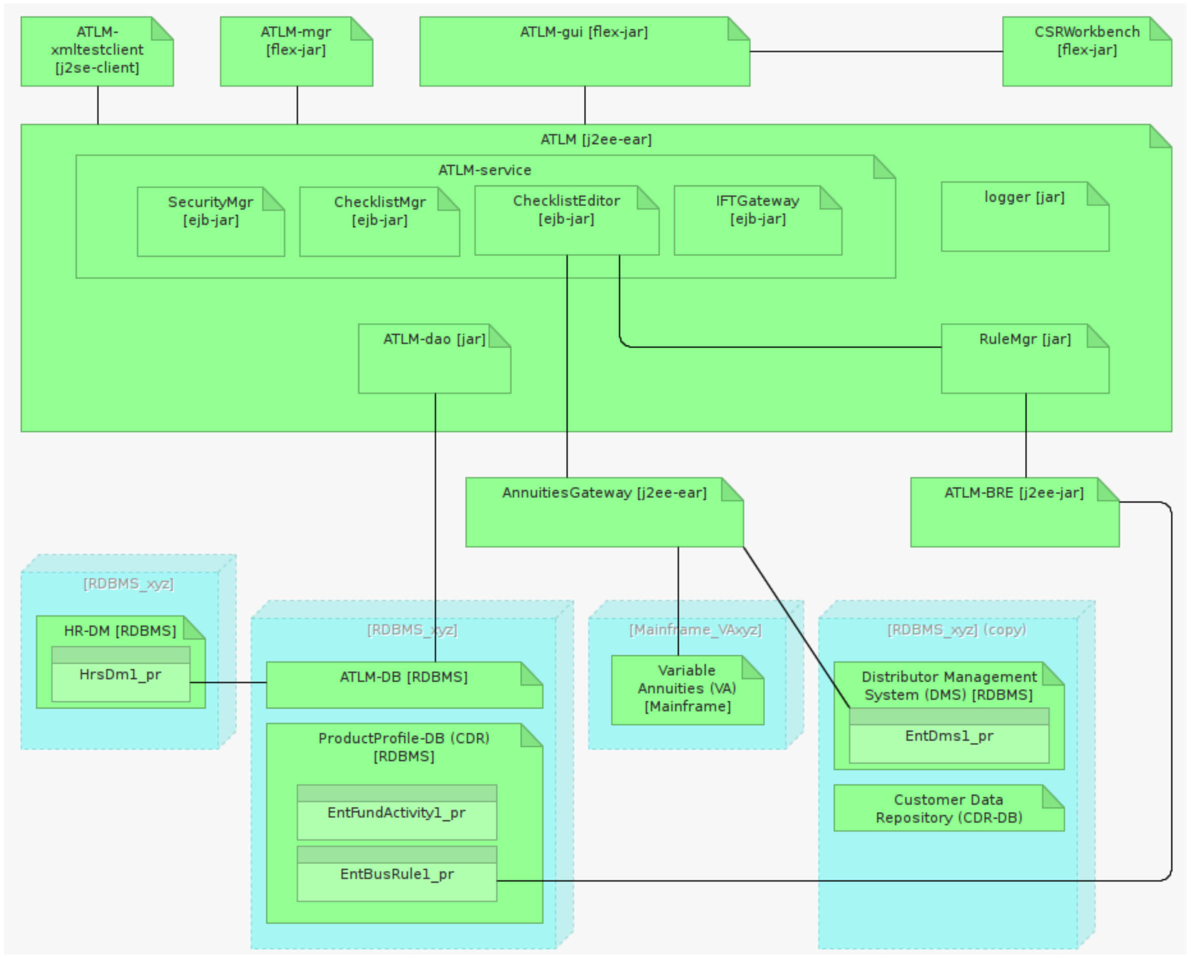
Purpose

The purpose of the View Type is to:

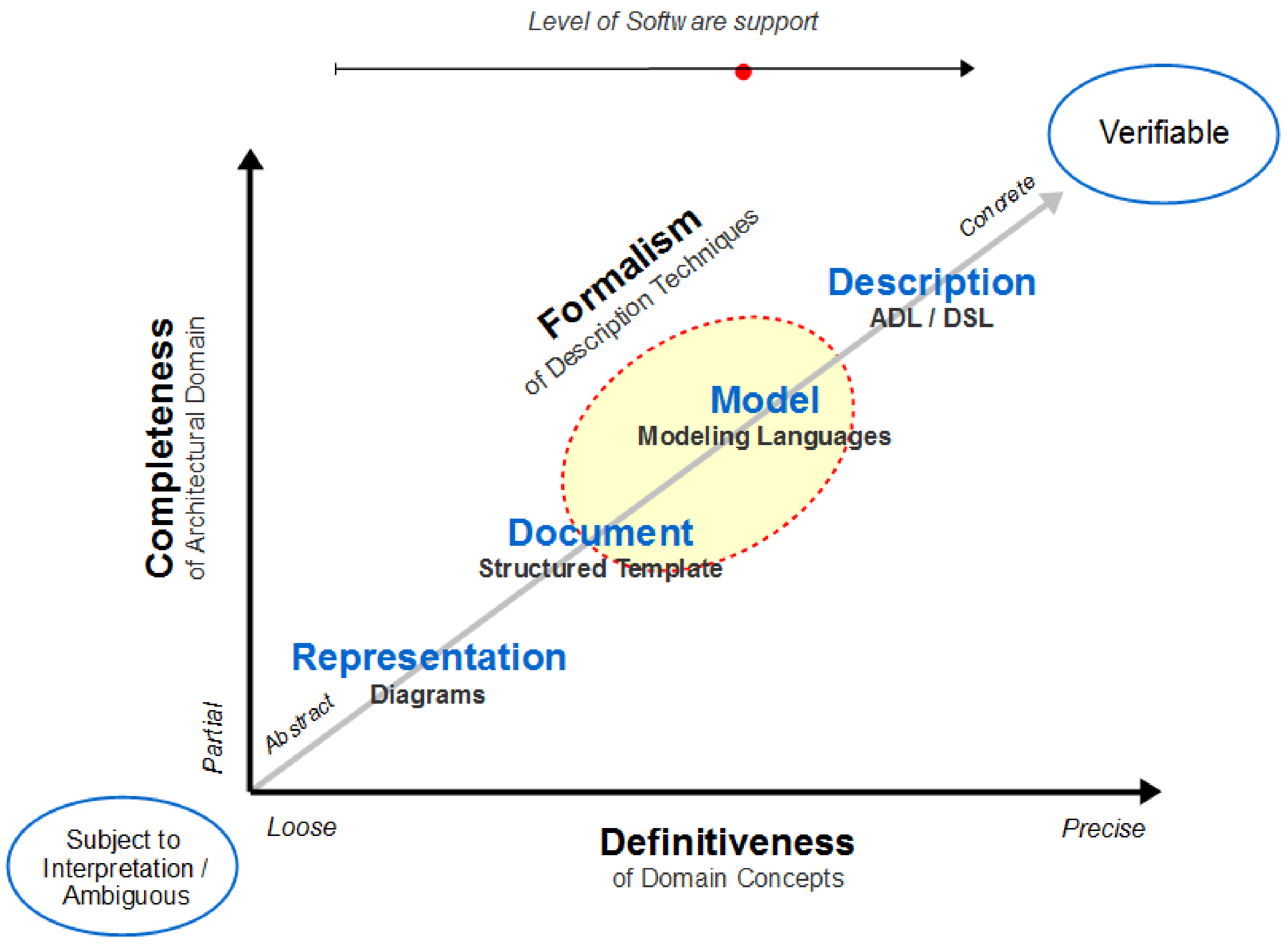
- ensure that the application is always release-ready

- so to defer the timing of when to push a solution into production to business stakeholders.

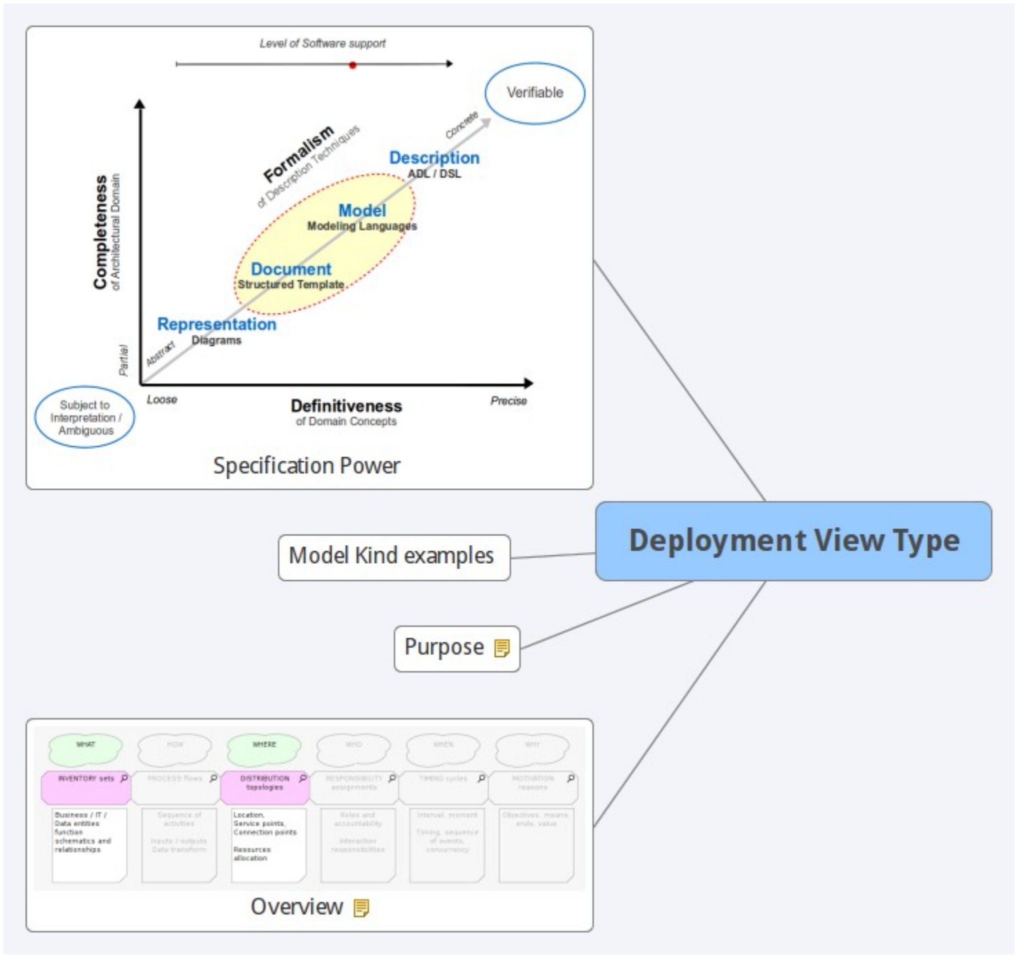
Example: Module dependency modeling



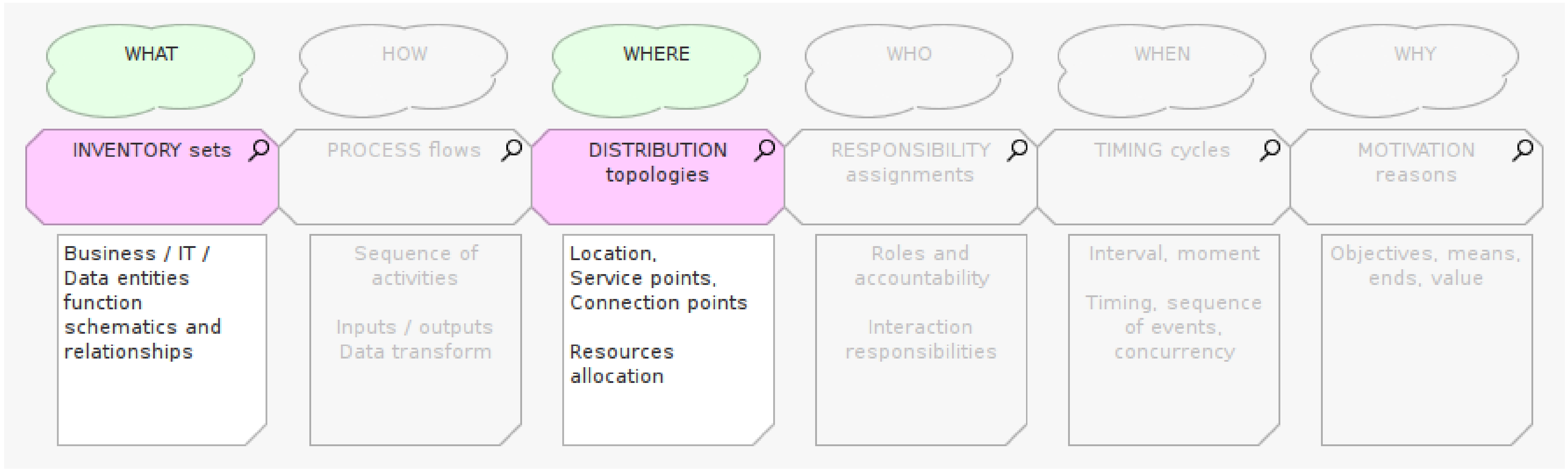
Specification Power



Deployment View Type



Overview


WHAT
HOW
WHERE
WHO
WHEN
WHY
INVENTORY sets
PROCESS flows
DISTRIBUTION topologies
RESPONSIBILITY assignments
TIMING cycles
MOTIVATION reasons
Business / IT / Data entities function schematics and relationships
Sequence of activities
Inputs / outputs Data transform
Location, Service points, Connection points
Resources allocation
Roles and accountability
Interaction responsibilities
Interval, moment
Timing, sequence of events, concurrency
Objectives, means, ends, value

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:

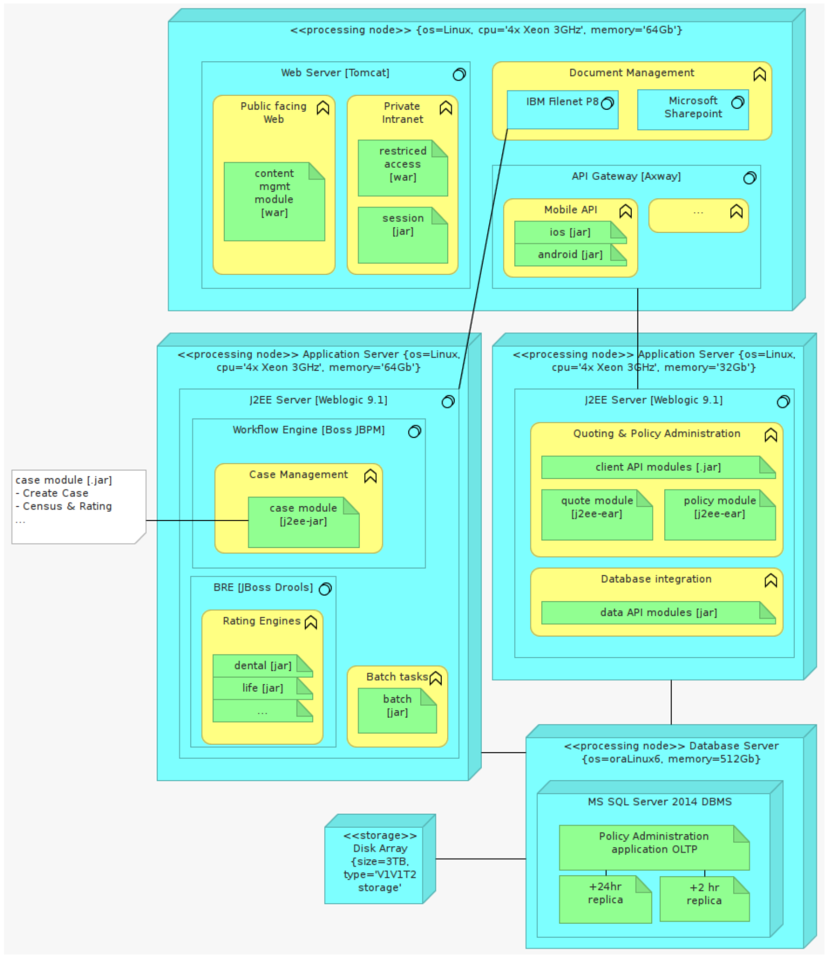
- the dependencies with the Solution runtime environment,

- the hardware infrastructure required,

- the technical requirements for each, and

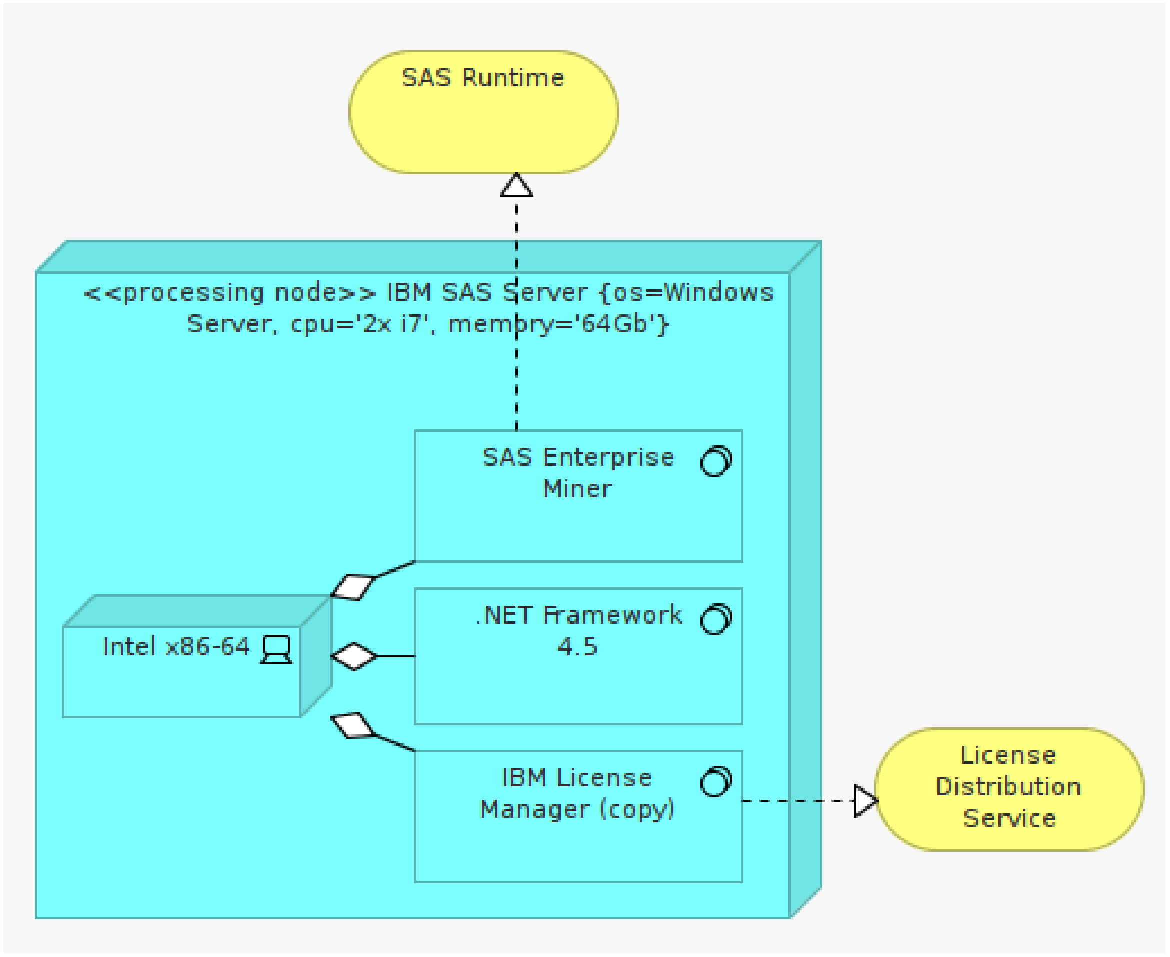
- the mapping of the software elements to the runtime environment that will execute them (release and configuration management).

Example: Deployment modeling (Application allocation)

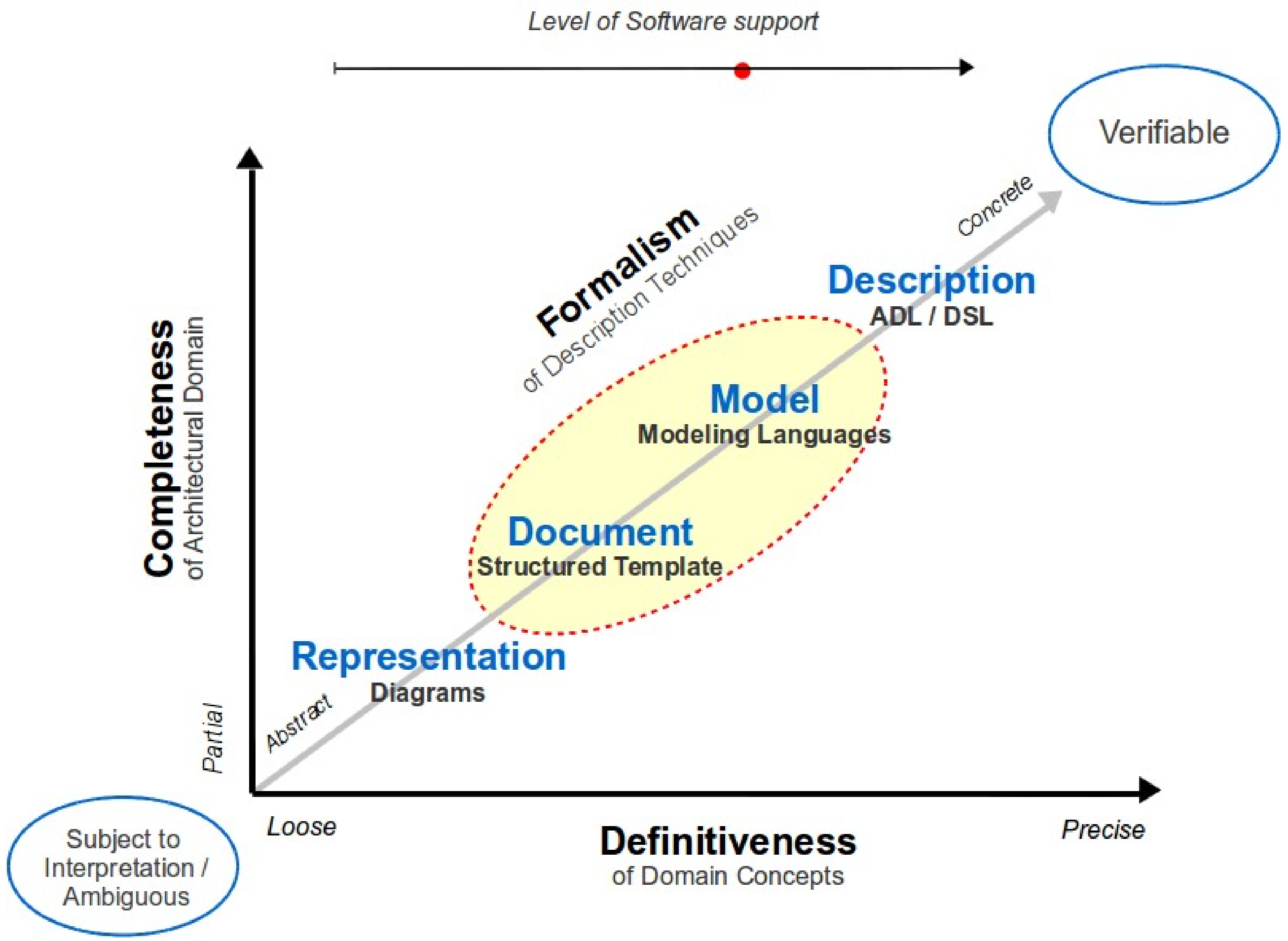


<<processing node>> {os=Linux, cpu='4x Xeon 3GHz', memory='64Gb'}
Web Server [Tomcat]
Public facing Web
content mgmt module [war]
Private Intranet
restricted access [war]
session [jar]
Document Management
IBM Filenet P8
Microsoft Sharepoint
API Gateway [Axway]
Mobile API
ios [jar]
android [jar]
...
<<processing node>> Application Server {os=Linux, cpu='4x Xeon 3GHz', memory='64Gb'}
J2EE Server [Weblogic 9.1]
Workflow Engine [Boss JBPM]
Case Management
case module [j2ee-jar]
case module [.jar]
- Create Case
- Census & Rating
...
BRE [JBoss Drools]
Rating Engines
dental [jar]
life [jar]
...
Batch tasks
batch [jar]
<<processing node>> Application Server {os=Linux, cpu='4x Xeon 3GHz', memory='32Gb'}
J2EE Server [Weblogic 9.1]
Quoting & Policy Administration
client API modules [.jar]
quote module [j2ee-ear]
policy module [j2ee-ear]
Database integration
data API modules [jar]
<<processing node>> Database Server {os=oraLinux6, memory=512Gb}
MS SQL Server 2014 DBMS
Policy Administration application OLTP
+24hr replica
+2 hr replica
<<storage>> Disk Array {size=3TB, type='V1V1T2 storage'

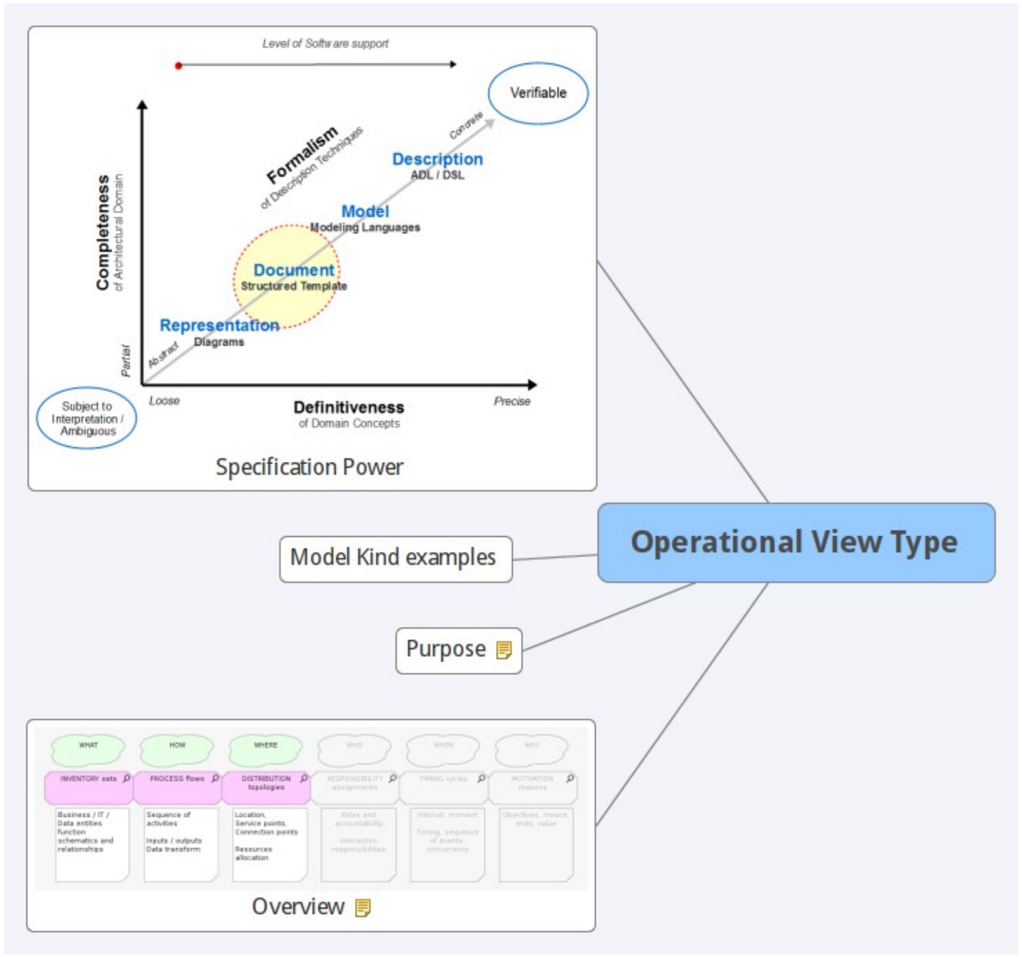
Example: Deployment modeling (System software allocation)


SAS Runtime
<<processing node>> IBM SAS Server {os=Windows Server, cpu='2x i7', memory='64Gb'}
SAS Enterprise Miner
Intel x86-64
.NET Framework 4.5
IBM License Manager (copy)
License Distribution Service

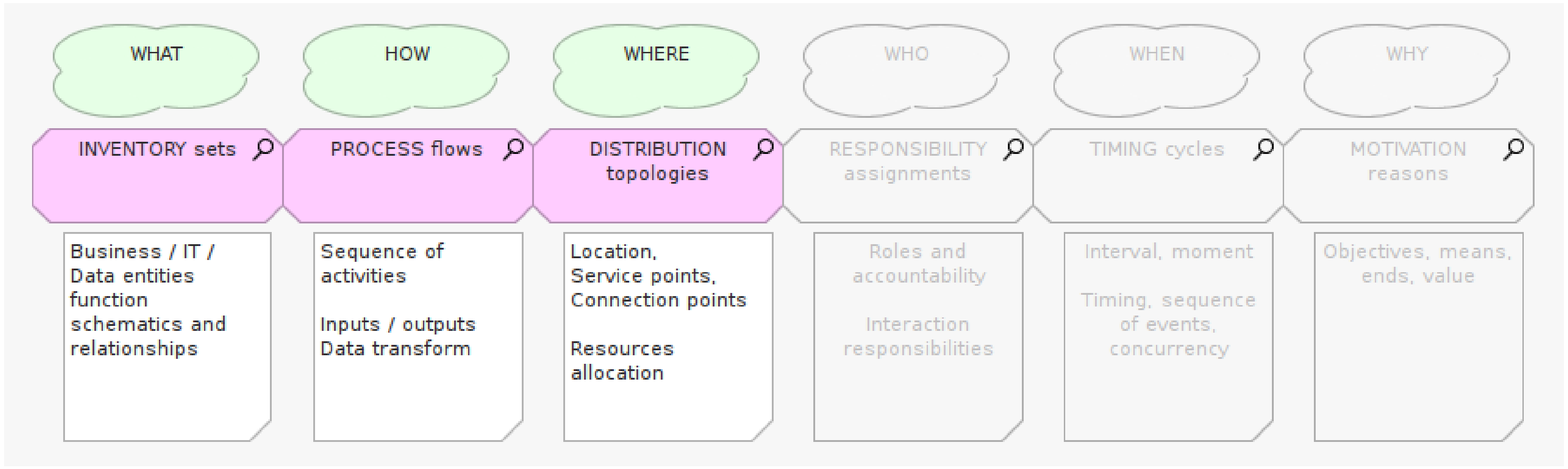
Specification Power



Operational View Type



Overview


WHAT
HOW
WHERE
WHO
WHEN
WHY
INVENTORY sets
PROCESS flows
DISTRIBUTION topologies
RESPONSIBILITY assignments
TIMING cycles
MOTIVATION reasons
Business / IT / Data entities function schematics and relationships
Sequence of activities
Inputs / outputs Data transform
Location, Service points, Connection points
Resources allocation
Roles and accountability
Interaction responsibilities
Interval, moment
Timing, sequence of events, concurrency
Objectives, means, ends, value

Overview

The "Operational View Type" describes, for each IT functional area, the infrastructure strategy (nodes virtualization, load balancing, networks isolation) supporting the operational concerns of the Architecture.

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.

Application architectures often assume that a solution can be allocated onto one server, one runtime

...until non-functional requirements like security, performance or scalability start posing difficult questions about distribution of system resources.

Purpose

The purpose of the Operational View Type is to:

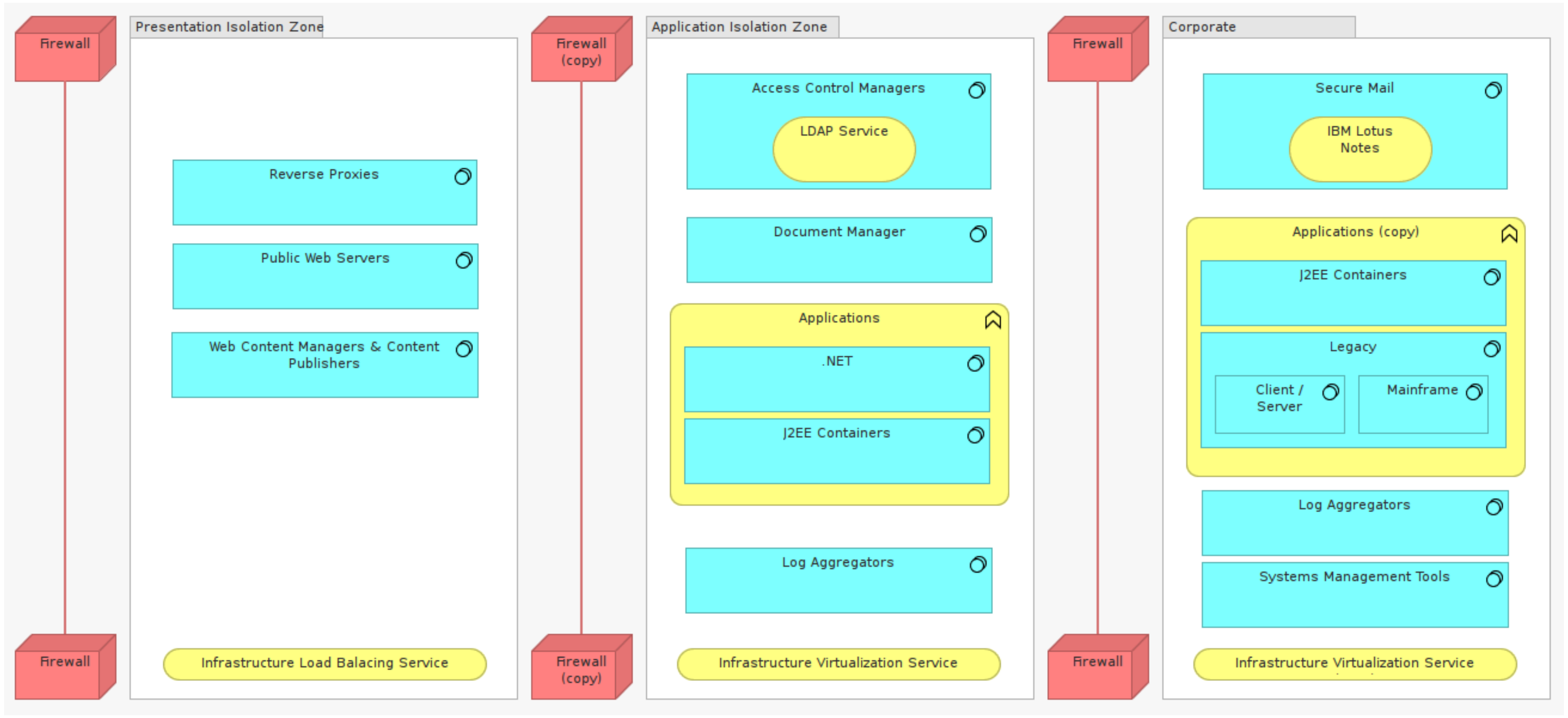
- define the characteristics by which Architecture will be operated, administered & supported (once deployed in its production environment).

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

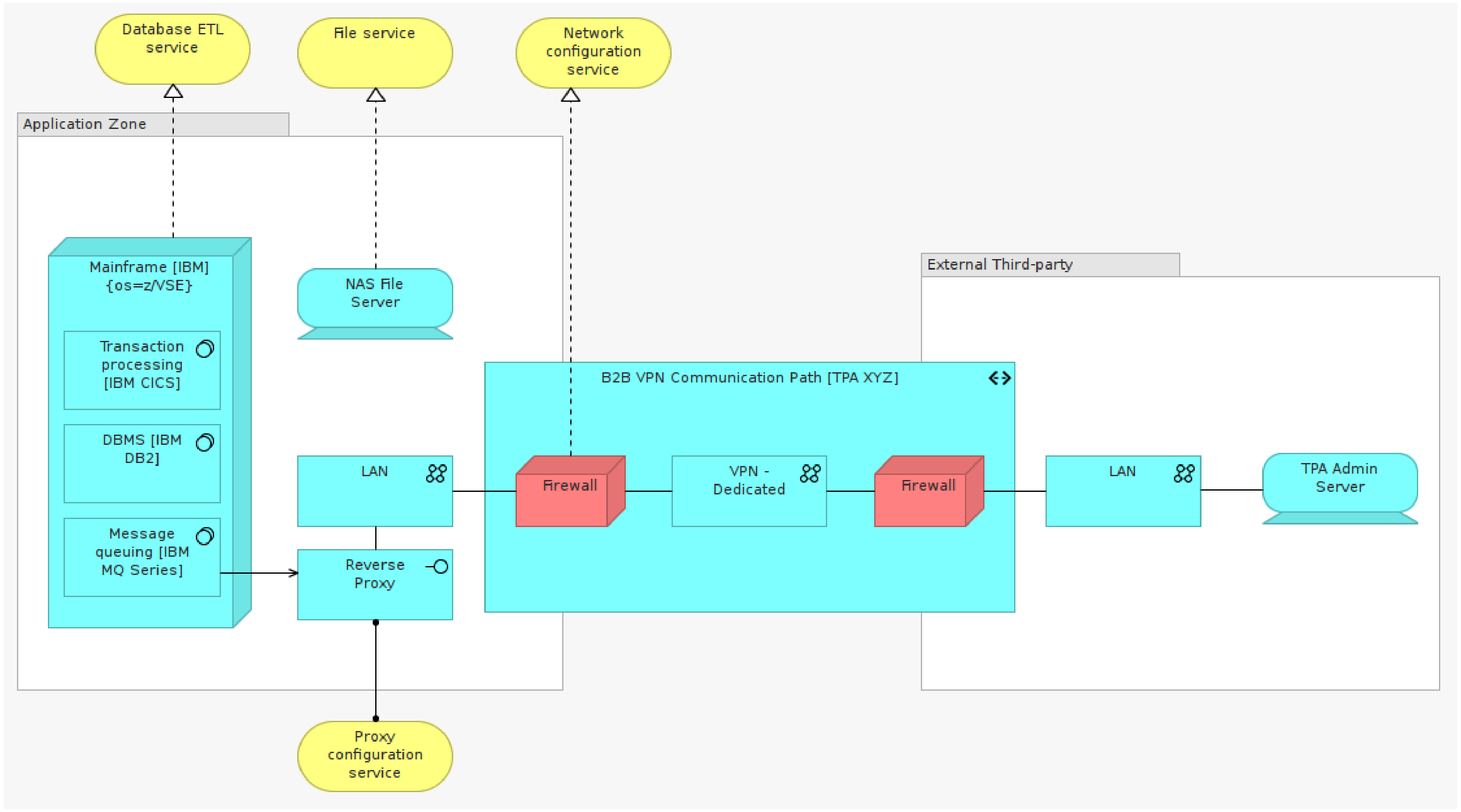
- specify the means of configuration and tuning of the solution architecture, without having to re-deploy the application.

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

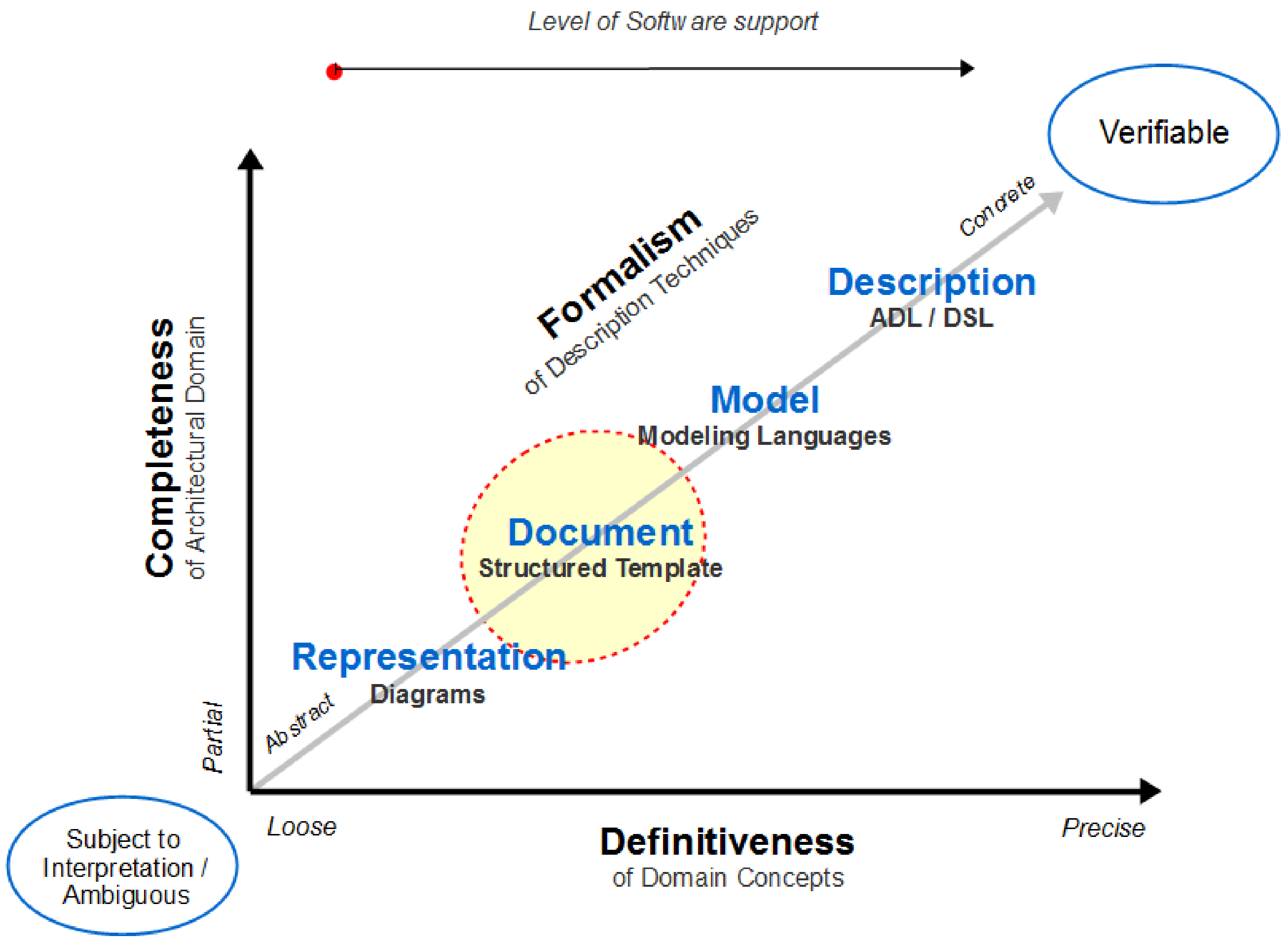
Example: Infrastructure modeling (Logical Isolation Zones)



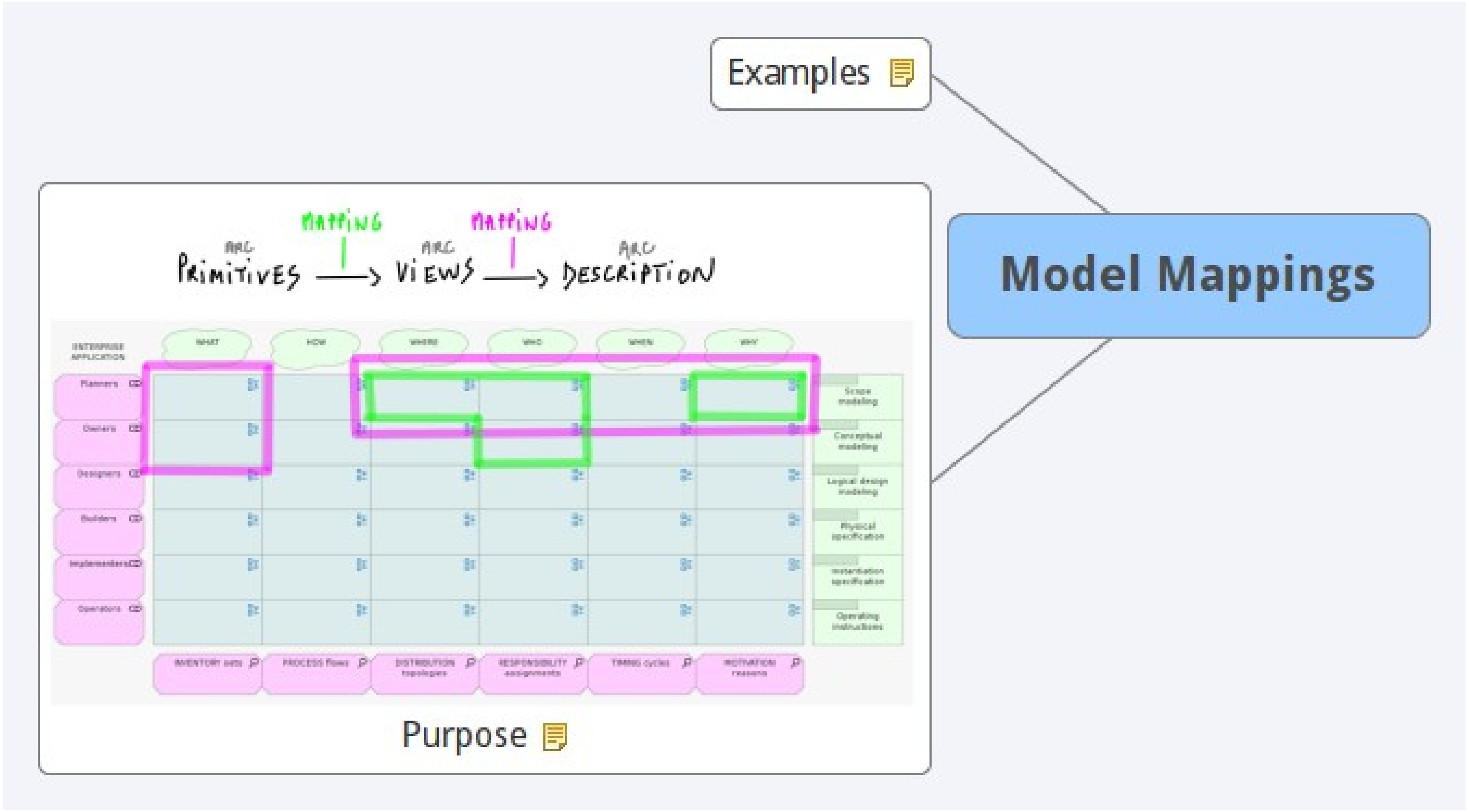
Example: Infrastructure modeling (Network Configuration)


Database ETL service
File service
Network configuration service
Application Zone
Mainframe [IBM] {os=z/VSE}
Transaction processing [IBM CICS]
DBMS [IBM DB2]
Message queuing [IBM MQ Series]
NAS File Server
LAN
Reverse Proxy
B2B VPN Communication Path [TPA XYZ]
Firewall
VPN - Dedicated
Firewall
External Third-party
LAN
TPA Admin Server
Proxy configuration service

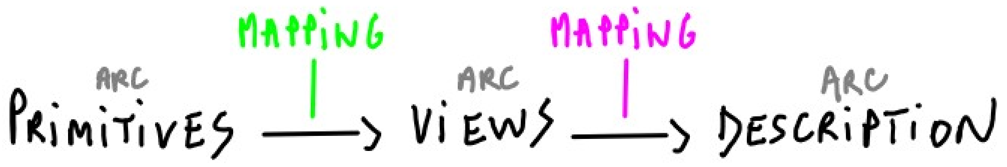
Specification Power

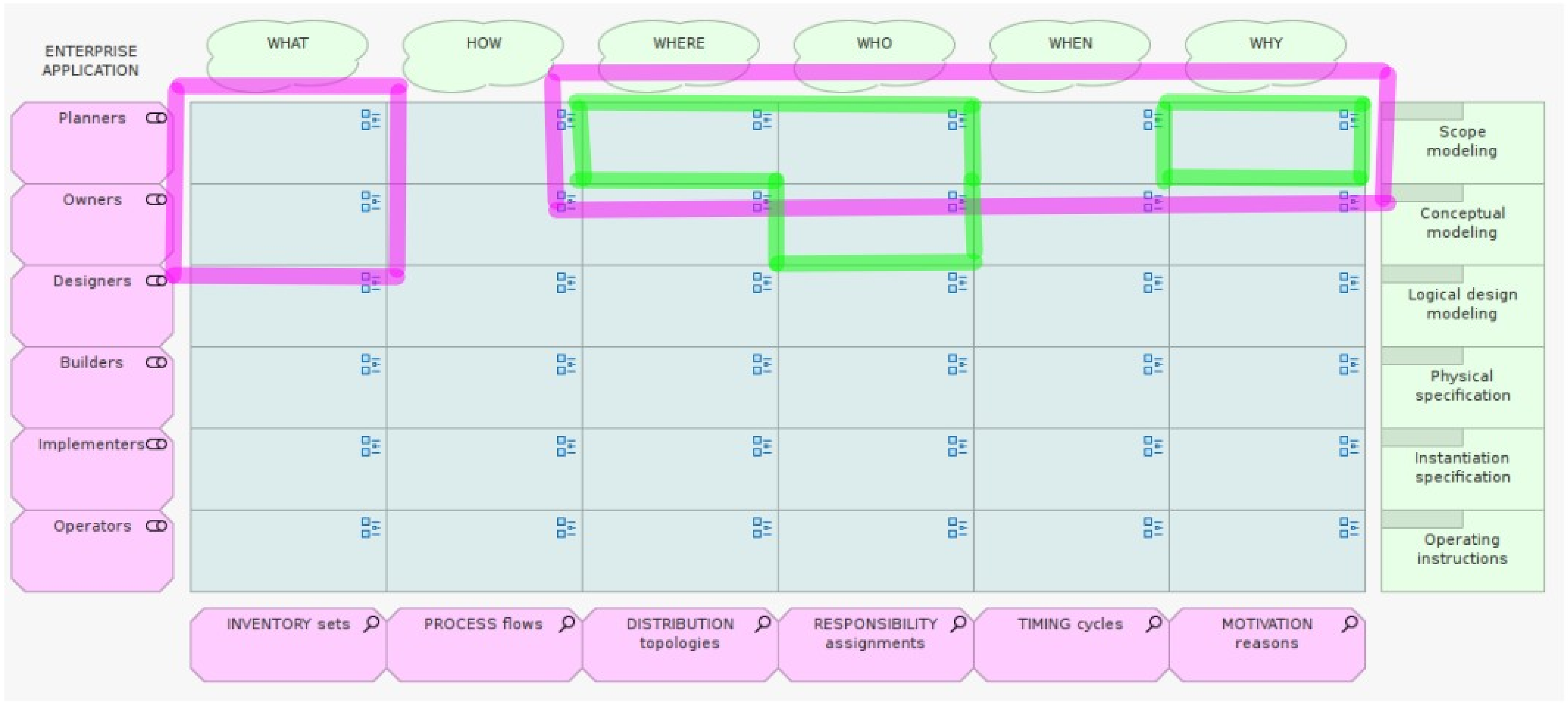


Model Mappings



Purpose


MAPPING
MAPPING
ARC
ARC
ARC
PRIMITIVES
VIEWS
DESCRIPTION


ENTERPRISE APPLICATION
WHAT
HOW
WHERE
WHO
WHEN
WHY
Planners
Owners
Designers
Builders
Implementers
Operators
Scope modeling
Conceptual modeling
Logical design modeling
Physical specification
Instantiation specification
Operating instructions
INVENTORY sets
PROCESS flows
DISTRIBUTION topologies
RESPONSIBILITY assignments
TIMING cycles
MOTIVATION reasons

Purpose

Mapping models helps to verify:

- The TRACEABILITY of the solution architecture, by navigating through the relationships of model elements.

- The CONSISTENCY of the solution architecture, by ensuring models are expressed at an equivalent level of detail so they can be mapped with one another.

The purpose of Mapping models (ex. Model Primitives, or Model Views) is to DETECT GAPS in the solution design.

- To understand what key aspects of the solution MUST be modeled: aiming to map architecture models help architects to iteratively decide what models are strictly required to define the design

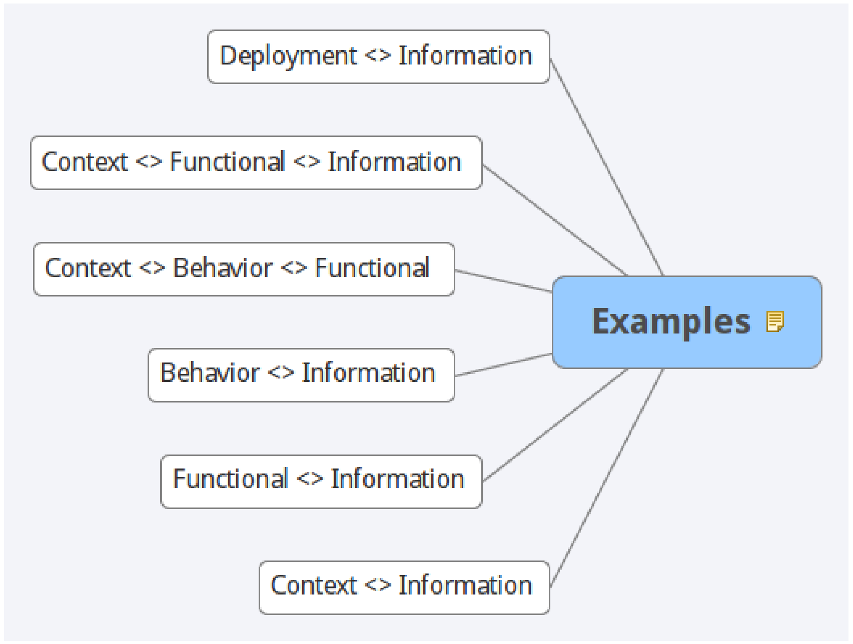
- To avoid inconsistencies: What doesn't "map" (i.e. doesn't have a correspondence or cannot be related to other modeling elements), is potentially pointing a gap in the design.

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

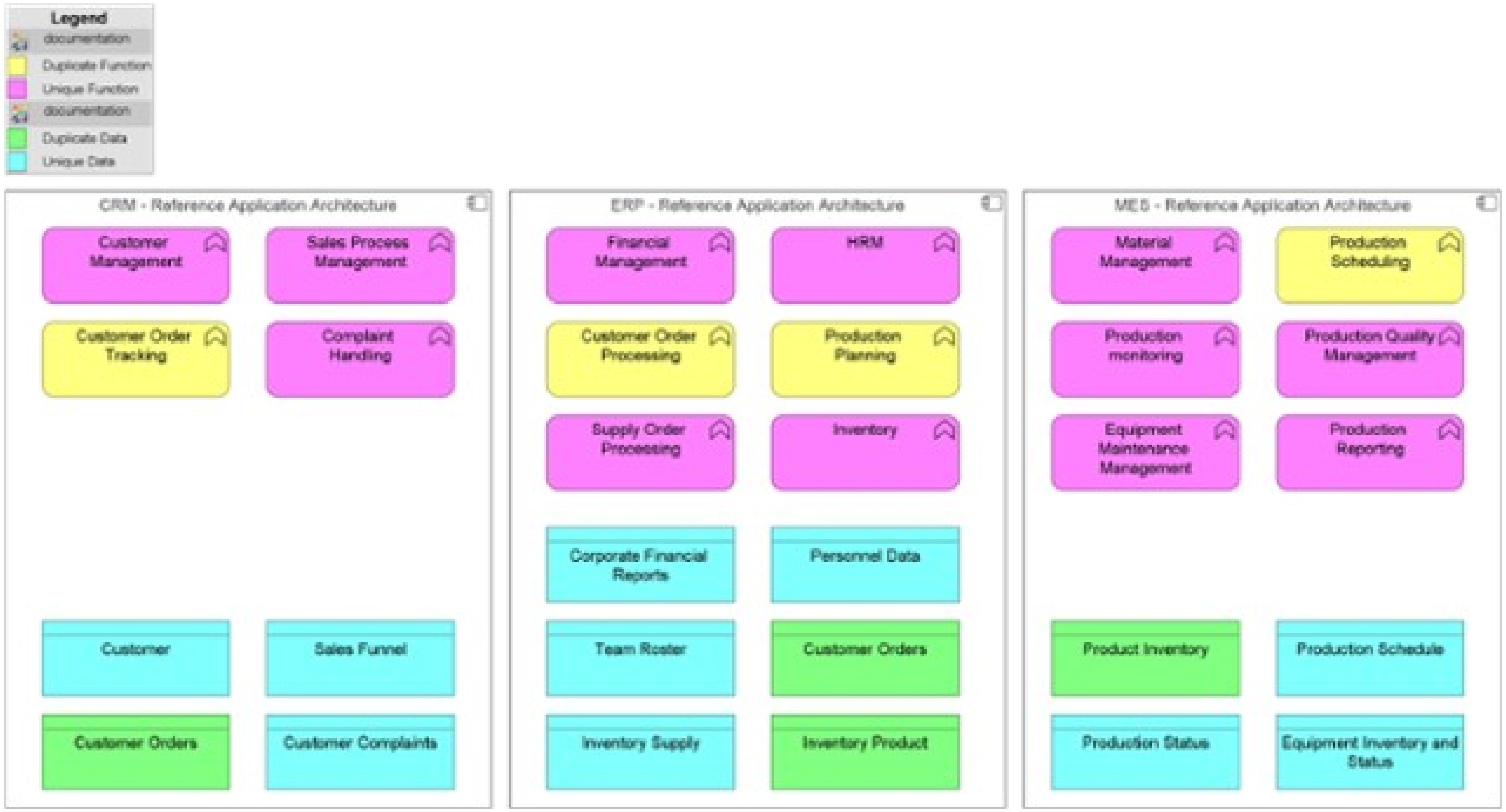
...hence forcing architects to increase the conciseness and expressiveness of their models.

Examples

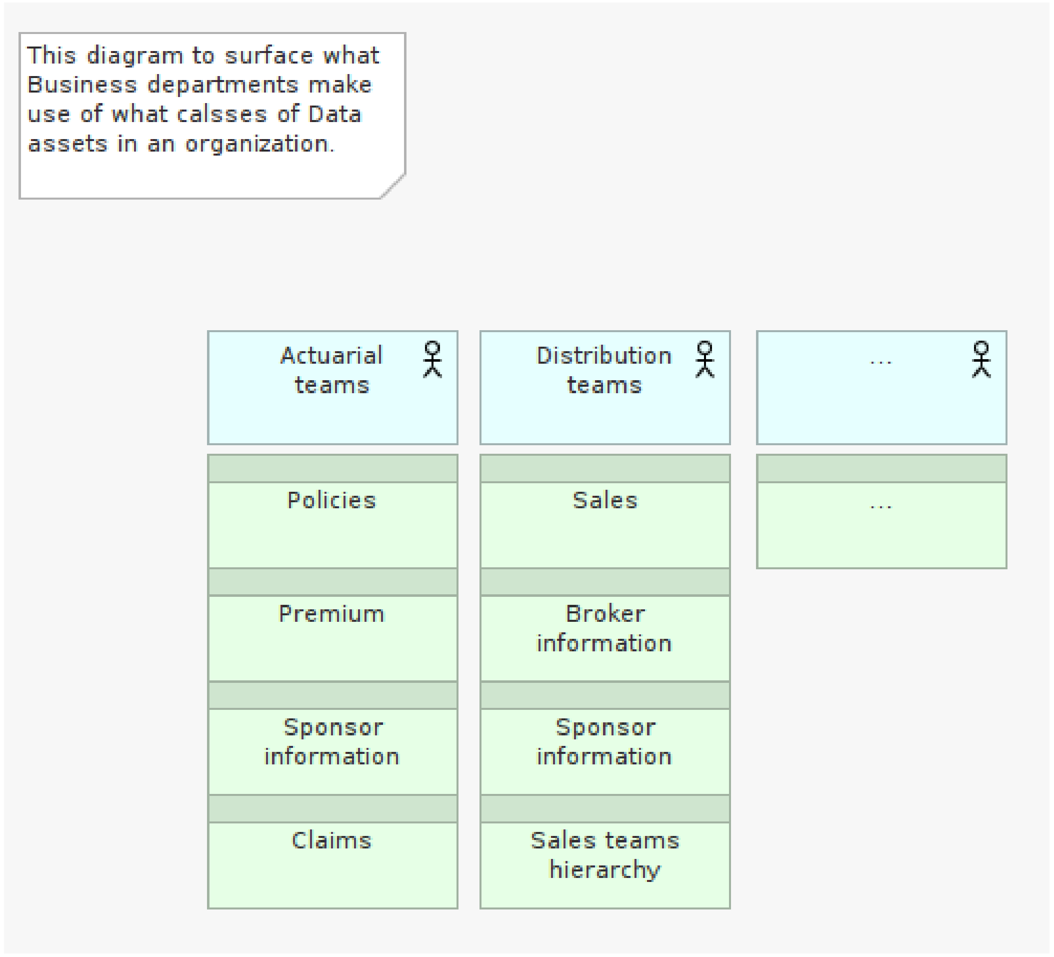
In the next Lectures we will see how VIEWPOINTS meta-models provide a formalized framework architects in the choosing of what models to map, to express what idea. The following examples do NOT constitute an exhaustive list of all possible model mappings.


Deployment <> Information
Context <> Functional <> Information
Context <> Behavior <> Functional
Behavior <> Information
Functional <> Information
Context <> Information
Examples

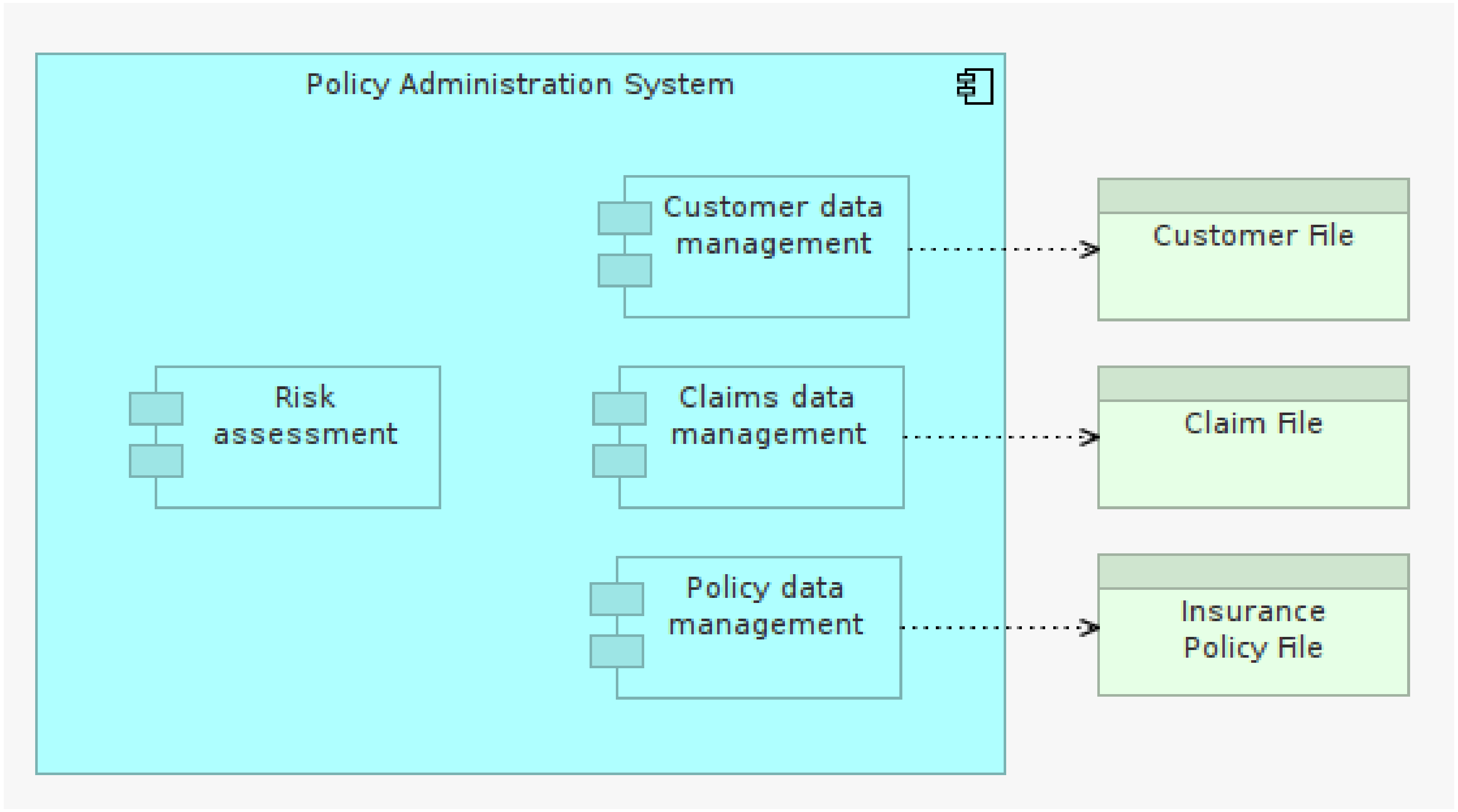
Context <> Information



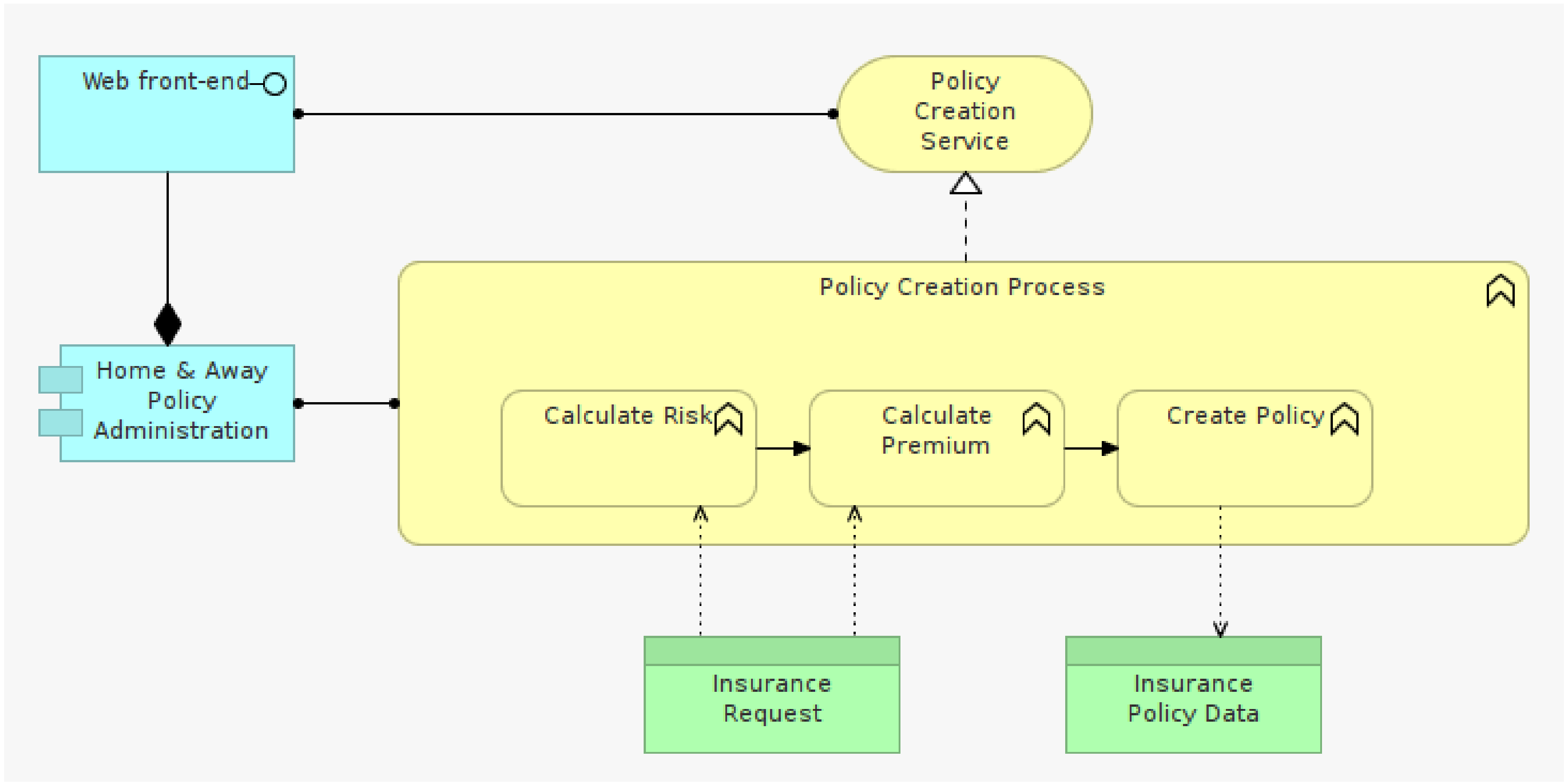
Context <> Information



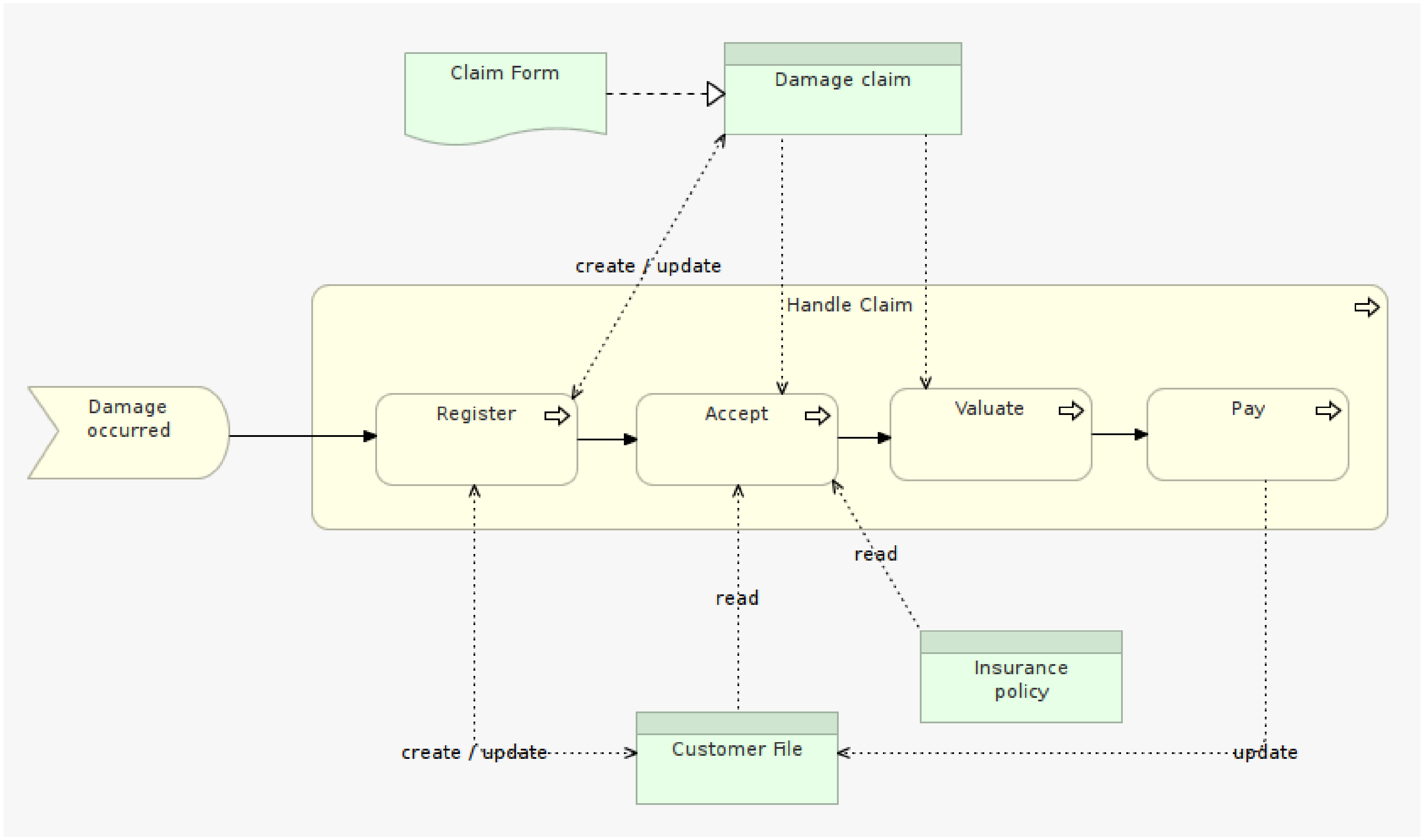
Functional <> Information



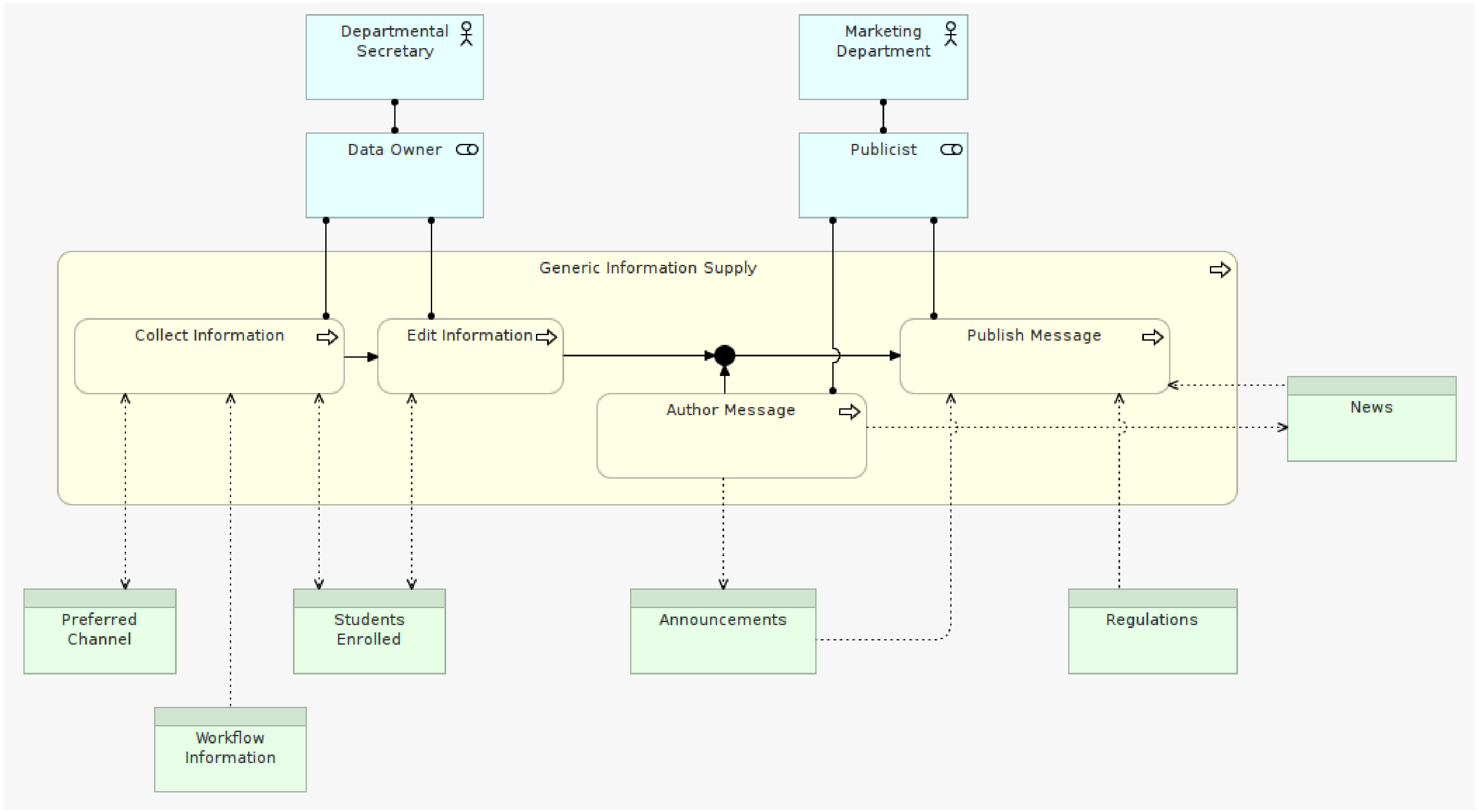
Functional <> Information



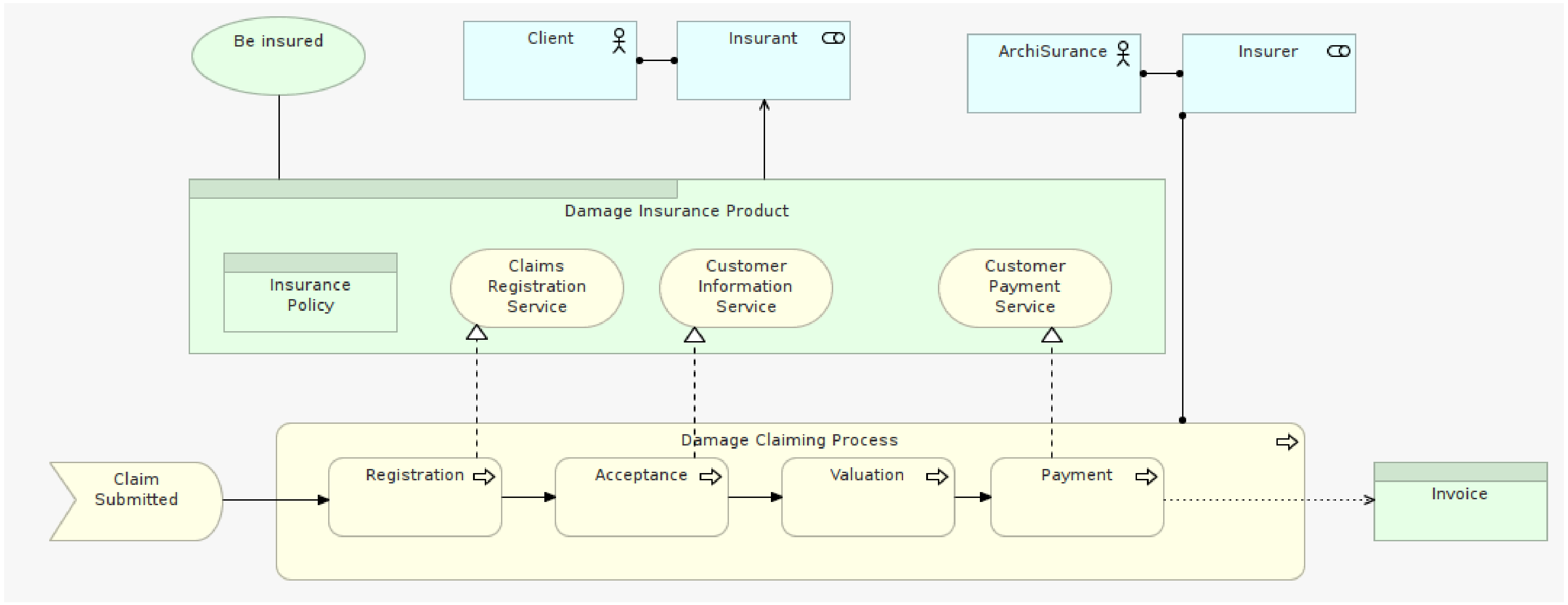
Behavior <> Information



Behavior <> Information

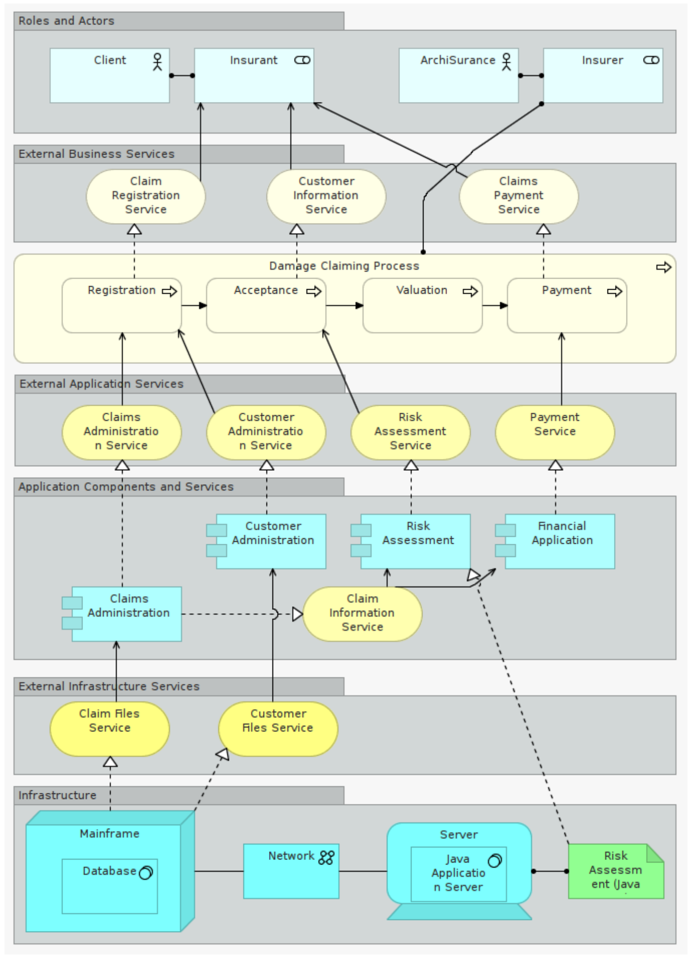


Context <> Behavior <> Functional

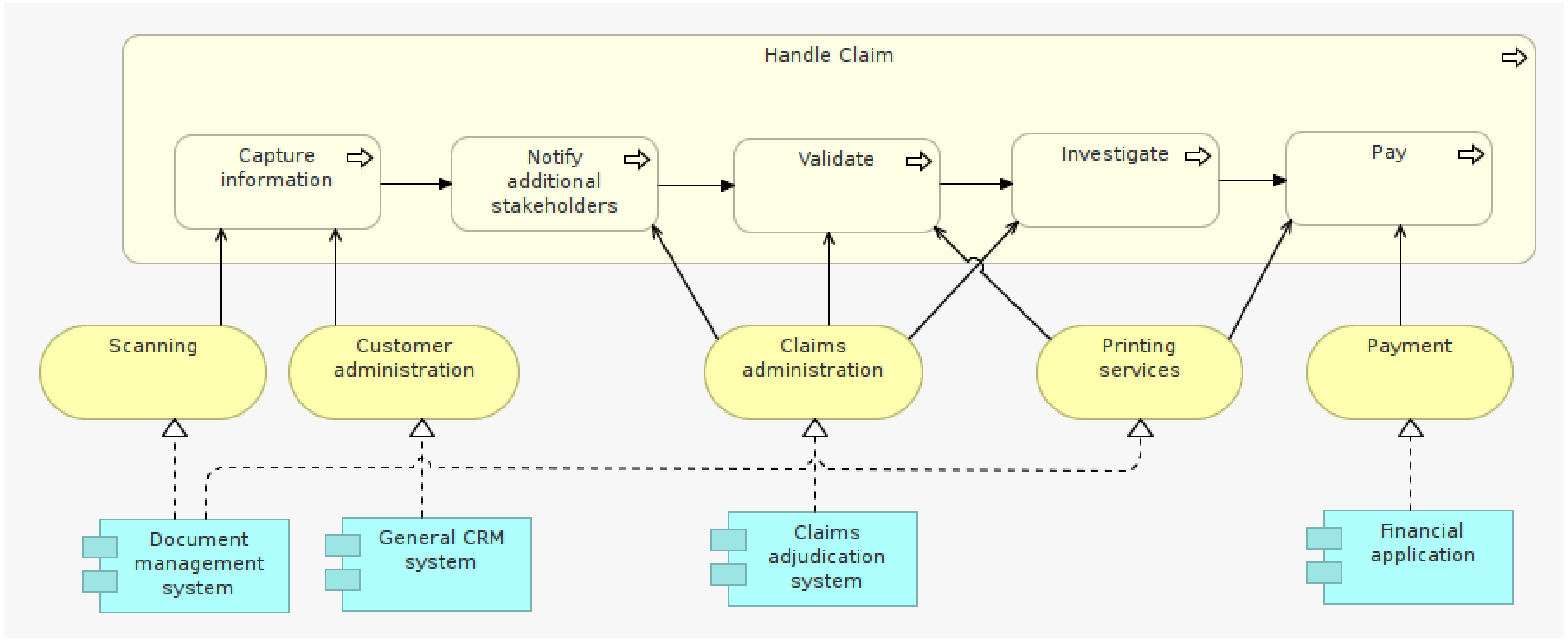


Context <> Functional <> Information

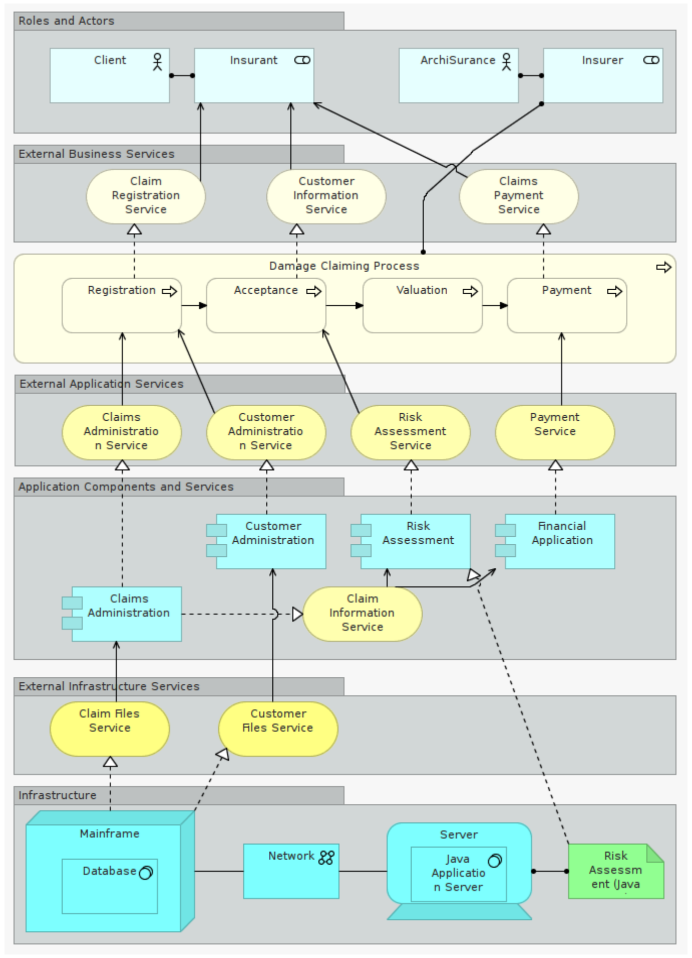
-



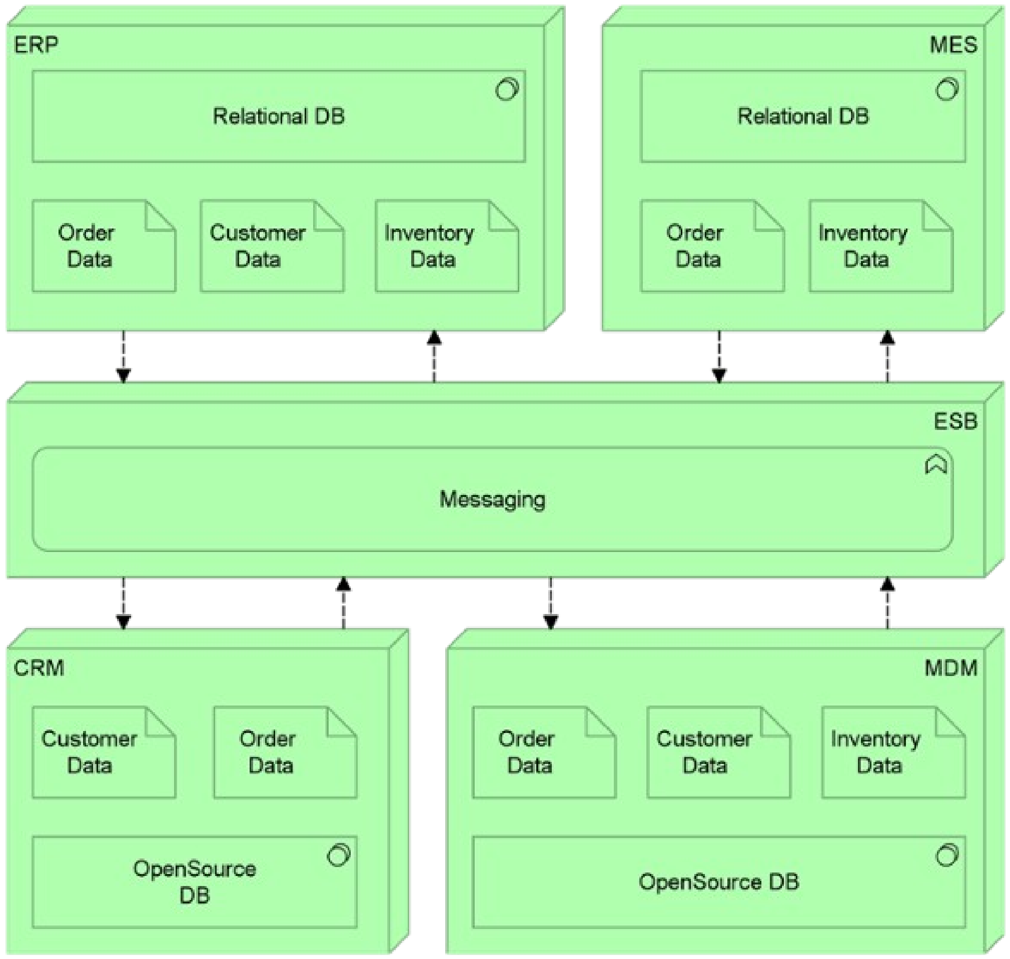
Context <> Behavior <> Functional



Context <> Functional <> Information



Deployment <> Information


ERP
Relational DB
Order Data
Customer Data
Inventory Data
MES
Relational DB
Order Data
Inventory Data
ESB
Messaging
CRM
Customer Data
Order Data
OpenSource DB
MDM
Order Data
Customer Data
Inventory Data
OpenSource DB