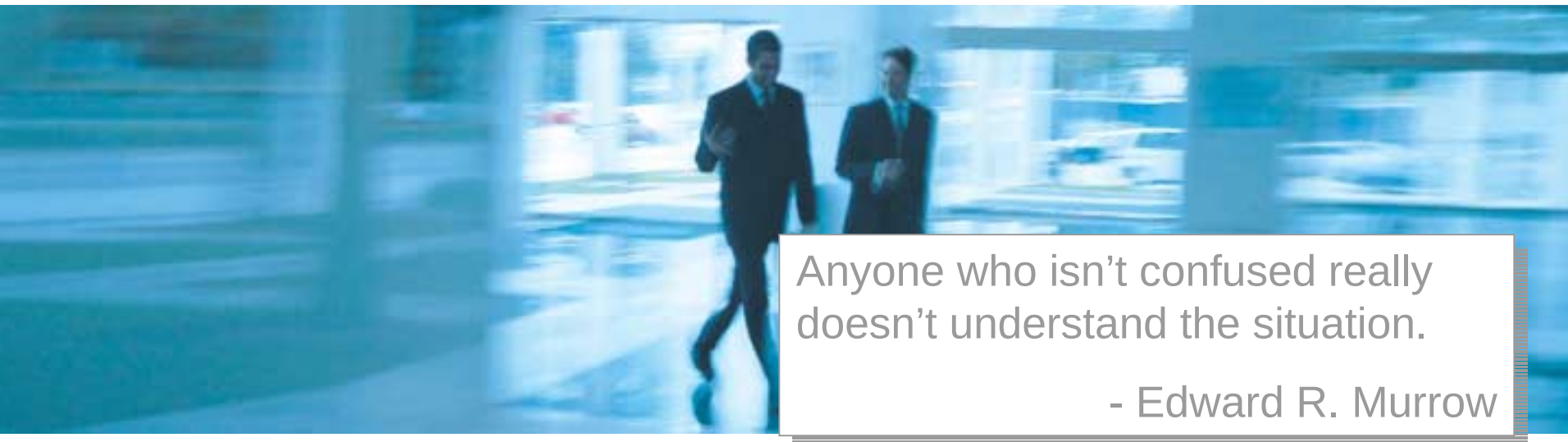
SOA ,Technical Risks, and Emerging Standards

An examination of the relationship between SOA, the conceptual integrity of required of the attributes, and the associated impacts

Victor Harrison
Partner, Federal Consulting Practice
Computer Sciences Corporation



> Anyone who isn't confused really doesn't understand the situation.
>
> - Edward R. Murrow

EXPERIENCE. RESULTS.

Agenda

- Going Beyond a Definition of SOA – The Characteristics
- SOA and Assurance – The Problem of Evidence and Trust
- The First Step: Characterize SOA – Views, Attributes, and Evidence
- Conclusions

Going Beyond a Definition of SOA

The Characteristics

EXPERIENCE. RESULTS.

No wonder SOA is so confusing

CSC
EXPERIENCE. RESULTS.

- There are Hundreds of SOA Definitions. Some emphasize –
  - Behaviors (*A SOA delivers late binding and loosely coupled …*)
  - Componentry (*A SOA utilizes service registries, Interfaces, …)*
  - Standards (*a SOA is a set of web services that* …)
  - So on and so on and so on…
- Depending upon Point-of-View, Services of a SOA can be—
  - Pure business capabilities (Removing snow or ingesting a signal)
  - Software things (the Registry Service , logon Service, etc.)
  - Infrastructure things (amount of disk, CPUs, or networks)

This is <u>not</u> another 'this is what SOA is' Workshop.

Adopting the OMG's Harmonized SOA Definition

CSC EXPERIENCE. RESULTS.

- a SOA is *an **architectural style** for a community of providers and consumers of services to achieve mutual value, that:*
  - *Allows participants in the communities to work together with **minimal co-dependence** or technology dependence*
  - *Specifies the **contracts** to which organizations, people and technologies must adhere in order to participate in specific communities*
  - *Provides for business value and **business processes** to be realized by the community*
  - *Allows for a **variety of technologies** to be used to facilitate interactions within the community"*

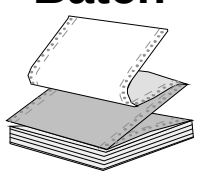
But does this answer the question:
How do you know a design is complete/incomplete or good/bad?

Assertion: SOA is a Superset of Previous Architectures

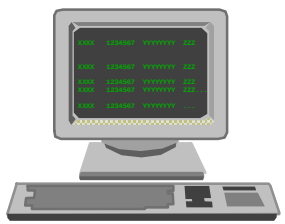
CSC
EXPERIENCE. RESULTS.

Capability Growth

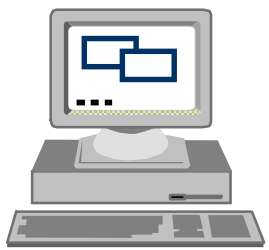
60s Batch



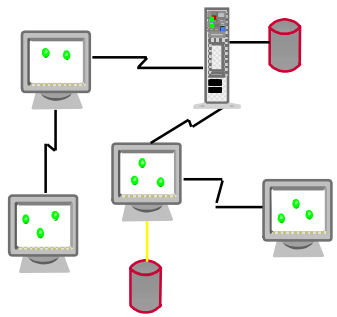
70s Online



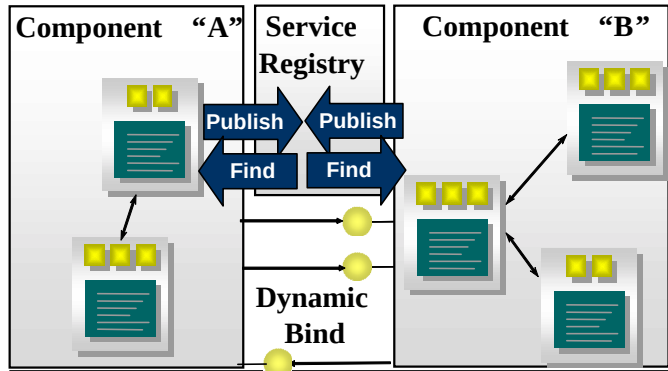
80s Interactive



90s – Today Distributed



Today SOA



Dynamic Architecture Capabilities, Emergence and Conjunctive Composition Capabilities

Distributed Applications, Objects, Interfaces

Layers/Tiers, Workflow, Session

Transactions, Unit of Work, Events

Data Separated from Program, Structure

A first step: Specify SOA Capabilities necessary to enable dynamic architectures

EXPERIENCE. RESULTS.

Capability	Description and Design Principles	How Applied
Late binding	The resolution of a service request as late as possible during execution.	Service Proxies; reflexive bindings, service registries, COIs
Loose coupling	The low degree of mutual interdependence between components.	Data adapters, SOAP messages, generalized Metadata repositories
Discrete functions	The specification of each method of a component is independent of all other methods.	SPRINT planning based upon which operations are to be delivered, when.
Service Discovery	Query, browse, offer, and select operations for specifying operations to satisfy a specific request.	Short term: Service Proxies; long term: "behavioral browsing", reflection.
Autonomous deployment	The ability to deploy components and interfaces into production with minimal constraints on how, when, or how often the deployment occurs.	Our SCRUM-based delivery process. Usage of Service Delegates, Facades, and Proxies.
Message-Based Integration and Interaction	The usage of event-generated messages from autonomous components to dynamically form application contexts; Support for mediated and non-mediated messaging from all types of connected services	SOAP based message formats for the data packets; deep crawling; COIs and application specific 'ontologies'; Support for mediated and non-mediated synch, asynch & pub-sub messaging.
implementation-neutral	Not associating a service with a specific set of programming languages or implementations	Interfaces abstraction, environment dispatching between enclaves; SAML
Coarse grained components	Design and selection of enabling components represent classes of business entities.	Product-centric nature of delivery –each IDP deliverable is a product.
Dynamic service collaborations	Dynamic formation of a process or a service from autonomously defined and deployed services.	Usage of Orchestration (BPEL) to enable processes and Choreography (WS-CDL) to for composite services.

Capabilities necessary to enable emergence and *conjunctive composition*.

CSC
EXPERIENCE. RESULTS.

Capability	Description and Design Principles	How Applied
Service contracts	The basis for finding and delivering a service that 'fits' by containing a detailed specification of actions to be performed and data to be provided.	Metadata defined application contexts represented by model for information collection and COI specification.
Dynamic Process Formation	Processes are created as dynamic service collaborations (above) that result in time-dependent sequences that satisfy specific objectives.	Dynamic (temporal and event driven) as well as pre-defined process support via metadata description,
Composite services	Associations of services in a stateless and peer-to-peer collaboration to satisfy a specific capability.	Best-practice formation of services to deliver IDP-specified capabilities
Application-neutral design	Services are not designed to presume a specific application context, but have application context imported as metadata during execution.	Usage of Facades, Delegates, Factories, and Containers; Interface Facades brokers.
Consumer & Provider Metadata	Metadata driven access/usage needs (consumer metadata) matched to provider description of service (provider metadata).	Session objects of metadata. Ontologies registered in service registry, Metadata usable by all services.
Concurrently developed	Components, interfaces, and data are all capable of being developed in parallel.	Value-drive and feature based SPRINT planning with concurrent execution.
Interoperable in a heterogeneous environment	Components (1) exchange and use self-describing data (2) discover and then accept/post operational requests to components irrespective of how or where the components are enabled.	Usage of Service Proxies, Adapters, Dispatching Services between enclaves, SAML-based entitlement and attribute access; dynamic and static COIs.
Reusable artifacts	Reuse is a measure of shared elements by, minimally, reuse of a service interface's signature.	Support for design time reuse (e.g., shared library of source and linkable objects as well as runtime reuse by the delivery of autonomous services.
Addressable Network Space	Means anyone, or any component, that is anywhere in the network addressable space should be able to participate in a service collaboration.	Federated and replicated registries across domains; Dispatch services; Proxies for cross domain services access.

SOA and Assurance

The Problem of Evidence and Trust

EXPERIENCE. RESULTS.

But there are Questions and Problems

EXPERIENCE. RESULTS.

- "*Trust* is established based upon sufficient and credible *evidence* leading to belief that the entity in some *system context* satisfies the specified requirements"
  - Which Requirements? The ones when the service was built, or the ones driving the selection of a Service for usage?
  - What, exactly, is THE SYSTEM that satisfies the requirements?
- Which capabilities should a SOA Design include?
- How do you deal with the multiple viewpoints regarding what the SOA characteristics are—
  - SOA describes business behaviors (emergence, interoperability, etc.)
  - SOA is a collection of behavioral characteristics (late binding, conjunctive, etc.)
  - SOA is a collection of artifacts (registries, metadata, contracts)
- How do you design, or measure, when there is no commonly accepted basis for what a 'good' SOA design should contain?

And More Questions and Problems

EXPERIENCE. RESULTS.

- What specific threats are introduced by the addition of a SOA architecture and what are the specific services and protection points to address those threats?
- What has to be added to the physical environment to ensure SOA security, given the above?
- What can be reused, maybe with additions or adjustments, in the physical environment to ensure SOA security, given the above?
- What references can be used to certify and accredit SOA services before they are added to ensure adequate security has been addressed?
- What metric or monitoring can provide an indication of how well I'm securing the SOA?
- What will the service provider, the infrastructure and the client be responsible for providing to secure the SOA?
- How will authorizations be applied when authorization repositories must be accessed across domains?

And yet more …

- How will data encryption be handled in the service-oriented architecture?
- How will SOA security 'fail over' for those services that don't/can't react to the SOA paradigm?
- If services can be pulled from 'anywhere', how do we restrict service requests to only pull 'approved' service for the data sensitivity level?
- How will SOAs perform in areas of limited bandwidth? (front line, wireless, dial up, etc)
- How will the metadata attributes attached to a service be inseparable from the service?
- How will the service repository be “self protecting”? What are the requirements for the repository? (Higher than the resulting network?)
- How will the transformation be handled? (Specifically, how to manage and secure a GIG which is evolving and only partially SOA-compliant?)
- What does a cross-domain service require of another service to accomplish information sharing across domains?
- How will we maintain Situational Awareness of SOA functionality and performance?

Each Dynamic Architecture Capability presents a Challenge

EXPERIENCE. RESULTS.

SOA Capability	Challenges
Late binding	Processes are formed dynamically therefore never completely sure what services will participate
Loose coupling	Loose coupling / tight coupling balance; the more loosely coupled the less deterministic the design.
Discrete functions	Implies method level authentication / authorization, not Component or Implementation level
Service Discovery	Policy Enforcement Point and context selection; runtime not design time selection; includes software agents and humans as event triggers
Autonomous deployment	Component “hot swap” requires sophisticated software and hardware design that does plug-and-play. How do you assure hot swap component valid?
Message-Based Integration and Interaction	Focused on correlating simple and complex relationships of events based on past trends and future predictions. Must react to new, external input arriving at unpredictable times. Temporal and event based validation of the service vector.
Implementation- neutral	In Object Management Group (OMG) parlance, how do you IA certify the Platform Independent Model (PIM) and the Platform Specific Model (PSM)
Coarse-grained components	Resolve the paradox with ‘Discrete Functions’; and ‘Application Neutral Design’; Requires proper domain analysis of business processing and ontology;
Dynamic service collaborations	What represents a ‘valid’ orchestration? PeP has to validate the collaboration and service vector.

And each *Emergence & Conjunctive Capability has a challenge, too*

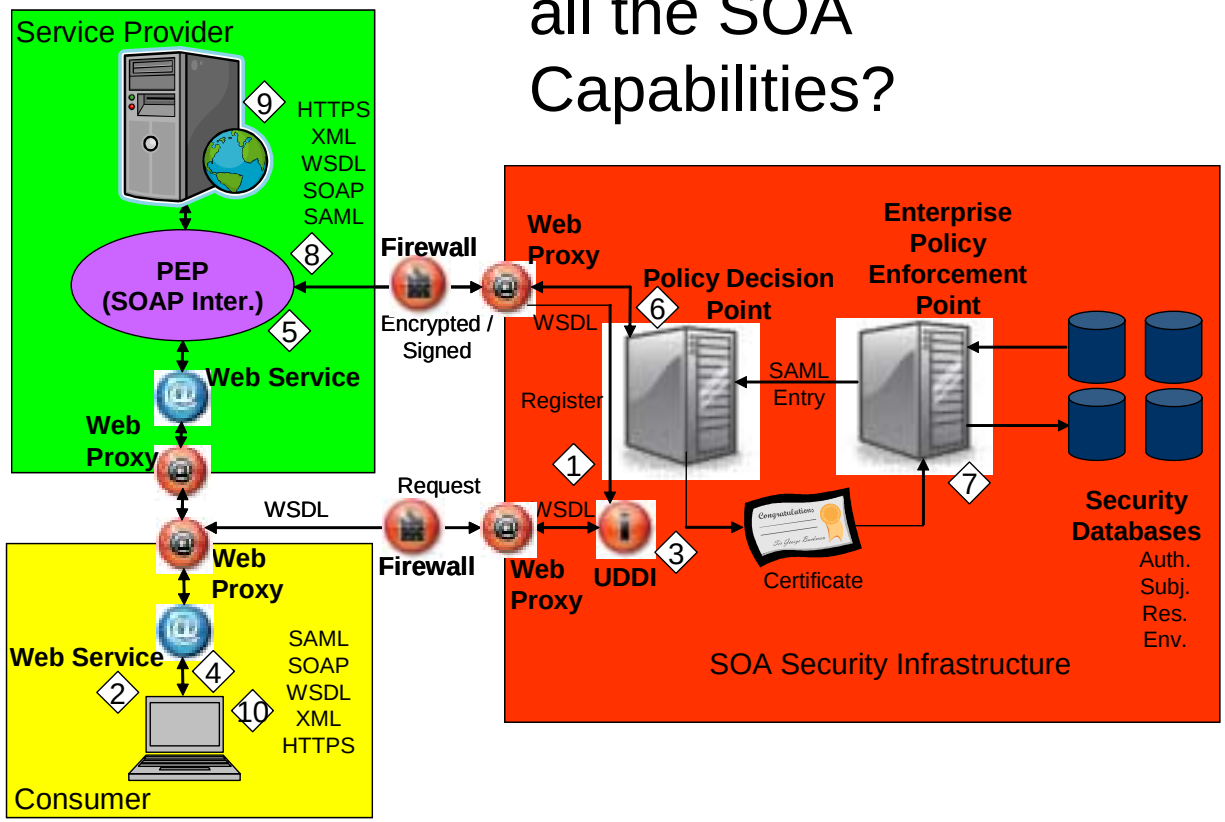
CSC
EXPERIENCE. RESULTS.

Capability	IA Challenges
Service contracts	Designing and defining voluntary agreements that mutually bind the participants to authorizations, obligations, and modes of interaction.
Dynamic Process Formation	Determine the value proposition and rights of the consumer; identifying and assembling services that will form a virtual service for the end consumer.
Composite services	Centralized reuse repository and effective reuse management; what represents a valid composition
Application-neutral design	Proper domain analysis, and good ontology specification and management
Consumer & Provider Metadata	How do you authenticate and authorize to Metadata? Proper domain analysis, and good ontology specification and management
Concurrently developed	Application context in the metadata, not the software component – how do you authenticate and authorize to Metadata?
Interoperable in a heterogeneous environment	How do we assure the origin and content of self-describing data; how do we know that the right context is being applied?
Reusable artifacts	Centralized reuse repository and management but how is registration for usage validated?
Addressable Network Space	What represents the ‘network addressable space?” How do we bridge air gaps? Service Replicas? How do we assure they are safe?

A Reference Implementation for a Secure SOA from CSC– Why is this 'correct'?

CSC
EXPERIENCE. RESULTS.

Does this accommodate all the SOA Capabilities?

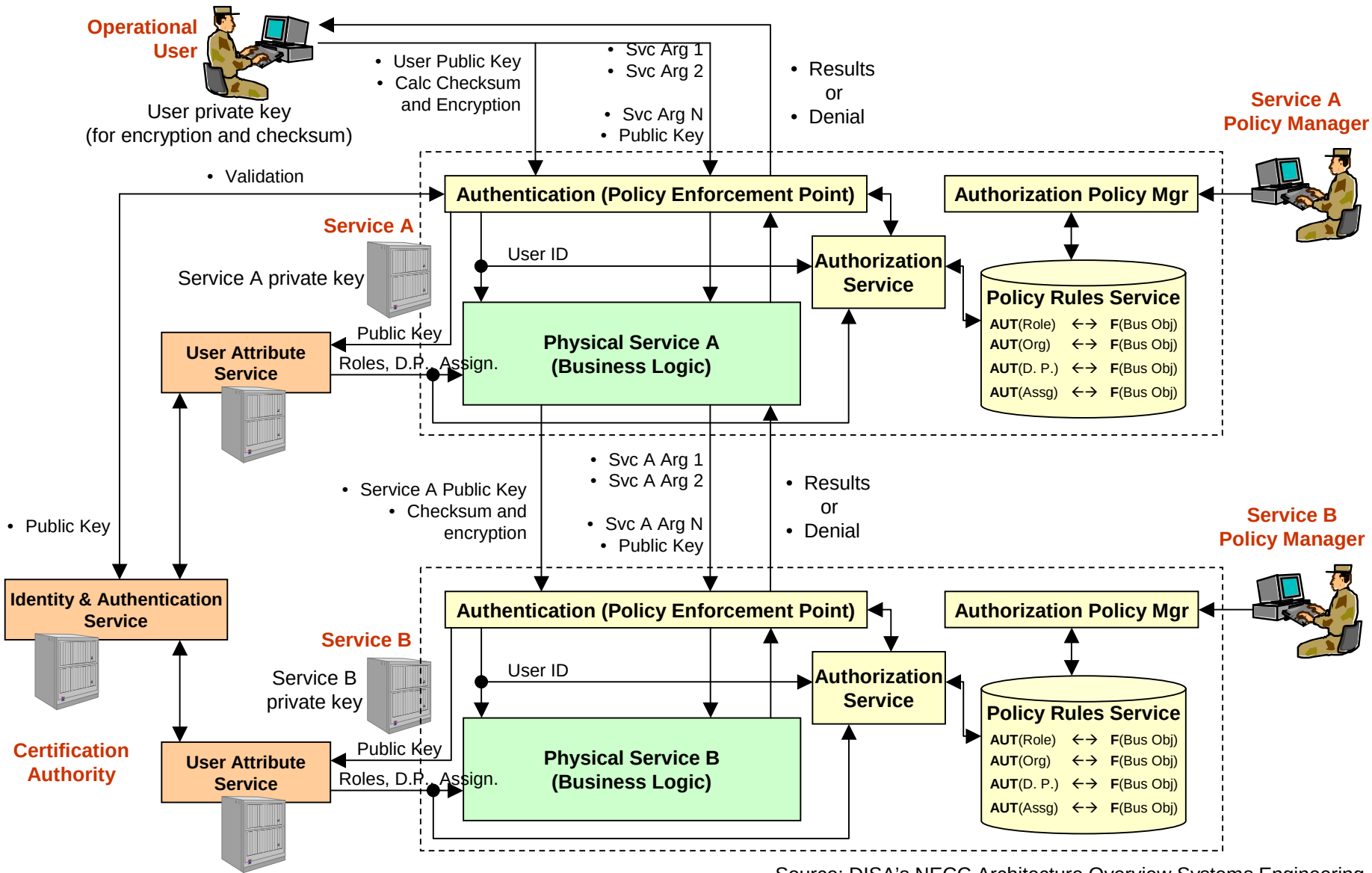


Source: Computer Sciences Corporation

1. Provider registers service (independent action, but done prior to access)
2. Consumer requests Service
3. System performs Registry lookup
4. System can now route Service Request in proper format
5. Policy Enforcement Point redirects message for Authorization check
6. Security Service sends an Authorization Request to Federated Authorization source
7. Federated Authorization source verifies ID, Role, Environment and Service Authorization
8. Federated Authorization source sends Authorization Grant
9. Service Provider accepts Service Request, processes request and returns Service
10. Consumer receives Service requested

A Reference IA Architecture from the US Government– Why is this 'correct'?

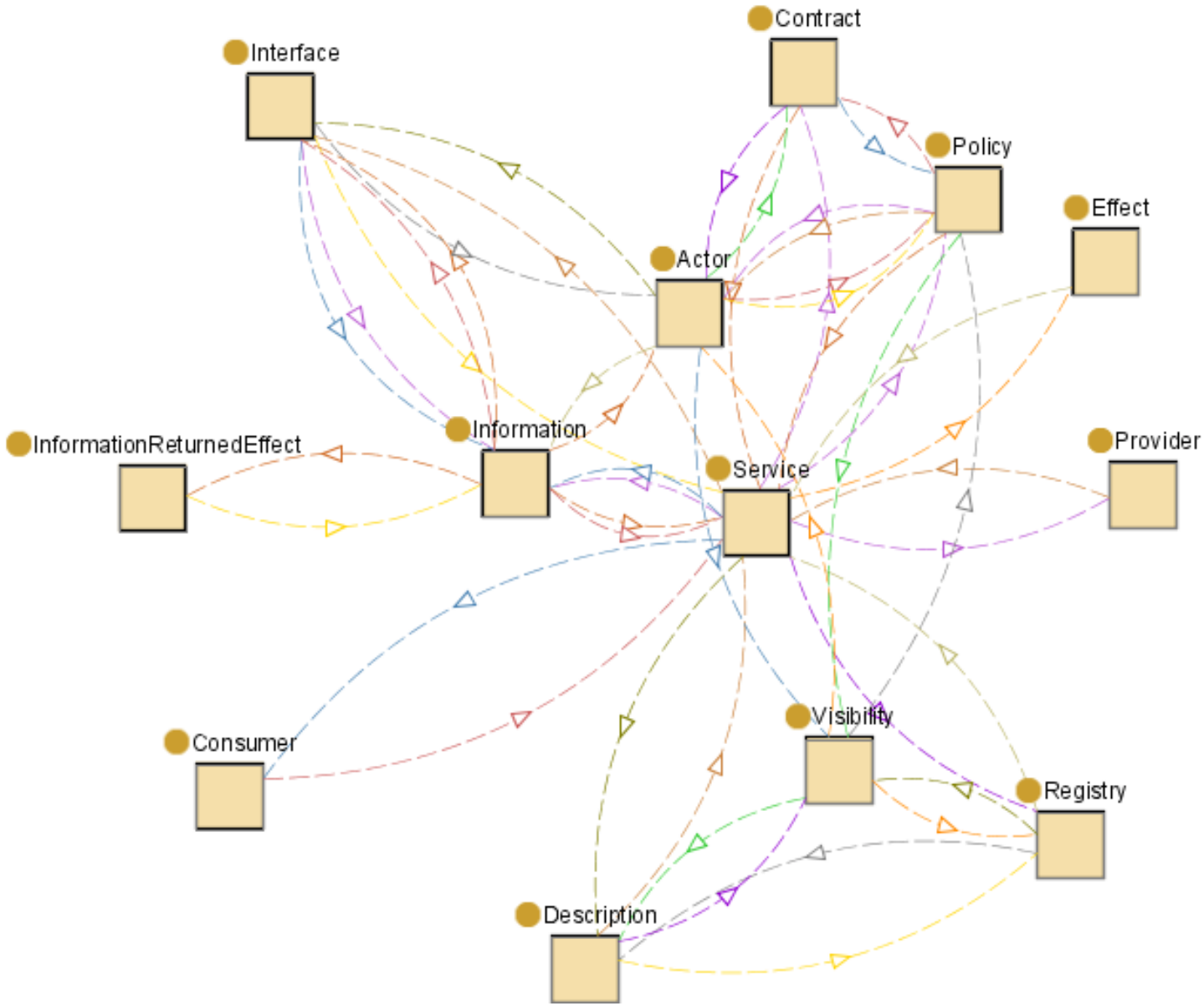
CSC EXPERIENCE. RESULTS.



Source: DISA's NECC Architecture Overview Systems Engineering and Integration Working Group, 15 Aug 2006

OASIS SOA Ontology– Does this accommodate all SOA Capabilities?

CSC
EXPERIENCE. RESULTS.



The First Step: Characterize SOA Characteristics Accurately

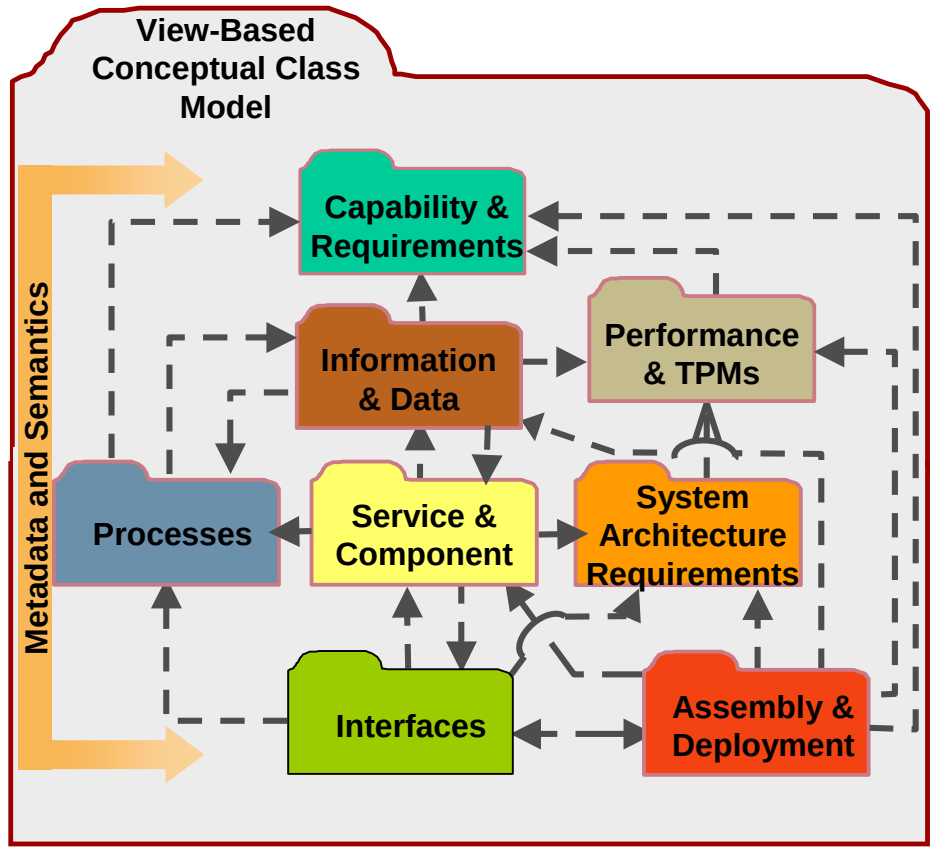
Views, Attributes, and Evidence

EXPERIENCE. RESULTS.

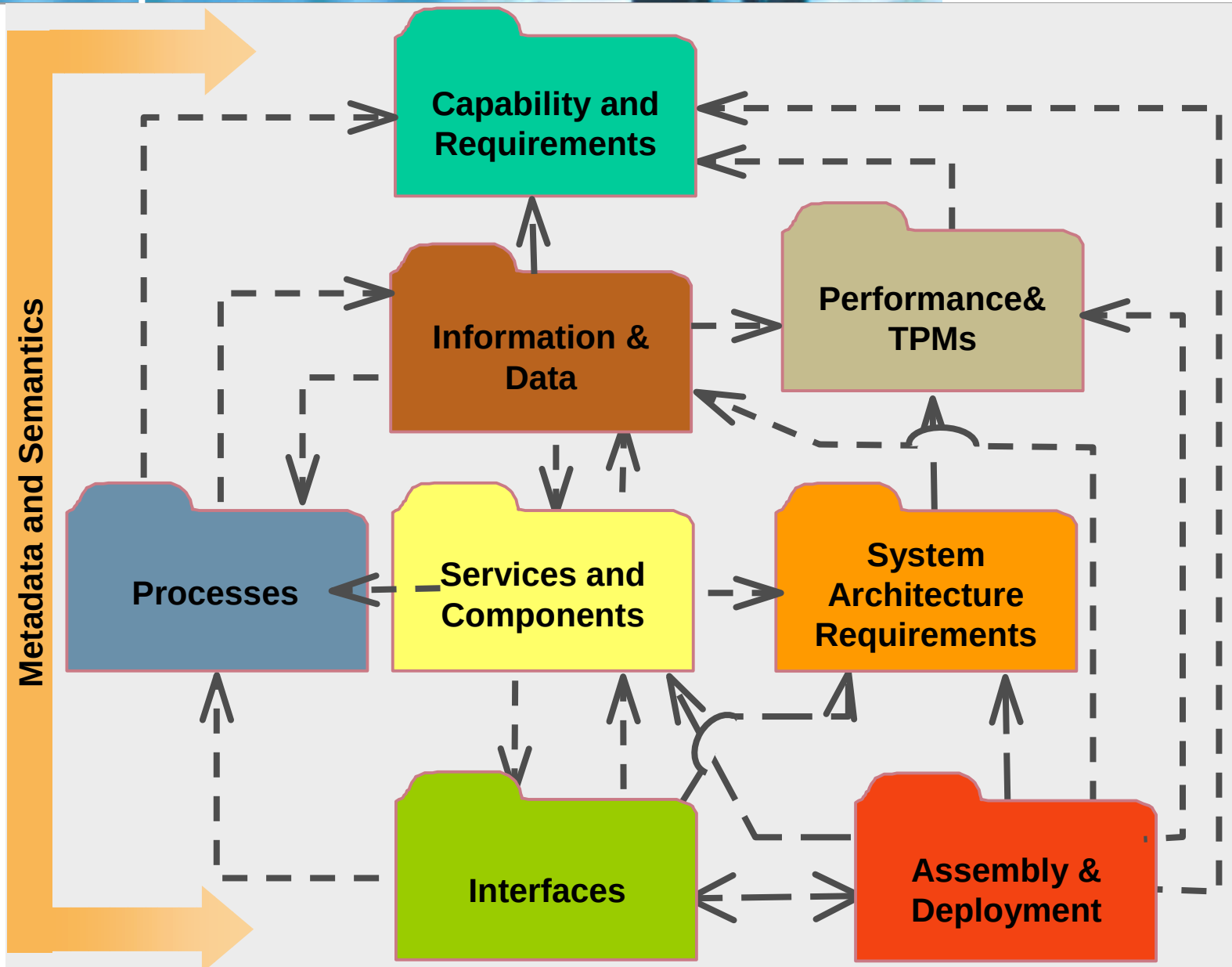
An Approach to these problems: a View-based Conceptual Class Model

EXPERIENCE. RESULTS.

- **Describes integrated, multi-view design data** for individual systems or system-of-systems
- **Specifies correctness** for necessary design-to-delivery information
- **Used for—**
  - **Defining and acquiring service components** into a common framework
  - **Reverse-engineering** dissimilar design information from multiple projects into a common format for analysis, assessment, and integration planning
  - **Validation and verification** of any set of design artifacts resulting in a quantifiable assessment of correctness
  - **Bottoms-up enterprise Integration** of multiple projects or top-down specification of the necessary pieces for an integrated enterprise
  - **Continuous and concurrent delivery** of software into an integration solution framework

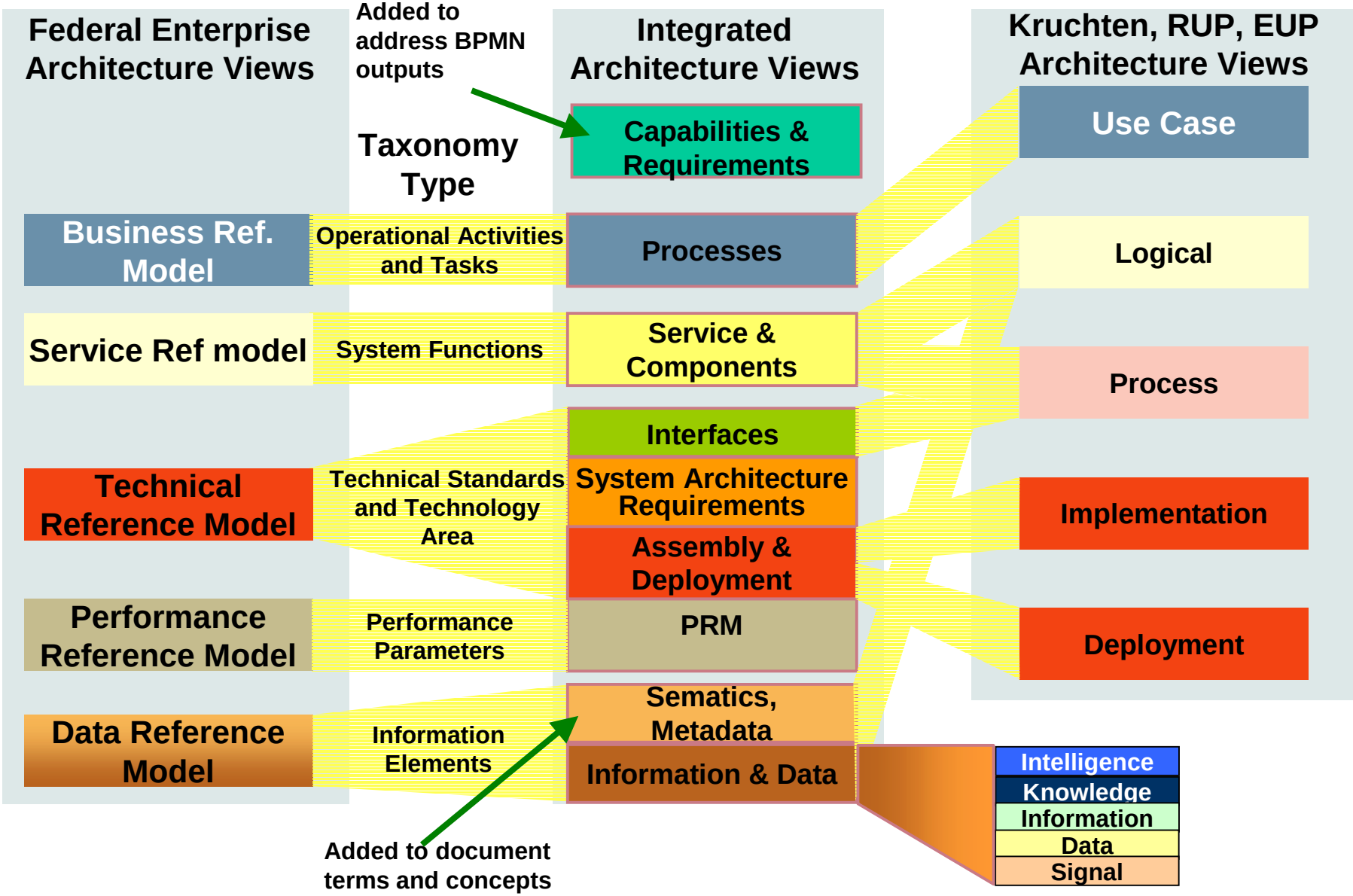


View Based Specification



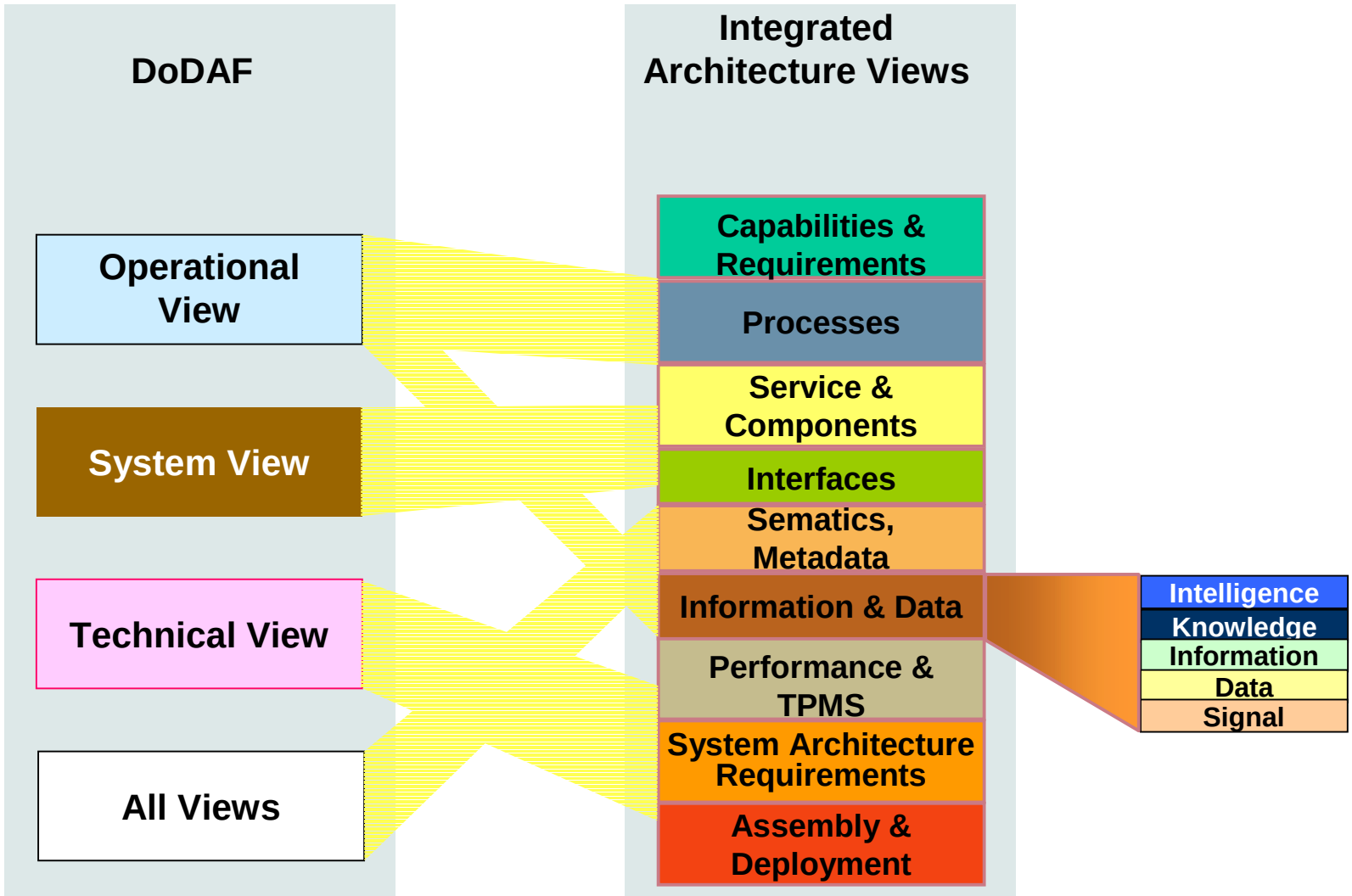
Relating the Viewset to FEA and Kruchten

CSC
EXPERIENCE. RESULTS.

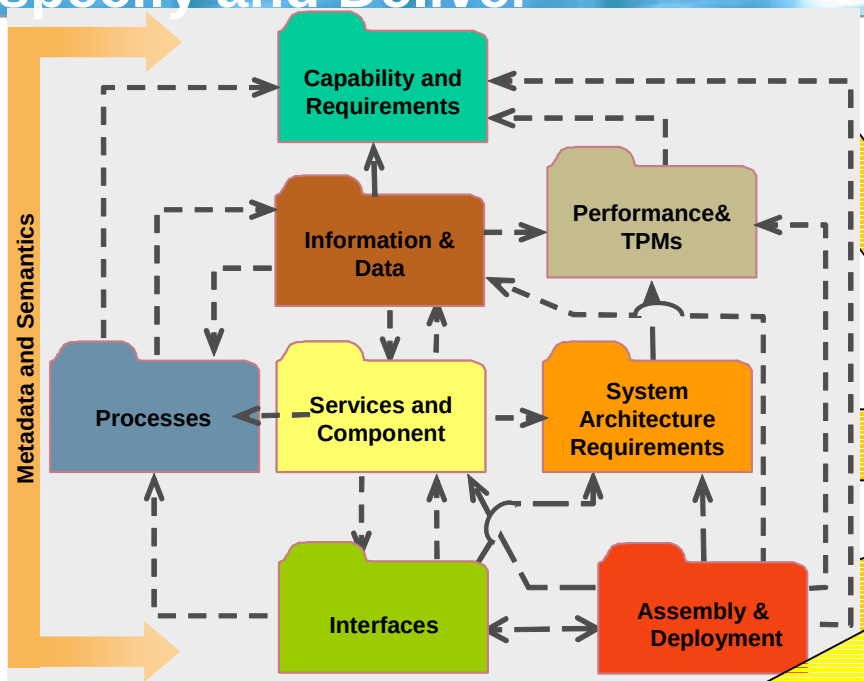


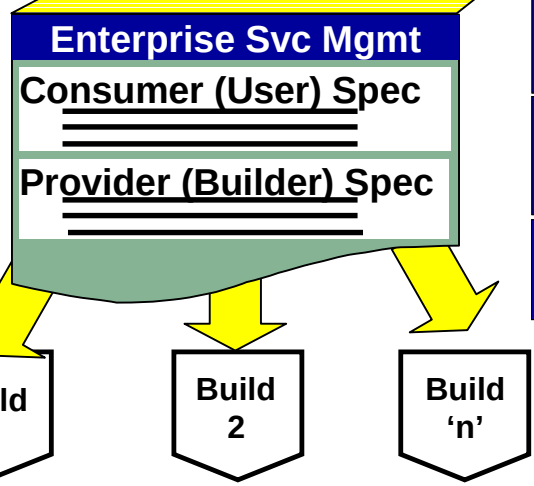
Relating the Viewset to DoDAF

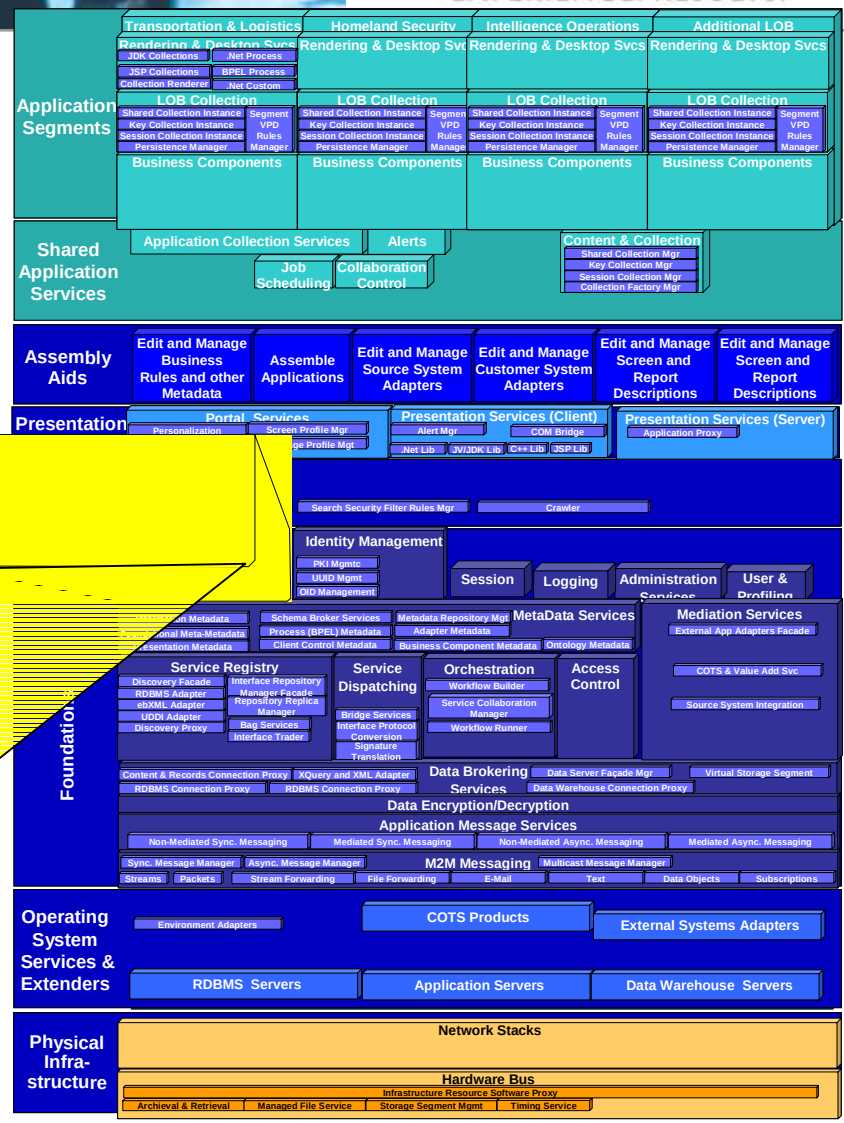
CSC
EXPERIENCE. RESULTS.



Using well-formed Views to specify and Deliver



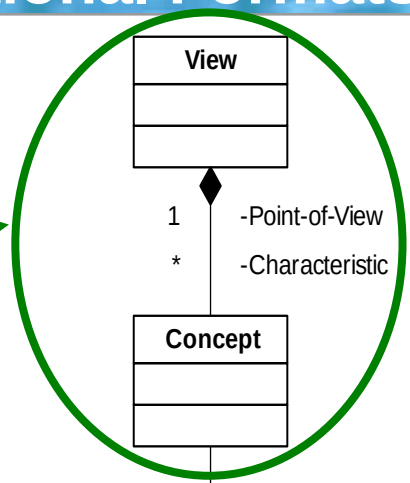


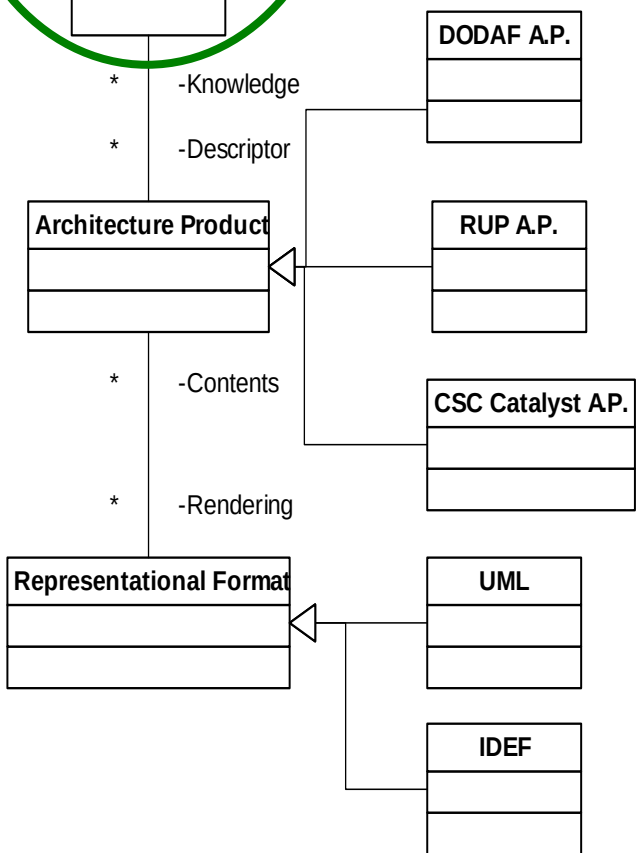


Views, Concepts, Architecture Products, and Representational Formats

EXPERIENCE. RESULTS.

Today's Focus





Example: Service Component View

Example: Component API

Example: SV-6c & SV-4

Example: Interface document artifact

Example: Interface Control Documen

Example: Parameterized Interface Class

Example: Mechanism

The Parts of a "Concept" in this Model

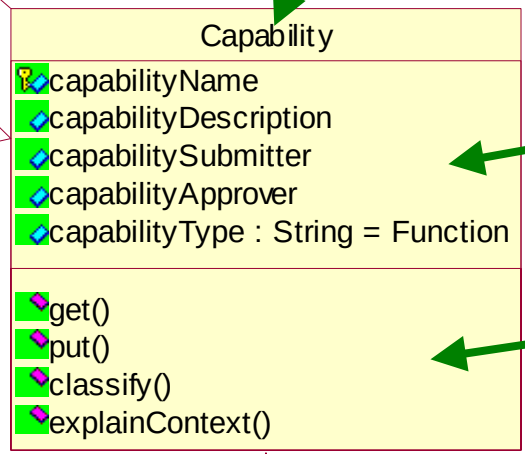
EXPERIENCE. RESULTS.

The concept has definition with terms such as actor-owner (another Concept) or desirable outcome (a specific and measurable semantic) defined in the model

Definition

A Capability is a discrete expression of a desired outcome that an Actor-Owner wants to see in a system. Capabilities are descriptions of behavior and for them to be unambiguous must be tangible and measurable

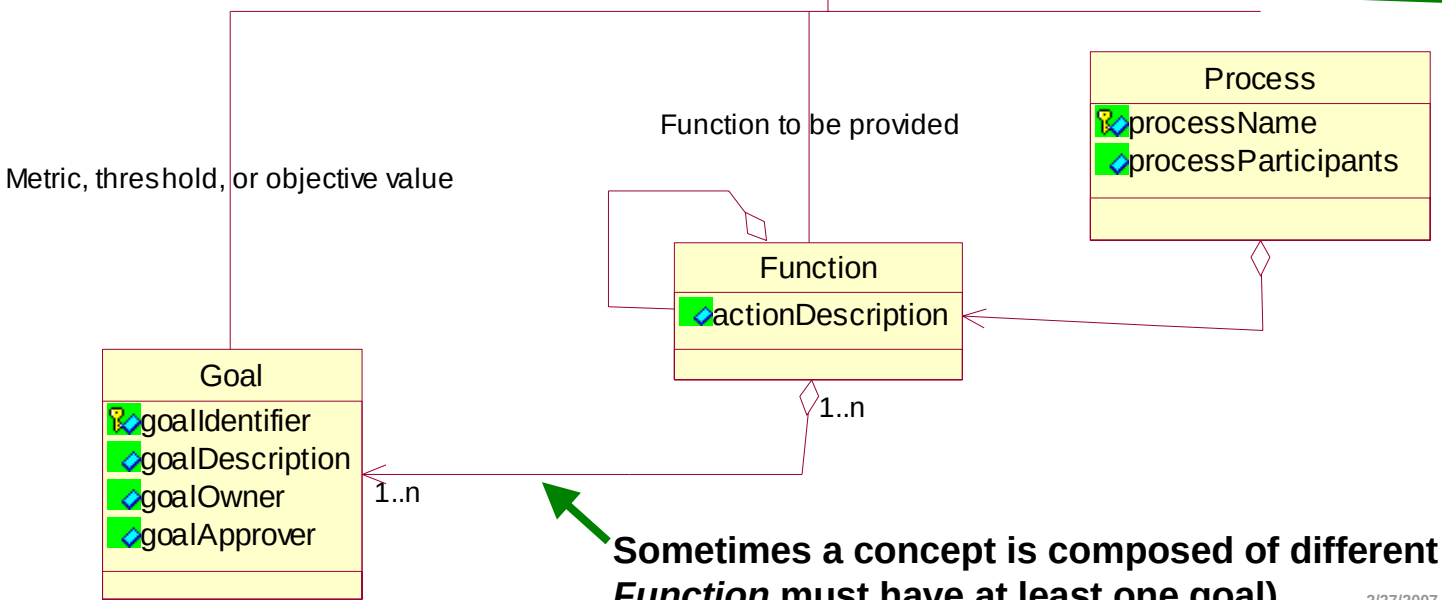
A concept expressed as a UML class



Attributes of the concept that must be included in a design

Actions that must be supported by the design

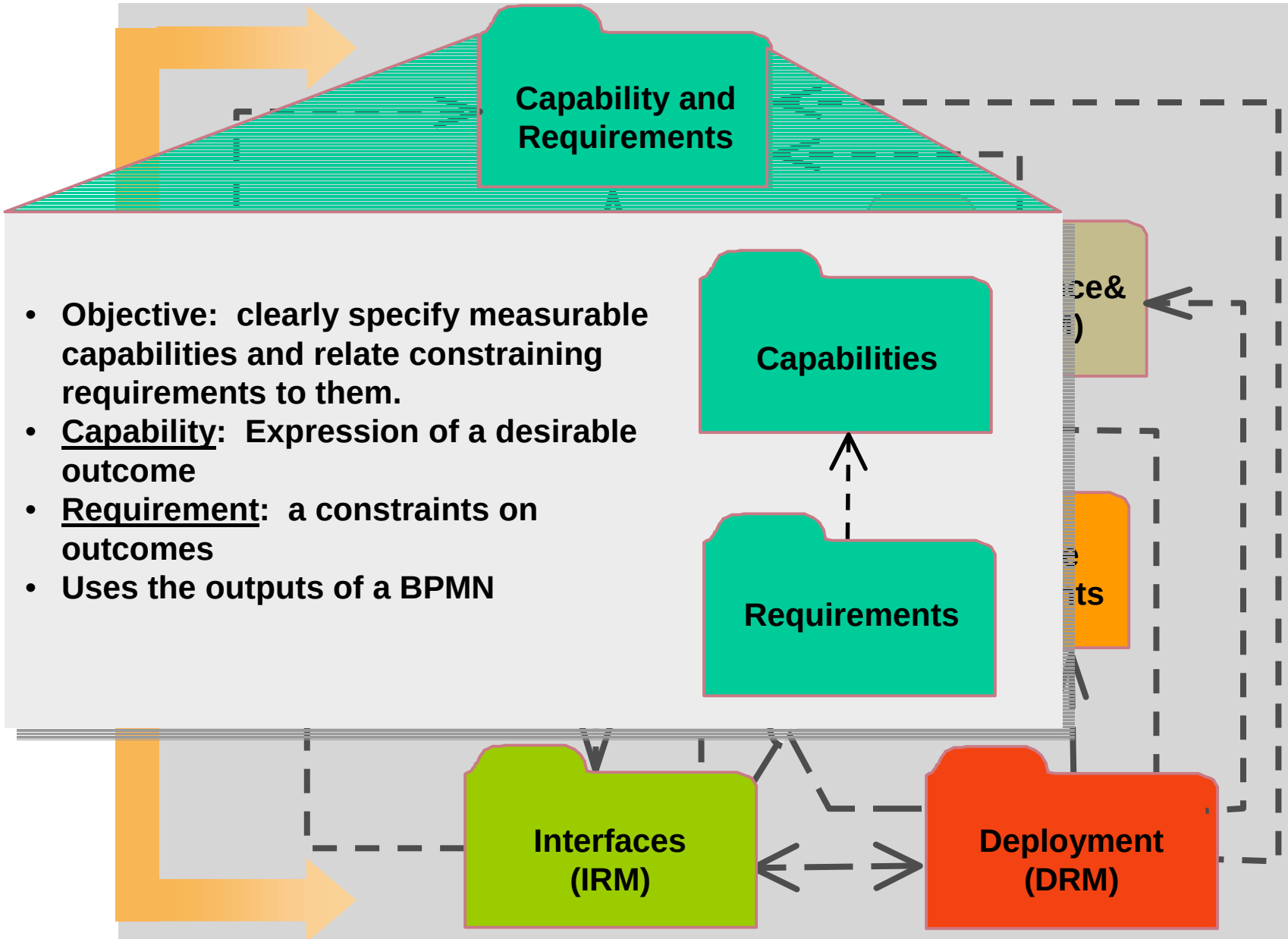
Sometimes Concepts take specific forms (e.g., a *Capability* is either a *Process*, *Function*, or *Goal*)



Sometimes a concept is composed of different things (e.g., a *Function* must have at least one goal)

Capabilities and Requirements View

CSC
EXPERIENCE. RESULTS.

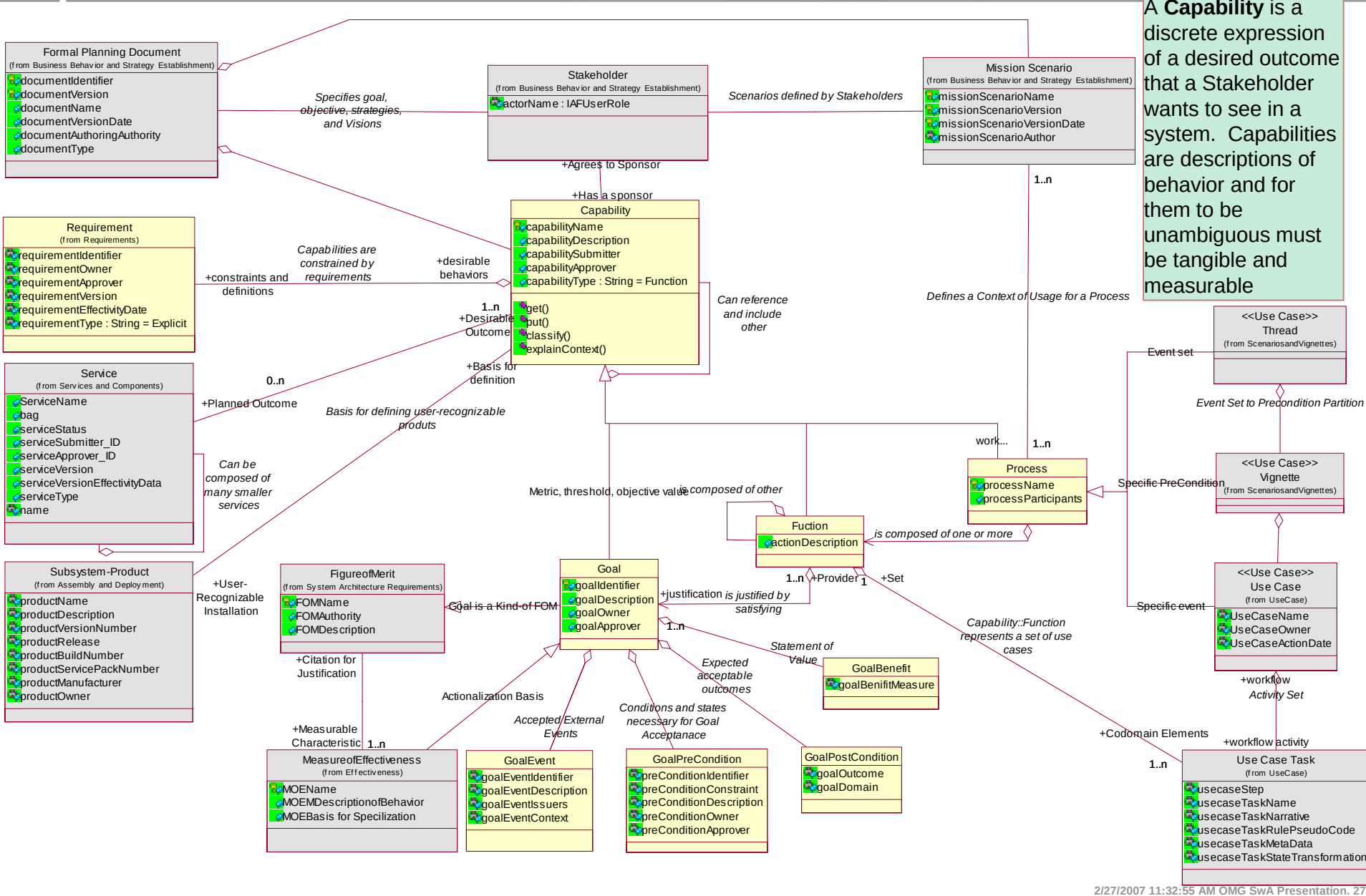


- **Objective: clearly specify measurable capabilities and relate constraining requirements to them.**
- **Capability: Expression of a desirable outcome**
- **Requirement: a constraints on outcomes**
- **Uses the outputs of a BPMN**

Capabilities and Requirements View: Capabilities

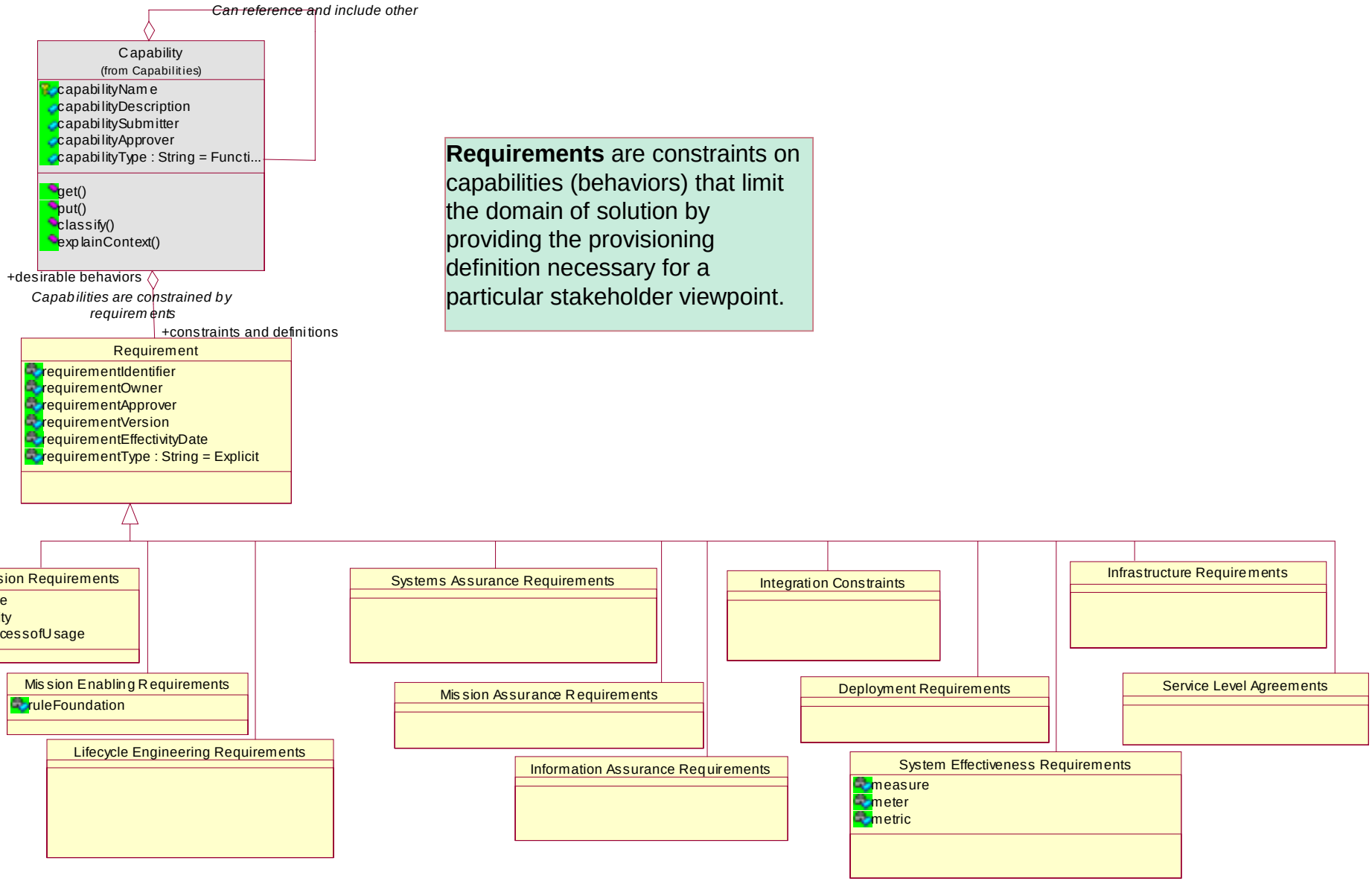
CSC EXPERIENCE. RESULTS.

A **Capability** is a discrete expression of a desired outcome that a Stakeholder wants to see in a system. Capabilities are descriptions of behavior and for them to be unambiguous must be tangible and measurable



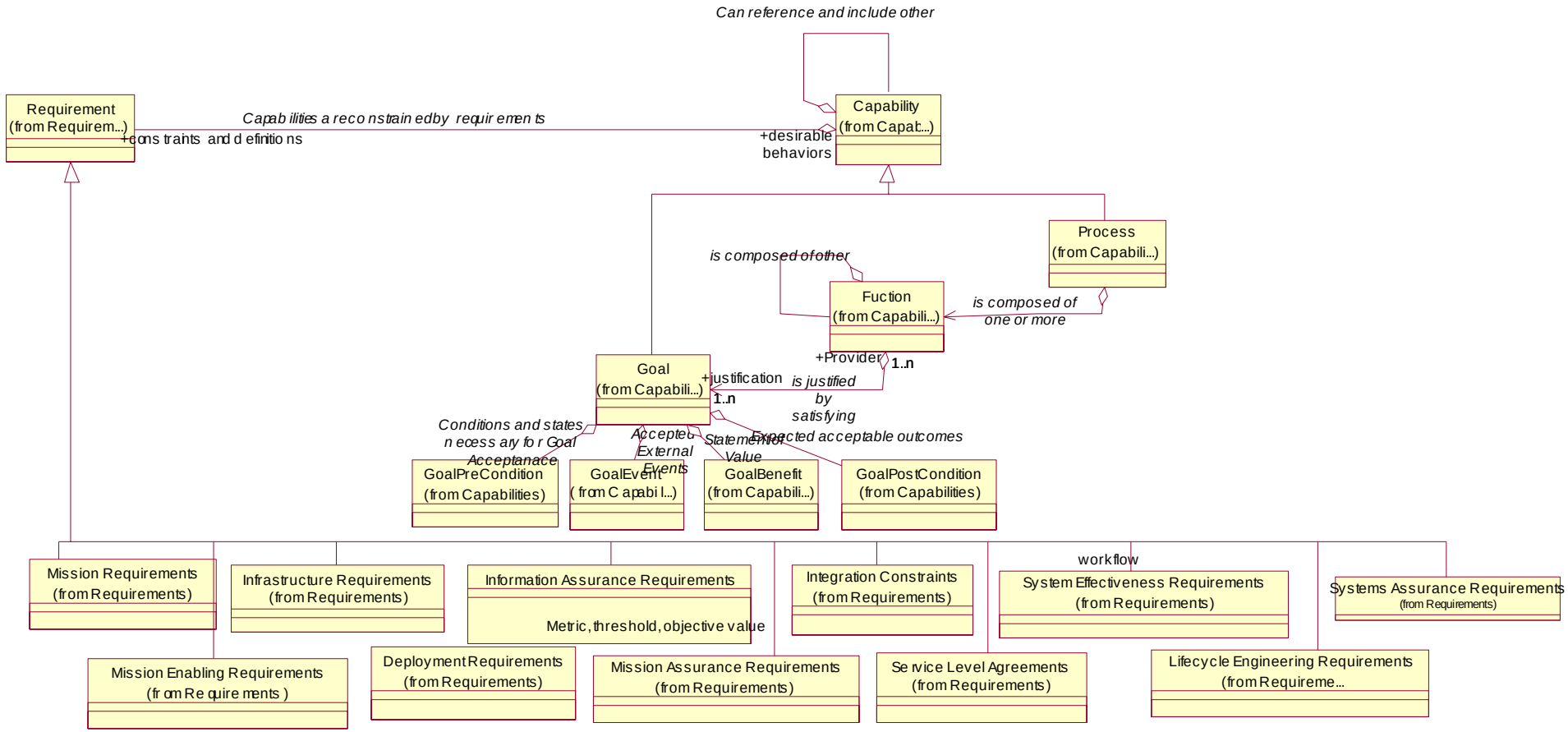
Capabilities and Requirements View: Requirements

EXPERIENCE. RESULTS.



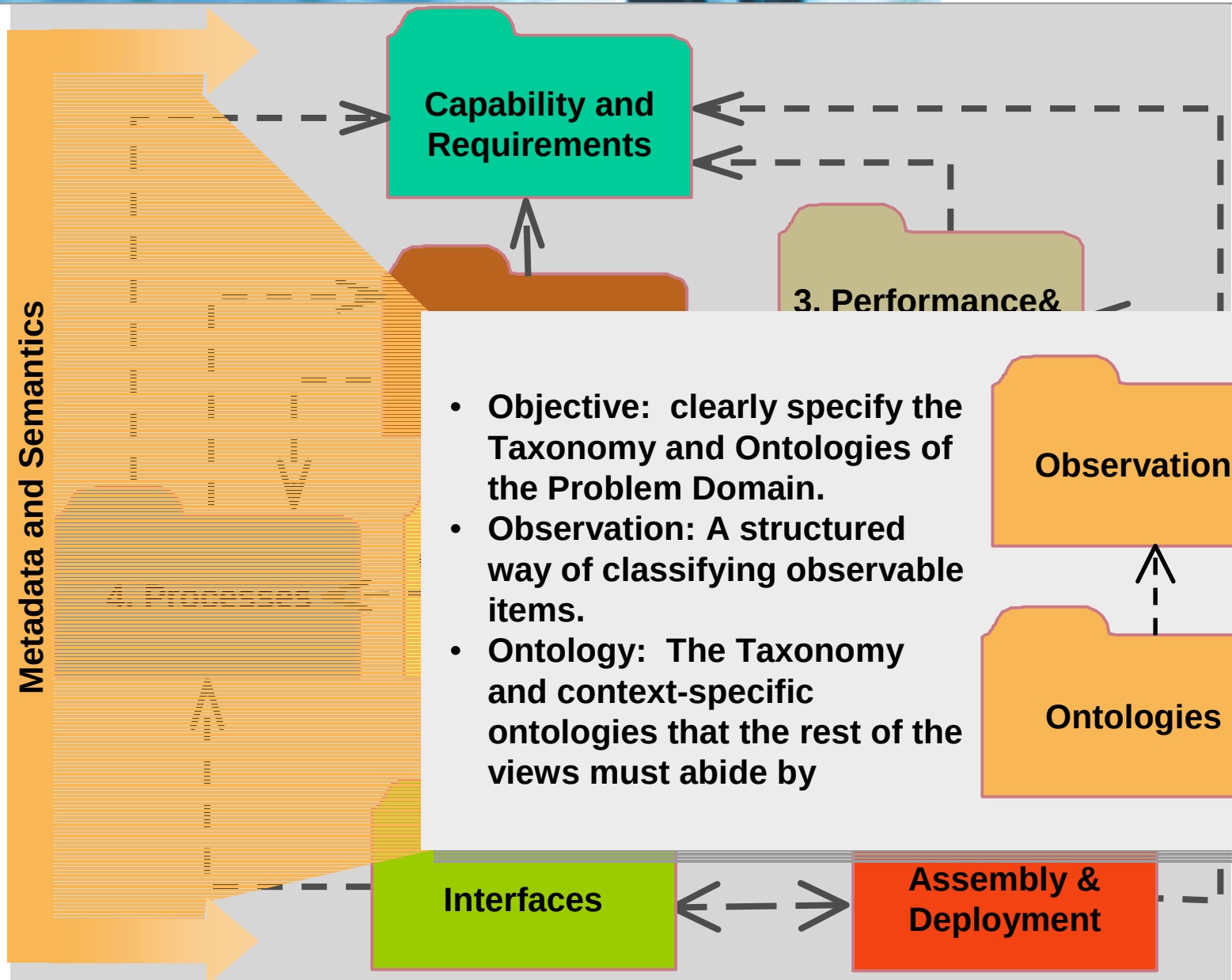
Requirements are constraints on capabilities (behaviors) that limit the domain of solution by providing the provisioning definition necessary for a particular stakeholder viewpoint.

Capabilities and Requirements View: Integrated



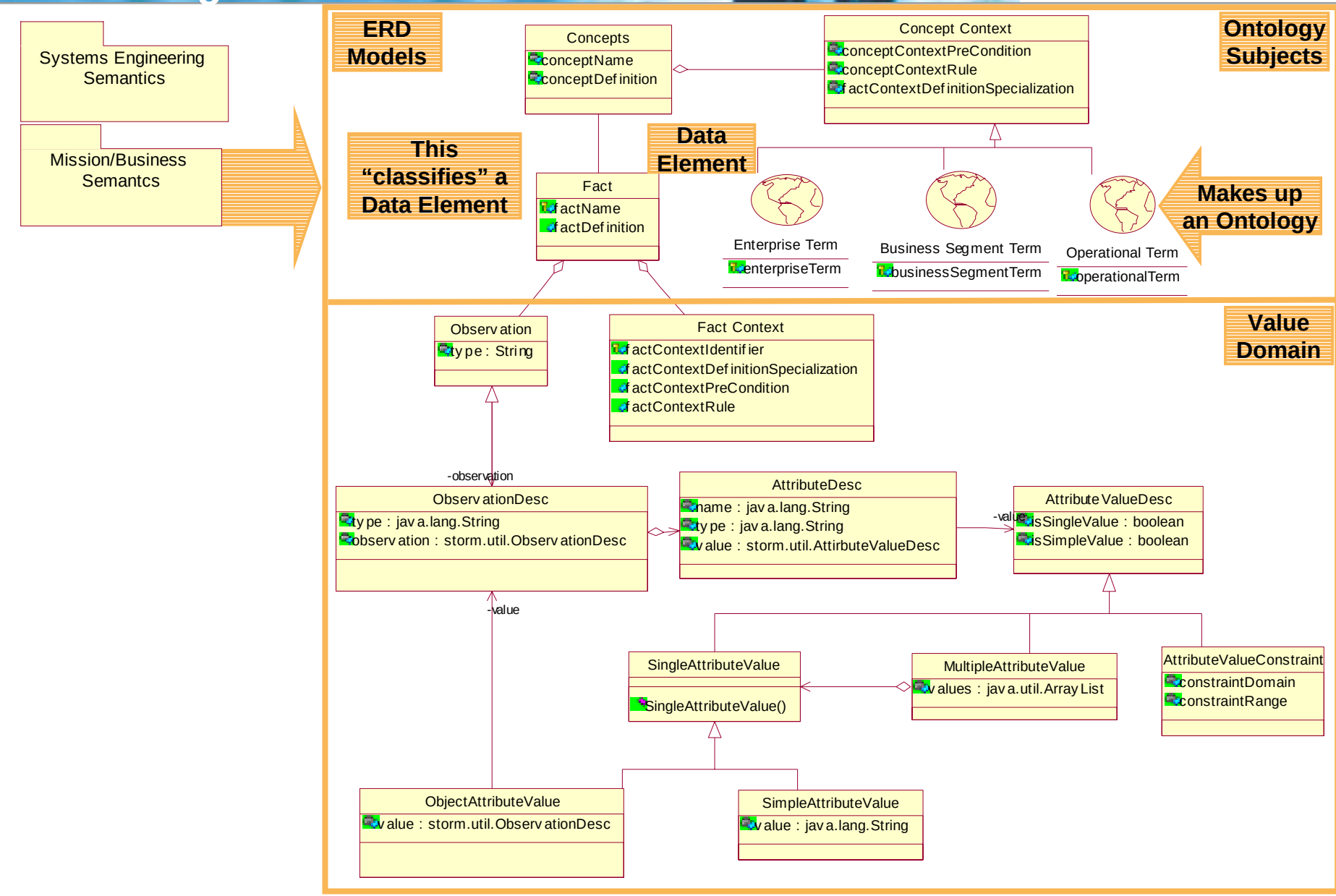
Semantics and Metadata View

EXPERIENCE. RESULTS.

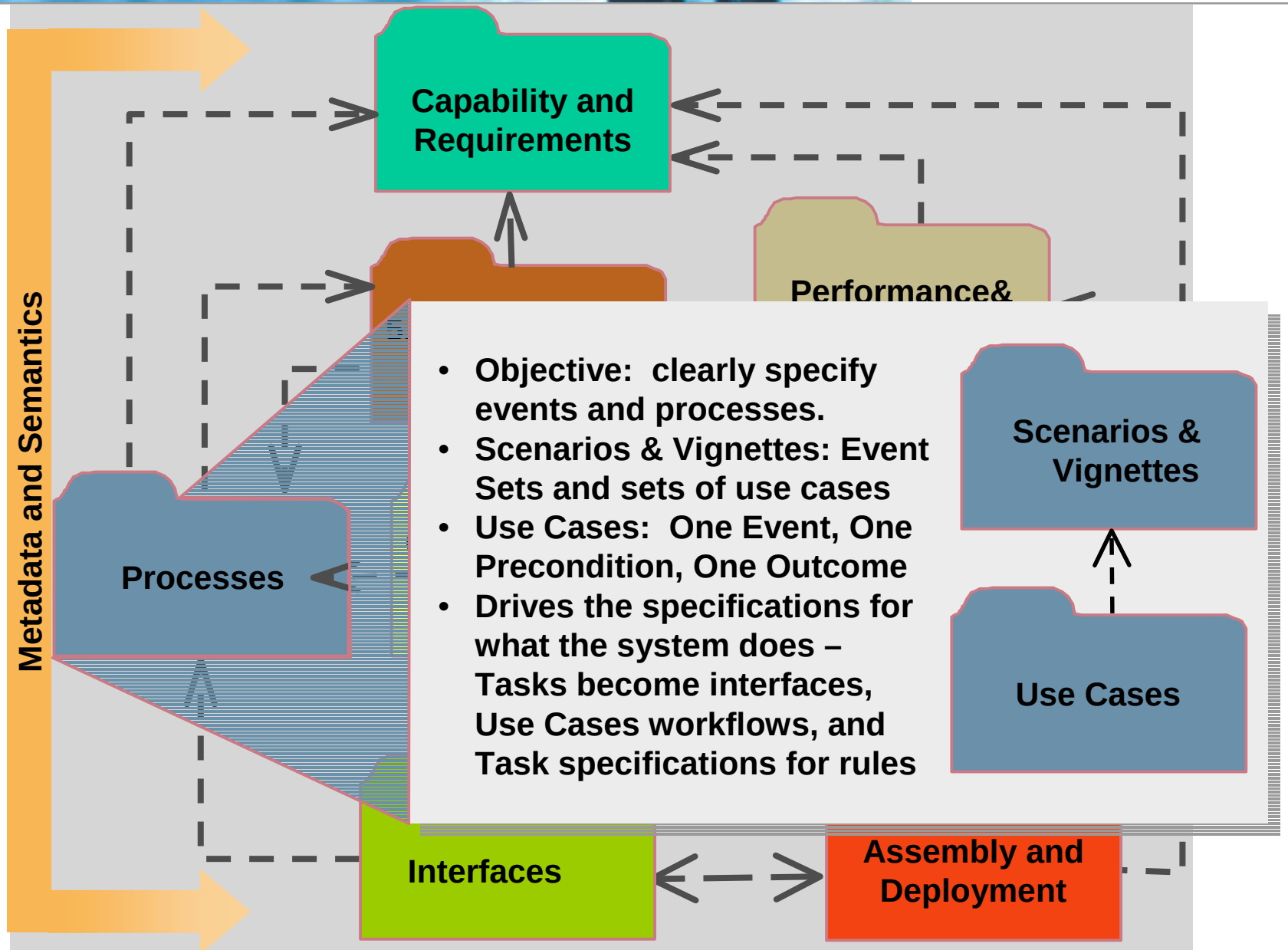


- Objective: clearly specify the Taxonomy and Ontologies of the Problem Domain.
- Observation: A structured way of classifying observable items.
- Ontology: The Taxonomy and context-specific ontologies that the rest of the views must abide by

Semantics and Metadata View: Integrated

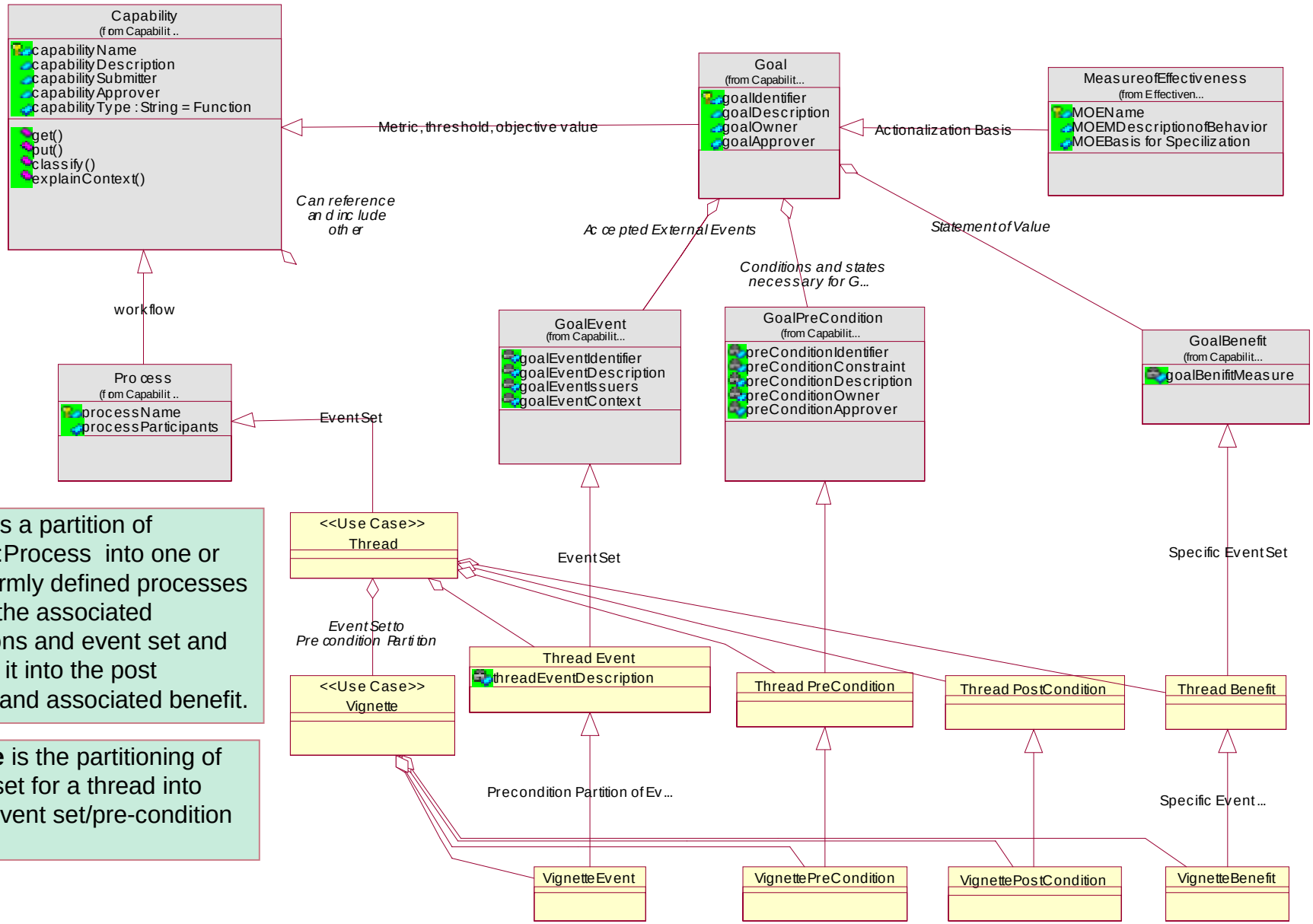


Processes View



Processes View: Scenarios and Vignettes

EXPERIENCE. RESULTS.



A **Thread** is a partition of Capability::Process into one or more uniformly defined processes that takes the associated preconditions and event set and transforms it into the post conditions and associated benefit.

A **Vignette** is the partitioning of the event set for a thread into separate event set/pre-condition sets.

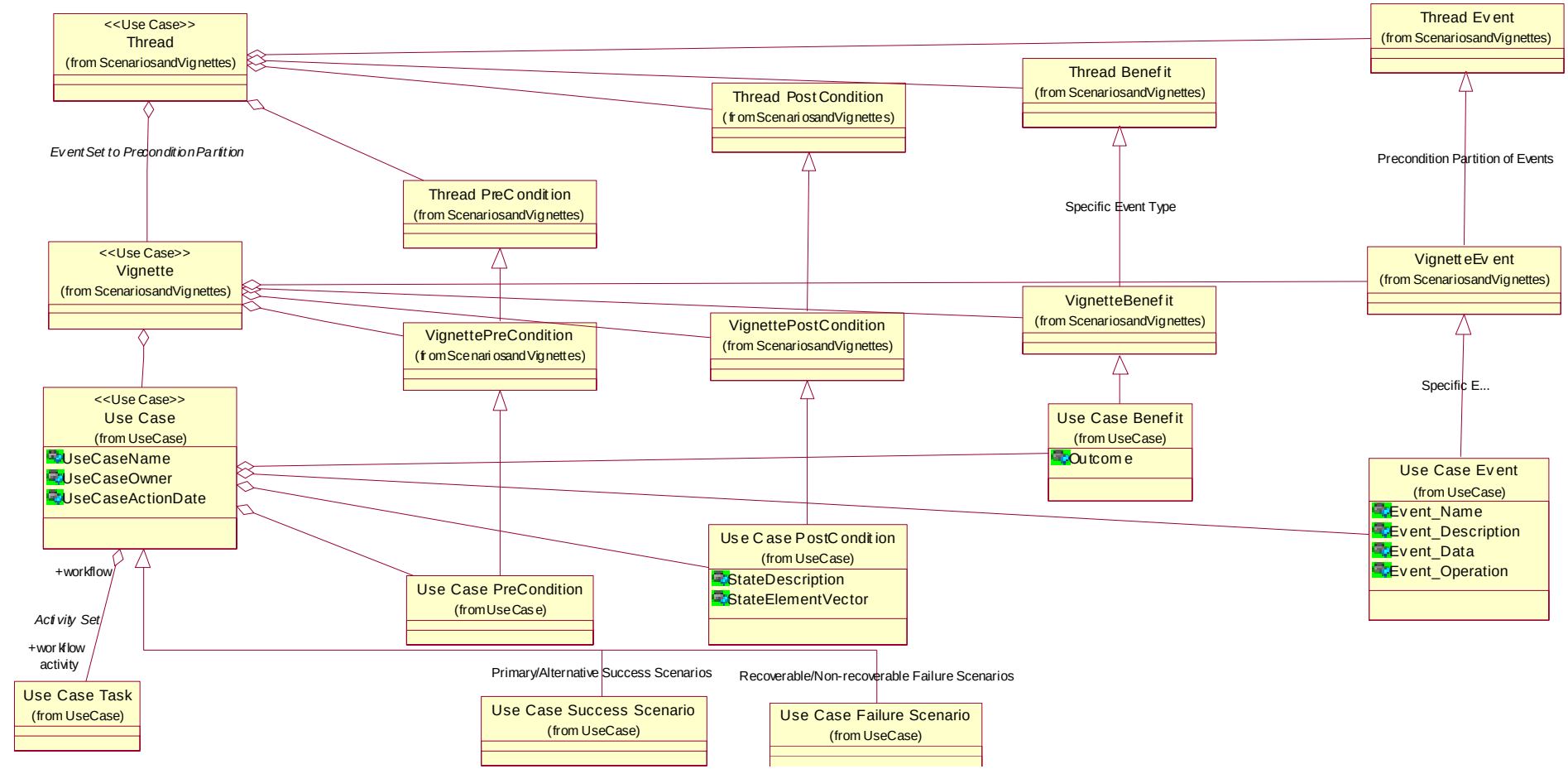
Processes View: Use Case

EXPERIENCE. RESULTS.

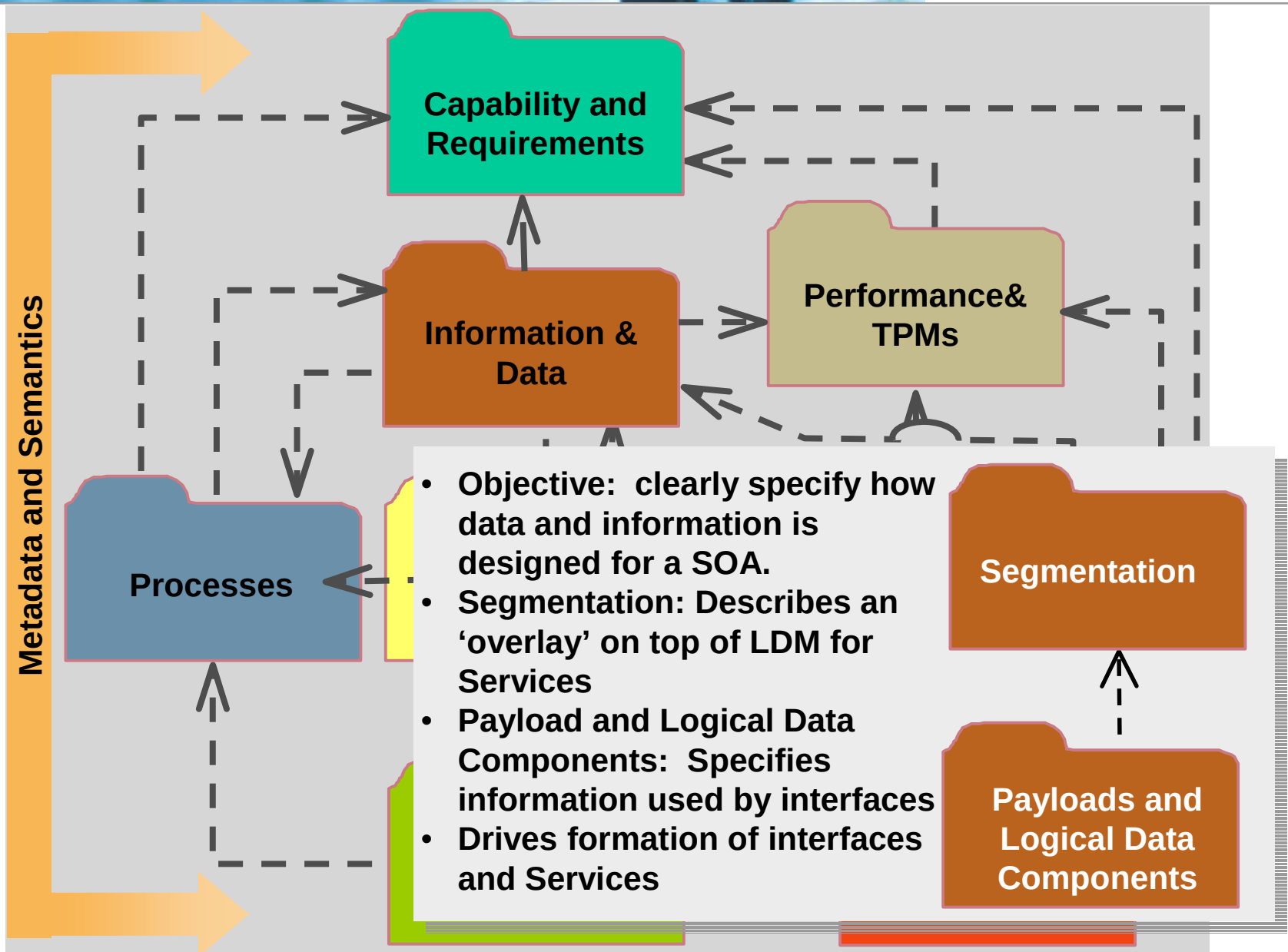
A **Use Case** represents the response to a single event resulting in a single benefit and having, on completion, a single PostCondition State.

A **Use Case Task** is the description of a step in the process necessary to deliver an outcome specified by the Use Case's Post Conditions. It represents a specification for a particular *feature* to be performed in the application context of the use case. In the case of software-enabled tasks, it represents the behavioral specification for enablement by some interface. Therefore, a use case task relates to a single interface specification, but interface specifications may satisfy one or more use case tasks.

Processes View: Integrated

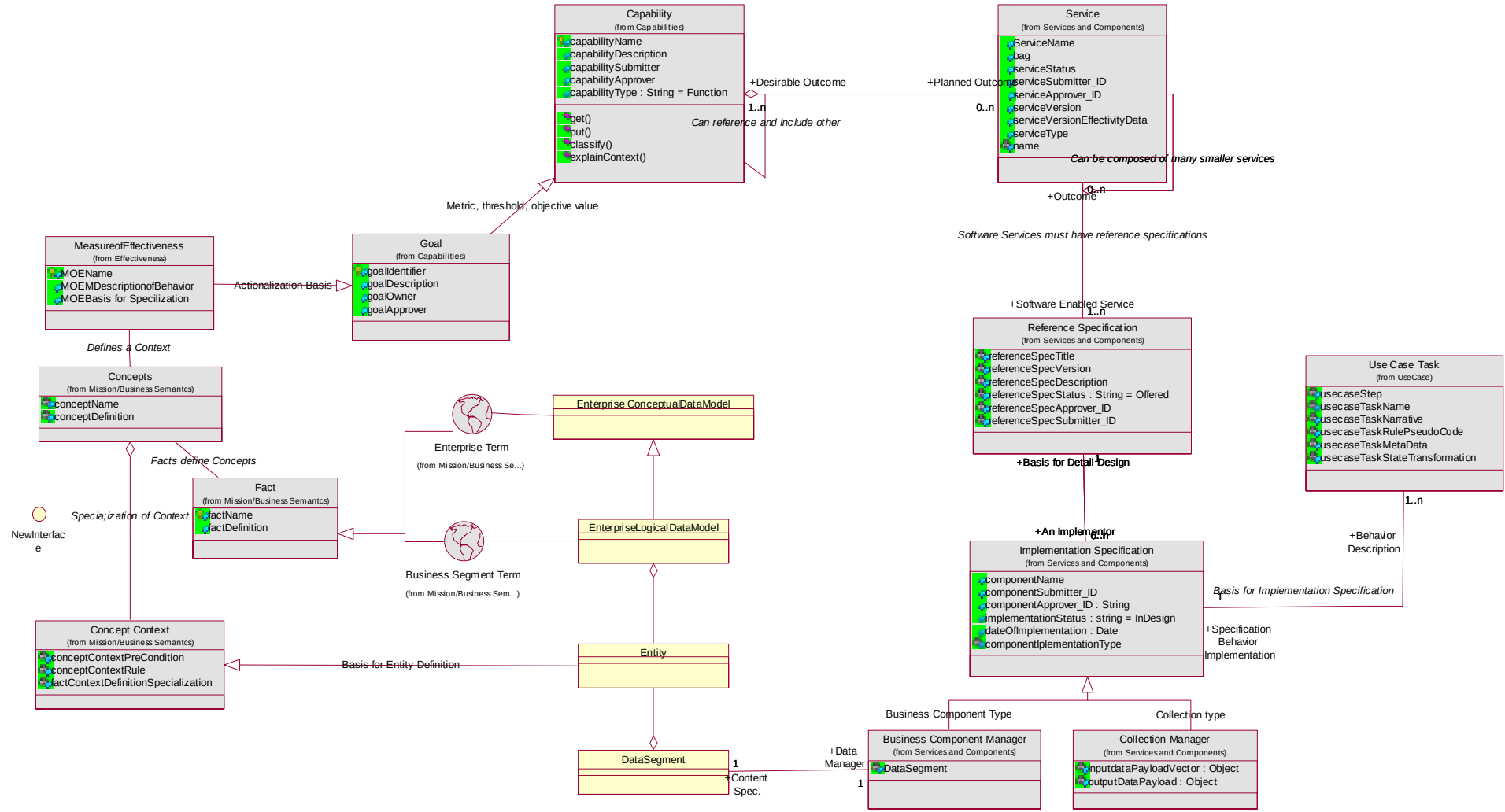


Information and Data View



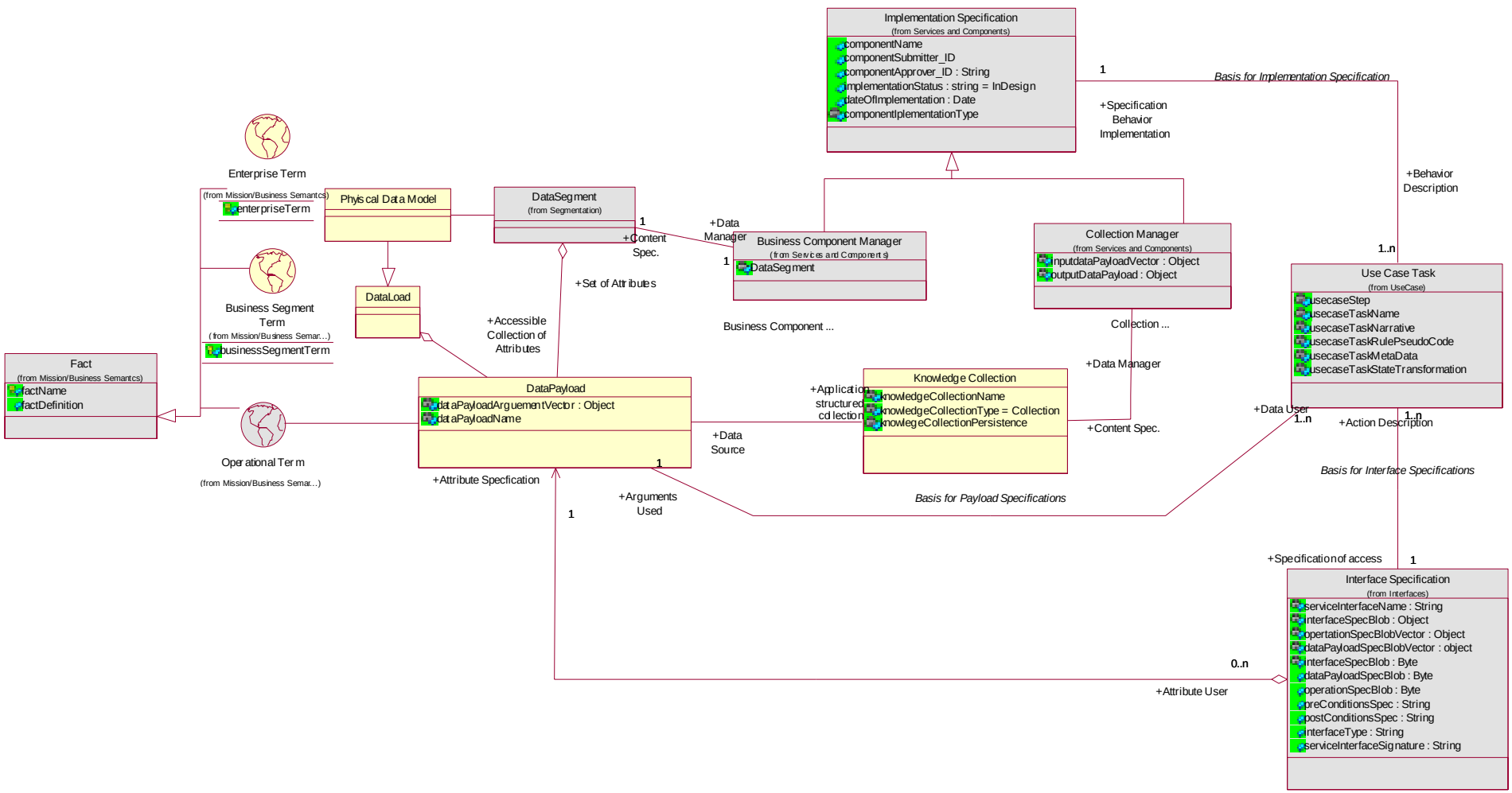
Information and Data View: Segmentation

EXPERIENCE. RESULTS.



Information and Data View: Payloads and Logical Data Components

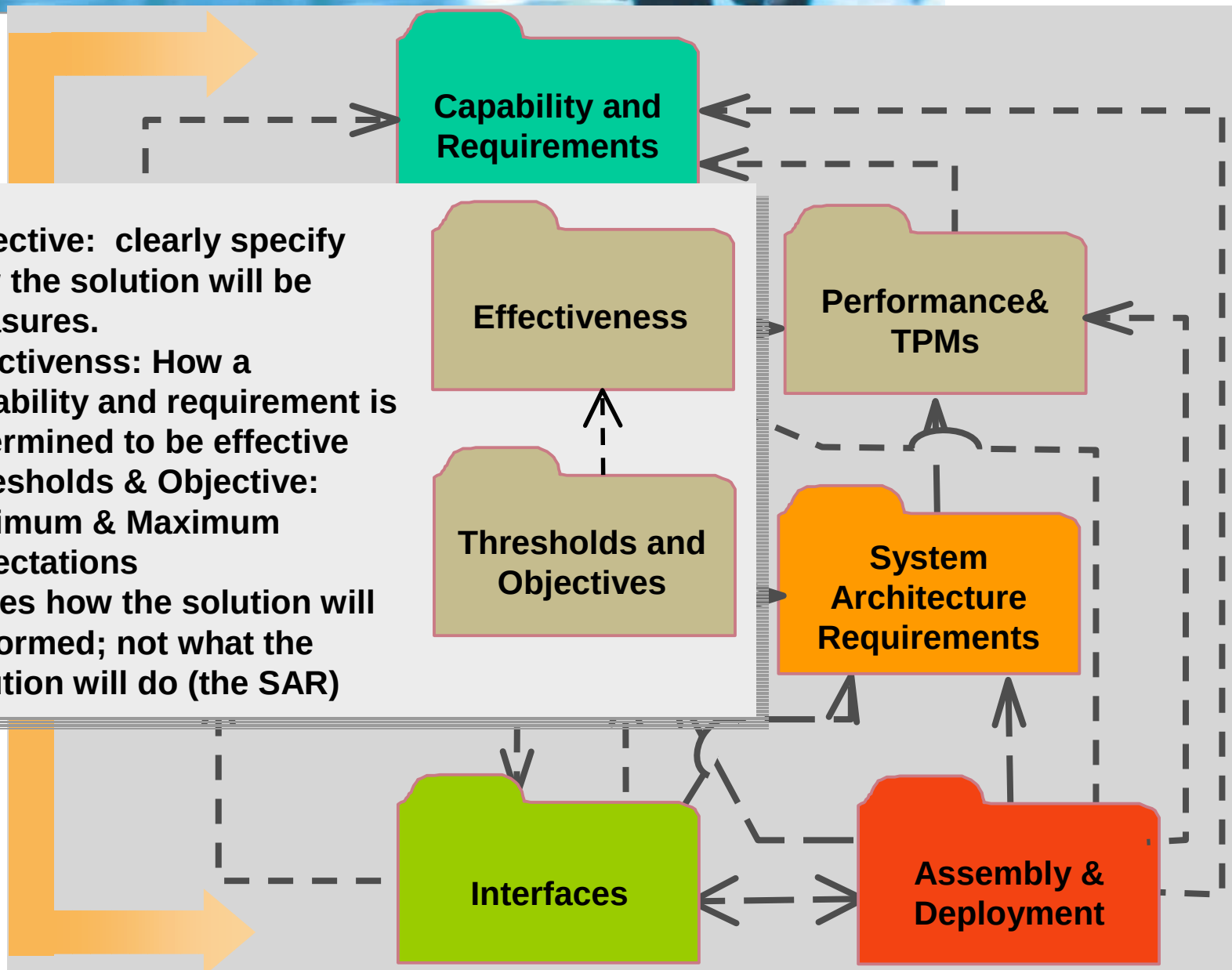
EXPERIENCE. RESULTS.



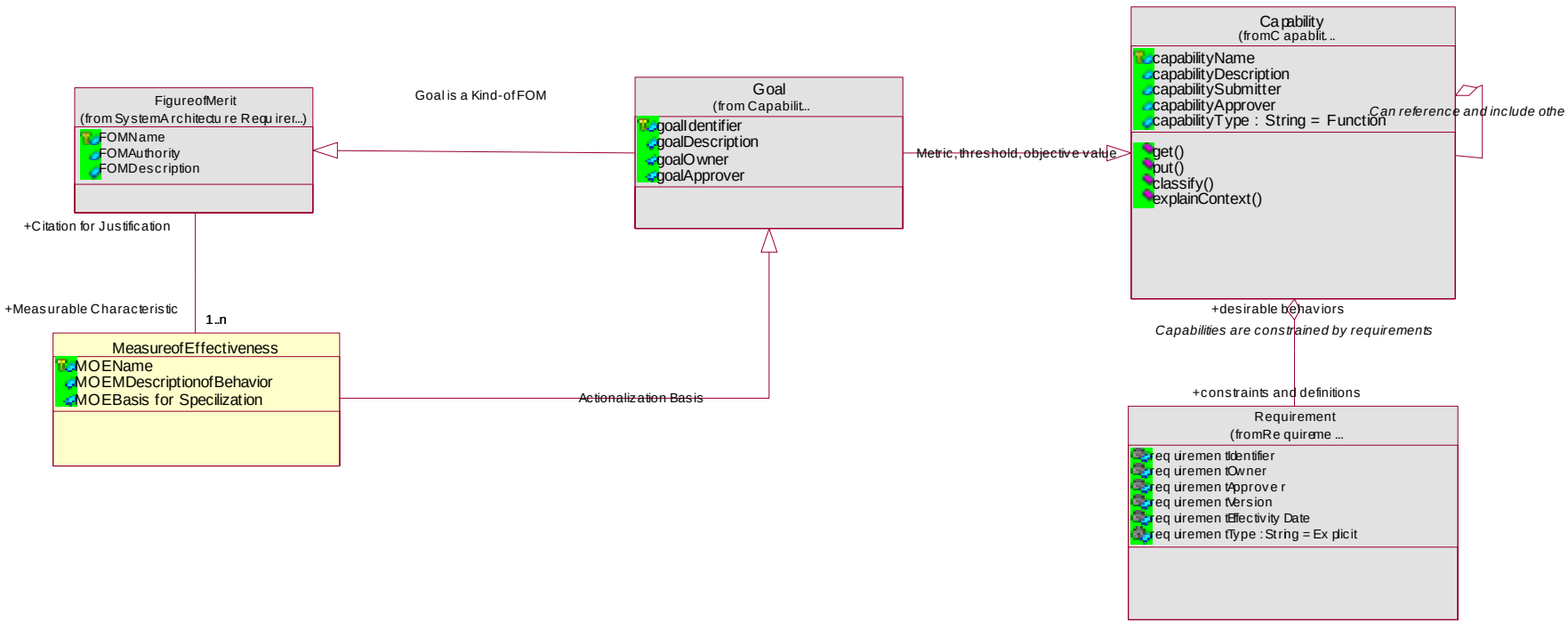
Performance and TPMS View

CSC
EXPERIENCE. RESULTS.

- **Objective: clearly specify how the solution will be measures.**
- **Effectivenss: How a capability and requirement is determined to be effective**
- **Thresholds & Objective: minimum & Maximum expectations**
- **Drives how the solution will be formed; not what the solution will do (the SAR)**

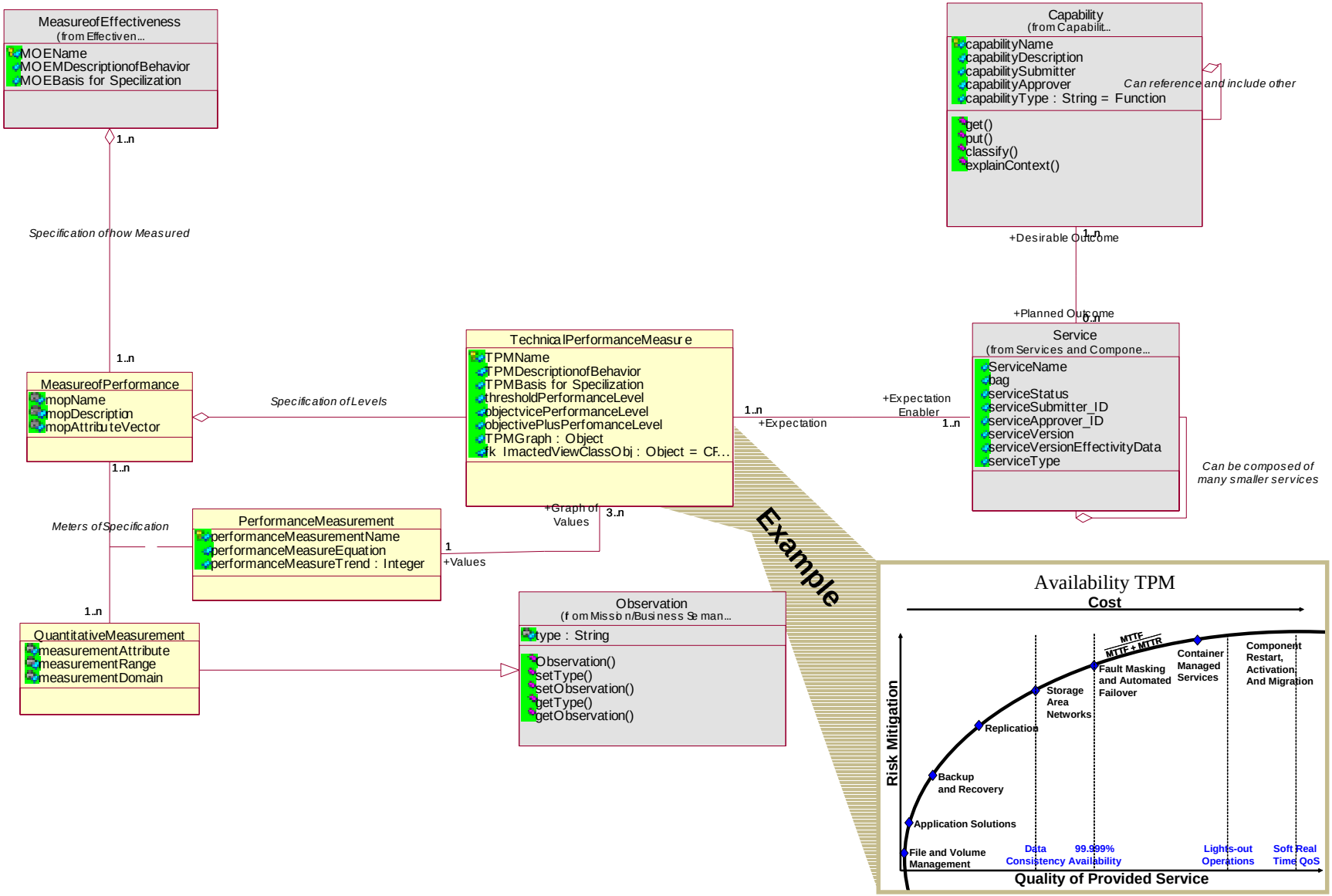


Performance and TPMS View: Effectiveness

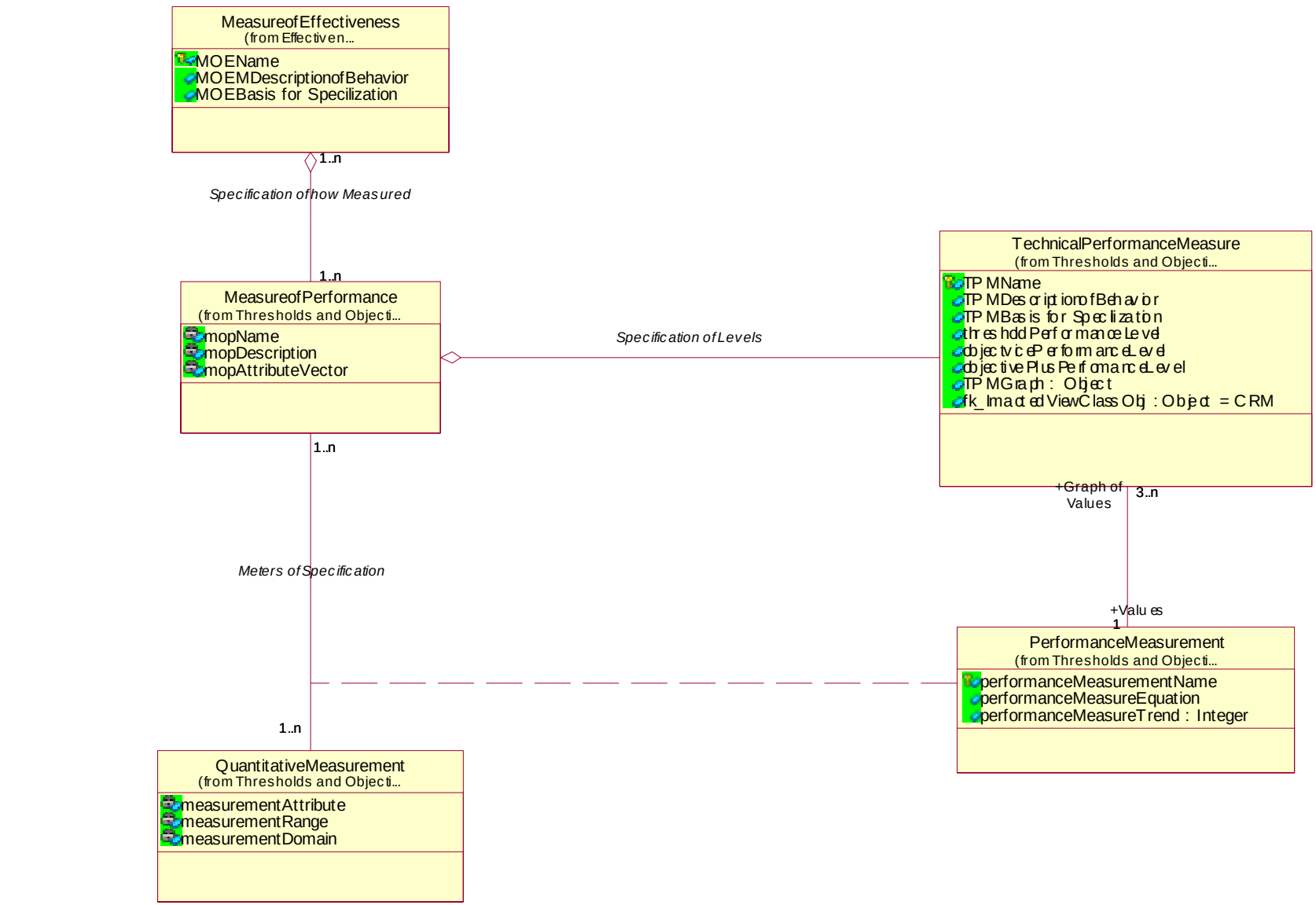


Performance and TPMS View: Thresholds and Objectives

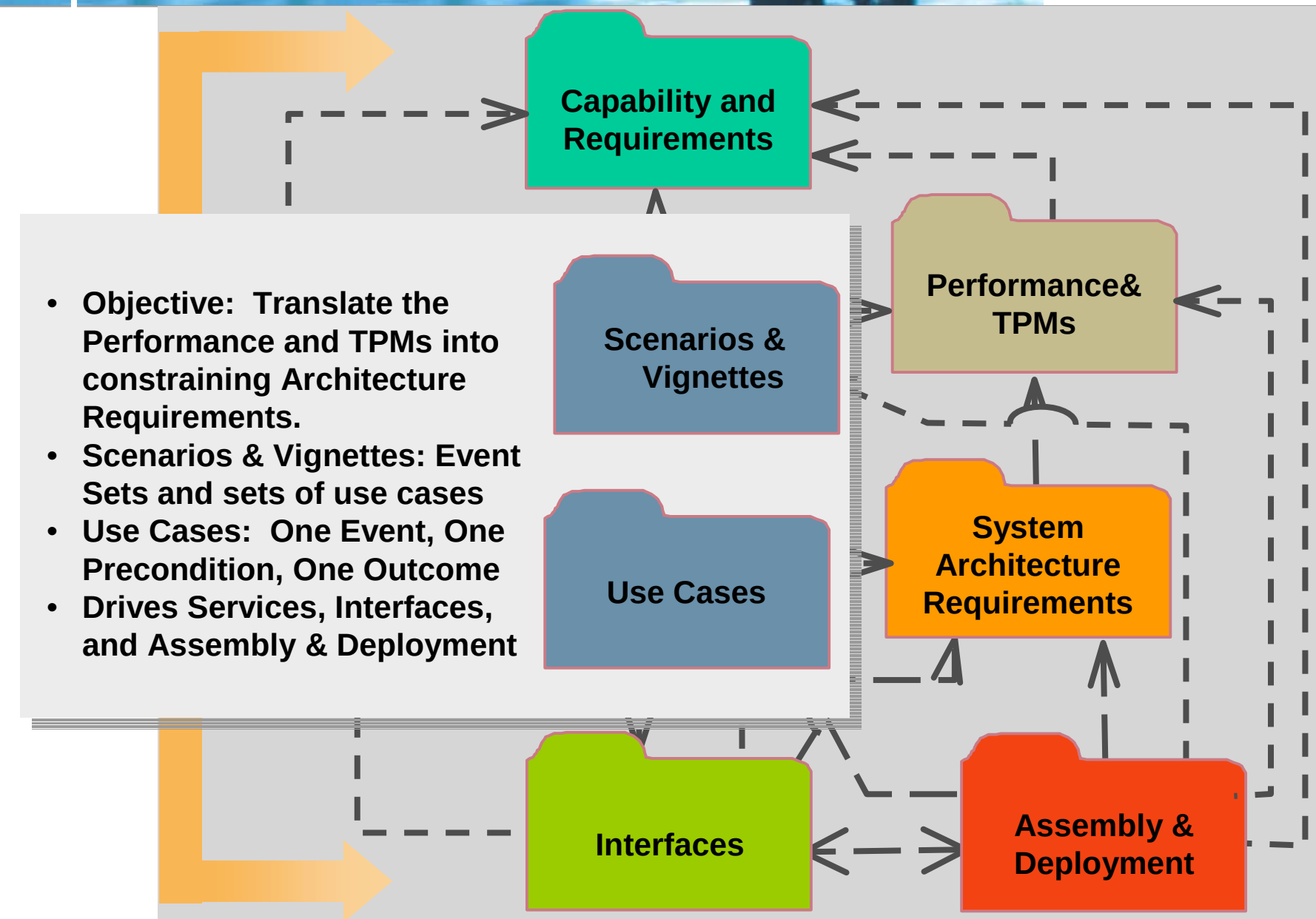
EXPERIENCE. RESULTS.



Performance and TPMs View: Integrated

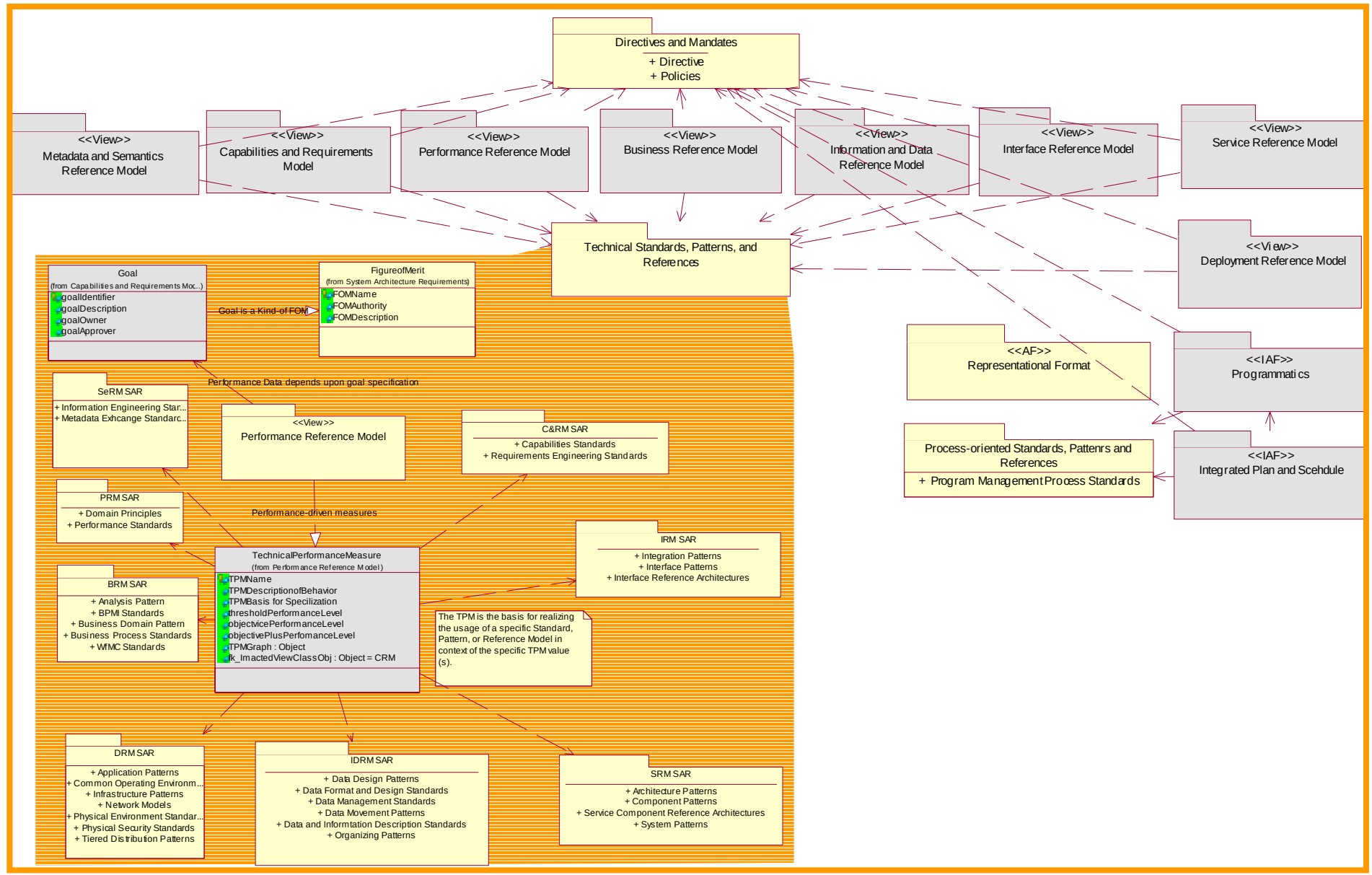


System Architecture Requirements

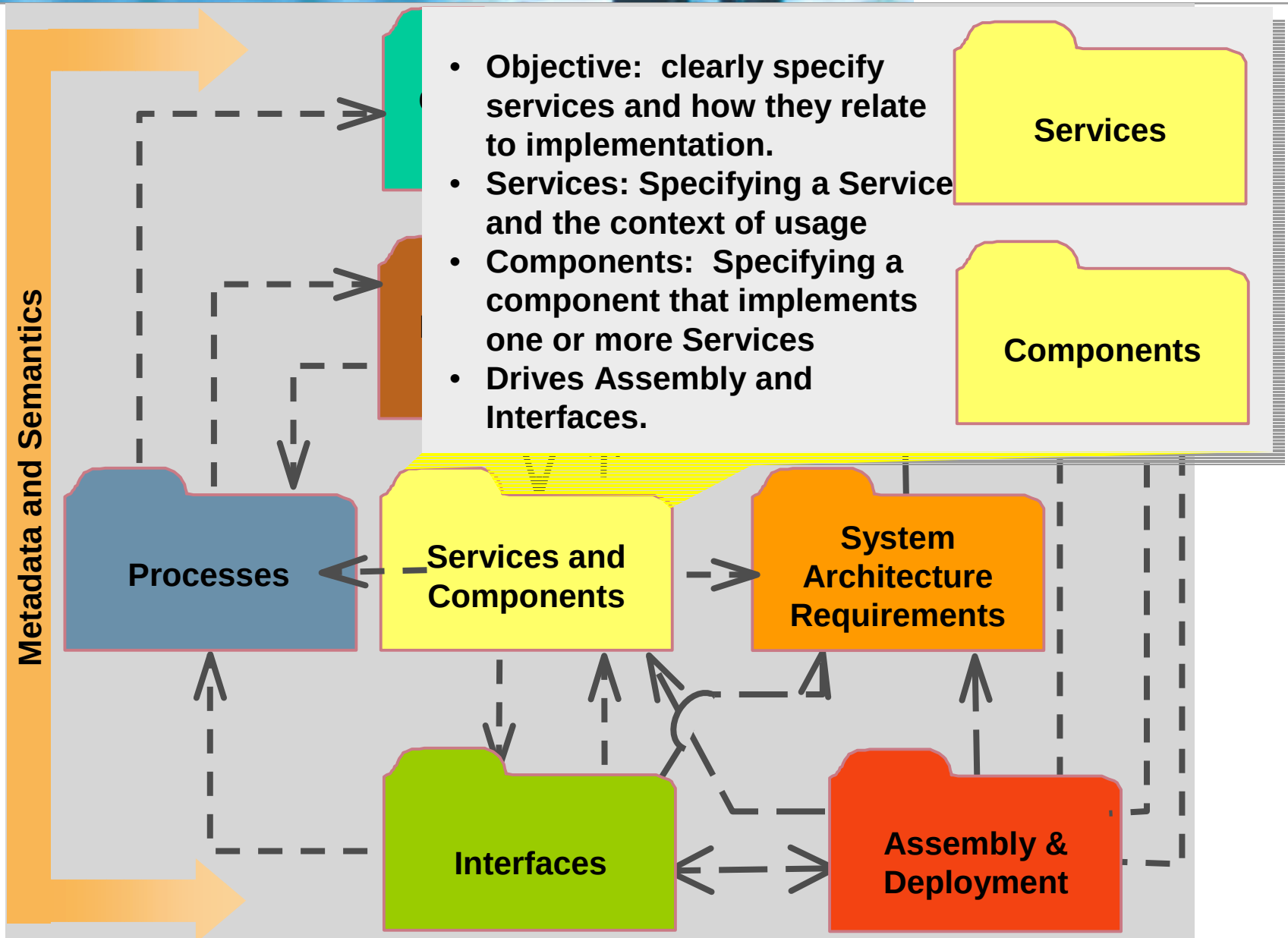


- Objective: Translate the Performance and TPMs into constraining Architecture Requirements.
- Scenarios & Vignettes: Event Sets and sets of use cases
- Use Cases: One Event, One Precondition, One Outcome
- Drives Services, Interfaces, and Assembly & Deployment

System Architecture Requirements



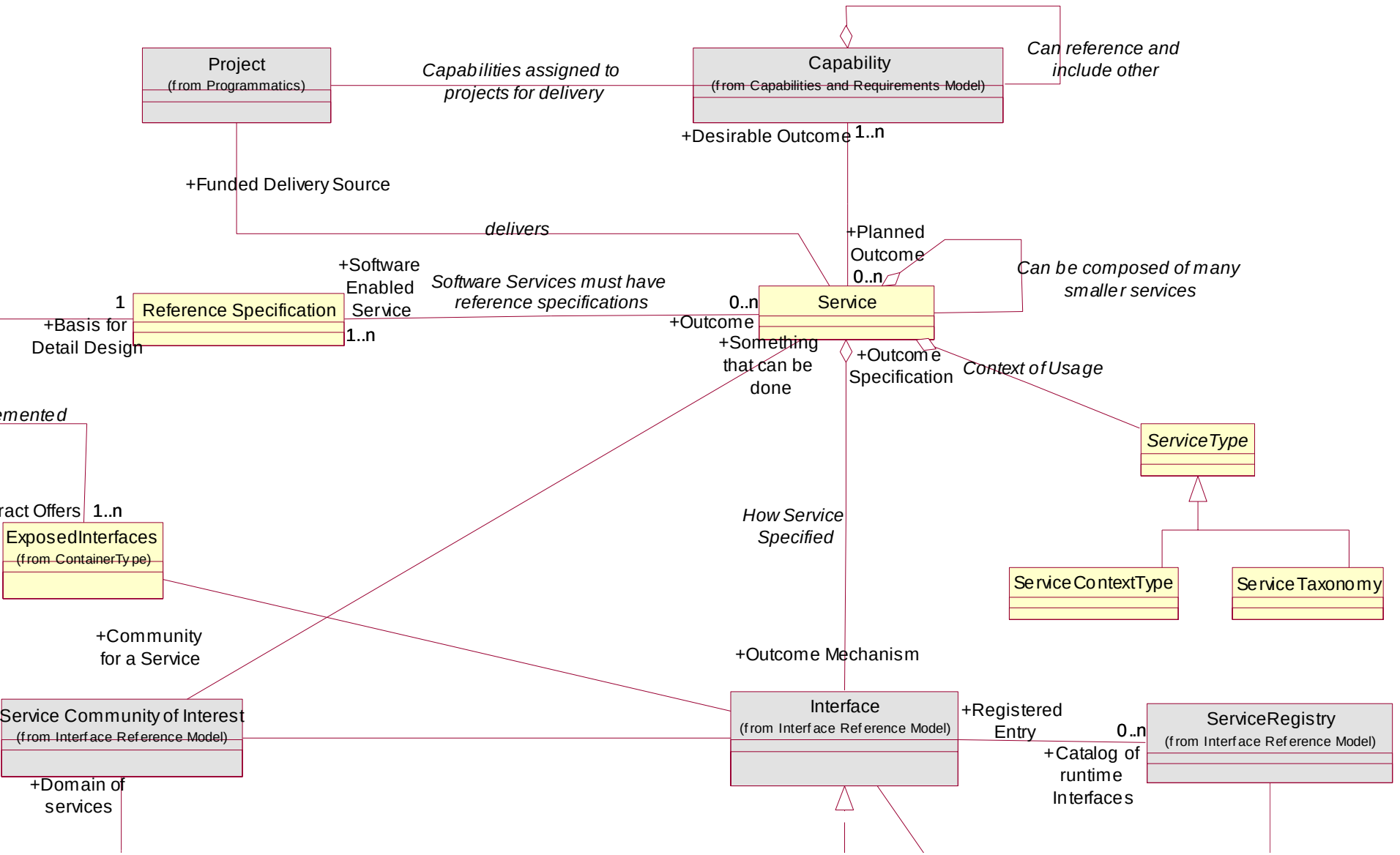
Services and Components View



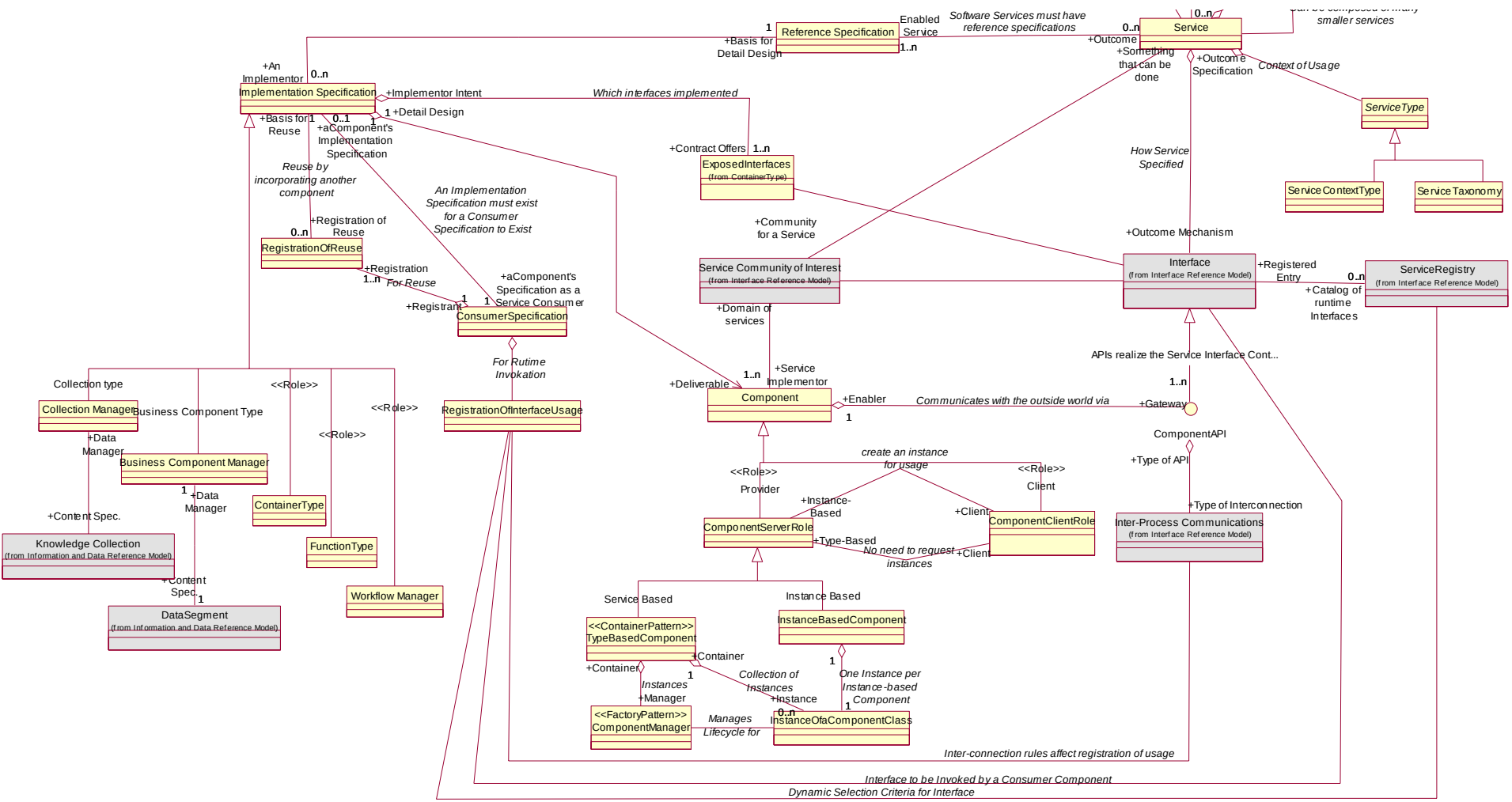
- Objective: clearly specify services and how they relate to implementation.
- Services: Specifying a Service and the context of usage
- Components: Specifying a component that implements one or more Services
- Drives Assembly and Interfaces.

Services and Components View: Services

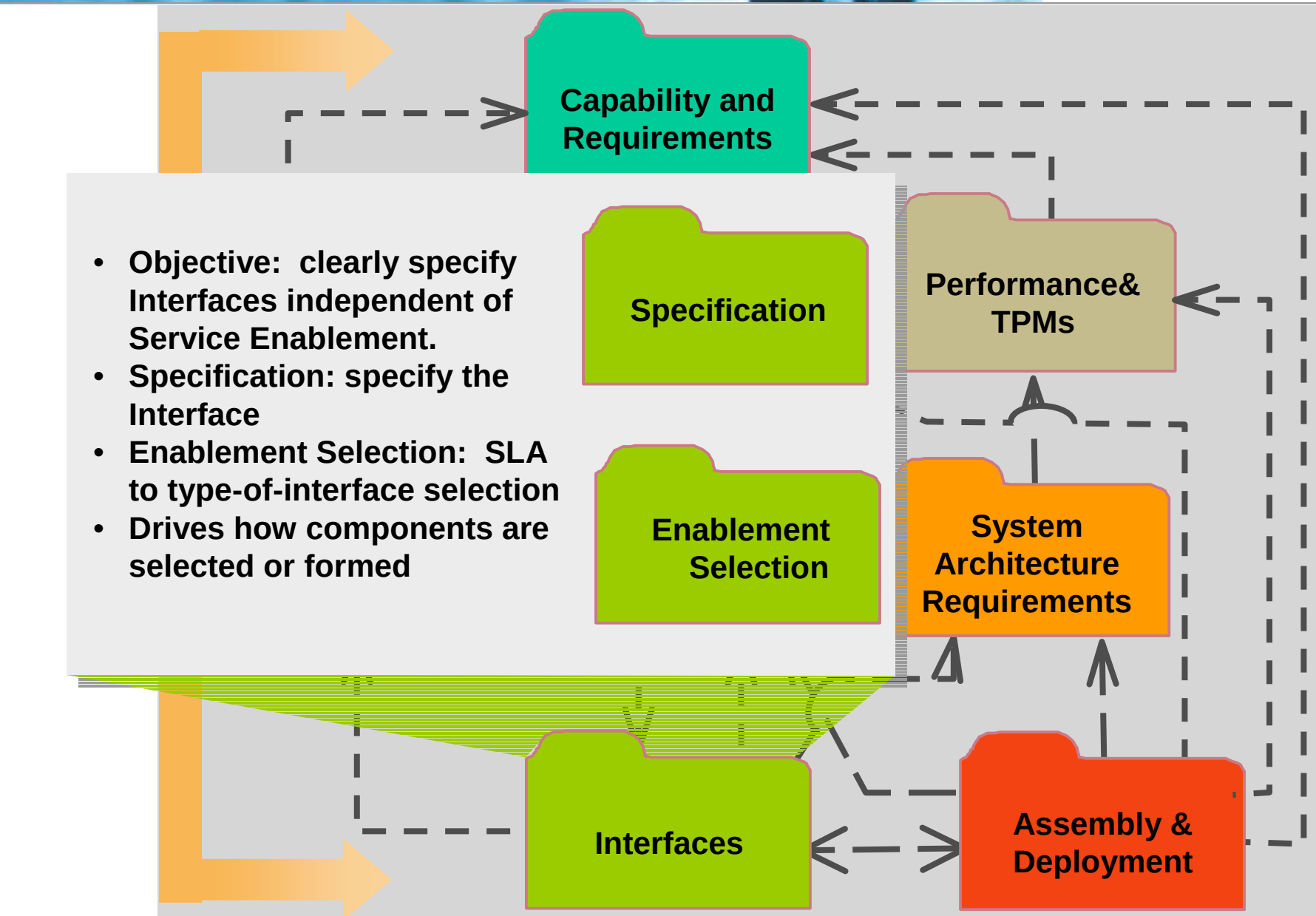
EXPERIENCE. RESULTS.



Services and Components View: Components



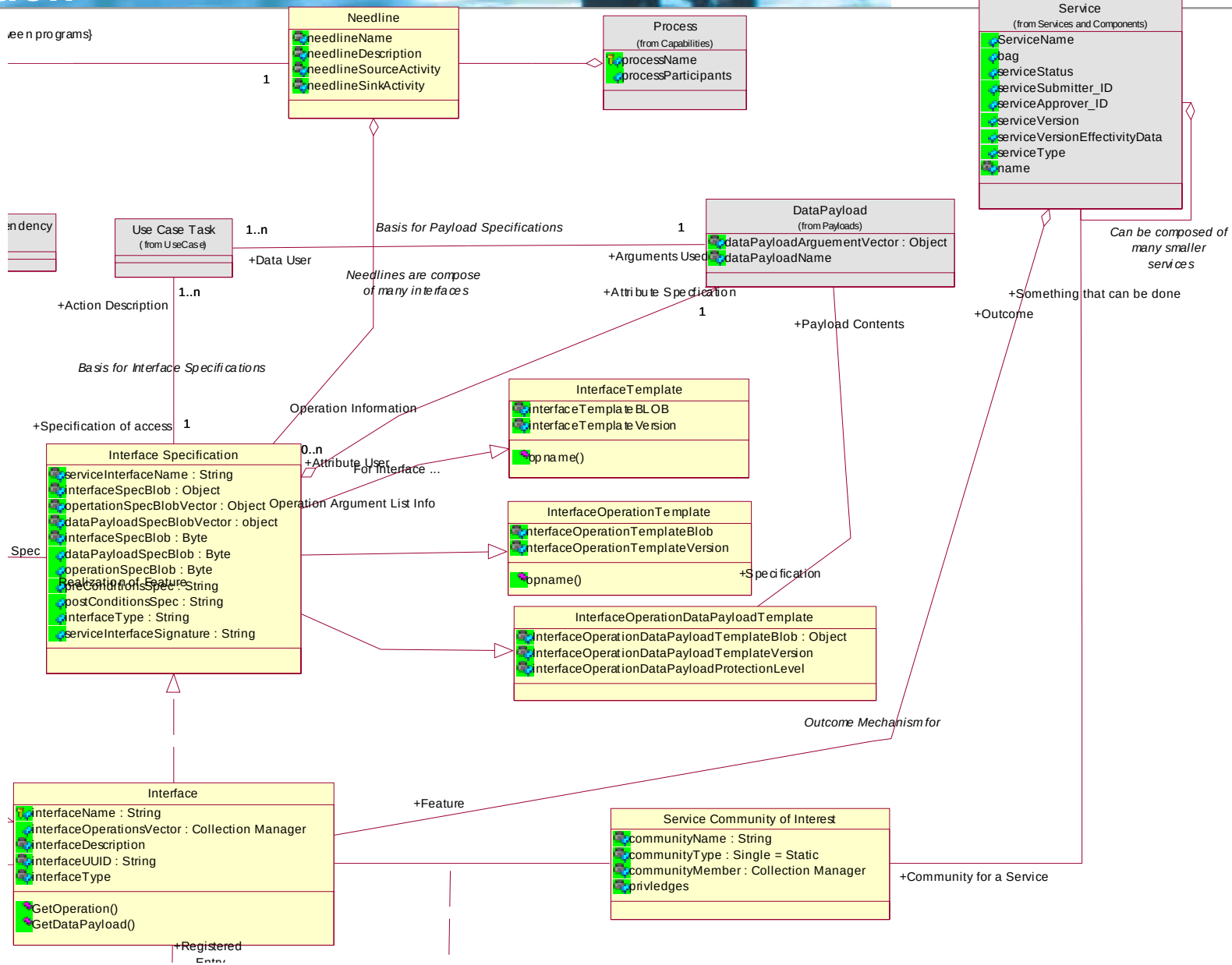
Interfaces View



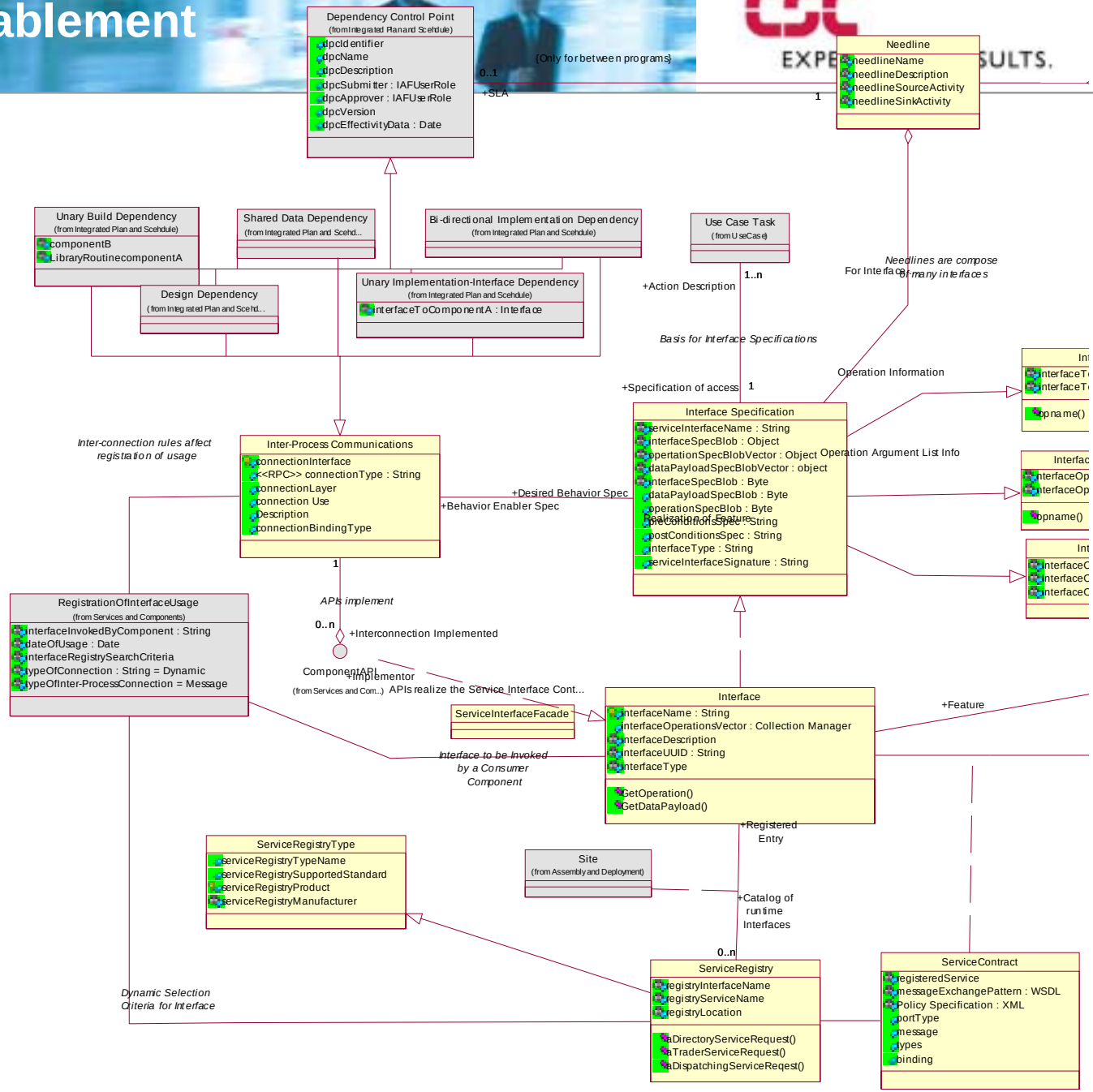
- Objective: clearly specify Interfaces independent of Service Enablement.
- Specification: specify the Interface
- Enablement Selection: SLA to type-of-interface selection
- Drives how components are selected or formed

Interfaces View: Specification

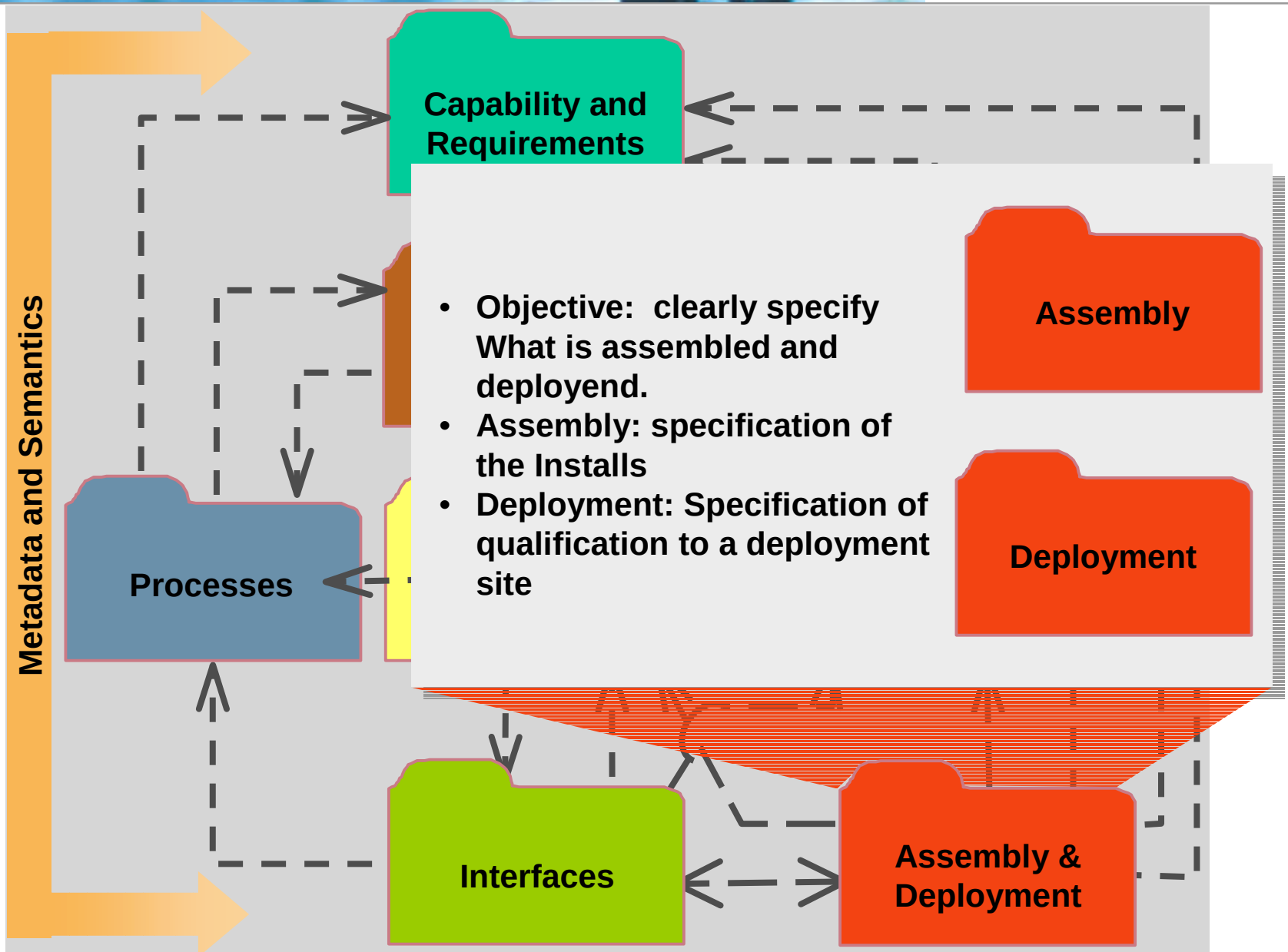
EXPERIENCE. RESULTS.



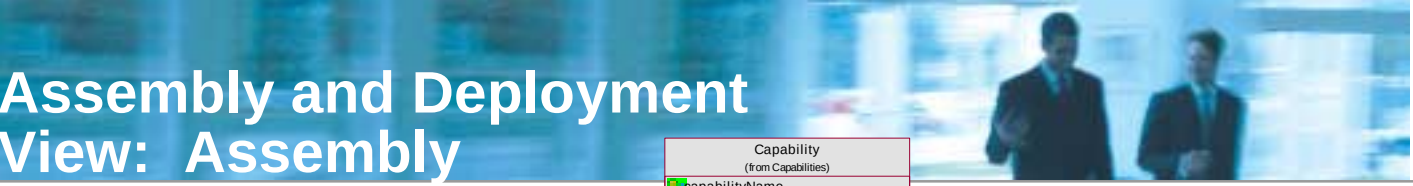
Interfaces View: Enablement Selection



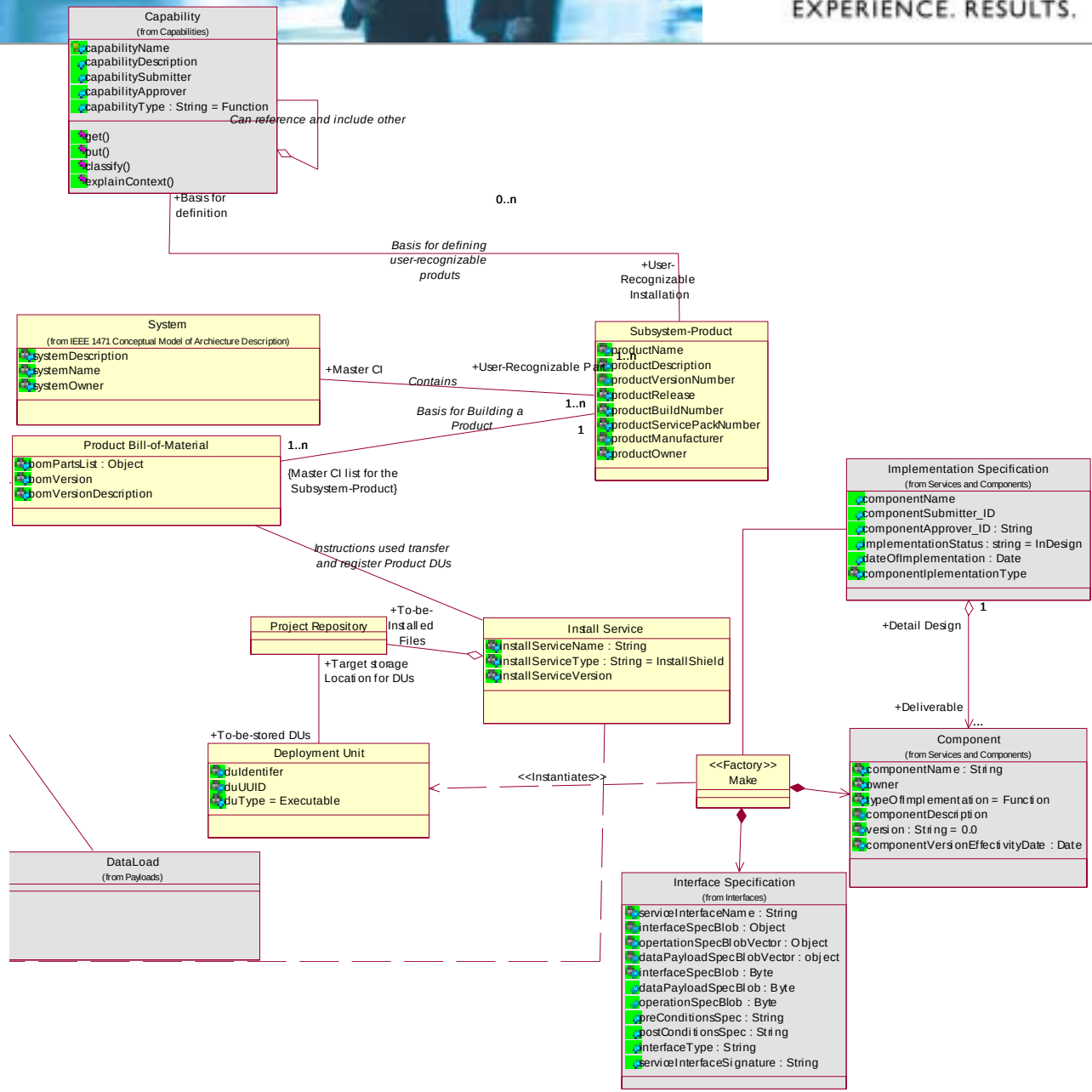
Assembly and Deployment View



Assembly and Deployment View: Assembly



EXPERIENCE. RESULTS.



Assembly and Deployment View: Deployment

EXPERIENCE. RESULTS.