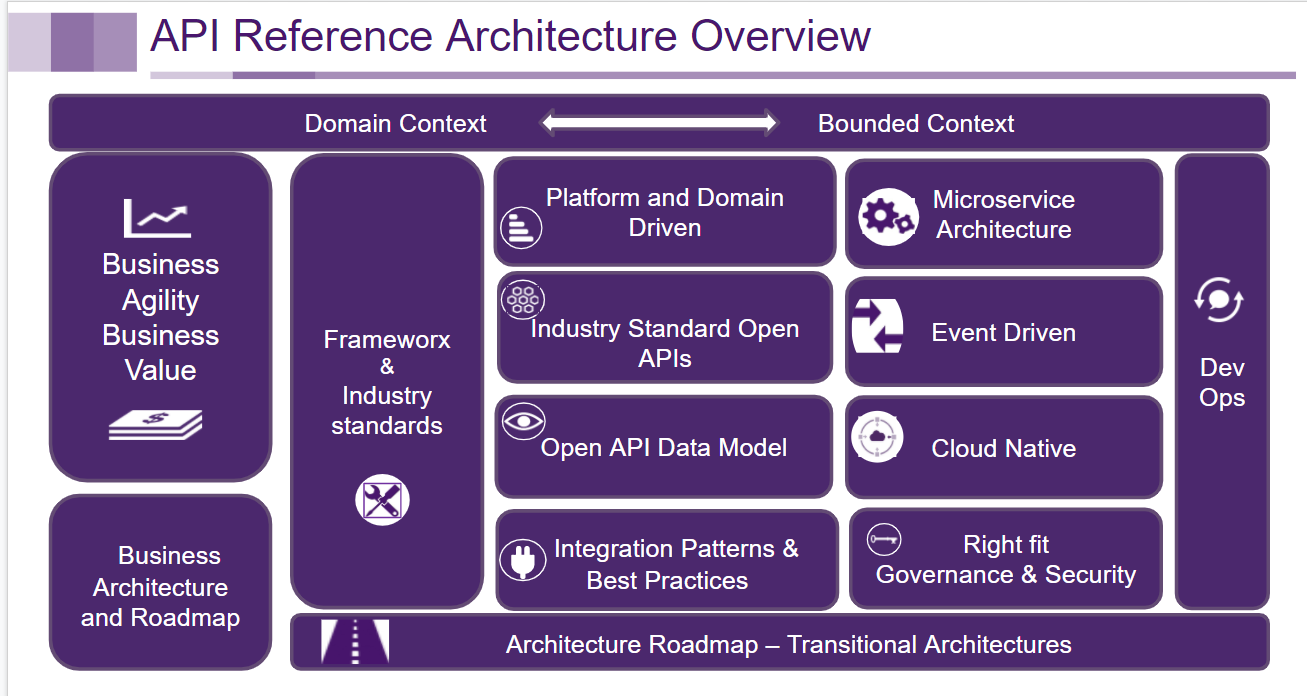
|  |  |
| --- | --- |
|  | **Overview**  The following section describes the Foundational Architecture Concepts of the target API Architecture. This section provides several architecture viewpoints from conceptual to physical. The purpose of the API Reference Architecture is to provide a view of the target state. The reference architecture provides a means of measuring the maturity of current architectures and the impact of proposed solution architectures in advancing enterprise architecture towards the target.  The underlying framework of the API Reference Architecture is based on the [TM Forum](https://www.tmforum.org/) Frameworks. TELUS has worked with the TM Forum to align the various TM Forum Frameworks: SID, TAM, eTOM and the Open API map and to move towards a more modern framework: a framework that recognizes the importance of describing the business as many separate domains which interact in order to provide and consume business capabilities. The business capabilities provided are the logical services and their definitions.  The API reference Architecture is comprised of the following components:   * Domain Frameworks * Platforms * Architecture Principles * Domain Context Mapping * Industry Standard Open APIs * Open Data Model * Integration Patterns * Design Best Practices * Microservice Architecture * Event-Driven Model * Cloud Native * Security |

The target API Reference Architecture is based on a number of guiding principles, architecture patterns and conforms to industry standards enabling interoperability between TELUS and other parties. Recommended integration patterns and design best practices are modeled and their application provided. The target API Architecture is, in part, enabled by the implementation of a Microservice Architecture. DevOps is fully enabled in the target architecture.

Modeling the business architecture of a specific area of concern, or domain context, provides a clear description of the business drivers, goals, outcomes and use cases. This view is required to assess whether the reference architecture and the implemented solutions support the needs of the business.



We recognize that a target architecture cannot be achieved without one or more transitional steps, therefore the API Reference Architecture describes a number of transitional patterns which can be applied to achieve the desired target architecture. These transitional steps are realized in a number of architecture plateaus.

# Domain Driven Design (DDD)

**Domain Driven Design** is a software development technique and set of practices, which includes strategic, philosophical, tactical and technical elements to implement a software based on the business domain.

Domain driven design (aka. DDD) aims to ease the creation of complex applications by connecting the related pieces of the software into an ever-evolving model.

DDD focuses on three core principles:

* Focus on the core domain and domain logic.
* Base complex designs on models of the domain.
* Constantly collaborate with domain experts, in order to improve the application model and resolve any emerging domain-related issues.

## Why use Domain Driven Design?

### Breaking down the silos

DDD, when applied, can help break down the silos inherent in traditional enterprise architectures that have developed often in a system-centric manner, driven by requirements specific to that system, by time, and by resource constraints. DDD takes the domain perspective rather than a single system perspective. A domain can contain any number of applications, processes, people, data, and systems bound into a context. Well-defined interfaces to the domain expose the information a domain wants or needs to provide in order to fulfill its relationship to other domains within the enterprise.

### Running away from Big Ball of Mud (BBoM)

“A Big Ball of Mud is a haphazardly structured, sprawling, sloppy, duct-tape-and-baling-wire, spaghetti-code jungle. These systems show unmistakable signs of unregulated growth, and repeated, expedient repair. Information is shared promiscuously among distant elements of the system, often to the point where nearly all the important information becomes global or duplicated.”

The BBoM can be created without writing a line of code, by designing functionality without taking into consideration the bounded context.

For TELUS architecture, we need to avoid this BBoM, and the right approach is using DDD. TELUS will have a lot of different and evolving needs, but there is value in strategically designing the new features to produce software that adapts to business rules, providing a good foundation for scalability and having the ability to reuse the software. For example, Product Catalog may receive new requirements but they need to be analyzed to the point of mapping the existing capabilities to enrich the API with the new features without breaking any existing functionality.

There are different challenges to get DDD going and avoid BBoM:

1. Invest time and resources in creating a Roadmap to analyze all the business processes involved, context diagrams, business rules, stakeholders, current state versus target state.

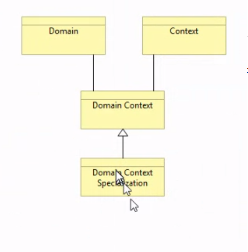
2. Integrate to and develop the TMF Open APIs, which provide standards to TELUS functionalities, every Open API maps to one bounded context.

3. Need for robust and fast performance API, as the services are reused and deliver all the functionality of the bounded context.

## Concepts

### The Domain

A Domain is a collection of Aggregate Business Entities associated with a specific management area, where you group all the business-related activities, actions, and logic, so the systems can be easily reused, understood, highly cohesive and practical to work with. Domains, domain context and bounded context are used to provide a clear scope of responsibility for systems, APIs and the data they manage.



### Domain Context Specialization

Domain context specialization recognizes that within a domain there may be a need for a specialization of the context. Business context is by definition within a domain. In Party for example we consider Supplier as a context. We can then specialize within the context. Within a domain context specialization you can extend but also remove properties in the generic context but still remain within the domain context.

### The bounded context

DDD helps break down complex problems through the creation of models that have a specific bounded context. Bounded context is a conceptual limit where a domain model is applicable. In any enterprise there are multiple models, therefore - specifically bounded - the context of each allows clear separation of roles and well-defined relationships between models. DDD advocates for the collaboration between the business and IT and uses business use cases to drive the design.

As you try to model a large domain, you will have great difficulties, because different groups of people will use subtly different terms and sentences. That means that any use of that vocabulary outside of that limit will probably mean something different and it will become a different context.

#### Ubiquitous language

A bounded context consists of a model that is described in its own ubiquitous language. The bounded context is where the ubiquitous language has its unambiguous meaning. Outside of that context, the sets of terms that comprise the ubiquitous language can have other meanings.

Each bounded context has its own ubiquitous language.

Using a ubiquitous language is key and assists in modeling domains to better understand what the bonded context, and the relationship between domains, is.

### Context Maps

Context Maps help in understanding the whole project, being able to show the relationships between the different Bounded Contexts.

It is extremely important to understand the relationship between Bounded Contexts so that you can build a domain model correctly.

The API Reference Architecture advocates for the application of the principles of domain driven design in order to determine the bounded context of each domain within the enterprise.

### Entity

Objects that have a distinct identity that runs through time and different representations. You also hear these called "reference objects". An object that is identified by its consistent thread of continuity, as opposed to traditional objects, which are defined by their attributes.

### Value Object

An immutable (unchangeable) object that has attributes, but no distinct identity. Objects that matter only as the combination of their attributes. Two value objects with the same values for all their attributes are considered equal

### Aggregate

An aggregate is an encapsulation of entities and value objects which conceptually belong together. It also contains a set of operations which those domain objects can be operated on. An aggregate will have one of its component objects be the aggregate root. Any references from outside the aggregate should only go to the aggregate root. The root can thus ensure the integrity of the aggregate as a whole. Two aggregates must not have overlapping invariants and entities which may or may not contradict with each other. The consequences of this would be that there would potentially be inconsistency across your data storage. Aggregates are the basic element of transfer of data storage — you request to load or save whole aggregates. Transactions should not cross aggregate boundaries. Examples of aggregates will be order, related party, invoice, trouble ticket, etc.

### Domain driven design principles and their relationships



## Applying Domain Driven Design

Every developer, designer, analyst, or architect must understand the domain driven design principles and relationships. TELUS has used the Domain Driven Design principles to create a domain context map of domains, bounded context, and aggregates. Using the context map, TELUS has determined the applicable TM Forum Open API specification for each.

Architects should use the DDD approach, principles and the [TELUS Domain Context Map](https://drive.google.com/a/telus.com/open?id=1yUNdmekHoBKbntIFZzgmP3Qvkj6NyLsxudZOmM1lhgI) to determine the scope and context of proposed APIs.

### Validate Design via Standardized Documentation

API Roadmaps, system diagrams, context maps, and any other documentation must follow a standardized format in order to facilitate clear communication between stakeholders.

## Business Architecture

***Please refer to*** [***Business Architecture Reference Viewpoints***](https://drive.google.com/a/telus.com/open?id=1tc_1Xhxg4NVouuhzmdelcQB2mmyCVCDAKNaRyX2CZ_I) ***for more detail.***

Business Architecture can be described as a blueprint of the enterprise that provides a common understanding of the organization and is used to align strategic objectives and tactical demands. Business Architecture can help express the capabilities, end to end value streams, organizational structure and the relationships between business views.

As the Domain Driven Design approach advocates for modeling the business use cases, creating business architecture models helps express the business context and breaks down the complexity of the problems. The business architecture models assist in clearly stating the scope of the bounded context. As the principles and concepts of the API Reference Architecture are applied it is critical to have a clear view of the business architecture in order to ensure the domains have a clearly defined separation of concerns and the solution architectures are driven by the business requirements and the reality of the business use cases. Any solution architectures should be directly tied to the defined business drivers and support the defined business outcomes.

## Portfolios

A portfolio’s systems should fit within one domain. If, when analyzing the collection of systems, more than one domain is represented then the definition of the portfolio should be reviewed and potentially decomposed. If a portfolio’s systems cross domains then the business capabilities will also cross domains and will not fit any one business context. This cross-domain business context and complex integration between many applications providing the same business capabilities has resulted in the complex integration of our current landscape and must be avoided. Each TELUS portfolio is contained within one domain or represents a relationship between domains.

## Roadmaps

* what they are
* why we do them
  + plan, track, and measure that we’re achieving our goals
  + guiderails for projects
* strategic vision of portfolio
  + The architecture portfolio roadmap demonstrates the applications of the patterns and principles of the API Reference Architecture presented as a blueprint for transitional architectures. The target architecture is used to govern, assess and measure the architecture of a domain and portfolio
  + current to target, with consideration of enterprise outcomes & goals
  + ties back to [definitions](https://docs.google.com/document/d/1gGbQ_Ea0zLxEE0kOLaVz7z0IW9zO--GpKkqvJppbcvc/view#bookmark=id.5kskea80spat) and enterprise outcomes

# Architecture Repository

TELUS has implemented an architecture repository. Information about the architecture repository can be found at [http://go/mega](http://go.telus.com/mega).

All architectures must be stored within the Architecture repository. The TELUS Capabilities models can be found within the architecture repository. The repository can be accessed at [http://go/mega/prod](http://go.telus.com/mega/prod).

API Roadmaps should be created and maintained by the API Architect. API Roadmaps must be reviewed by all API Roadmap stakeholders as defined with the roadmap. API Architects can use the sample API Roadmap template. [API Roadmap information](https://drive.google.com/drive/u/0/folders/0AIC2J2Sd9Y3vUk9PVA)

API Roadmap artifacts should be stored and maintained in the Architecture Repository.

### 

# Why Adopt Industry Standard APIs?

TELUS has adopted the TM Forum Open APIs as a foundational component of our architecture. TM Forum Open APIs provide an industry standard interface that will allow for interoperability between both TELUS’ own platforms and domains as well as to TELUS’ partners and vendors.

TM Forum has demonstrated APIs are the key to enabling digital transformation, when nine of the world’s largest service providers – Axiata, Bharti Airtel, BT, China Mobile, China Unicom, NTT Group, Orange, Telefónica and Vodafone – officially adopted TM Forum’s suite of Open APIs for digital service management. These service providers have committed to adopt TM Forum Open APIs as a foundational component of their IT architectures, to promote global adoption of the API suite by their partners, and to expect technology vendors and systems integrators to support these APIs in their products and cloud-based services. Adoption of the TM Forum Open APIs will also enable inter-carrier exchange of data for areas like revenue settlement.

One example of the need to adopt an industry standard for APIs is the current transformation of TELUS’ infrastructure through network functions virtualization (NFV), software-defined networking (SDN), and cloud technologies. APIs are critical to making those internal support system platforms available to partners and customers in a dynamic, on-demand way capable of supporting almost-instant, mashed-up services.

## MEF LSO (Lifecycle Service Orchestration)

MEF 3.0 services are a specification of APIs and service lifecycle orchestration for layer 1 through layer 7 services that span more than one operator. As the two industry standards continue to collaborate, the MEF LSO APIs are aligning with the TM Forum Open APIs and provide the ability to manage the lifecycle of network services. The Open APIs, MEF and ONAP support the concept of network as a service and will open new revenue streams for TELUS. MEF focuses on services that cross provider boundaries.

### MEF Relationships to other Information Models

#### TM Forum SID

The MEF Core Information Model is inspired by portions of the TM Forum SID model (taken from Frameworx Release 15.5) with the core classes of the MEF Core Information Model being simplified versions of what one finds in the TM Forum SID. The structure of the MEF Core Information Model closely aligns with the partitioning of the TM Forum SID into Customer / Product / Service / Resource/ Engaged Party domains. The MEF Core Model does not include all the elements of the SID, the latter not always including industry-specific elements of critical importance for delivering MEF-defined services such as EVC and OVC Endpoints, UNIs and ENNIs. Collaboration between MEF and TM Forum is designed to enable the eventual inclusion of those elements in TM Forum SID. Conversely, numerous TM Forum SID classes and associations have been removed in the MEF Core Information Model in order to simplify and focus on the use cases for MEF-defined services.

#### ONF Core Model

The ONF Core Model is leveraged by it being 'plugged into' the MEF Core Information Model at the Resource layer, where the ONF core network classes are generalizations of the "LogicalResource" entity in the MEF Core Information Model. The ONF Core Model itself is thus referenced, but not redefined as part of the MEF Core Information Model.