Enterprise Insight

How to Create a Custom Architecture Modeling Standard

When implementing your Enterprise Architecture (EA) tool, you have many essential design decisions to make. One of the earliest decisions is which standards, frameworks, and methods you are going to select that best serves your organization's strategy and your team's goals and objectives. As part of this decision, you will need to decide which metamodel(s) and notations to implement.

Most organizations choose these models:

- Data Model
- ArchiMate
- TOGAF Content Metamodel
- BPMN
- UML

There are tools and vendors that either support all desired standards, have partial support, or e prescribe a proprietary metamodel of their own design. The most successful organizations use metamodels specific to their needs, but few tools provide the option to easily customize because many tools, mainly legacy ones, have limited support for metamodels.

This paper will focus on how architecture teams can build a custom architecture modeling standard by defining a metamodel, detailing how to build a custom metamodel in four steps with a notation standard, building the model, and finally, putting the model in context.

Do I need a Metamodel?

Most organizations want to build a structured model, by enforcing some rules around the data in their architecture model.

One example of a commonly used rule that can only be enforced with a metamodel:

- An application can be composed of another application

Therefore

- An application cannot be composed of things that are undesirable I.e. a Person, a goal

It is possible to avoid modeling the things you don’t want, but as data volumes and users expand, the need to control the model expands with it. Building an architecture repository without traditional metamodel rules may be appealing initially, but most organizations ultimately need a metamodel that enforces some sooner or later.

Enterprise Insight has open support for standards, allows easy GUI-based customization, and has support for loosely defined metamodels.

Building a Custom Metamodel and Notation-based Standard

A word of caution: for most organizations, it’s best to start with a predefined standard rather than starting with a blank slate.

Here we will show how to do exactly that.

Step 1 – Define the object types

This may seem like an uncomplicated task, but with object types come object type definition explanations. Creating a “Capability” or “Application” object type is easy but defining it may be more complex. In reality, it would be necessary to write formal definitions and create exemplars.

For the sake of simplicity, we will define three object types in our metamodel:

- Capability
- Process
- Application

Step 2 – Define attribute types

This step starts with a basic set of attribute types and then evolves to include many more complex fields.

Here, we define three basic fields which will apply to all objects:

- Description (Type= Text)
- ID (Type= Text)
- Quality (Type = High, Medium, Low)

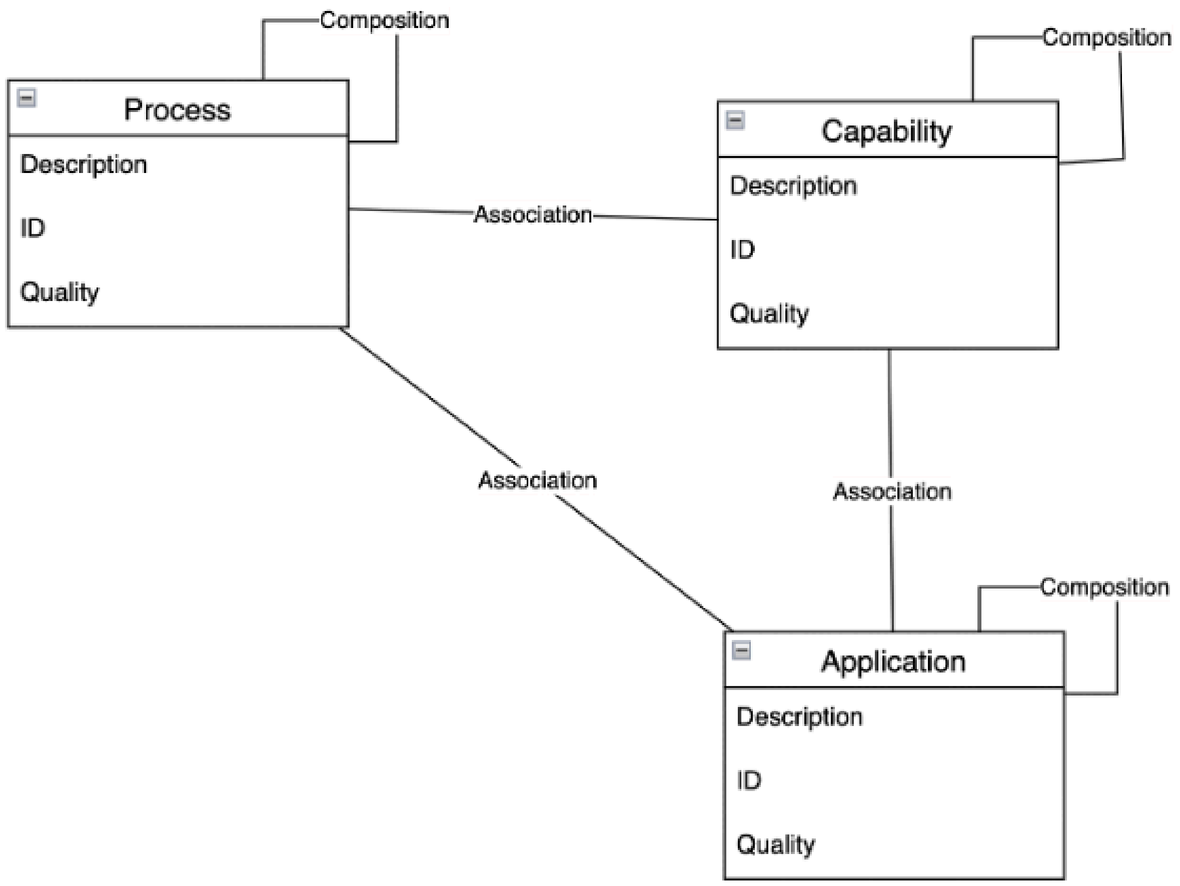
Step 3 – Define relationship types

This step could be deceivingly manageable. In a low-touch metamodel, we might define a single relationship type that could apply to anything or a small set, and applying some simple rules. However, setting up a simple set of rules early on creates challenges down the road when we need to change these rules where remediating non-conforming content can become laborious..

In line with our scope to create a simple metamodel, we will create some simple metamodel rules:

Relationship Type	**From Object Type**	**To Object Type**
Association	Any	Any
Composition	Capability	Capability
	Process	Process
	Application	Application

We may prefer to look at this metamodel in pictural form:

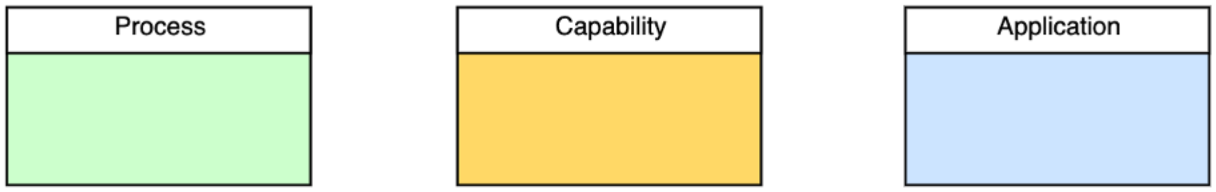


Step 4 – Define a notation

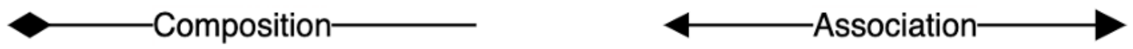
While the previously defined object types, attribute types, and relationship types may be functional, diagrammatically they would be indistinguishable, so it is best to define a notation. The notation could vary by deliverable. For example, if you draw a BPMN diagram, a process might take on the style of that notation for that deliverable.

Again, we will define a simple notation:

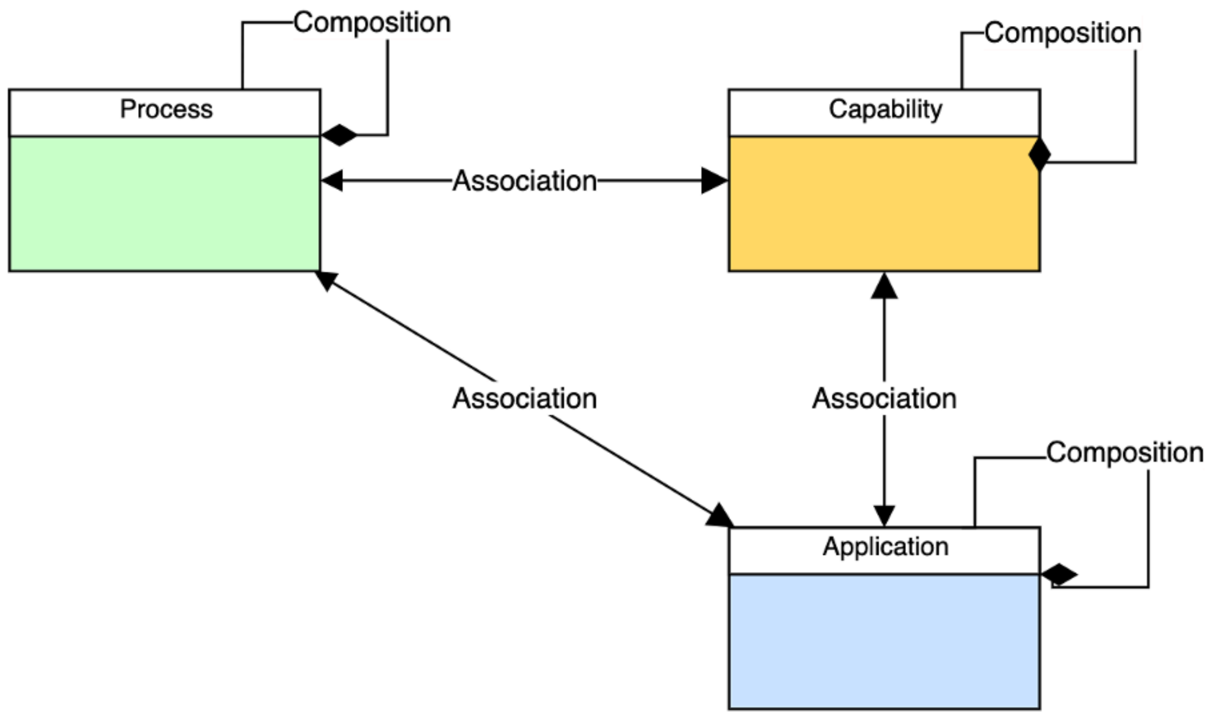
Shape Notation:



Relationship Notation:

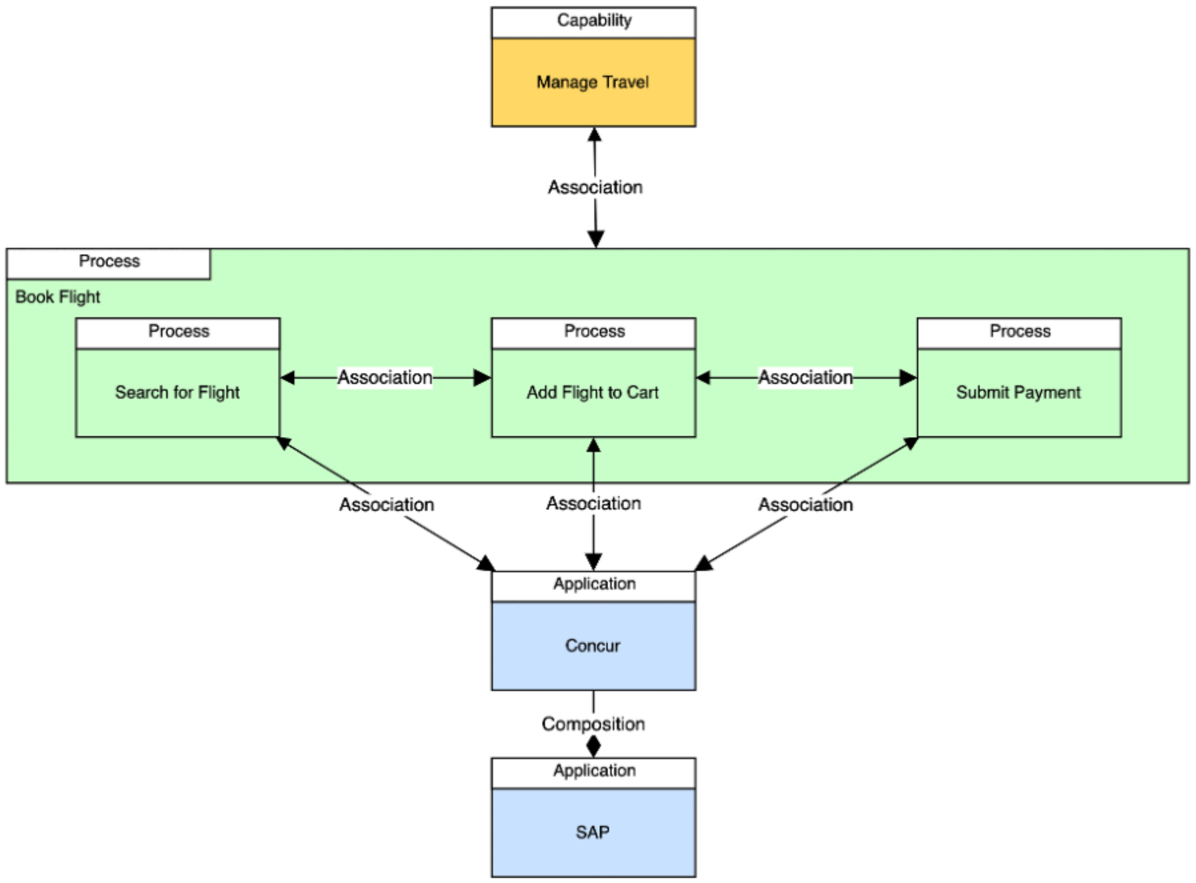


Final Metamodel Diagram



Putting the Metamodel into Context

The ultimate deliverable produced using this metamodel and notation is illustrated in the diagram below.



Key Takeaways

Building a proprietary metamodel is an unorthodox approach because it is often significantly more time-consuming to build and implement. That said, standards typically offer heavyweight metamodels and expect the community to tailor them to reduce the complexity to suit their needs.

With the rise in low-touch metamodels prescribed by some vendors and tools today, there may be some appetite for fully custom metamodel definitions. Enterprise Insight has a range of options to support all use cases.