Modeling Strategy and Implementation

The **strategy layer** is a layer on top of the **business**, **application**, and **technology** layers. Because everything in the enterprise must happen to realize the strategy, having strategy elements within the repository that are supported with well-designed models can tell management which enterprise elements are helping to realize the strategy.

It will help management to predict what would happen if changes were made to the state of an organization. Changes to the state of the organization can be any changes that require the business to respond, and they can be positive or negative changes. They can be financial, political, natural, or technological. Some changes bring opportunities and others bring threats, and your business must be ready to respond. Otherwise, it will be in danger of being left behind.

Business capabilities are at the heart of strategic planning and modeling. Modeling business capabilities is one of the most powerful tools that can help organizations to document how they are able to provide their services to customers, and how they can build on their abilities if they don't have them.

In this chapter, you will learn how to utilize the power of the strategy models and how to use them to identify the gaps between business today and business in the future. In architecture words, this chapter will cover the following two topics:

- Introducing strategy elements
- Introducing implementation elements

Introducing strategy elements

The strategy layer uses elements for modeling strategy artifacts, just like all the other layers. Strategy elements are also of two types, **structural** and **behavioral**, just like the elements in other layers. In this section, we will explore the four elements of the strategy layer, which are as follows:

- Capabilities
- Value streams

- Courses of action
- Resources

Defining capabilities

"A capability represents an ability that an active structure element, such as an organization, person, or system, possesses".

The topic of **business capabilities** is one of the most controversial topics in **Enterprise Architecture** (**EA**). One of the reasons behind this huge disagreement among enterprise architects is a result of the vague definitions that are provided by the different standards.

In this section, we are introducing you to our understanding of business capabilities, which are based on ArchiMate®'s definition. Always keep in mind that what matters the most, as a practical enterprise architect, is to know how to model business capabilities and present them to management rather than keep arguing about them.

ArchiMate®'s definition of a capability is also very abstracted and does not answer any of the questions that enterprise architects will be looking for, so we will try to elaborate on it. To start with, a **capability** is a strategic behavior element, which means it is an intangible element. It is a behavior that an active structural element can have or own (*possess*). Owning this behavior enables the structural element to be *capable* of realizing a business outcome by providing value. This value can take the form of services that can be provided or products that can be sold.

Important Note

Sparx combined elements from the strategy and the motivation layer into one toolbox, which is called **motivation**. If you want to create a strategy model, you need to choose motivation as the type of the diagram.

ArchiMate® 3.1 provides two notations for modeling capabilities, as you can see in the following diagram:

Figure – Capability modeling notations

The following focused metamodel shows how every core behavioral element in the enterprise realizes the capability:

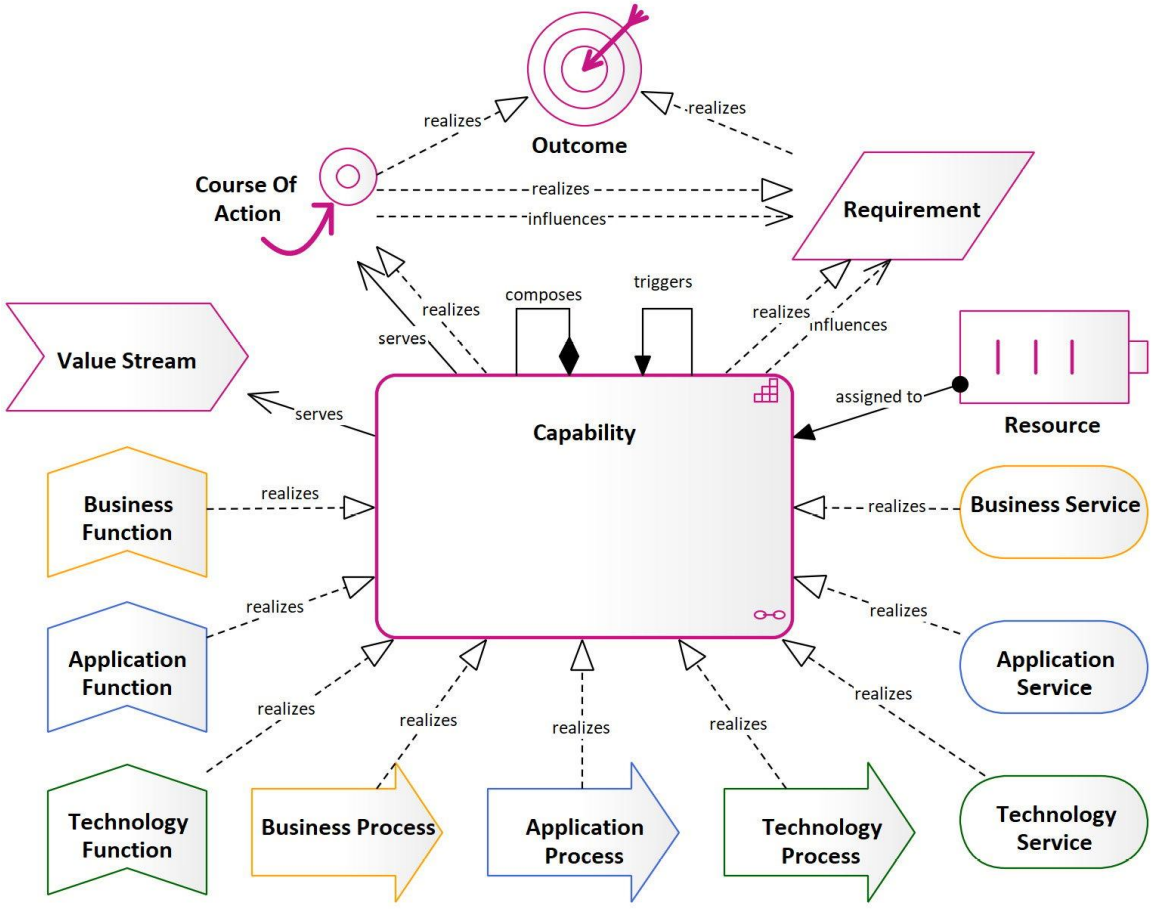

Figure – The capability focused metamodel

As you can see, every function, service, and process, from the business, application, and technology layers, *realizes* the capability. In other words, every enterprise

behavior element *realizes* the capability. We can confidently say that capabilities are the center of the enterprise, and this statement is made by us, not by any standard.

There are a few things to notice in the previously focused on metamodel. The first thing is that capabilities realize and influence requirements. If a car manufacturer, for example, decided to build motorcycles, there must be some strategic requirements behind this decision. This requires the business to develop the ability to make new products and provide new services.

When the required capability is developed, the business will be able to achieve (or realize) the goals. At the same time, requirements can be influenced by existing and targeted business capabilities. The decision for a car manufacturer to build motorcycles, not computers, for example, could have been influenced by the existing capability of manufacturing cars, which are relatively closer to motorcycles than to computers. In general, our requirements have always been influenced by our possessed capabilities, even at a personal level. We all want to have everything, but what limits us are our capabilities.

Another thing to notice is that the capability realizes and serves the course of action. A course of action is simply the plan that you need to follow to achieve the desired outcome based on the capabilities that you have. But in a nutshell, if you have the capability but you do not have the proper plan to achieve your desired outcome, that capability will be wasted. Therefore, capabilities need to realize the courses of action needed to actualize (realize and influence) business outcomes.

Remember that capabilities are meant to deliver value, so let's understand what value and value streams are.

Value streams

"A value stream represents a sequence of activities that create an overall result for a customer, stakeholder, or end user".

Value is anything that a stakeholder wants to have from an organization. Value from the perspective of a customer takes the form of the products or services that the organization offers, while value for a shareholder takes the form of profit, market share, reputation, and customer loyalty. The products that the organization produces may not be of interest to the owners as they look for different value.

Some values are dependent on other values, which in turn depend on other values. This is what is known as a **value stream**. It is a sequence of values, activities, and actions that together participate in realizing an outcome. Do not confuse value streams with processes, as they both involve a sequence of actions. Value streams represent a strategic high-level sequence of actions, while processes represent an operational-level sequence of actions. They are both behavioral elements, and processes need ultimately to realize the value streams, but each has a different scope of coverage and different level of detail.

ArchiMate® 3.1 provides two notations for modeling value stream elements, as you can see in the following diagram:

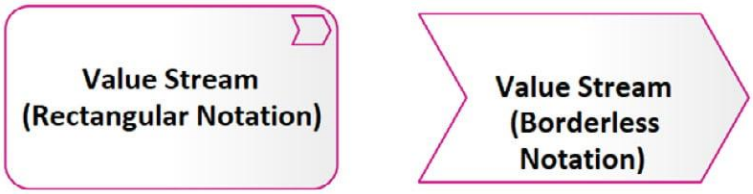

Figure – Value Stream modeling notations

Value streams run if the organization can do so or, in other words, is capable of doing so. To deliver value, you must have the right capability. In architectural terms, we can say that capabilities serve value streams.

Since value streams are strategic behavioral elements, they are realized by other behavioral elements, such as processes, functions, and services from the business, application, and technology layers. The following diagram shows the focused metamodel for value streams:

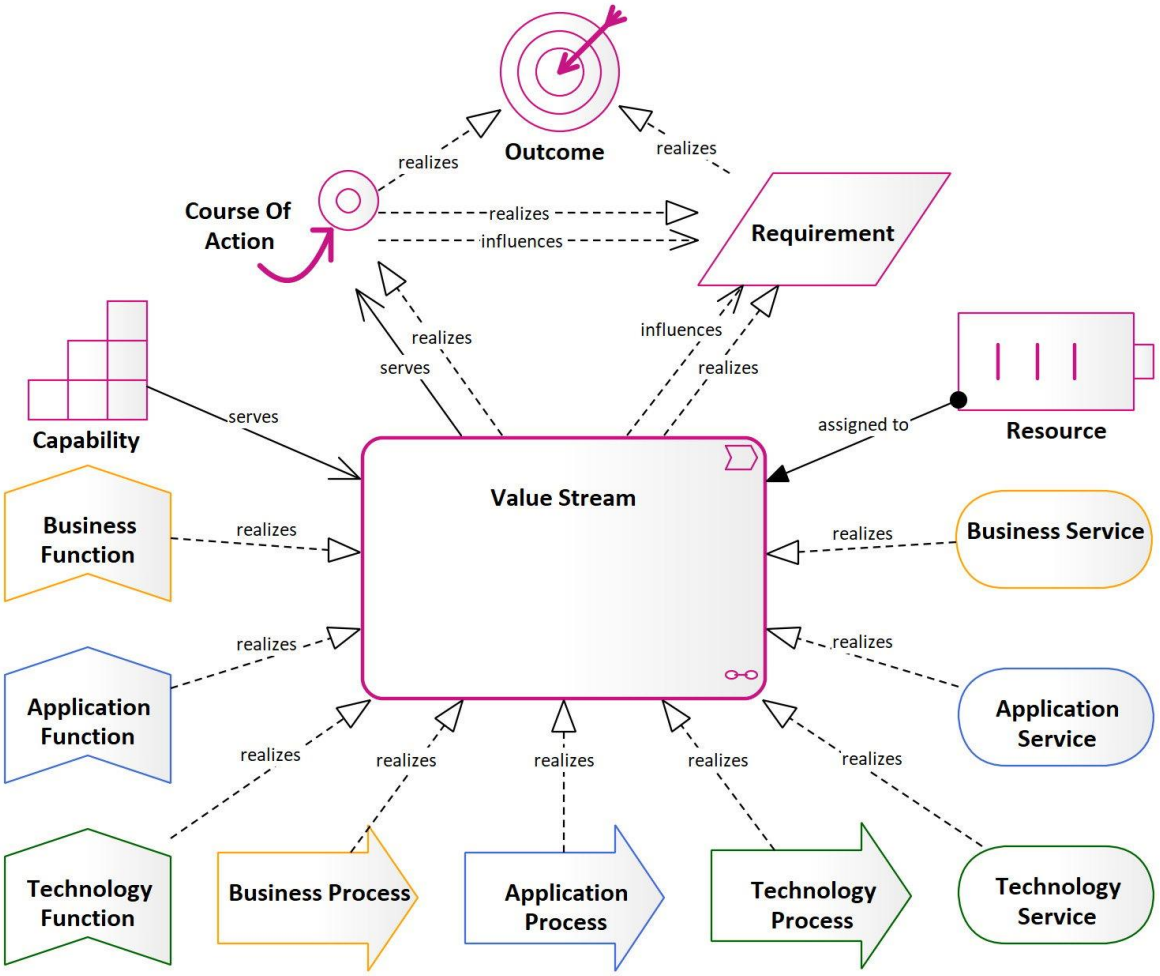

Figure – The Value Stream-focused metamodel

As you can see, it is identical to the capability-focused metamodel because both value streams and capabilities are strategic behavior elements. You can also see that value streams need resources assigned to them, just like capabilities. Without resources, such as actors, application components, technology components, data, information, technology components, and other structural elements, no behavior will be performed. Talking about resources, let's learn more about them.

Defining resources

"A resource represents an asset owned or controlled by an individual or organization".

As you can see again, the definition is very vague and does not answer many of the questions that you may have about capabilities. As mentioned in the previous *Defining capabilities* subsection, a capability is a behavior that needs

people, information, and technologies. All of these are forms of resources, so we must assign a specific number of resources to be able to deliver a specific capability.

ArchiMate® 3.1 provides two notations to model resource elements, as you can see in the following diagram:

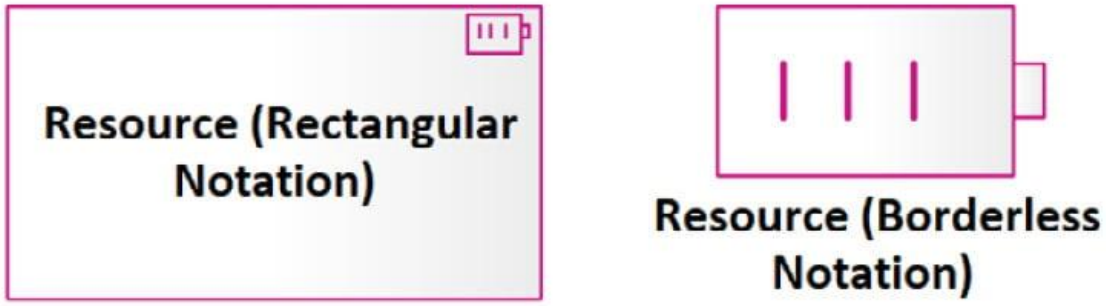

Figure – Resource modeling notations

The following focused metamodel shows how resources are assigned to capabilities and value streams, and they all realize and influence requirements:

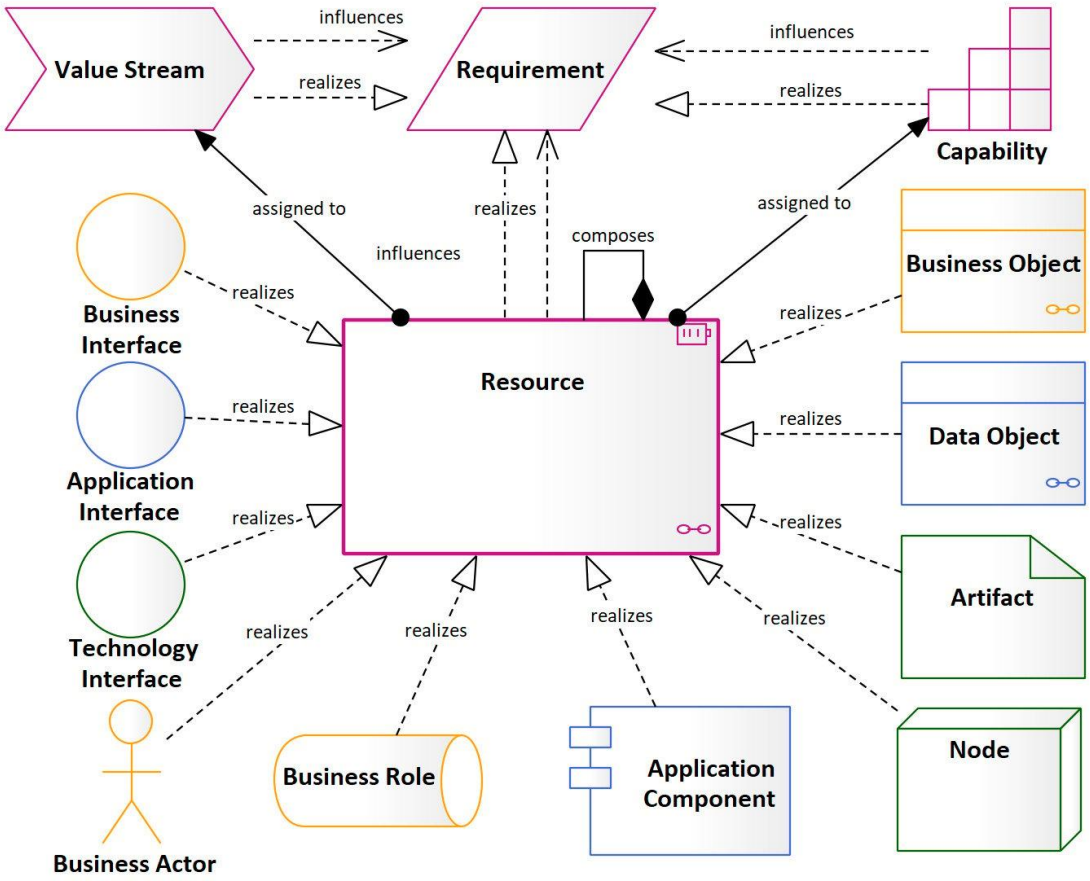

Figure – The resource focused metamodel

Resources are structural elements, and they are realized by the active structural elements in the business, application, and technology layers.

Resources in the project management world include time and money in addition to human resources and physical assets. Time and money are not part of the ArchiMate® specification at any layer, so you cannot model them. However, you can still model what they can buy (or hire) for you. So, you can model the human resources that you can hire, the application and technology components that you can buy, and the information that you can gain with time and money.

The last strategy element that we will explore before giving you some practical examples to model strategy elements is the course of action element.

Courses of action

"A course of action represents an approach or plan for configuring some capabilities and resources of the enterprise, undertaken to achieve a goal".

Organizations, in general, have multiple capabilities. These capabilities can be grouped and organized in different ways to achieve different outcomes. The courses of action is where you plan and configure this grouping of capabilities and value streams based on the desired outcome. You can think of courses of action as long-term strategic initiatives rather than thinking of them as short-term detailed plans. They just outline what needs to be done without detailing how.

Take, for example, the goal of sending people to Mars. To achieve this goal, one of your courses of action would be to build a spaceship that is capable of landing on Mars and returning. The details on how to develop the needed business capabilities to build that spaceship are at a more tactical short-term level.

Keep in mind that courses of action can compose larger courses of action and can be composed of smaller courses of action. So, being at a high level of abstraction does not mean that they cannot be part of some other higher-level or lower-level strategic initiatives.

ArchiMate® 3.1 has two notations to model course of action elements, as you can see in the following diagram:

Figure – Course of action modeling notations

courses of action are strategy behavioral elements, so they are realized by capabilities and value streams, as you can see in the following focused metamodel:

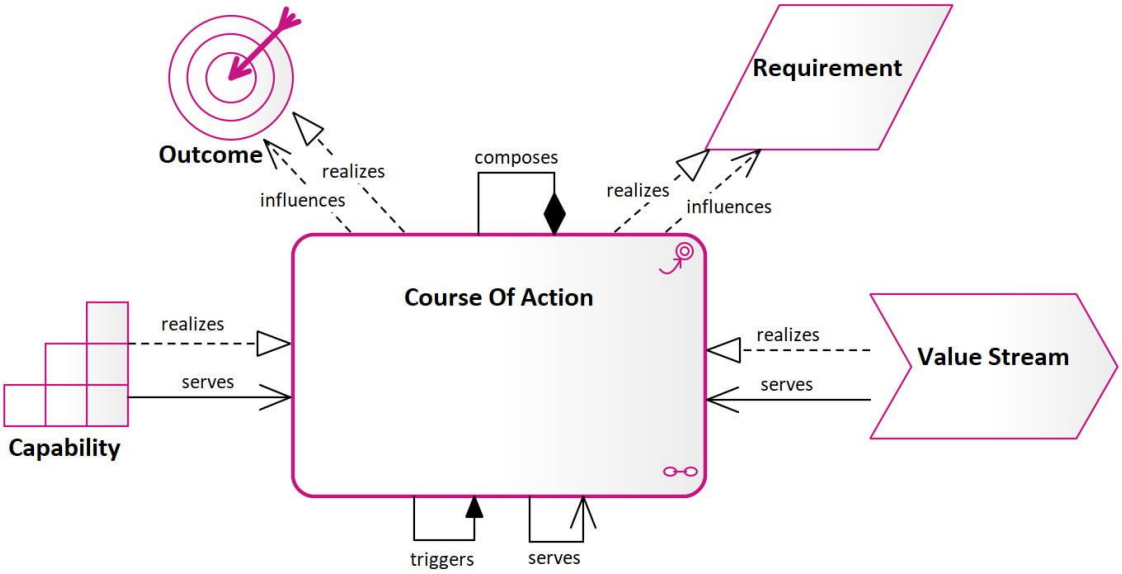

Figure – The Course of Action focused metamodel

Courses of action can be composed of smaller courses of action, and they can serve, be served, trigger, or be triggered by other courses of action.

Let's look at some examples to illustrate the relationships between the strategic structural elements, the strategic behavioral elements, and the structural and behavioral elements from other enterprise layers.

Modeling the ABC Trading strategy

ABC Trading has decided to establish online retail sales to sell goods directly to customers and reduce its dependency on retail partners. According to feasibility studies that *ABC Trading* has conducted, the CEO has decided to proceed with the initiative and asked you to develop a high-level business architecture document. You need to help the CEO to model their thoughts, describe the targeted online service, and cooperatively identify what *ABC Trading* needs to do to provide the service. In this subsection, we will be doing the following:

- Modeling the value stream
- Realizing capabilities by behavioral elements
- Differentiating between the **as-is** and the **to-be** elements
- Identifying the needed resources

We will start by modeling the value stream and the capabilities that serve it.

Modeling the value stream

Before we start modeling capabilities in detail, it is best to start by answering the question of *why* we need them. The best person who can give these answers is the CEO or manager who assigns the building and development of those capabilities. It is very important to understand that as an enterprise architect, you are not the one who is finding these answers. Your role is to capture them from the person who is making the decisions, challenge them architecturally, and finally, present them in a model.

The CEO told you that the target outcome is to increase annual sales, and this can be achieved if *ABC Trading* is able to sell goods online directly to consumers instead of being only a retail sales company. Based on the answers that you have collected, you came back with the following diagram to have it approved by the CEO:

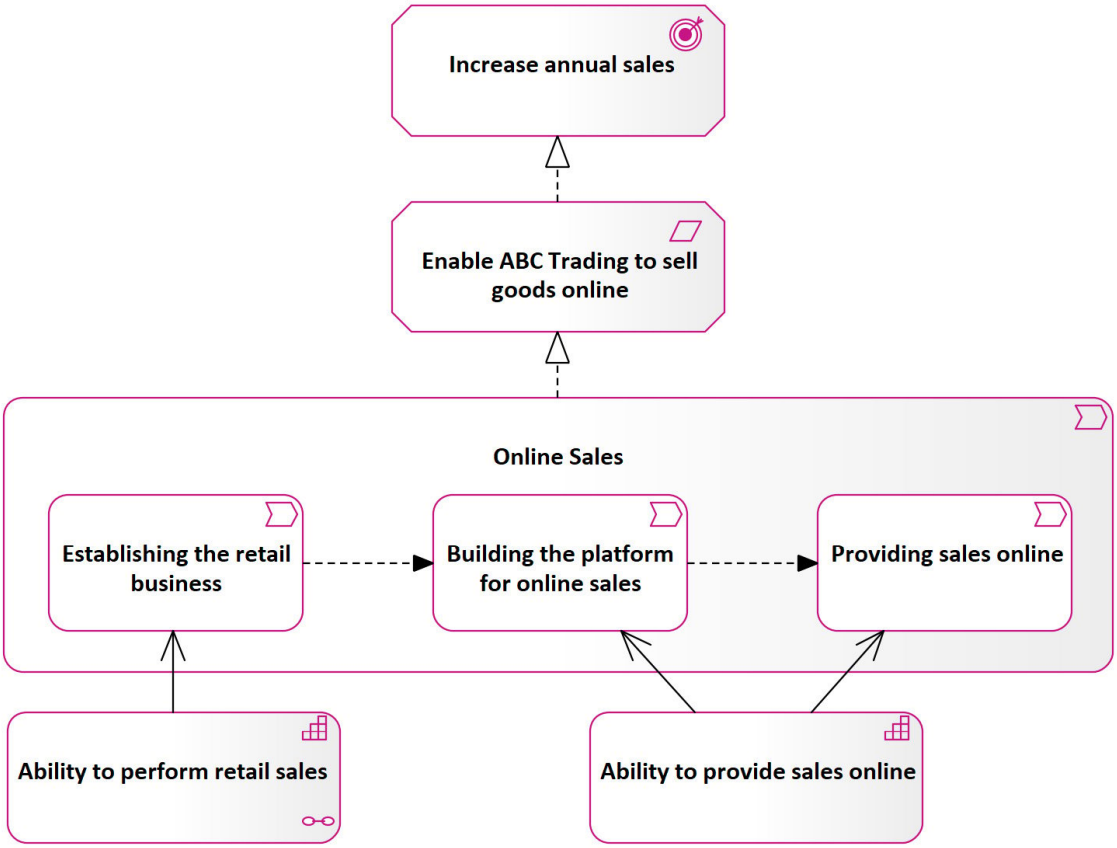

Figure – Business capabilities serving the value stream

The diagram shows that offering online sales from a retail-oriented company requires performing three main steps (for the value stream):

- **Establishing the retail business**
- **Building the platform for online sales**
- **Providing sales online**

The online sales value stream will realize the requirement to enable *ABC Trading* to sell goods online, which in turn will realize the increase in annual sales outcome. The diagram also tells us that two capabilities need to be developed to deliver the value stream – the first is **Ability to perform retail sales**, and the second is **Ability to provide sales online**.

Important Note

Do not expect to get your diagrams right the first time. Final acceptance can take several rounds of refinements, so don't feel frustrated.

We have identified at a high level what we are planning to do and outlined how it is time to present much deeper details about the two required capabilities. We need to identify how to build and develop them and, ultimately, serve the value stream. In other words, we need to answer two questions:

- *What behavioral elements are needed to realize the identified capabilities?*
- *What structural elements are needed to realize the resources that will be assigned to the capabilities?*

Whether we have these elements today or not is another question to be answered later, but first, we need to identify what the needed elements are; therefore, let's start with the behavioral elements.

Realizing capabilities by behavioral elements

A capability is a behavior at the strategy layer, and it needs to be realized by one or more core behavioral elements from the business, application, and technology layers. If we want our business to have the ability to perform retail sales, it means that we want it to provide a specific type of services.

Without these services, a retail business will not be considered a retail business. The following diagram shows an example of the business services that realize the **Ability to Perform Retail Sales** capability:

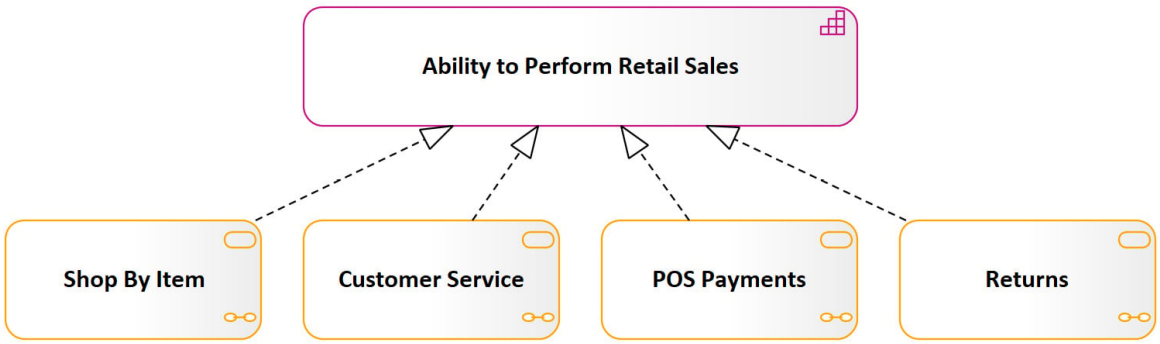

Figure – Business services realizing a capability

Each of these business services can be realized by one or more behavioral elements, such as business processes, business functions, application services, and technology services. To realize the customer service business service, for example, you will need a set of functions and processes to realize it.

If it is automated, this means that it is realized by an application service or a technology service. In the following diagram, there is an example of a business service and how it can be realized by other business behavioral elements. The diagram also shows that the business service is completely realized by the **Customer Relationship Management** application service:

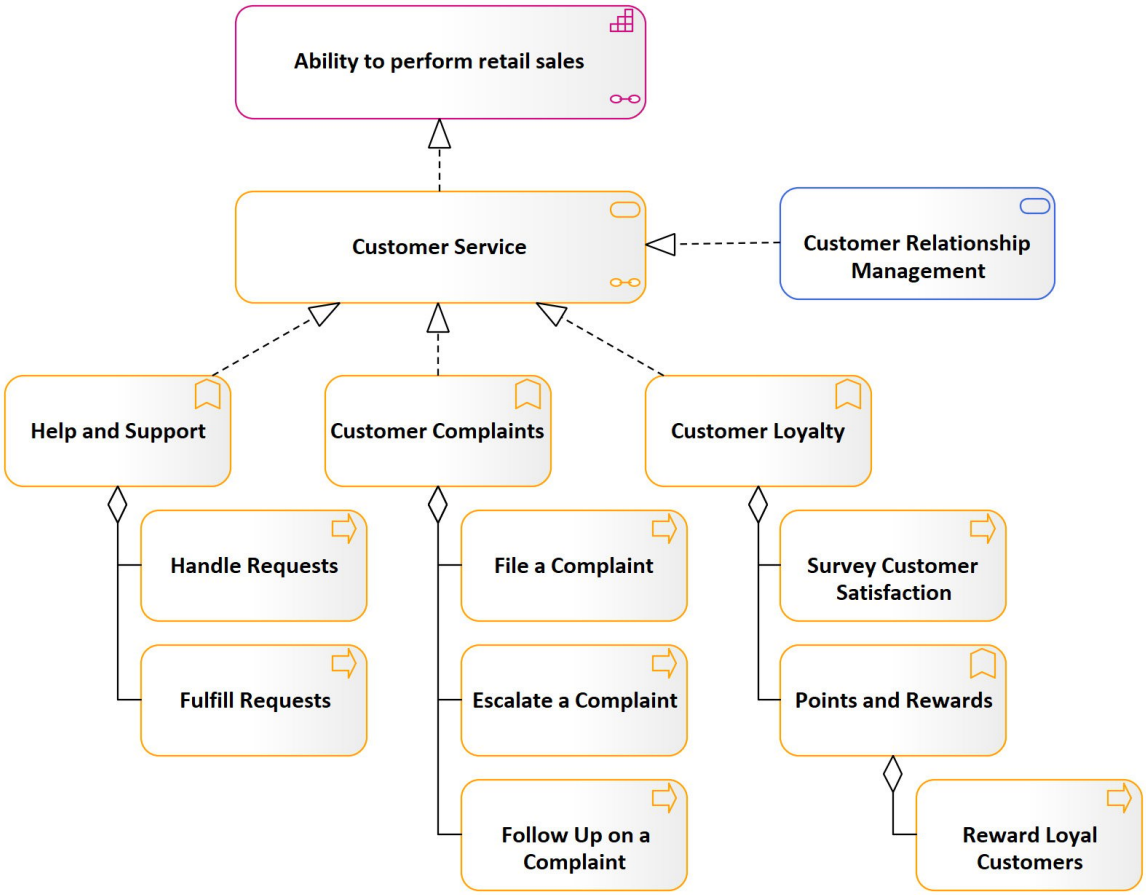

Figure – A business service decomposition

If the business service is partially automated, it means that it is not realized completely by an application service. In this case, you must show the realization relationship only on the automated elements.

For every business service that you have identified, you can create a similar decomposition model, showing the elements that are composed (or aggregated). If your diagram grew in size and became busy with too many elements, you can put each main element in its own diagram.

After identifying the capabilities that we need to build and the behavioral elements that will realize them, it is time to identify which of these elements exist already and which ones don't.

Differentiating between the as-is and the to-be architectures

Whenever you plan for something, you will always have two architectural states at least – the present state and the planned future state. They are also known as the **as-is** and **to-be** states of the architecture.

We say at least two states because, in many cases, you will need some transitional states between the as-is and to-be architectures. One main reason for having transitional states is when the gap between the as-is and to-be is too large and requires building smaller steps at a time. One good example is when you are migrating from a legacy application to a modern one. You may build or use some components that are required only during the migration, but once it is completed, there will be no need for these transitional components.

One way to differentiate between the states in Sparx is to use the version value of each element. By default, every element you create in Sparx will be set to version **1.0**. Let's assume that the **Customer Relationship Management** application service does not exist now, and it is a future service in the to-be architecture if we want to fully automate the customer service business service.

To change the version to **2.0**, right-click on the **Customer Relationship Management** application service element and enter `2.0` in the **Properties** > **Properties** > **General** > **Version** field, as shown in the following screenshot:

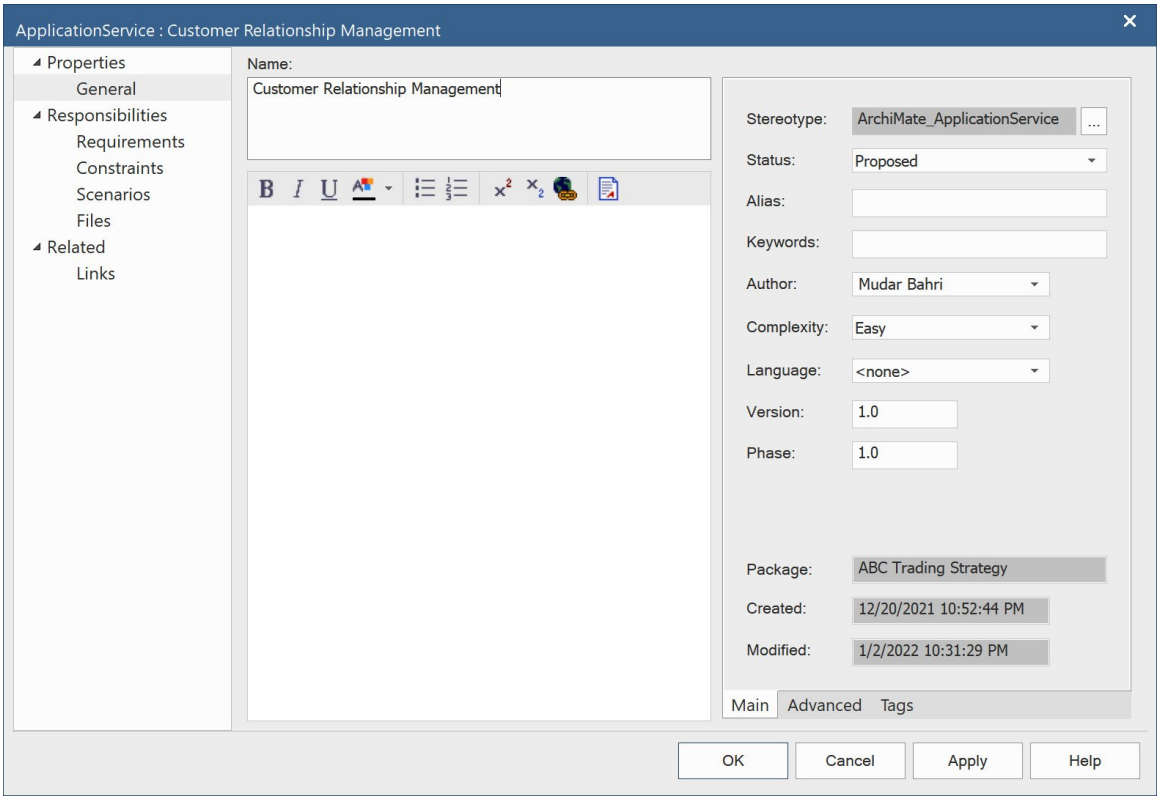

Figure – Setting the version of an element

The **Version** field is a free text value, which means that you can put any value you want to use for versioning and not only numeric values. It is all up to you and the architecture team that you work with. You can use the values as-is, to-be, and in transition if you want. We will keep the default of `1.0` to indicate an as-is element, use `2.0` for the to-be, and `1.5` for any transitional element.

Do the same steps to change the version for the **Customer Loyalty** business function and all the composing elements. This indicates that this business function does not exist today as part of *ABC Trading's* internal functions, so it must be built and developed as part of the to-be architecture.

Now, we need to use color codes to visually differentiate elements based on their version. One way is to use the filters and layerss. This is very useful if the diagrams will be accessed from Sparx. However, if you publish the image, the layers will be ignored and not published. So, we need a different way to color-code the elements based on their versions that are still effective even on printed diagrams and exported images, and this way is by using the **Legend** element.

From the **Common** section in **Toolbox**, find the **Diagram Legend** element and place it on the diagram. Double-click on the **Diagram Legend** element to open the dialog that you can see in the following screenshot:

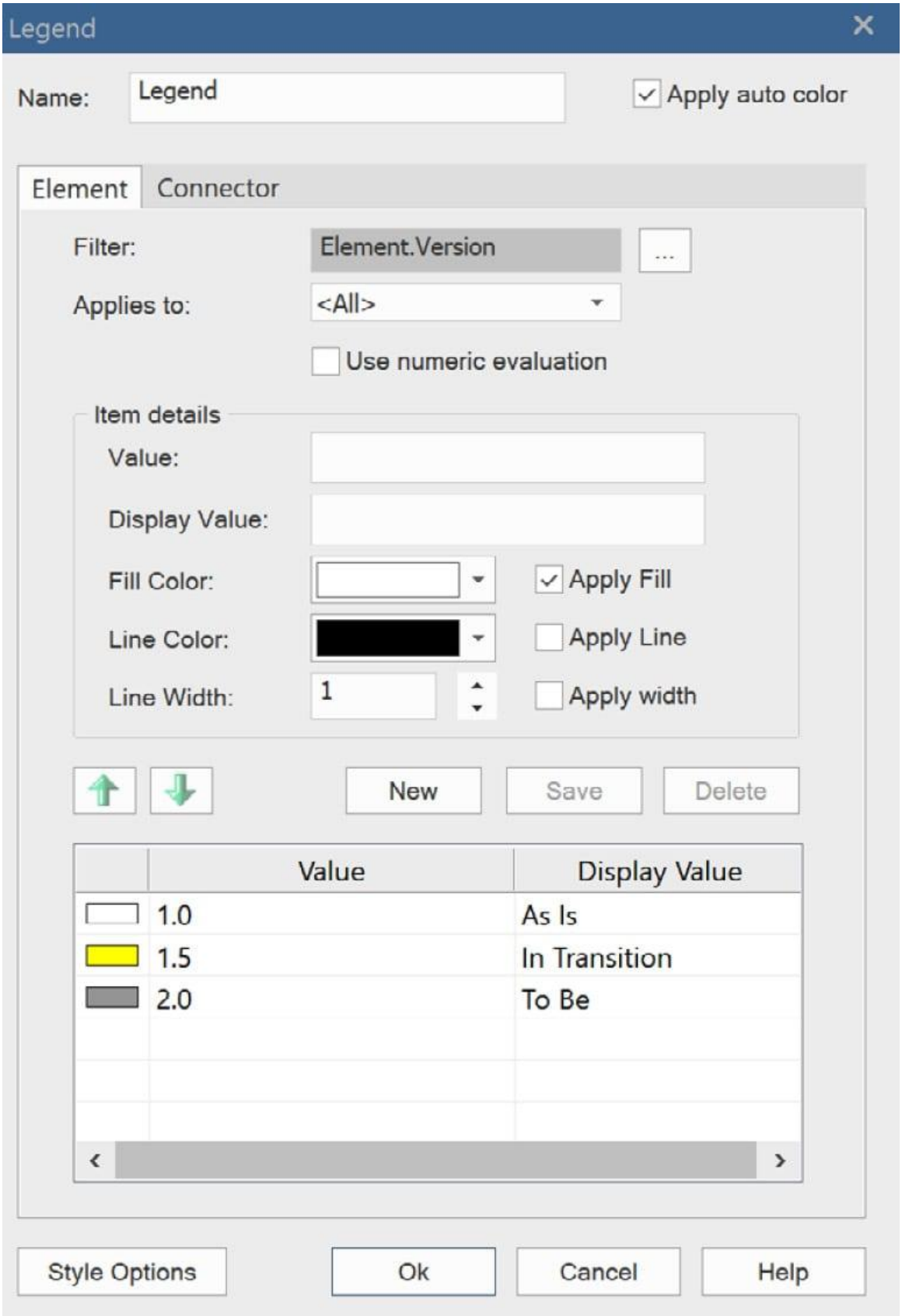

Figure – The diagram legend dialog

Follow these steps to create a color-coded legend:

1. Keep the default **Legend** value in the **Name** field or rename it if needed. This will act as the label that will be displayed on the legend box.
2. Make sure that the **Apply auto color** checkbox is checked. This tells Sparx that you want to apply special formatting on elements that match the criteria that we are going to define now.
3. From the **Filter** field, click the three dots and select **Element** > **Version**. This tells Sparx that your legend formatting criteria will be based on the version field of the diagram elements.
4. Leave the **Applies to** value as the default value, **<All>**. You can use this field to filter on a specific element type.

Now, we need to tell Sparx to use three different color codes to differentiate elements based on the previously defined criteria. We will use **1.0** for **As Is**, **1.5** for **In Transition**, and **2.0** for **To Be**.

5. Enter `1.0` in the **Value** field and `As Is` in the **Display Value** field.
6. Select the **Fill Color** for version **1.0** elements, which I kept as the default white fill color.
7. Make sure that the **Apply Fill** checkbox is checked.
8. Since we are not applying any color codes on the line (the element's borders) in this example, uncheck the **Apply Line** checkbox. We're also not applying any formatting on the borders' width, so we can uncheck the **Apply width** checkbox.
9. Click the **Save** button to save this legend value.
10. Click the **New** button and repeat *step 5* to *step 9* for the **1.5** and **2.0** versions. This time, select different fill colors for the two versions.
11. Click **Ok** to accept and close the dialog.

If you have done everything correctly, you can see how Sparx now differentiates the fill colors of the elements based on their versions, as shown in the following diagram:

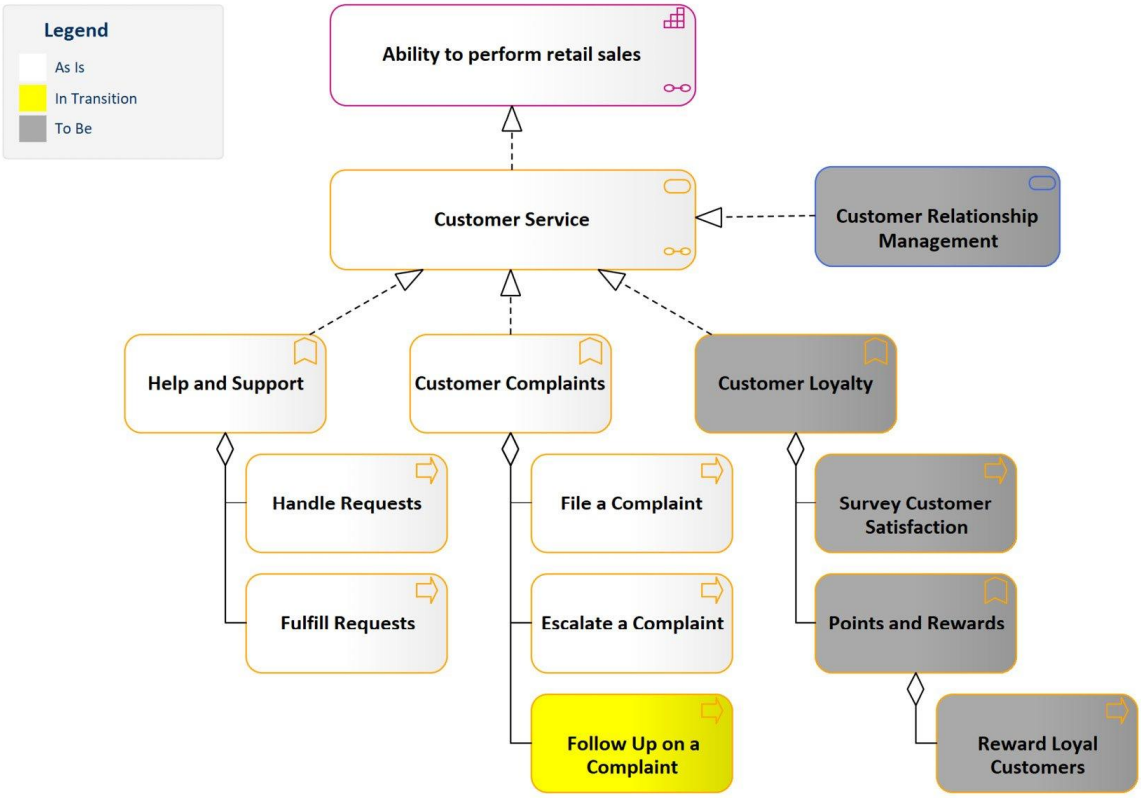

Figure – A color-coded business service diagram

The legend element is reusable and can be placed on any other diagram in the repository. Furthermore, it will apply the defined color code automatically on the elements of the diagram that it is placed on. Try it yourself:

1. Select the legend element on the diagram.
2. Press *Alt + G* to find the legend element in the browser.
3. Drag the element and place it on any diagram that you have created earlier.
4. Change the version values of elements on the diagram.

You will see how the defined color codes will be applied automatically to the elements. Being a reusable element, changing values in the legend such as adding a new version or changing the colors will automatically be reflected in all the diagrams that use the same legend element.

Identifying resources

Earlier, we identified the capabilities that are required to serve the targeted value stream and the core behavioral elements that will realize the capabilities. In this section, we need to identify the resources that will be assigned to the identified capabilities and their realizing behavioral elements.

Since we went into more detail about the ability to perform the retail sales capability, we will continue using it for this example. You will still need to do the same for all the capabilities that you have identified. For *ABC Trading* to be able to perform retail sales, it needs to have the following types of resources:

- **Business actors and roles**: The managers, accountants, security guards, janitors, forklift operators, salespersons, and all the other required human resources to enable this capability. You can nest business actors and assign them to business roles.
- **Information**: No matter whether the information is documented on paper, in a knowledge base system, or undocumented, with the proper knowledge, people will be able to deliver value. Information about supply and demand, trending products, market changes, team-building techniques, project management, operations, and other information is essential to be capable of performing retail sales.
- **Interfaces**: Customers will need interfaces to be able to use services. These interfaces can take the form of a showroom, a call center, a web interface, and many other forms of human and application interfaces. Remember that when a business interface is realized by an application or a technology interface, it indicates that it is an automated (or digital) interface.
- **Technologies**: These can be anything from heavy machinery, security doors, badge scanners, and ceiling fans to computers, network devices, and computer applications – the list is long.

The following diagram shows sample resources assigned to the ability to perform the retail sales capability. We can now identify which resources we have and which resources we need to have in the future (to-be). We will identify the ones in the to-be architecture state by changing their version value to `2.0`, and the ones in transition by the `1.5` value as a version:

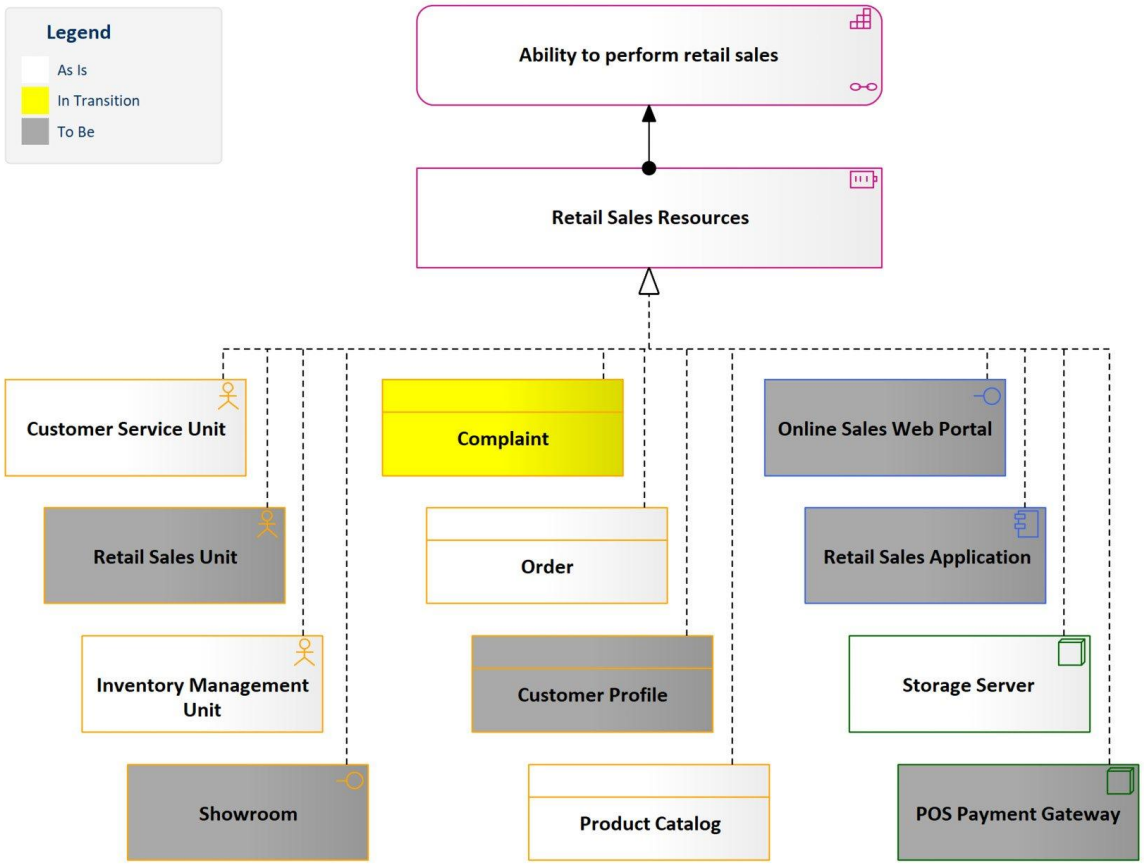

Figure – Retail sales resources assigned to a capability

Note that we have reused the same legend element that we created, so it automatically applied the fill colors on elements based on their version.

To recap, enterprise resources are realized by structural elements from every enterprise layer (business, application, and technology). Capabilities are realized by behavioral elements from every enterprise layer. Resources are assigned to capabilities to realize and influence requirements. Capabilities with the resources assigned to them realize the courses of actions, which will help the enterprise to achieve the desired outcomes.

The differences between the two versions of the architecture mean that there are gaps between the two. To transform from an as-is state to a to-be state, these gaps must be closed.

Introducing implementation elements

Without being implemented, all your architectural work and artifacts will remain theoretical. Good presentations and nicely published enterprise content are great achievements, but they will not make things happen. Your next action is to convert your architectural artifacts into actionable plans and start building what you have been planning for. In this subsection, we will introduce you to the implementation elements and how to use them to model your plans, so let's start with the plateau element.

Defining plateaus

"A plateau represents a relatively stable state of the architecture that exists during a limited period of time" .

A plateau represents the state of the architecture. It can be a past, present, or future state. When we mentioned the as-is, in transition, and to-be architectures in the previous *Introducing strategy elements* section, each one of these three states can be represented with a plateau element. You can also think of plateaus as phases or milestones within a bigger effort to achieve goals and address requirements. Let's look first at the plateau's focused metamodel before elaborating:

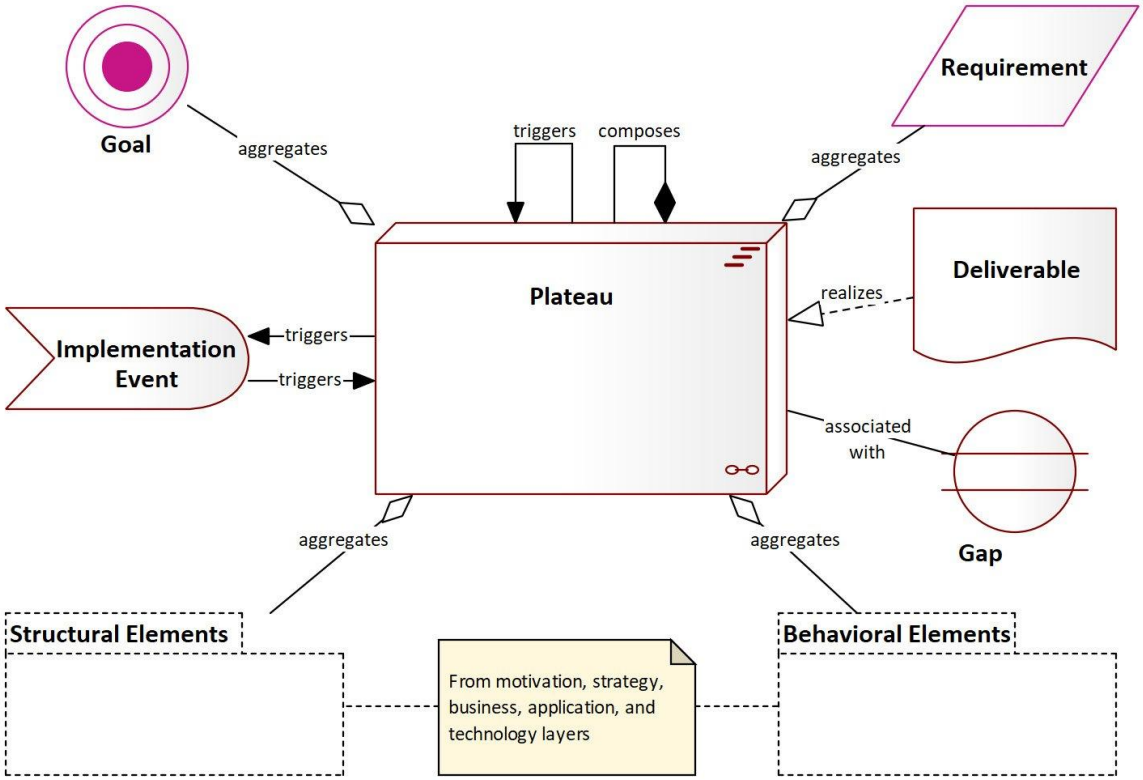

Figure – The Plateau focused metamodel

As you can see in the focused metamodel, the plateau can aggregate any number of structural and behavioral elements from any layer of the enterprise. The plateau can compose a bigger plateau, and it can be composed of many smaller plateaus as well. It is always recommended to put the right amount of information in a single diagram. You can consider modeling a plateau for each value stream element, as shown in the following diagram:

Figure – Three plateaus representing high-level transformation phases

Within each plateau, you can identify what elements will belong to it. The same elements can belong to any number of plateaus. Transitioning from an as-is plateau to a to-be can involve keeping all or some of the current elements to the targeted state. In real-life examples, a plateau can contain hundreds of elements, which makes it difficult to fit them all in one diagram. It is highly advisable to make

diagrams that convey a single idea at a time. The following diagram shows two plateaus with some shared and new elements:

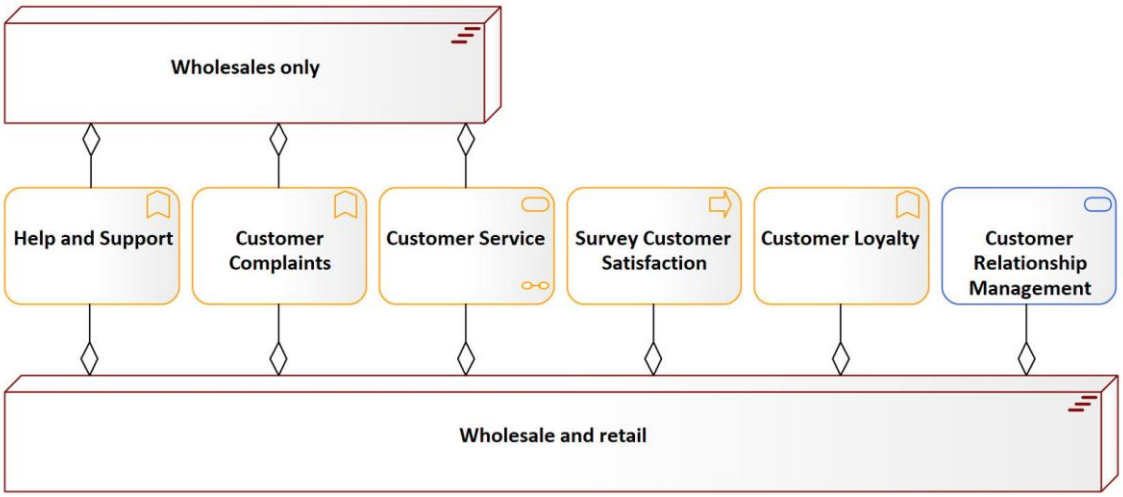

Figure – The difference between two plateaus

As you can see, there are elements that are part of the as-is Wholesales Only plateau but that will continue to be used in the targeted Wholesale and Retail plateau. There are also elements that belong only to the targeted plateau, and those represent the gaps that must be closed. If there are elements in an as-is plateau but not in the to-be, they indicate elements that will be retired or decommissioned from the enterprise.

Defining gaps

"A gap represents a statement of difference between two plateaus".

Gaps are passive structure elements, so they are a type of data or information. This data concerns the decision-makers, as they can prioritize the gaps and allocate the required resources to them. If you have one plateau representing the as-is and another plateau representing the to-be, the differences between the two are described by the gaps that need to be closed to complete the transition. Gaps usually indicate missing elements. However, they can also indicate additional elements that will be retired from the as-is element and will not be taken forward.

The following diagram shows the focused metamodel of the **Gap** element:

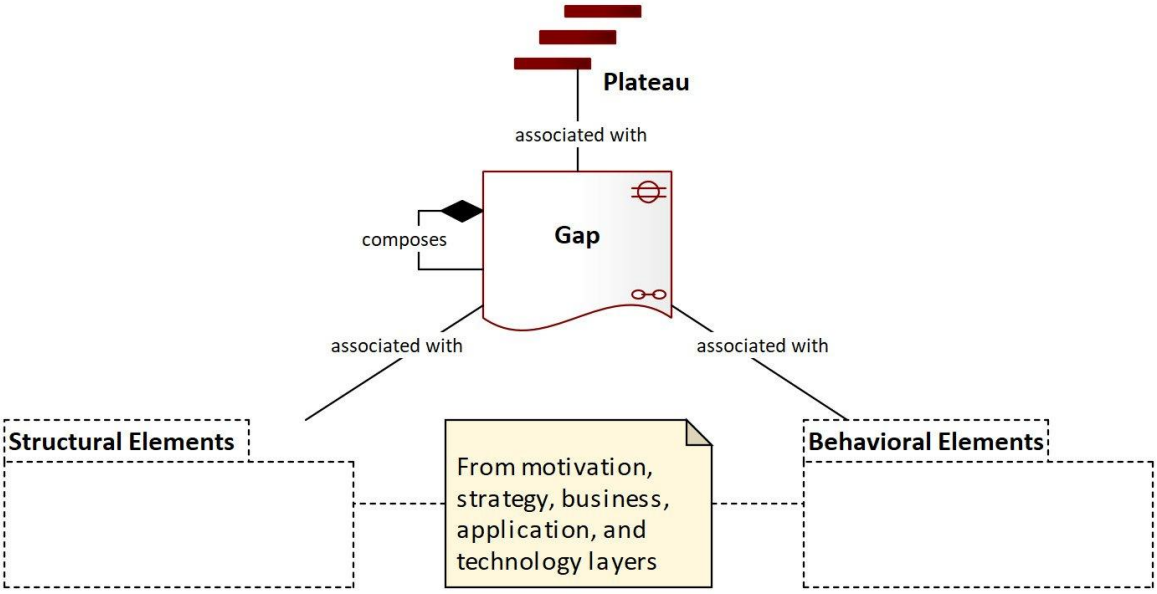

Figure – The Gap focused metamodel

An example of how gaps between two plateaus can be modeled is depicted in the following diagram:

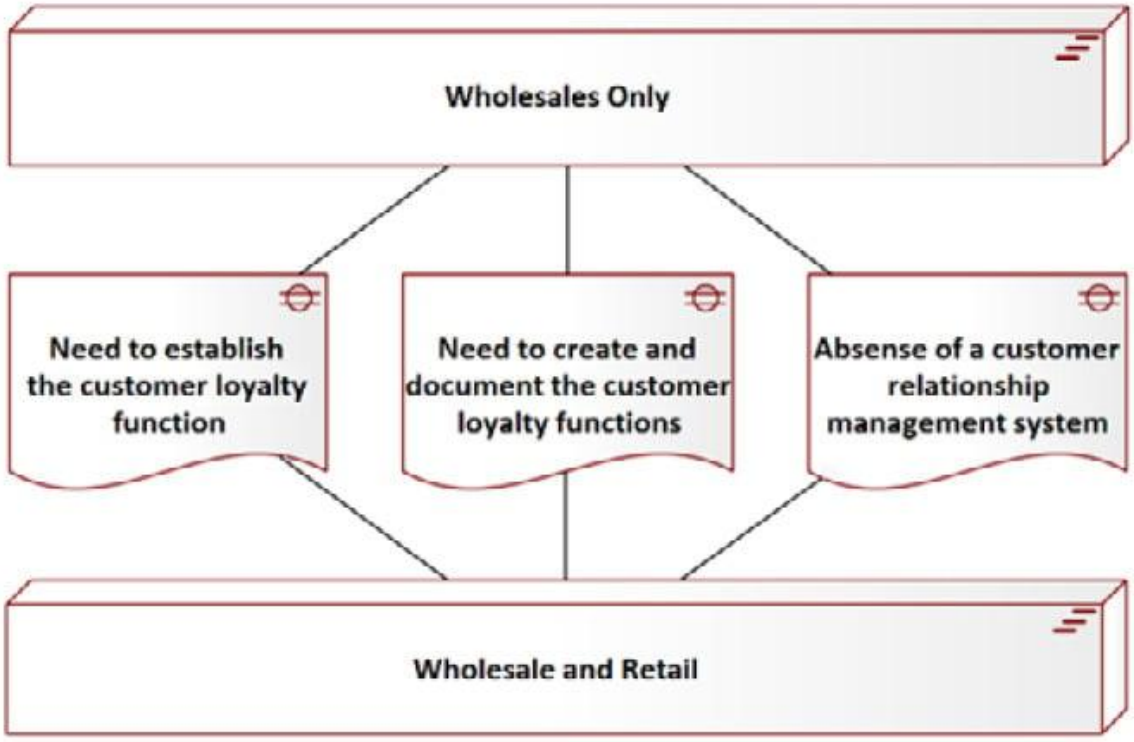

Figure – Modeling gaps between two plateaus

Gaps can be associated with any structural and behavioral elements from any enterprise layer. You can have a single gap that involves some missing roles,

processes, strategies, applications, and technologies, of any number, as shown in the following diagram:

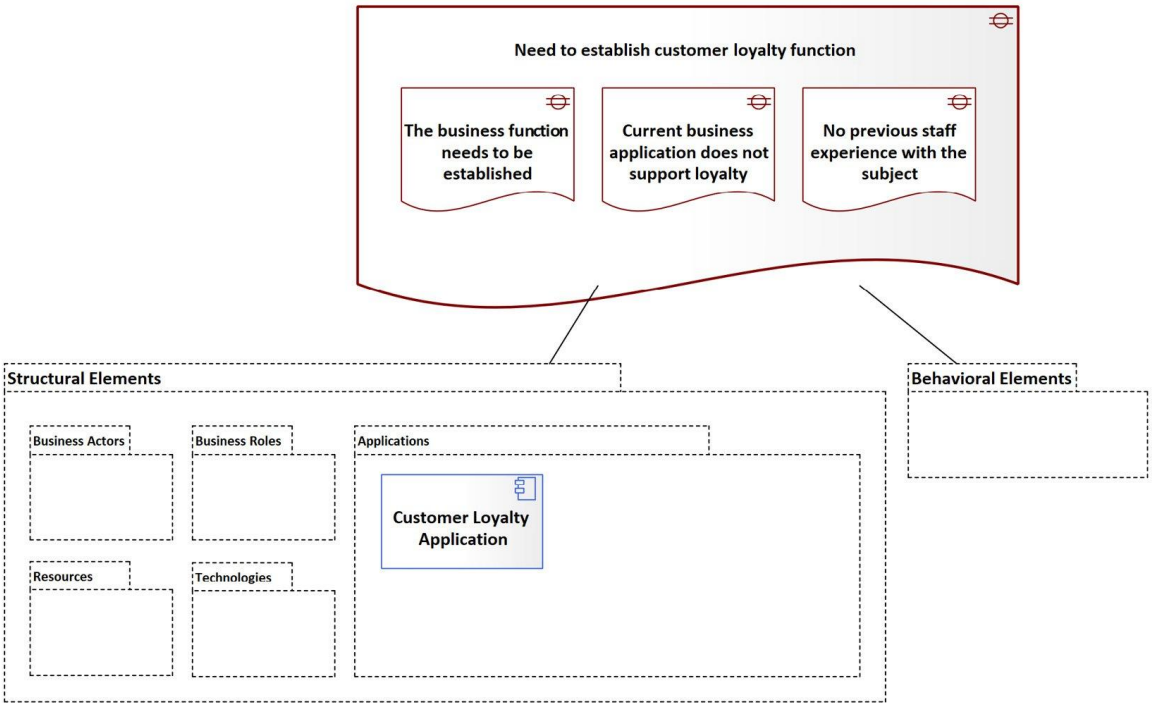

Figure – Modeling nested gaps

As you can see in the diagram, we have broken down one of the gaps into smaller gaps to ease turning them into smaller actionable items known as **work packages**. The diagram also indicates that closing this gap will use the resources that are allocated for building the ability to perform retail sales. You can develop as many diagrams as needed where each can highlight a specific point of interest. Some diagrams will show how many business roles the enterprise needs to be capable of to close the gap. Another diagram can show us what processes we need to learn, develop, and automate. A third diagram can show what technologies are missing and are required.

At any point in time, you may have limited information, so you add what you know and keep adding elements whenever you learn something new. Developing a new capability is a learning experience for everyone. If time is not on your side, then you have to use part of the allocated resources and hire people to fast-track the process.

Defining work packages

"A work package represents a series of actions identified and designed to achieve specific results within specified time and resource constraints".

Work packages are internal behavioral elements. If you are familiar with the **Project Management Body of Knowledge** (**PMBOK**), you can think of work packages as the equivalent of work breakdown structure items. From this point in the architecture definition, it will be very helpful to involve the project managers to participate in defining and modeling the work packages and their composition hierarchy. Work packages can be composed of smaller ones, and there is no limit to how small a work package can be. Here is the **Work Package** focused metamodel diagram:

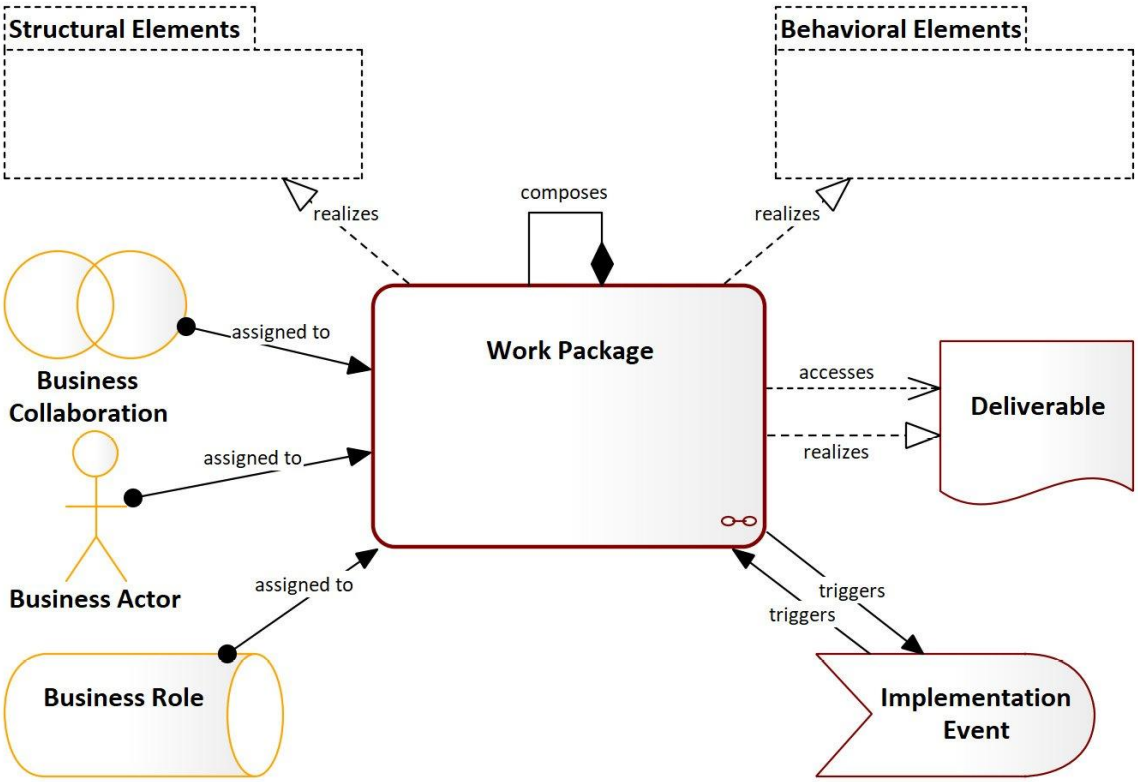

Figure – The Work Package focused metamodel

You need to avoid making them as granular as the tasks are, so they still need to maintain the concept of *packaging* multiple actions that together will realize a deliverable.

We need to remember that Sparx is not a project management software, so we don't need to move all project management activities into it. Even with the ability to create

some models such as Gantt charts, Sparx is still an EA repository, and artifacts must remain as EA artifacts. Once a project is kicked off, the project manager can use more sophisticated project management software to manage the project while maintaining the trace back to the related EA artifacts, such as the work packages and the deliverables.

The following example explains how a work package can be composed of many smaller work packages, each one realizing some of the elements that are associated with a gap:

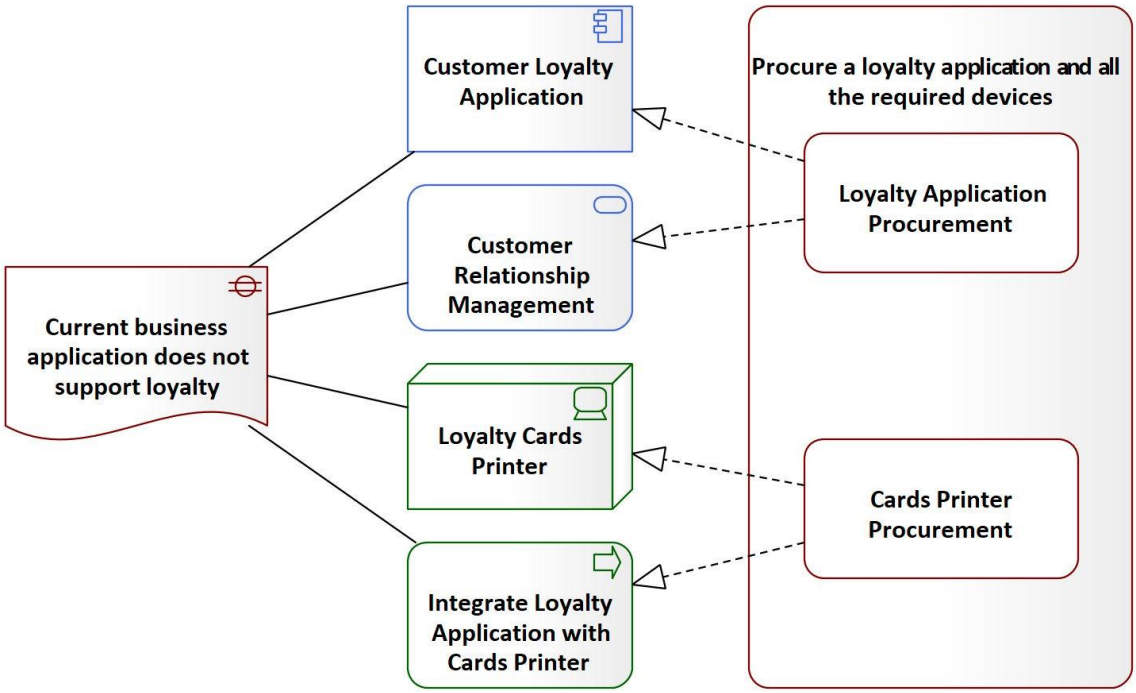

Figure – Work packages realizing gap elements

The next element is the deliverable, which is the result of completing a work package, so let's learn more about it.

Defining deliverables

"A deliverable represents a precisely-defined result of a work package".

Deliverables are passive structure elements, so they are treated the same as data or a piece of information. They can be anything that a work package is intended to deliver. This means that they realize either structural or behavioral elements based on what work packages were intended to deliver. Deliverables can be tangible, such as products, software, modules, documents, and reports. They can also be

intangible, such as the maturity of an organization, customers' satisfaction, or employees' loyalty to their workplace.

Deliverables are like milestones that, when achieved, mark the end of a work package. Based on the size of the work package, a deliverable can mean the completion of a single user story, a whole project phase, or the entire project. Remember that these requirements were originally aggregated within a plateau, so completing the deliverables means that we have realized all parts or a part of the plateau. It's all based on the requirements that the deliverable has realized.

The following diagram contains the focused metamodel of the deliverable element:

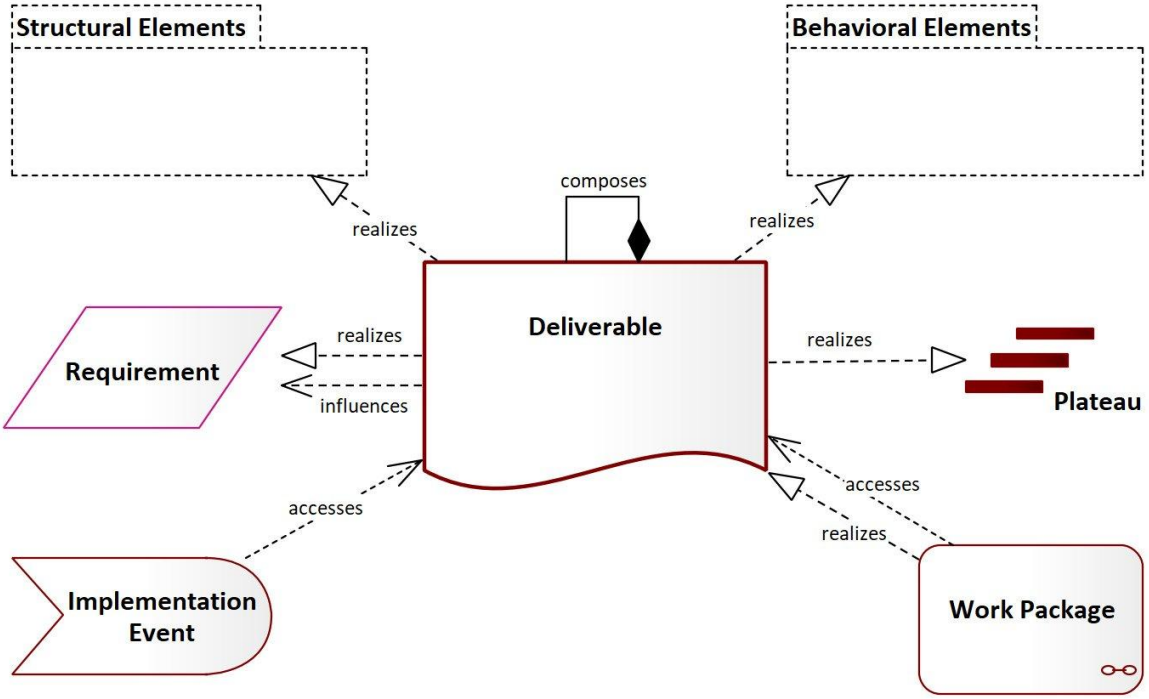

Figure – The deliverable-focused metamodel

Look at the following example to understand how a work package realizes a deliverable, which itself realizes a plateau:

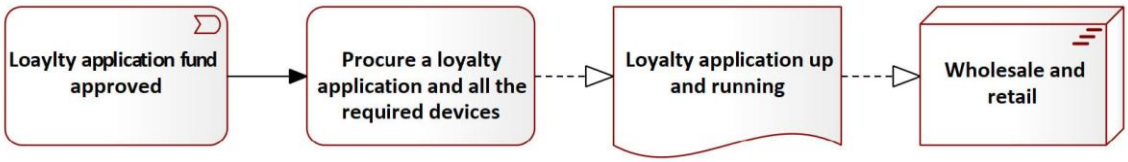

Figure – Work packages realizing a deliverable, realizing a plateau

The left-most element is an implementation event, which we will talk about in the next subsection. It triggers the execution of a work package, which once completed will realize a deliverable, which in turn realizes a plateau.

The last element that we will discuss in this chapter is the implementation event before wrapping up with the summary.

Defining implementation events

"An implementation event represents a state change related to implementation or migration".

Implementation events are events that initiate a change in relatively stable plateaus or work packages. They are instantaneous and have no duration, but they may initiate changes that last for long periods of time. The following diagram shows the implementation event-focused metamodel:

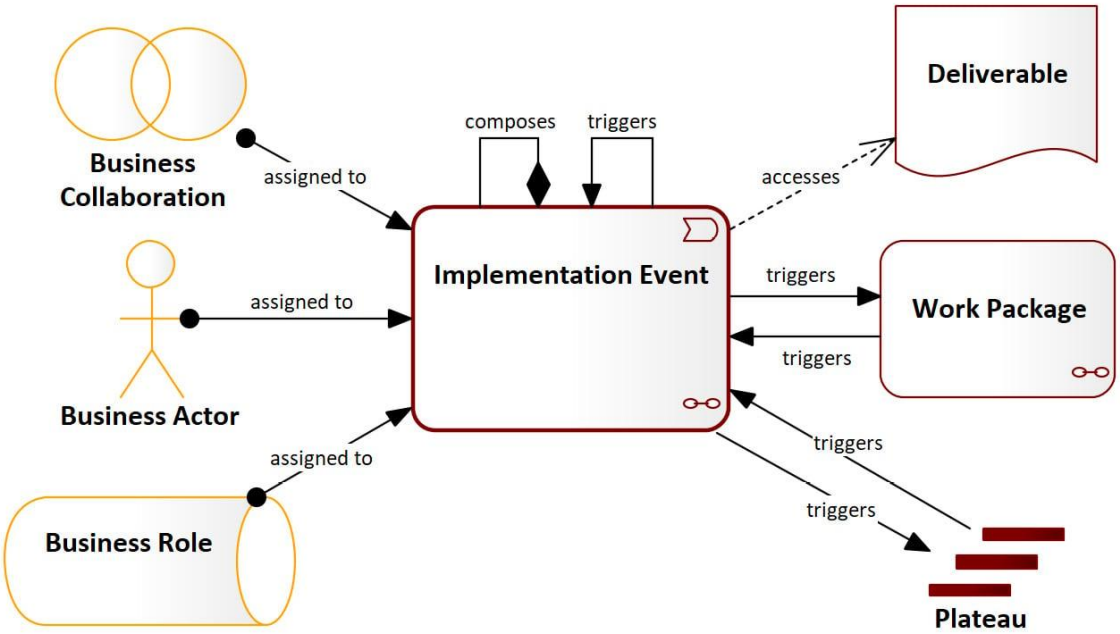

Figure – The Implementation Event focused metamodel

Internal business structural elements are assigned to implementation events to manage and control their flow, as shown in the following diagram. This could be a business unit, a specific person, a specific role, or a combination of all:

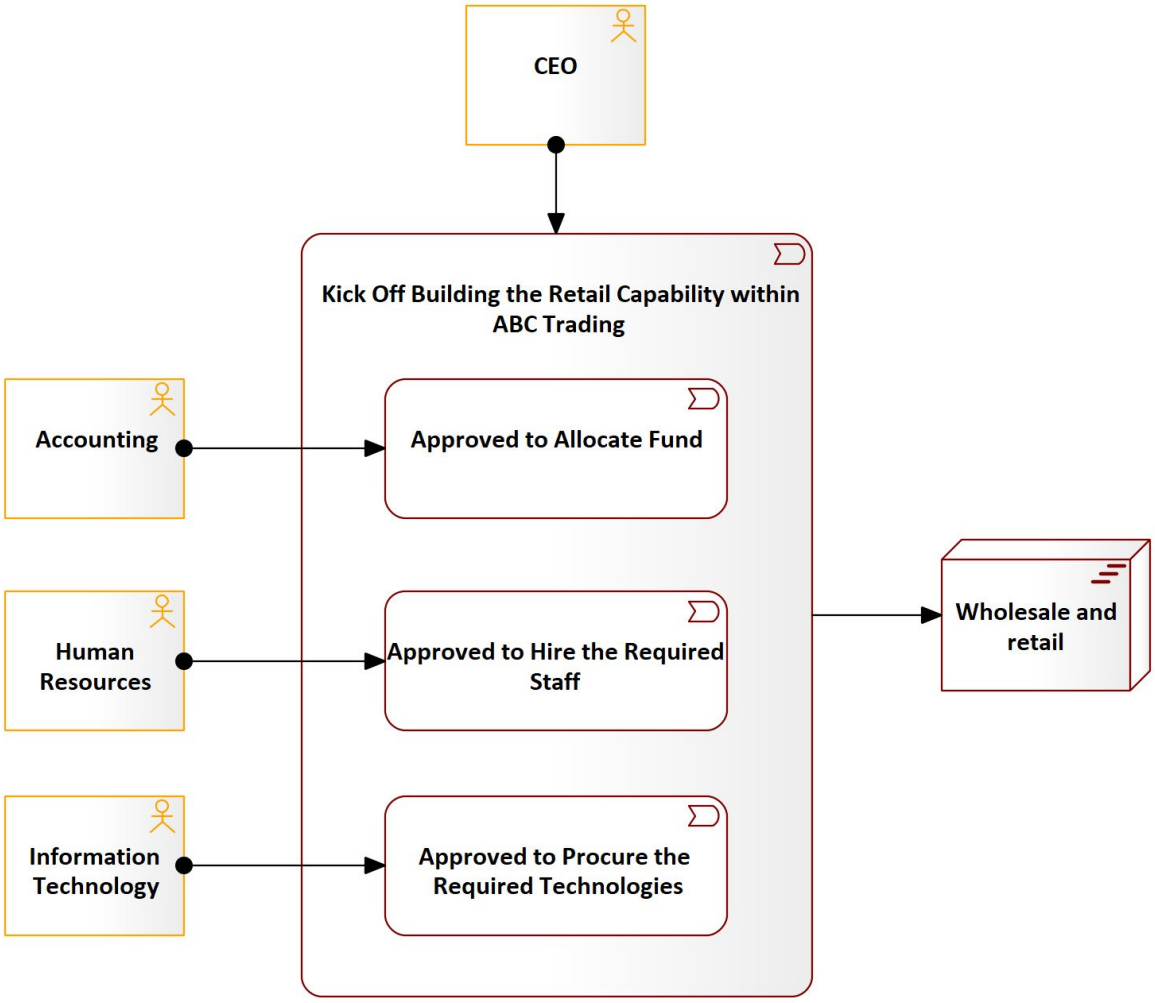

Figure – Business structural elements assigned to implementation events