Maintaining Quality and Consistency in the Repository

You now have an enterprise architecture repository with a single artifact, an application context diagram for the **Tracking App** package. At this point, we need to step back and imagine where all of this may lead. This is just the first of many artifacts that are needed to represent your model-driven enterprise. You might be wondering how you can scale this effort so that others in your organization can help. You need a standard against which you can measure the quality and consistency of artifacts from various authors. Without such a standard, you would quickly run into problems with consistency and quality in your repository. Asking everyone in your organization to become experts in the ArchiMate® 3.1 notation is unreasonable.

We delve into modeling best practices and how to represent your model standards using the concept of a **metamodel**. Just as metadata is data that describes other data, a metamodel is a model that describes other models. Here, we explore the ArchiMate® standard metamodels and learn to condense that information into what we call a **focused metamodel**.

Technical requirements

We will be referring to the *ArchiMate® 3.1* standard, so access to The Open Group®'s ArchiMate® reference (https://pubs.opengroup.org/architecture/archimate3-doc/) is important. It would also be helpful, but not necessary, to have some familiarity with that work.

Building the application component focused metamodel

Metamodels are the blueprints that guide the development of architectural artifacts such as diagrams. We developed our first diagram without a metamodel, but that is because we were referring to the ArchiMate® 3.1 metamodels.

Focused metamodels are our interpretation of ArchiMate®'s standard metamodels. ArchiMate® metamodels are great references, but they are organized by layer. To

get the full picture of an element and to find all possible relationships across the different layers, you will need to go back and forth between multiple diagrams in different chapters of the ArchiMate® specification. Focused metamodels combine all that you need to know about a specific element in one place, so they are *element-oriented*, not *layer-oriented*.

Each focused metamodel has one element in focus at a time and shows the possible relationships to other elements. You can enrich your enterprise architecture repository with dozens of focused metamodels, each focused on a different element. On the other hand, you can still work without them and use the standard metamodels, but they will help make your enterprise architecture practice easier. Remember that when you work in a team, each member can have a different level of experience in ArchiMate® and enterprise architecture, so having an easy-to-follow reference can be very handy to unify team efforts.

Metamodels are diagrams that guide us in making diagrams. To create a metamodel, we will follow the same steps that we took to create the application component context diagram. The only difference between the two diagrams will be the content. Sparx treats both diagrams the same and does not provide any special treatment to one diagram over the other. It is your responsibility as an enterprise architect to organize the repository in a way that makes sense to you and the audience.

We will create our first focused metamodel and it will be the application component focused metamodel. We will achieve this by doing the following:

- Establishing the metamodel diagram by creating a package and a diagram
- Using ArchiMate® 3.1 as a reference to guide the focused metamodel creation
- Adding the proper elements to the diagram according to the reference material

Establishing the metamodel diagram

The first thing that we need to start with is to create a new package that will contain the new diagram and the elements that will be placed on it. In other words, we will be doing the following:

- Creating a **Metamodels** package and the application component-focused metamodel diagram

- Adding a focus element to the diagram

We have already created a root node package and named it **Architecture Content**. We will create a new **Metamodels** package under the root node package but at the same level as the **Tracking App** package that we created. Metamodels are references for architects working on any enterprise architecture artifact for any element within the enterprise. On the other hand, the **Tracking App** package contains the **Tracking App Context Diagram** and the elements that are specific to the diagram. Therefore, we need to place the **Metamodels** package under the **Architecture Content** package instead of under the **Tracking App** package.

Creating a diagram in a package

We will follow the same method that we followed to create the context diagram by creating a package with a diagram and then adding elements to it, as follows:

1. Click on the **Architecture Content** node package, and then click on **Design** > **Model** > **Add** > **Package**, which will bring up the **New Package** pop-up window. Type `Metamodels` in the **Name** field, select **Create Diagram**, and click **OK**.
2. The **New Diagram** pop-up window will appear automatically, so enter the name `Application Component Focused Metamodel` in the **Diagram** field.
3. From the **Type** list, either choose **All Perspectives** to list all diagram types or choose **Enterprise Architecture** > **ArchiMate 3.1** to filter the list to ArchiMate® 3.1 diagrams only.
4. Click on **ArchiMate 3.1**, and a list of available ArchiMate® diagrams will be shown in the right section of the window under **Diagram Types**.
5. Choose **Application** because we are creating an application-type diagram.
6. Click **OK** to confirm your choices.

You will see that a new **Metamodels** package has been created with a single empty **Application Component Focused Metamodel** diagram in it. Next, we need to add elements to the diagram.

In the previous model, we used a text element as a diagram label, which gave us the flexibility to change the font size, the location of the label, and the color, and we are free to type anything we want to type. This sounds great but it comes with a small cost, which is maintaining its content. If you rename your diagram (right-click on the

diagram, then **Properties** > **General** > **Name**), you need to rename the label content as well to reflect the new name, or else your content will be outdated.

Sparx has provided another way to add labels to diagrams, which has different advantages and disadvantages. Let's add a diagram label next.

7. Right-click on the diagram, and then select **Properties** > **Diagram** > **Appearance** > **Show Diagram Details**.
8. Click **OK** to accept the changes.

Notice that the diagram now shows four lines of information in the top-left corner: **Name**, **Package**, **Version**, and **Author**. The best advantage is that when you rename the diagram or package that contains this diagram, the label will update automatically to reflect the new names of the diagram or package. However, the biggest disadvantage is that you have no control over what to show or what to hide in this label, no control over the font size or color, and no control over the location. The label will always be displayed in the top-left corner, with the same four lines of information, the same font size, and the same font color. You either accept it as it is or reject it completely as there is no current possibility to customize it.

You have two ways to choose how to display diagram labels, so it is up to you to follow either of them. Just make sure that you remain consistent with your choice. The diagram still has no elements on it, so let's start by adding the focus (the primary) element.

Adding a focus element to the diagram

The focus element in our case is the **Application Component** element. This is the element that we are creating a diagram about, so it will be visually differentiated from other elements on the diagram by the following styling characteristics:

- It will be in the center of the diagram.
- It will be relatively larger than the other elements.
- Its border will be thicker than the borders of the other elements.

Let's add the application component to the diagram and make it the focus element. If you do not have the application component metamodel diagram opened, please do so by double-clicking it from the **Project Browser** and then go through the following steps:

1. From the **Toolbox**, locate the **Application Component** element and drag and drop it to the center of the diagram area.
2. Click on the **Application Component** element either on the diagram or in the **Project Browser**, and then press *F2* on the keyboard to rename it from `ApplicationComponent1` to `Application Component`.
3. Add the definition of **Application Component** in the **Notes** area, and use the complete definition from ArchiMate® 3.1 specification.
4. Style the application component as **App Arch**, change the thickness of the border to **2**, and, optionally, use the borderless notation or the rectangular notation.
5. Press *Ctrl* + *S* to save.

We have kept the rectangular notation for the application component in this diagram while using the borderless notation for the other elements. This is a styling choice, and we will be using the rectangular notation for all focus elements within the metamodels. Any style is a good style if you remain consistent in using it.

Interpreting an ArchiMate® metamodel

We need to add elements that can relate an element of the application component type to the diagram. We are following the ArchiMate® 3.1 specification, so will use the online material as a reference to derive our easier-to-read metamodels. The complete reference is available at https://pubs.opengroup.org/architecture/archimate3-doc/toc.html, but we will mainly use the following ArchiMate® 3.1 chapters in this book:

- *Chapter 5*, *Relationships* (https://pubs.opengroup.org/architecture/archimate3-doc/chap05.html#_Toc10045310)
- *Chapter 8*, *Business Layer* (https://pubs.opengroup.org/architecture/archimate3-doc/chap08.html#_Toc10045365)
- *Chapter 9*, *Application Layer* (https://pubs.opengroup.org/architecture/archimate3-doc/chap09.html#_Toc10045389)
- *Chapter 10*, *Technology Layer* (https://pubs.opengroup.org/architecture/archimate3-doc/chap10.html#_Toc10045407)

- *Chapter 12*, *Relationships Between Core Layers* (https://pubs.opengroup.org/architecture/archimate3-doc/chap12.html#_Toc10045440)

You can bookmark the provided references in your browser, download them to your computer, or print them if you like—whichever you feel is more convenient for you.

Looking carefully at ArchiMate®'s metamodel, you can see that it has a lot of useful information, but it can be slightly difficult to read because of the use of generic type names such as **Application Internal Active Structural Element** to refer to an application component.

We will show you how to interpret the standard ArchiMate® 3.1 metamodels in step-by-step instructions, to build an easier and more focused metamodel that can be used as a future reference. We will do the following:

- Start by analyzing the reference metamodels and understand how to read them.
- Extract the statements that are conveyed within the metamodels and use them as building blocks for our focused metamodel.
- Build our first focused metamodel.

The more you get familiar with the ArchiMate® specification, the easier it becomes to interpret its metamodels, so let's start developing this skill.

Analyzing the reference metamodels

Look carefully at *Figure 70: Application Layer Metamodel* in the *ArchiMate® 3.1 Specification*, *Chapter 9* (https://pubs.opengroup.org/architecture/archimate3-doc/chap09.html#_Toc10045390) and notice the following:

- There is one element called **Application Internal Active Structure Element**. This, according to the specification, can include the **Application Component** and **Application Collaboration** elements.

Application collaborations are simply groups of application components collaborating to fulfill a given objective. An **enterprise resource planning** (**ERP**) solution, for example, is an application collaboration as it can aggregate multiple different

application components, but they all serve the same objectives. Structure elements describe *what* makes up the application components.

- The **Application Interface** element is also a structural element, but it is the part of the application that is exposed to the enterprise , so it is considered as an **Application External Structure Element**.
- The **Application Internal Behavior Element** includes **Application Process**, **Application Function**, and **Application Interaction**.

All these elements describe the behavior of a component, such as—for example—transforming data from one format to another. Behavior elements describe the *how* part of the application components.

- The **Application Event** and **Application Service** elements also describe behavior, but they are exposed to the external enterprise, so they are **Application External Behavior Elements**.
- Finally, there is the **Data Object** element, which is a **Passive Structural Element**. This means that it cannot perform any actions on other elements, but other elements can *access* it and perform actions on it.

We've just introduced lots of new terminologies, do not worry about understanding all of them at once because we will introduce them with complete definitions and supportive examples as we progress. All that we need you to do now is to look at the **Application Internal Active Structure** element because this element **generalizes** the application component that we will use for our focused metamodel.

Identifying general statements

The next step is to identify the relationships that exist between the **Application Internal Active Structure** element and the other elements and put them into statements. We can identify the following generalized relationships from the diagram:

- An **Application Internal Active Structure** element can be *assigned to* an **Application Internal Behavior** element.
- An **Application Internal Active Structure** element can be *assigned to* an **application event**.

- An **Application Internal Active Structure** element can be *composed of* **application interfaces**.
- By default, an **Application Internal Active Structure** element can be *self-composed*, so it can be composed of or decomposed into other **Application Internal Active Structure** elements.
- Finally, an **Application Internal Active Structure** element can be *served by* an **application service**.

All of the mentioned relationships are defined in the referenced ArchiMate® metamodel diagram. If you cannot find them, please print a copy of that diagram on paper, put it side by side with the preceding list, take as much time as you need, and make sure that you can see and find all of them.

Building the focused metamodel

Let's take a moment to analyze the diagram and use the application component as a specialized element of the **Application Internal Active Structure** element. We can rephrase the first statement into three specialized statements, as follows:

- An **application component** can be *assigned to* an **application process**.
- An **application component** can be *assigned to* an **application function**.
- An **application component** can be *assigned to* an **application interaction**.

Let's have a quick overview of the formal definition of the *assignment* relationship before we go any further.

The assignment relationship

"*The assignment relationship represents the allocation of responsibility, performance of behavior, storage, or execution.*" (https://pubs.opengroup.org/architecture/archimate3-doc/chap05.html#_Toc10045314)

You can see a visual representation of this relationship here:

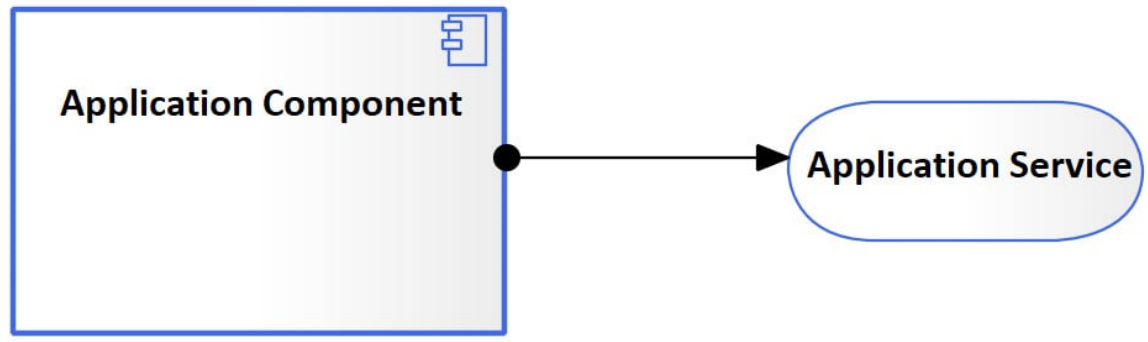

Figure – The assignment relationship

This means that part of the application component's behavior is defined within elements that it is assigned to, such as an application service. If an application component performs transactions such as processing claims, *how* claims are processed is defined in the form of claim processing functions, processes, services, and interactions, which are all behavioral elements. As a general rule, structural elements are assigned to the behavioral elements, because you need to have a structure in order to perform the behavior. You cannot perform the behavior from nothing.

Application events also describe behavior that affects an application component; a very good example of application events are batch jobs that run at specific times. Now, we have a fourth assignment relationship statement, as outlined here:

- An **application component** can be *assigned* to an **application event**.

Before our list of bulleted items grows too long, let's add the identified ones to the diagram. As we mentioned earlier, the behavior of an application component can be described using different behavioral elements such as application processes, functions, interactions, and events.

We need to add these elements to the diagram and define the possible relationships that can be created between the application component and these behavioral elements, as follows:

1. From the **Toolbox**, drag a new **Application Process** element and drop it onto the diagram area.
2. Rename the newly created element from `ApplicationProcess1` to `Application Process`.
3. Style the application process in **App Arch** style.

4. Select the desired modeling notation, whether rectangular or borderless.
5. Find the *assignment* relationship in the **Toolbox** and create an assignment relationship from the application component to the application process.
6. Repeat all previous steps to add **Application Function**, **Application Interaction**, and **Application Event** elements to the diagram, and don't forget to create an assignment relationship too.
7. Press *Ctrl* + *S* to save.

The diagram should look similar to the one shown here; you can, of course, organize the elements in a different order if you like, so you do not have to place them in the same locations that we did:

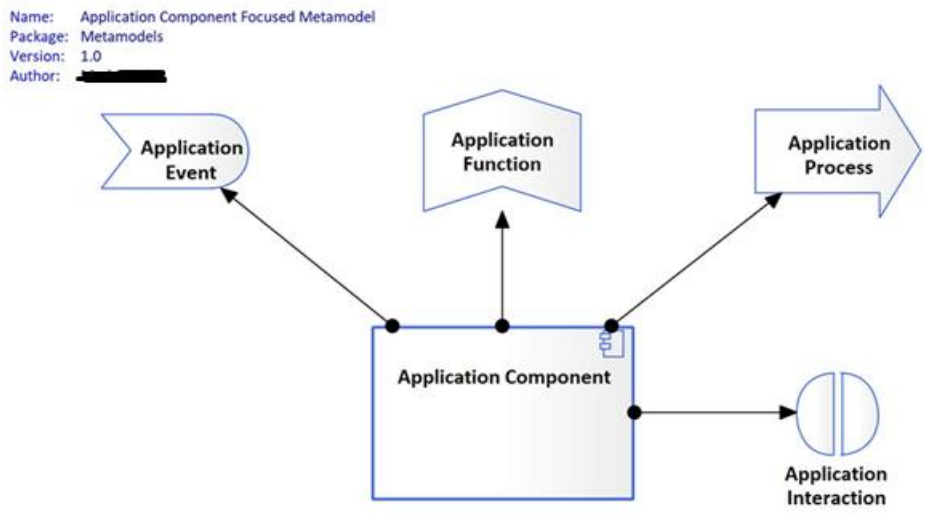

Figure – The application component focused metamodel so far

Notice that we are still adding elements, so we may need to move some of the elements later to make room for additional ones.

The composition relationship

The third and fourth generalized statements that we have captured from interpreting ArchiMate®'s metamodel tell us about another type of relationship, which is the **composition** relationship.

"*The composition relationship represents that an element consists of one or more other concepts.*" (https://pubs.opengroup.org/architecture/archimate3-doc/chap05.html#_Toc10045312)

You can see a visual representation of this relationship here:

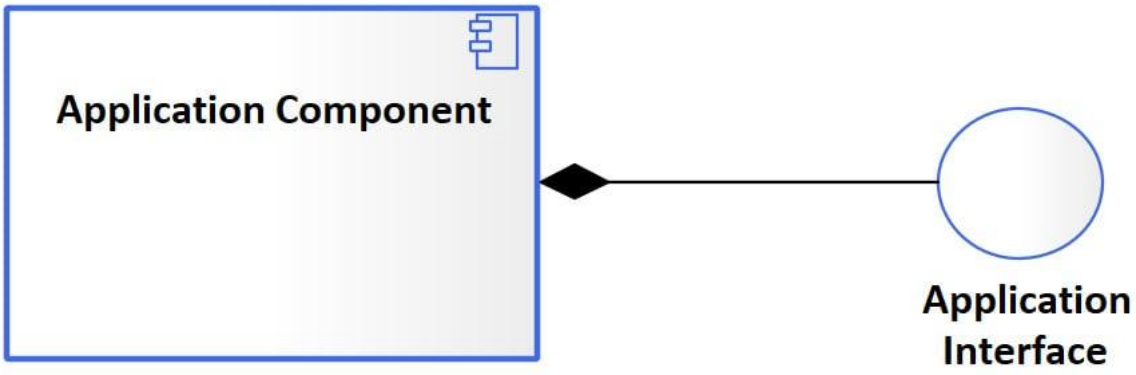

Figure – The composition relationship

Composition is the well-known *parent-to-child* or *whole-to-part* relationship. When an element, A, is composed of elements B, C, and D, it means that element A is the whole, and elements B, C, and D are its parts.

An interesting thing to know about the composition relationship is that it is **transitive**. This means that if a composed element is related to another element, it implies that the parent composing component also has the same relationship with that element.

Another interesting thing about the composition relationship is that any element can be composed or decomposed of larger or smaller elements of its type. So, an application component can be composed of many smaller application components, an application process can be composed of many smaller application processes, and so on.

Look back at ArchiMate®'s metamodel and we will explain how these relationships apply to the application component. First, there is a clear composition relationship between the application component and the application interface. There is also an implied self-composition relationship between the application component and itself. Let's write these down before we forget them, as follows:

- An **application component** is *composed* of **application interfaces**.
- An **application component** is *self-composed* of **application components**.

Additionally, there is an assignment relationship between the application interface and the application service. Since the composition relationship is transitive, it also means that the same relationship exists between the application component and the application service. Therefore, we can add the following statement to our list:

- An **application component** is *assigned to* an **application service**.

We now have sufficient elements and relationships to add to the diagram. We need to add the **Application Interface** element and add the proper relationships as we have stated them, so continue with the following steps:

1. From the **Toolbox**, drag a new **Application Interface** element and drop it onto the diagram area.
2. Rename it `Application Interface`.
3. Style it in the **App Arch** style.
4. Select the desired modeling notation, whether rectangular or borderless.
5. Find the *composition* relationship in the **Toolbox** and create a relationship from the application component to the application interface.

The black diamond-shaped end of the relationship must be connected to the parent element, which is the application component.

6. Now, you need to create a self-composition relationship for the application component, so click the *composition* relationship from the **Toolbox**, click on the application component, move the mouse a little to the top, and click once again on the application component.
7. Do not forget to add the **Application Service** element and create an assignment relationship from the application component to it.
8. Press *Ctrl* + *S* to save your work.

Your diagram should look close to the following:

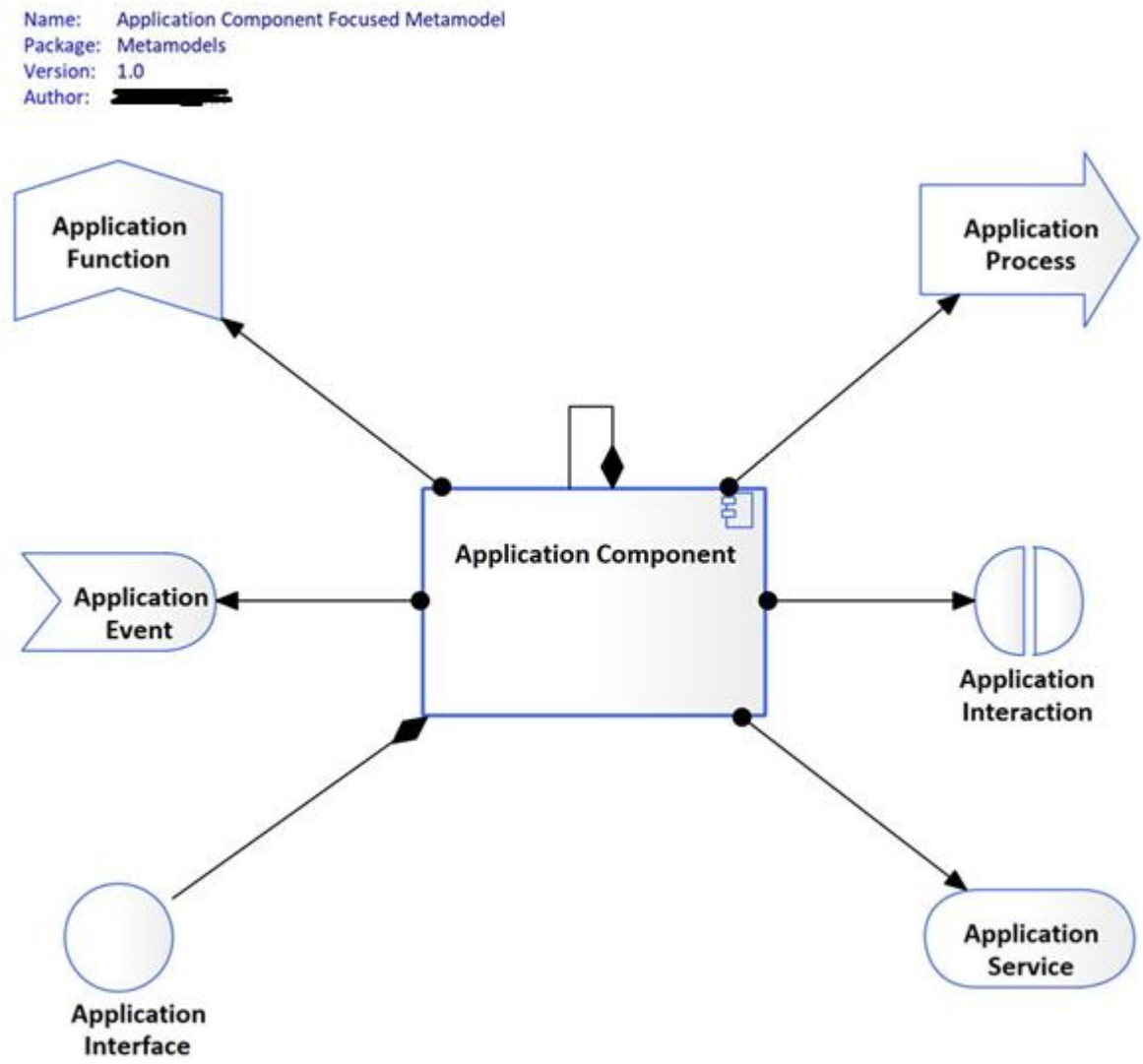

Figure – The application component focused metamodel up to this point

Notice how we have reorganized the diagram to fit the newly added elements, but everything still looks well organized. This is a habit that we want you to develop while building diagrams—always keep them organized as much as you possibly can, even if you are still drafting them.

The aggregation relationship

Another form of the composition relationship is known as **aggregation**.

"*The aggregation relationship represents that an element combines one or more other concepts*." (https://pubs.opengroup.org/architecture/archimate3-doc/chap05.html#_Toc10045313)

You can see a visual representation of this relationship here:

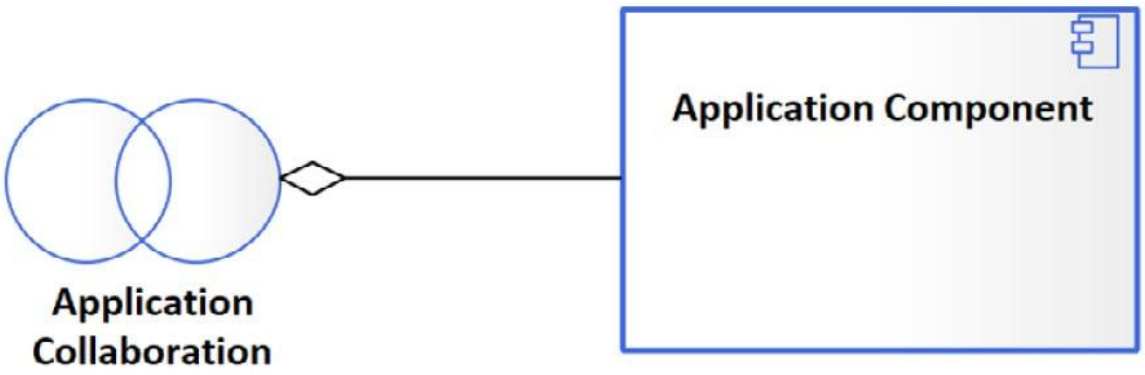

Figure – The aggregation relationship

Aggregation is similar to composition as they are both two forms of the *parent-to-child* or *whole-to-part* relationship; however, aggregation is *weaker* than composition and the child can live completely independently from the parent. In composition, the child element is a *part* of the parent element and cannot function outside it, so it is more like containment. In aggregation, on the other hand, the parent element is a *combination* of its independent children. Each can work independently, and each can work combined with other elements under different parents.

To help you understand the difference between composition and aggregation, think of a house, for example. It is composed of bedrooms, a living room, and a kitchen. All these components are indivisible parts of the house, and they cannot exist by themselves. They must exist within a house to have a useful purpose, and the house is not a fully functional house without them. On the other hand, a detached shed or a detached garage are independent components, but when they are related to the house, they form a better set of options. You can disassemble the shed, move it, or sell it without affecting the integrity of the house, and the shed will be perfectly installed somewhere else.

Another example is an ERP solution that can be made up of accounting, inventory management, sales, billing, **human resources** (**HR**), marketing, and much more. Each of these can be a standalone solution and can work separately, but together, they are parts of a bigger solution. ArchiMate® calls the big solution that is a combination (aggregation) of multiple application components an **application collaboration**. Based on this, we can state the following relationship:

- An **application component** can be *aggregated in* an **application collaboration**.

Next, we need to add this element to the diagram. Following the steps that we took to add the other elements, we need to add an **Application Collaboration** element and connect it with an aggregation relationship to the application component, as follows:

1. Find the **Application Collaboration** element in the **Toolbox** and use it to add a new element to the diagram.
2. Rename the element `Application Collaboration`, style it with the **App Arch** style, and optionally change the notation to borderless.
3. Find the *aggregation* relationship in the **Toolbox** and create one from the **Application Collaboration** element to the **Application Component** elements. The diamond-shaped end must always be connected to the parent element.

The diagram now shows that an application collaboration is an aggregation of application components.

The access relationship

"*The access relationship represents the ability of behavior and active structure elements to observe or act upon passive structure elements.*" (https://pubs.opengroup.org/architecture/archimate3-doc/chap05.html#_Toc10045319)

You can see a visual representation of this relationship here:

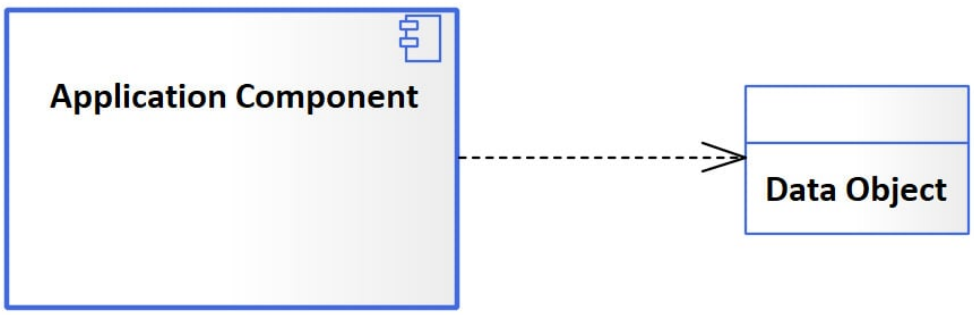

Figure – The access relationship

In simpler English, the access relationship means that an element (whether structural or behavioral) can access the indicated data object. Based on the ArchiMate® application metamodel, we can see that an application component is assigned to an application behavior element, which accesses the data object. Therefore, we can say the following:

- An **application component** *accesses* a **data object**.

Now, we need to add a **Data Object** element to the diagram and connect it with an access relationship, as follows:

1. Drag a **Data Object** element from the **Toolbox** and drop it onto the diagram area.
2. Rename it `Data Object`, and style it as **App Arch**.

You cannot change the notation of data objects, so there is no possibility to choose between the rectangular and borderless notations.

3. Find the *access* relationship in the **Toolbox** and create one from the application component to the data object.

Keep in mind that an access relationship can *only* exist between active elements on one side and passive structure elements on the other side. It cannot exist between two active structures, two active behaviors, or even between an active structure and an active behavior element. The arrowhead of the relationship must always be connected to the data object.

The serving relationship

The last relationship that we have identified is **serving**.

"*The serving relationship represents that an element provides its functionality to another element.*" (https://pubs.opengroup.org/architecture/archimate3-doc/chap05.html#_Toc10045318)

You can see a visual representation of this relationship here:

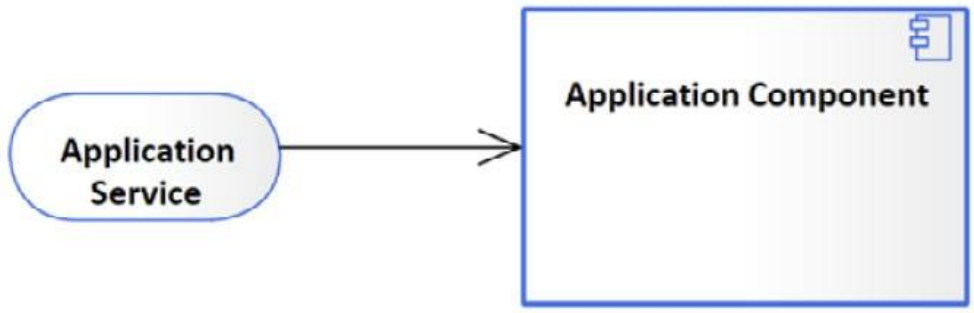

Figure – The serving relationship

If element A is serving element B, it means that element B is using the functionality that is provided by element A. It can be read in the opposite direction, as element B is served by element A. A web application is served by a web server, for example.

As per ArchiMate®'s metamodel, we can add the following two statements:

- An **application component** is *served* by an **application service**.
- An **application component** is *served* by an **application interface**.

We already have the **Application Component**, **Application Service**, and **Application Interface** elements on the diagram, so we only need to add the relationships, as follows:

1. Find the *serving* relationship in the **Toolbox** or use the arrow-shaped action menu item to create a serving relationship going from the application service to the application component.
2. Repeat the previous step to create a serving relationship going from the application interface to the application component.

Because there are relationships already in place between the component and the service and between the component and the interface, Sparx will place the newly added serving relationships above the old ones, so you will need to manually adjust and separate them. Click and hold the relationship, then drag one away from the other. This is what the diagram should look like at this point:

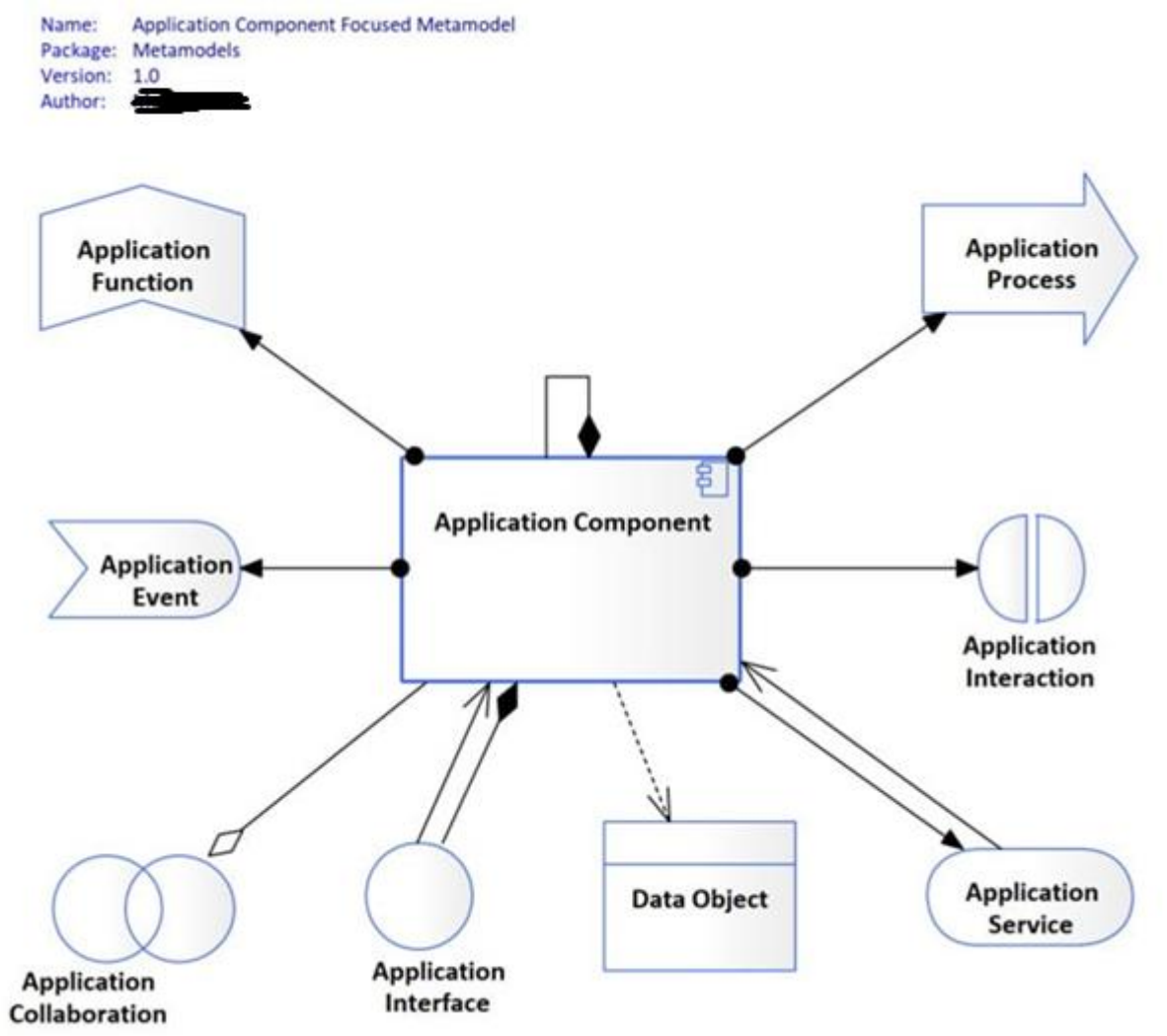

Figure – The application component-focused metamodel up to this point

Don't forget to save your work before we continue. There are still more elements and relationships to add to the focused metamodel. They belong to different architecture layers, so we need to refer to additional reference materials to know what these elements are and how they are related to the **Application Component** element.

Adding elements from other architecture layers

All the elements that we have placed on the metamodel so far belong to the Application architecture layer, but the application component is also related to other elements from other layers such as the Business and Technology architecture layers. ArchiMate®'s metamodels do not show all the possible relationships in one place, and you will find yourself moving back and forth between multiple locations to

get the full picture and find out all the possible relationships to an element. we will do the following:

- Find the elements from the Business and Technology architecture layers that can have a relationship with **Application Component** and identify these relationships.
- Add elements from the Business and Technology layers to the application component-focused metamodel diagram.

Once you see the completed focused metamodels, you will understand why they are handier and easier-to-read references than standard metamodels.

Finding and reading references

In ArchiMate® 3.1, the relationships between elements from different layers are detailed in *Chapter 12* of the specification (https://pubs.opengroup.org/architecture/archimate3-doc/chap12.html#_Toc10045440). As we mentioned earlier, the biggest advantage of focused metamodels is that they save you time searching for the complete picture in multiple chapters and diagrams within the ArchiMate® specification.

For now, we still need to use the specification until our focused metamodels become complete and reliable. We can see the following:

- An **Application Internal Active Structure** element (such as **Application Component**) can be served by **Business Service** and **Business Interface** elements.
- An **Application Internal Active Structure** element can be served by **Technology Service** and **Technology Interface** elements.

Now we know all this information, we need to add it to the focused metamodel to make it as complete and reliable as we can. Remember that we can always come back to these metamodels and add new elements when we need to, so it does not have to be 100% complete at this point and it is anti-agile to make it so.

Adding business and technology elements

Follow these steps to add business architecture elements to the diagram:

1. Change the **Toolbox** to the business architecture toolbox. You can do that by clicking the hamburger menu on the left corner of the **Toolbox** window and selecting **ArchiMate 3.1** > **Business**.
2. Locate the **Business Interface** element, drag it, and drop it onto the diagram area.
3. Rename it `Business Interface`, style it as **Biz Arch**, and optionally change the notation to borderless.
4. Find the serving relationship in the **Toolbox** and create a connection from the business interface to the application component.
5. Repeat *Steps 2* to *4* to add a **Business Service** element and create a serving relationship to the application component.
6. Press *Ctrl* + *S* to save your work.

Repeat almost the same steps to add the technology architecture elements (**Technology Interface** and **Technology Service**). We will leave this as an exercise for you, but remember to change the **Perspective** setting to **ArchiMate 3.1** > **Technology** and apply the **Tech Arch** style on the technology elements this time. Your diagram must look like this, or close to it:

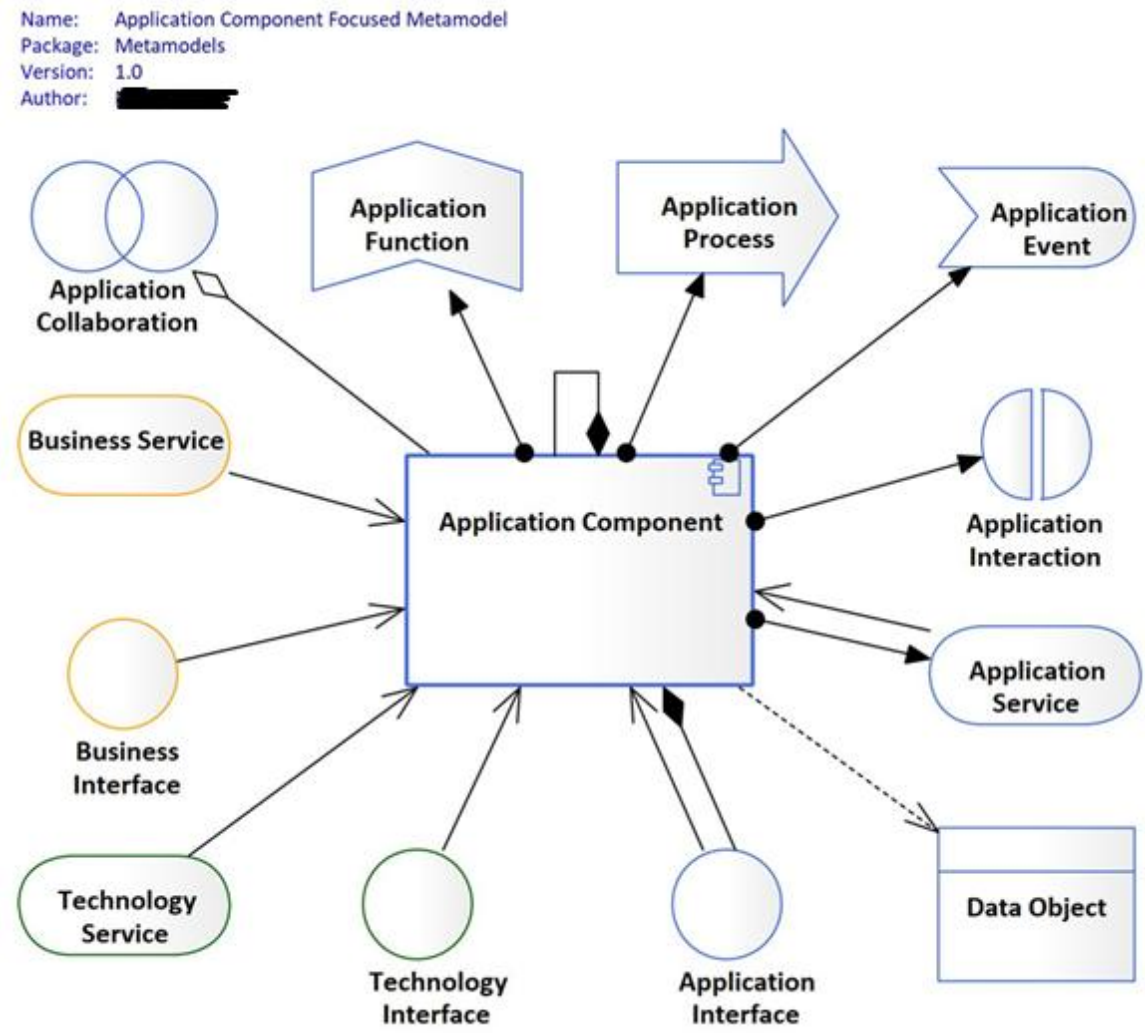

Figure – Application component-focused metamodel

We have sufficient details on the focused metamodel to guide us in developing new application component diagrams. We have the definition of what an application component is (in the notes), we know most of the elements that can relate to it, and we know which types of relationships we can use. Having all the information that we need in one place is way easier than digging through the reference material every time we need to know something.

We do not need to use all these elements in every application component diagram, but we only use what makes sense to the audience per diagram. The application component context diagram that we developed in the previous model used some of the elements that are shown in the metamodel. Other application component diagrams may use a different set of elements based on what they are addressing. Say, for example, that you are developing a new application services catalog

diagram. The only two elements that you need to use are the **Application Component** and **Application Service** elements.

The relationships between elements will remain the same as per the metamodel in every diagram. The relationship between an application component and an application service, for example, is always assignment, regardless of the type, scope, and audience of your diagram.

Modeling best practices

It is always a good practice to organize your diagrams in a way that makes them attractive to readers. Aligned elements on a diagram look better than misaligned ones, elements with similar spaces between them look better than elements with random spacing, and elements of the same size look better than elements of random sizes. So, do your best to have your diagrams as organized as possible because the difference this practice makes it worth the effort.

We will be giving some guidelines that we had set for us when creating diagrams. You can follow these guidelines as they are, or you may add to them from your own experience as well. These are not part of any standard but are based on our personal experience in modeling, so nothing is written in stone. You may agree or disagree with us on some, we may learn something new tomorrow that results in changing some of these guidelines, or you may have a different experience background than ours, so you may add your own touch and flavor. We all learn something new every day, no matter how old or experienced we are.

Keeping your diagram focused

A single diagram better tells a single main idea. If you put many ideas in a diagram, the readers will get confused about what are you trying to address. As you have seen in the application context diagram we have created, the focus was on the **Tracking App** component, and the entire diagram was made to serve the single idea of describing it. If we had added some unnecessary details about the **Trading Web** application component, for example, the diagram would be confusing to the reader as it would be describing two components at the same time.

You can see an example of such a diagram here:

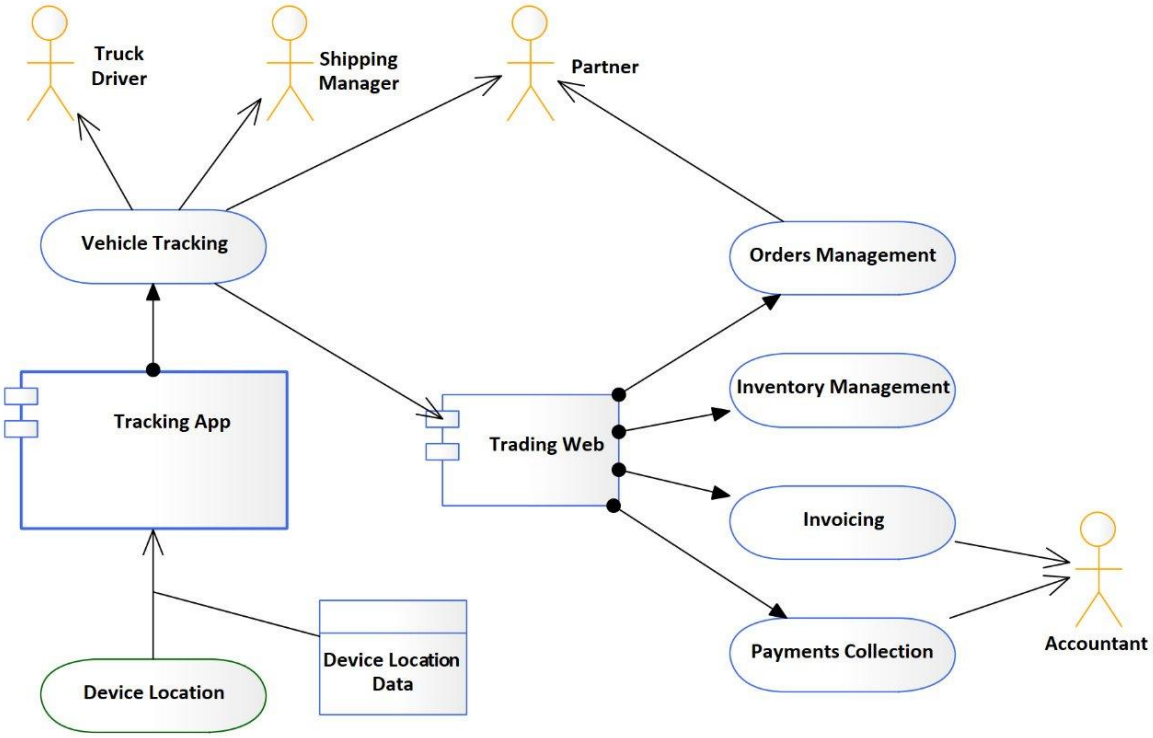

Figure – Avoid introducing multiple ideas in a single diagram

As you can see in the preceding diagram, we have added some of the services that the **Trading Web** component provides, and added a new **Accountant** business Actor who will be using **Trading Web** but has nothing to do with **Tracking App**. This information is irrelevant to the **Tracking App** context diagram and only adds noise to it. Even though the preceding diagram is still correct, it has more information than the reader needs to know. Readers will be confused about whether you are telling them about **Tracking App**, **Trading Web**, or all the available application services.

You can tell readers more about the **Trading Web** component and who will use it, but this is better off in a separate diagram that has **Trading Web** as the focus and adds the other elements as context around it. Knowing how the **Accountant** Actor will be served by an application that is not **Tracking App** is unnecessary noise and will only cause confusion. On the other hand, it will be helpful to know how all Actors are being served and by which applications, but again, this is better off in a separate diagram that only focuses on Actors and services.

Sometimes, you may need to have many elements in one diagram, which is fine if the idea that you are trying to show in the diagram is how complex the situation is. If the purpose of your diagram is to list all the application components—for example, all the application services that exist within the enterprise, and all the data objects they exchange—then you will end up with a complex diagram with lots of overlapping

connectors and shared data sources. But in this case, *complexity is the idea* that you are conveying, so avoid introducing a second idea such as where the application instances are deployed or which technology stacks each application requires.

Remember that in Sparx, you can create child diagrams for any element in the repository, which allows you to create drill-down diagrams showing different levels of detail. If your diagram looks complex and your purpose is not to show complexity, we highly advise you to break down the diagram into multiple smaller and simpler diagrams.

Fitting your diagram onto a single page

It may sound difficult, but if you follow the first advice of keeping your diagrams focused, this one will be easy to achieve. There is no harm in having large diagrams that span over multiple pages, but keep in mind that not every reader will be reading your diagrams from the Sparx **user interface** (**UI**), and not every reader has a monitor the same size as yours.

Additionally, if you publish your enterprise architecture content in a Word document, large diagrams will be shrunk automatically to fit within the page. This is acceptable if your diagram uses a little extra space beyond the page border as this will not affect its readability, but if it uses more than two pages, readers will find it extremely difficult to read it, and it will be of no value to them. It is hard to realize that such an effort that might have taken hours of your time is ultimately useless.

You can set the size of a diagram to any standard paper size or set a custom size if needed by right-clicking on the diagram and selecting the suitable page size from the **Properties** > **Diagram** > **Advanced** > **Page Setup** > **Paper Size** drop-down list. You can use the same dialog box to change the page orientation from **Portrait** to **Landscape** or the opposite. You can show/hide page borders by checking/unchecking **Properties** > **Diagram** > **Hide Page Border**. Notice that you can do this for a single diagram or all diagrams.

Adding only the necessary information

In some cases, showing details can be beneficial, and in other cases, hiding them can be more beneficial. It all depends on the purpose and the target audience. Take, for example, the **Tracking App** context diagram. We have shown that it uses a single technology service, which is the **Device Location** service, but this is not a

complete list of services, to be honest. For an application to work on a device in an environment (whether it is a personal computer, a smartphone, a gaming console, or a **virtual machine** (**VM**)), it needs to be provided by many other services, such as storage, computing, network gateways, subnets, load balancers, and containers, to name but a few, as the list can go on much longer. Having all these services listed in a diagram that is supposed to provide an overview of what an application is can end up containing more details than what the target audience needs to know, and it needs to be avoided.

However, if our target audience is a cloud architect, for example, we will need to list all the cloud services that will be needed, but that will be another diagram: a separate diagram that targets a different audience and tells a different message.

Paying attention to your diagram's appearance

As an architect, most of your deliverables will be in the form of diagrams. The quality of your diagrams *will* reflect your quality as an architect, so make sure that you are reflecting the best possible image of yourself. Even if you have the ultimate knowledge in all architecture domains, do not compromise the appearance of your diagrams at any cost. Modeling is an art, and your diagrams will carry your signature for as long as the enterprise architecture repository is alive, so make sure that you are putting your name on the most beautiful artifacts that you can make.

Important Note

In a nutshell, never compromise on making outstanding diagrams at any time. To make outstanding diagrams, you must pay attention to the details of how they look.

You need to keep this advice in your mind always, no matter whether your diagram has 2 elements or 20, whether your diagram will be shared with your coworkers or with the **chief executive officer** (**CEO**), or whether it is a draft version or a final version. Sparx provides a fair number of sizing, spacing, and alignment tools in the **Layout** > **Alignment** toolbar that will make your job easier, so use them effectively.

Knowing your audience

This is the most important and the most obvious rule that you need to follow in every diagram, presentation, and publication you write. This rule is not mine, in fact, and

every person who is responsible for building and developing products knows it, whether the products are books, electronic devices, vehicles, or houses. In each of those cases, you must know your target audience, or else you will end up building something that no one is interested in.

In your case as enterprise architects, if you are targeting a C-level audience, you need to provide completely different content than what you would do if you were targeting technology gurus or professionals for a specific product. It is extremely important to realize that your job is to close the gaps of understanding among the different stakeholders and to avoid creating new ones. Creating diagrams—or creating architecture content in general—that do not address the concerns of the target audience will create new gaps for sure, and the least damage you would cause will be your wasted time and the additional confusion that will arise as a result.