THE Open GROUP

ArchiMate® 2.0 Understanding the Basics

A White Paper by:

Gerben Wierda

February 2013

ArchiMate® 2.0 – Understanding the Basics

Document No.: W130

Published by The Open Group, February, 2013.

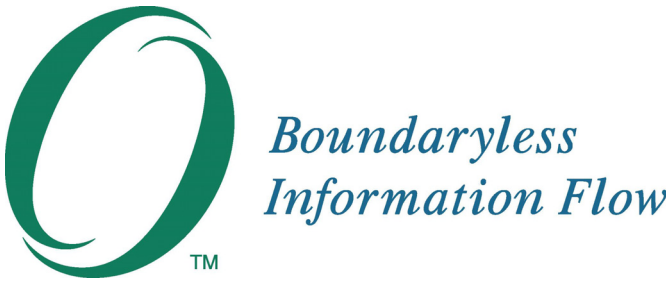
Any comments relating to the material contained in this document may be submitted to:

The Open Group, 44 Montgomery St. #960, San Francisco, CA 94104, USA

or by email to:

ogpubs@opengroup.org

Table of Contents



Boundaryless Information Flow™
achieved through global interoperability
in a secure, reliable, and timely manner

Executive Summary

ArchiMate, an Open Group Standard, is an open and independent modeling language for Enterprise Architecture that is supported by different tool vendors and consulting firms. ArchiMate provides a notation to enable Enterprise Architects to describe, analyze, and visualize the relationships among business domains in an unambiguous way.

This White Paper, ArchiMate® 2.0 – Understanding the Basics, is intended as a first introduction to the ArchiMate modeling language for those who want to become proficient in the language (e.g., Business and IT Architects), as well as for those who just want to be able to understand diagrams presented to them (all others).

Introduction

This White Paper is intended as a first introduction to the ArchiMate modeling language. It is both an introduction for those who want to become proficient in the language (e.g., Business and IT Architects), as well as for those who just want to be able to understand diagrams presented to them (all others). It is a slightly amended excerpt from the book *Mastering ArchiMate* by Gerben Wierda.

Elements and Relations: 3x3x3

When modeling Enterprise Architecture, we need a language that knows about the concepts of Enterprise Architecture, and the ArchiMate language does a good job. To start with, you need to know that the ArchiMate language is built from three types of elements:

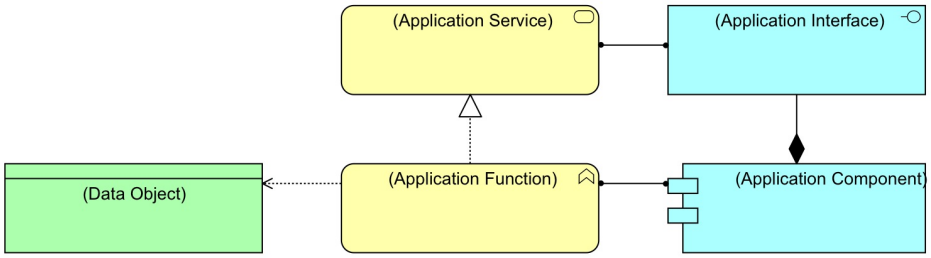
- Elements that act (active elements)
- Elements that represent the behavior of those elements that act (behavioral elements)
- Elements that cannot act and which are acted upon by that behavior (passive elements)

The three element types, connected by relations, can form sentences of sorts. A pickpocket (the application) steals (the application function) a wallet (the data). This is what makes the ArchiMate standard a grammar/language. Your model is a story of sorts. It tells the reader the basic structure of the story: who acts on what. The structure of the ArchiMate grammar is partly based on the subject-verb-object pattern from natural language.

Let's start somewhere in the middle, at the application and data level of Enterprise Architecture.

Applications and Business

An application is modeled in the ArchiMate language, as seen in View 1.



View 1: The Basic Application Pattern

This is possibly the first snippet of the ArchiMate language you have ever encountered, and it already has five element types and four relation types, so we are going to take time to describe it.

Roughly, the two yellow and two blue elements in the image together make up the "application" and the green element is the "data" on which the application operates; i.e., think of it as the blue and yellow elements representing a word processing application and the green element representing the document being edited. The lower three elements represent the internals of the application and the data. The two upper elements represent how the application is used by and is visible for a user.

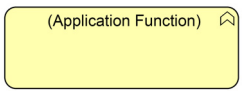
One of the most essential aspects of the ArchiMate language is that modeling the behavior is separated from modeling the structure. The blue elements in View 1 represent the active structure ("who") and the yellow elements represent the behavioral aspects of the "who" elements – they are two sides of the same coin.

It seems rather excessive that you need four elements to model one application. We will later see that you can simplify this, even to a single one of these elements, but for the moment it is very important to understand what the underlying structure looks like. Actually, the lack of addressing this in documents and courses I

have seen has been a major reason for writing the book *Mastering ArchiMate* from which this White Paper is excerpted. An understanding of the foundation is required to model well in ArchiMate.

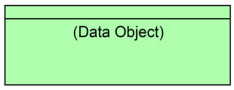
Having said that, here is a short explanation of the five element types in the image:

This is the Application Component. It stands for the "actor" that an application in your Enterprise Architecture landscape actually is. It is one side of the coin of which Application Function (its behavior) is the other.



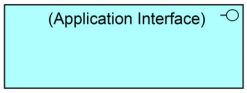
This is the Application Function. It stands for the behavior of the Application Component, how the application can act. It is one side of the coin of which Application Component is the other.

Application Component and Application Function are, in fact, inseparable. You cannot have an actor that does not act unless the actor is dead, and in that case it should not appear in your architecture. So, you cannot have an act without an actor. Again, later we will see how to leave things out of our views and models, but for now we will stick to the details.



This is the Data Object. It is what the Application Function acts upon. The Application Function might create, read, write, update, or delete the Data Object. Conceivably, the Data Object might not be needed; you can imagine behavior that does not access a passive element. You can also imagine that you do not model Data Objects that only exist *inside* your application. That variable in the application code is not modeled. But as soon as the Data Object is visible to other parts of your landscape, or when it is persistent, it should be there. Generally, that means we only model (semi-)persistent Data Objects. Note: this is not the file or the database itself – those are represented on a lower level: the infrastructure level. We are still one abstraction level up. To illustrate the difference: an RTF file can be both an MS Word Data Object or an Apple TextEdit.app Data Object, depending on which application is used to access it.

Two elements in the image have not been explained yet. They have to do with how the application is used/seen (by the business or by other applications):



This is the Application Interface. It stands for the route via which the application offers itself to the business or to other applications. Note: both separate concepts (used by people and used by other applications) are supported by this one element. One example would be a Graphical User Interface (GUI), but it can as well be an Application Programming Interface (API), a Web Service, or one of the many other ways an application offers itself to other "actors" in your landscape. Given that difference in use (and thus, as Uncle Ludwig[1] would say, meaning), it is unlikely that the same interface will be used by both a human or another application. But it can be; e.g., in the case of a Command Line Interface (CLI) used by a scheduler system. The Application Interface is a "handle" of the "actor" that is the Application Component. It is one side of a coin of which Application Service is the other.

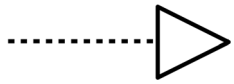
This is the Application Service. It stands for the "visible" behavior of the Application, how the Application Interface can act *for a user*. It is one side of the coin of which Application Interface is the other. Here again, the service may be "technical" in that it is offered by one application to other applications, or it may be part of your Business-IT integration: services offered to business processes (behavior of humans). The same type of element is used for both.

Now, apart from the elements, there are relations between the elements. There are four in the initial image:

[1] "Uncle Ludwig" stands for Ludwig Wittgenstein, the 20th century analytic philosopher. His work is relevant for the relation between grammar/language (e.g., the ArchiMate language) and meaning.

This is the Access relation. The access relation always depicts a behavioral element accessing a passive element. Here it depicts the behavior of the application (its function) accessing a passive Data Object (e.g., something that in the end resides in a file or a database). The arrowhead is optional and it may depict read or write access.

This is the Composition relation. It means that the element at the end with the diamond is the *parent* of the element on the other end and that the child *cannot exist independently* from the parent. The relation depicts the composition of larger wholes out of smaller parts, but it does not mean the set of children modeled must be necessarily complete in your model: there may be parts that are not modeled. This relation could also, for instance, be used to show that an Application Component has various subcomponents.

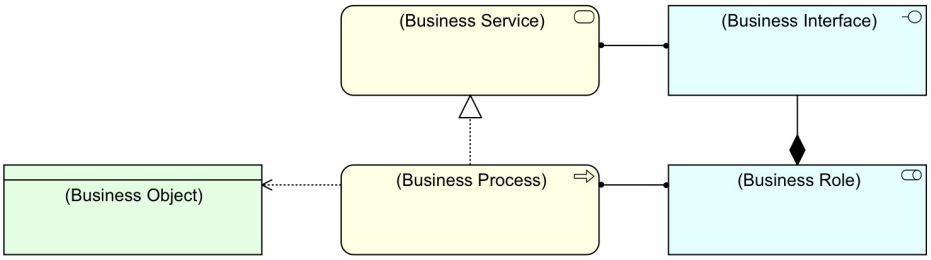


This is the Realization relation. This has two types of use in the ArchiMate language. Here it means that the element at the end without the arrowhead is the element that "creates" the element at the end with an arrowhead: the application's internal functionality realizes a service, which is the externally usable functionality of the application.

This is the Assignment relation. This also has more than one meaning in ArchiMate. Here, it means that one side (the active element) *performs* the behavior that is the behavioral element on the other side.

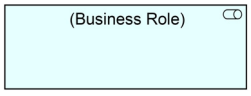
Important and possibly initially confusing aspects of ArchiMate are thus that – while we have multiple elements (structural and behavioral) representing what is in the mind of many a single thing (an application or an application interface) – we also have single elements (and as we will later see, relations) that can be used with multiple meanings. When you get the hang of it, it all becomes pretty natural just like it is with natural language, but if you are looking for a strictly disjunct (made of independent concepts) approach to a modeling language (e.g., like a programming language or like UML), it might be confusing in the beginning.

Having modeled an application, we can turn to modeling the way this application is used by the business. And before that, we need to look at the way the business and information level of Enterprise Architecture is modeled in the ArchiMate language. Luckily, it looks a lot like what happens on the application level so it is easy to understand now and it can be seen in View 2.



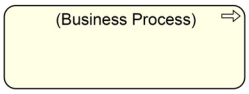
View 2: Basic Business Process Pattern

I have left out the actual human actor – the one that fulfills the Business Role – for now, to stress the equality between this pattern and the one about the application in View 1. It looks quite the same and that is not a coincidence. The relations are the same as with the application image above. The new element types are:

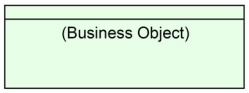


This is the Business Role. The Business Role is an "actor" in ArchiMate, but it is slightly more complicated than that, because it is an abstract sort of actor based on being responsible for certain behavior. The real actors are people and departments

and such. ArchiMate has an element type for those as well, but we leave that for later. Business Roles can perform Business Processes.

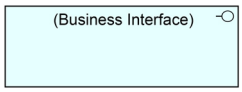


This is the Business Process. It stands for a set of causally-related activities that together realize services or create elements. Roles can be assigned to a process, they perform the process, just as the Application Component performs the Application Function. Just as on the application layer, a Business Process cannot exist without a Business Role (which does not mean you must model both, I am talking about the reality you are modeling), they are two sides of the same coin.

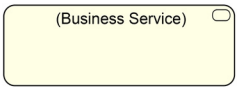


This is the Business Object: the abstract element that is created or used by a Business Process. Think of elements like a payment or a bank account or a bill. Though it is named Business Object, it is more like a Concept.

Again, just as with the application, these elements can make sentences of sorts. The proverbial "second-hand car sales" role performs the "sell second-hand car" process, which creates a "bill". Criminal, crime, and – in this case, possibly – proof of crime.

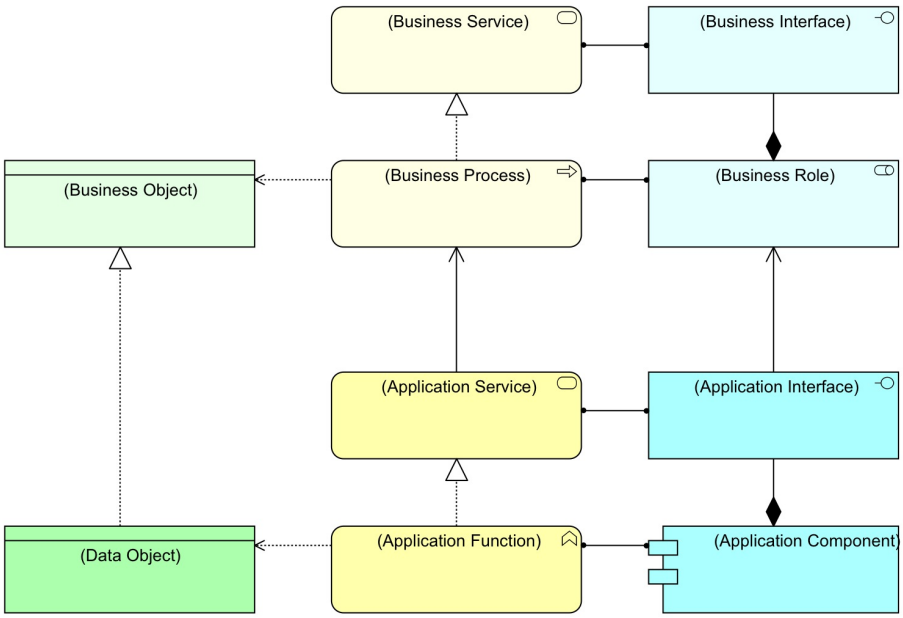


This is the Business Interface: the way the role interacts with others. You can think of it as a "channel"; e.g., phone, mail, meeting, etc. The interface is the visible manifestation of a role.

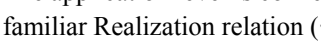


And this is the Business Service: more or less what it is always about in an organization. This is the reason for the existence of the process. This is the service it offers to (and thus can be used by) others (either inside or outside the company). A company might offer many services; e.g., a bank offers at the most abstract level a "savings" service or a "loan" service.

Having set up a couple of basic elements and relations, we can now fill in the missing links. Because on one hand we have the business that offers services to others, on the other hand we have IT that offers supporting services to the business. Together they look like View 3.



View 3: Basic Application is Used by Basic Business Pattern

The application level is connected to the business level by three relations. On the left we see the already familiar Realization relation (). Here it means that the Data Object realizes the Business Object. In a concrete example: the "bank account" Business Object may be data (a Data Object) in an accounting application; it is the same item's representation in a different architectural layer.

In the middle and on the right we see a new relation:

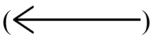
This is the Used-By relation. It means that the element at the end without the arrowhead is used by the element at the end with the arrow head. The Application Service, for instance, is Used-By the Business Process. The Application Interface (e.g., the Graphical User Interface) is Used-By the Business Role. Note especially that the definition is passive: it is not "uses" but "is used by". The reason for that is that the direction of relations is important in the ArchiMate language, something that will be explained later on.

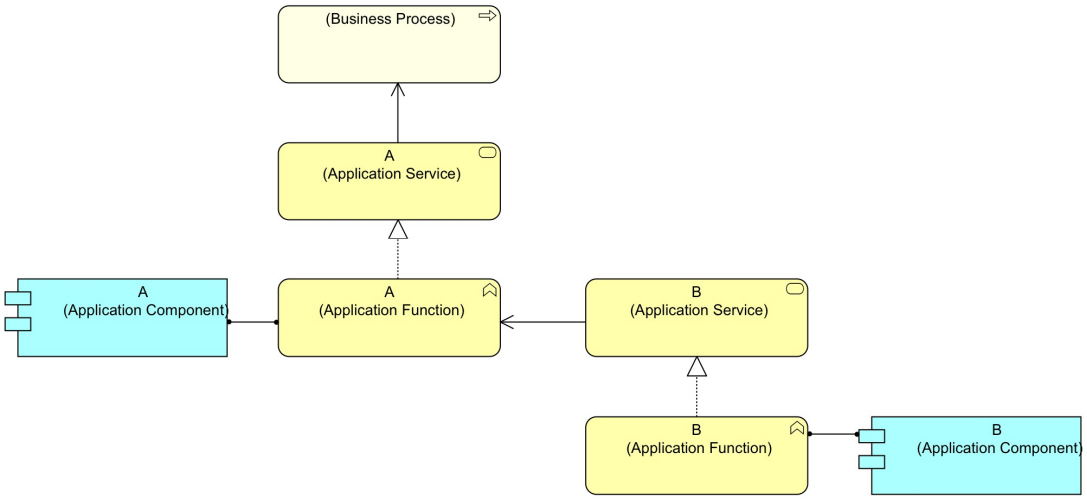
The fact that an Application Interface can be used by a Business Role is the other side of the coin of the same Used-By relation between Application Service and Business Process. They are twins. Both illustrate the "service-oriented" way that ArchiMate relates one layer to the next as far as actors and their behavior goes.

With what has been explained so far, you can already do much of the modeling you need in terms of "Current State" or "Project" Architecture. There are two more things that need to be explained before the basic set-up is complete: applications using other applications, and business processes/roles using other business processes/roles. Here again, what happens at business level and application level is identical, so we are going to illustrate one.

Double Use of Used-By

So far, our example has only shown Used-By as a relation *between* levels in your architecture. But the same relation type can also be used *within* a level. The business may use an application, but an application can also be used by another application.

In View 4 you see the same Used-By relation () twice. Once between the "A" Application Service and the Business Process that uses the "A" application, and once between the "B" Application Service and the "A" Application Function.

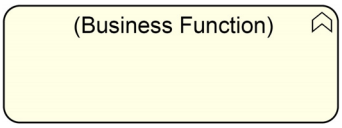


View 4: An Application Using Another Application

This means that the "A" application makes use of the "B" application. Though the relation is Used-By in both cases, it has quite a different role to play. Often, the definition of an Application Service that is used by the Business is pretty business-like in its description, something you discuss with a senior user or a process owner. But the relation between applications is more of a technical nature and you discuss it with application architects. The difference generally shows itself clearly in the types of names and descriptions of the service. It is hard to truly imagine an Application Service that is both used by another application and a Business Process. After all, both uses will be pretty different, unless we are talking about a fully-automated business process, which we will handle later.

Business Function

So far, we have modeled business behavior as a Business Process. But ArchiMate actually has two main work-horses for business behavior: Business Process and Business Function. The differences are subtle and in most situations you can use either. In the *Mastering ArchiMate* book, the difference is studied in more depth and there you can see that there is a nice way of combining them both when modeling your business layer. The Business Function looks like this:

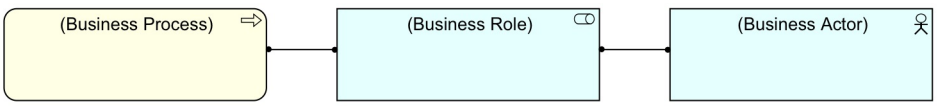


For now, as we are doing a superficial introduction, it is best to use the following guidelines:

- You use a Business Process if you are thinking of a causally-related set of behaviors ("activities") that in the end produce something, normally a Business Service. The Business Process is an outside-in view of business behavior, based on what the behavior produces. You assign a single Business Role to a Business Process or Business Function, but that leaves you free to have sub-roles assigned to sub-processes or sub-functions. If disjunct roles do something together, you should (officially) use a Business Interaction from the ArchiMate language (see Collaborations and Interactions) but you often have freedom to choose a different "not-proper" pattern (we'll get to this later).
- You use a Business Function if you are thinking of a grouping of related behavior based on, for instance, the same tools, same skills, or same role that performs it. In fact, the Business Function is best seen as an inside-out view of the business behavior.

Business Actor

Earlier we encountered the Business Role. The Business Role is an abstract sort of actor. But in a business architecture there are of course *real* actors: people, departments, or even business units or companies or maybe regulators. ArchiMate has an element for that: the Business Actor, as seen in context in View 5.



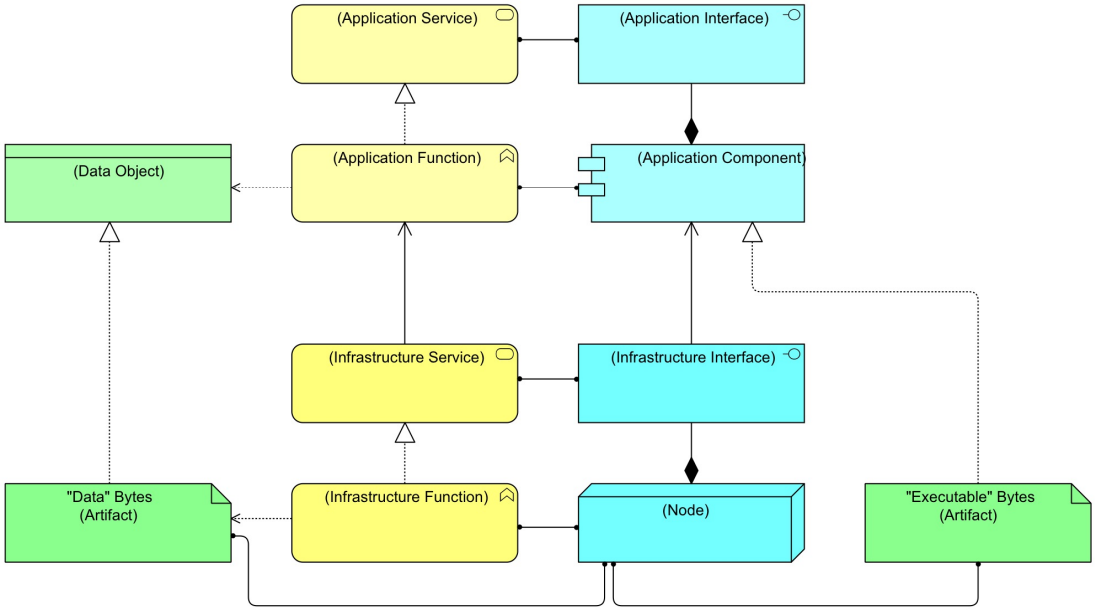
View 5: Business Actor in Context

On the left we see the already familiar Business Process to which a Business Role is Assigned. The Business Role *performs* the Business Process. On the right we see the new element type Business Actor, which is Assigned-To the Business Role. This must be read as the Business Actor *fulfills* the Business Role. The

ArchiMate language was designed to be economical with relation types, so it re-uses the relation for a slightly different meaning.

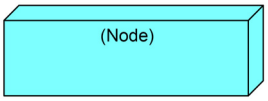
Adding Technical Infrastructure to the Mix

Just as the Business Process needs the Application Service to be able to be performed, the Application Function needs infrastructure to run. If we add the infrastructure to the application layer, we get View 6.

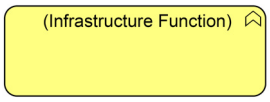


View 6: Basic Application uses Basic Infrastructure

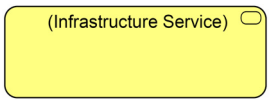
There are no new types of relations here, but there are new element types:



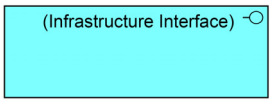
This is the Node. This is a slightly complicated concept in ArchiMate and more details follow later, but for now, think of it as the hardware and its system software, where files are stored or applications can run.



This is the Infrastructure Function. Just as with the Application Function and the Application Component, the Infrastructure Function stands for the behavior of the Node. They are two sides of the same coin. Note: the Infrastructure Function is new in the ArchiMate 2.0 standard. If you use older tooling or you encounter older views, it might not be there. Instead, in the ArchiMate 1.0 standard, the Node directly Realizes the Infrastructure Service. The ArchiMate language designers did the right thing and made sure all layers in the ArchiMate 2.0 standard have the same basic structure.



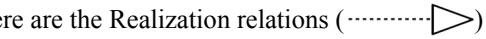
This is the Infrastructure Service, the visible behavior of the Node. In many ways, this is what the infrastructure is all about, what it can do for the applications, its reason of existence. This is what the applications need to function. Typical examples of an Infrastructure Service are, for instance, a "file share", "application execution", or a "database service". The latter may cause confusion, because you might wonder why that is an infrastructure service and not, for instance, something at the application level. The *Mastering ArchiMate* book goes into more detail, but for now, it is enough to say that the Node comprises both the hardware and the system software. A database system is generally modeled as system software and the database as an Artifact (see below).



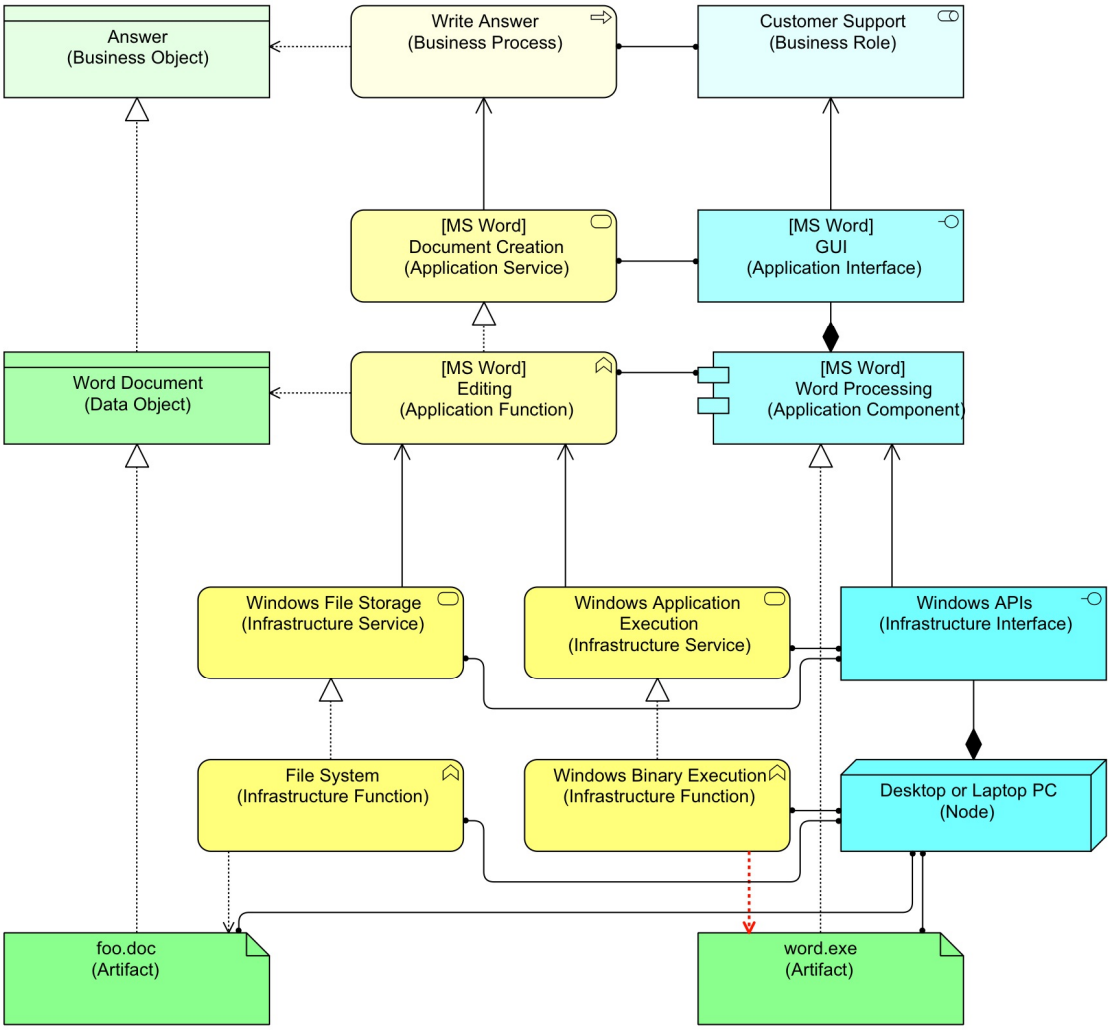
This is the Infrastructure Interface. The ArchiMate standard says it might be best thought of as a kind of contract that the "user" (the application) has to fulfill. An example would be a protocol, like the SMB or NFS protocol for file sharing and the TCP/IP or UDP/IP ports where the service is offered. Unless you are a dedicated Infrastructure Architect, you can generally do without this one. I mention it here to be complete as leaving it out would introduce the concept of "pattern" before it is wise to do so.

The last new element (for now) is the Artifact. This one is pretty simple. The best example is a file. Another often-used example is the actual database where your application's data resides. Another example is the actual "executable" (file) also known as the "binary"; that is, what your application is when you look at the "byte" level. Your application. The "data bytes" Artifact in the model above is the one that realizes the Data Object that is accessed by the Application Function. The "executable byte" Artifact are the bytes (a file, or a set of files often called a "software distribution") that the system can read and interpret as a program. On the infrastructure level, "a byte is a byte" in the sense that both passive (Data) elements and active (Application) elements are in the end nothing but a collection of bytes. Deep down, we get to the basic level of zeros and ones as we should.

The relations are mostly pretty straightforward. The Assignment relations (●——●) between Node and Artifacts stand for the fact that the Artifacts *resides* on the Node. In other words, it depicts where in your infrastructure you can find a file. Also pretty simple are the Used-By (<——) relations between Infrastructure Service and Application Function and between Infrastructure Interface and Application Component. If an Application Function needs access to a file that resides on a file system, the file system is an Infrastructure Service that is Used-By the Application Function. And its mirror is the Used-By relation from Infrastructure Interface to Application Component. This mirroring is depicted by the Assignment relation (●——●) between Infrastructure Interface and Infrastructure Service, exactly as happens in the levels above between the "interface" and the "service".

And lastly, there are the Realization relations () between an Artifact and Data Object and between Artifact and Application Component. This one is also pretty simple and easiest to explain by example. Suppose your application is Microsoft Word, then the file you are editing could be called "foo.doc". And if the application is MS Word, the Artifact realizing the Application Component could be "word.exe".

I have a detailed example model for you in View 7 to see everything in a single context.



View 7: Write Answer Process, supported by MS Word and a Standalone PC

The model shows someone writing a letter to a customer that contains an answer, supposedly to a question the customer has asked. Two infrastructural services are needed for this to work. The application should run and the document must be stored. In this example, everything happens on a standalone PC.

If you are an Enterprise Architect, you may think that all this detail is irrelevant and the language looks like a language for detailed design. Don't worry: using the ArchiMate language does not mean you absolutely must model all these details, the language can handle both: a roughly sketched Business Operating Model down to the nitty-gritty details you have to confront when you are in a project. I am simply using an example everybody knows to illustrate how the ArchiMate language works. And even in View 7, I have left some things out, and I have combined the Infrastructure Interfaces into one element. I could also have combined

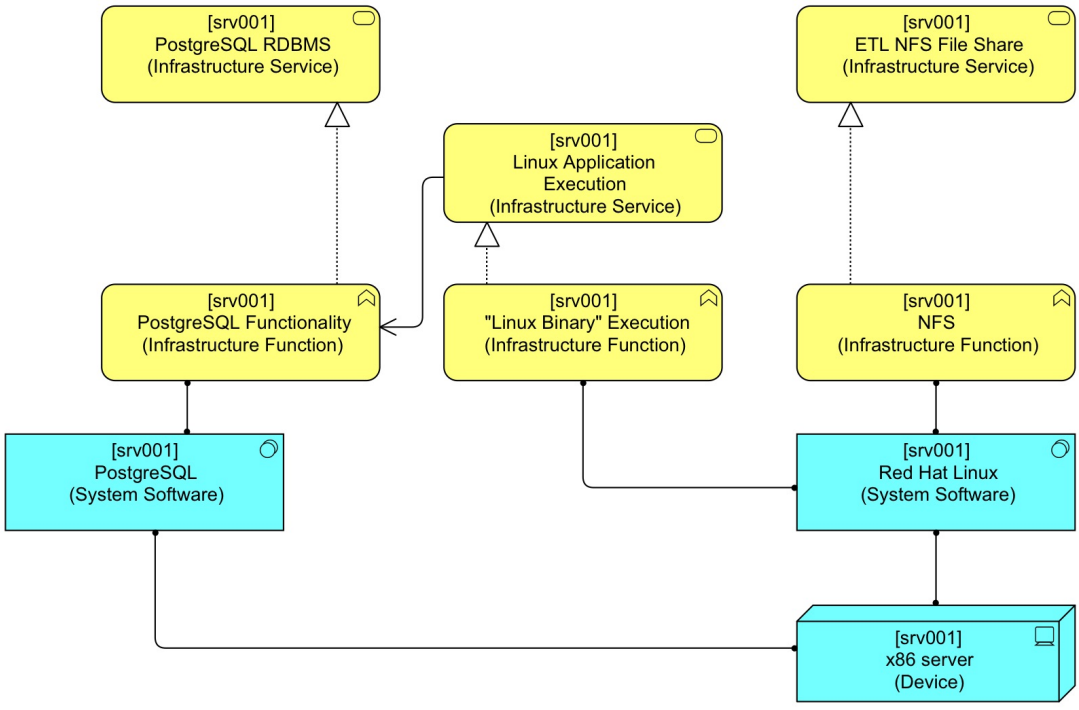
the Infrastructure Services into one element; this is a question discussed in the *Mastering ArchiMate* book under "Modeling Patterns".

One more comment, before we go on with the rest of the language. As you might have noticed, the initial example with infrastructure in View 6 had no Access relation between the Infrastructure Function and the "executable" Artifact. But the "write answer" example of View 7 does have that kind of relation (shown in red). Here an explicit Infrastructure Function for application execution was modeled and that function needs access to the "executable" artifact. So which one is correct? The answer is: whatever your brief is and what you want to show. The ArchiMate standard describes a language, but it is your choice what you want to say in that language and how you say it. You have considerable freedom and you will certainly develop your own style. You can certainly model incorrectly in the ArchiMate language, just as you can write false statements in any language. So there are wrong models. The syntax of the ArchiMate language does not by definition lead to correct statements, just as with other languages. In the *Mastering ArchiMate* book, I address choosing your style and patterns, the latter being like "common phrases in the language".

System Software and Devices

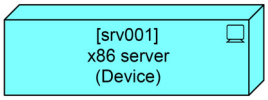
Our first use of infrastructure elements was pretty limited. We added the Infrastructure Function, the Infrastructure Service, the Node, the Infrastructure Interface, and the Artifact. The ArchiMate standard adds two useful active structure concepts to model the actual infrastructure layer of your architecture: Device and System Software. The best way to explain them is by giving an example of how they can be used.

In View 8 we see a model of a database server in our landscape. The name of the server is "srv001" and that could be the name it carries in a CMDB, for instance.

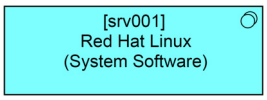


View 8: Device and System Software (without Infrastructure Interfaces)

The new element types are:



This is the Device, the actual hardware, the "iron" of our infrastructure. In this example, two elements of the type System Software have been installed on it (Assigned-To it). The little logo of the element is an image of a keyboard/screen.



This is the System Software element. It stands for software that we consider to be part of our infrastructure layer and not our application layer. Most common uses are operating systems or database systems. In our example both are available: the Red Hat Linux operating system and the PostgreSQL database system. Modeled too is that the PostgreSQL software uses the Red Hat software. (If we have our existing landscape modeled in such detail, we could by analysis find all PostgreSQL databases that run on Red Hat, handy if we are planning an update and there is something about PostgreSQL running on Red Hat that merits extra attention).

Both are a type of Node. A Node as in View 6 can be either System Software or Device, or a collection of both (see below). That means that both may also have an Infrastructure Interface, but these are not modeled here. Note that this kind of detail is not often required and even sometimes frowned upon. It is shown here only for educational purposes.

As the ArchiMate standard has two elements for software systems, Application Component and System Software, it is important to know when to use one and when to use the other. The *Mastering ArchiMate* book goes into more detail, but in short the following guidelines apply:

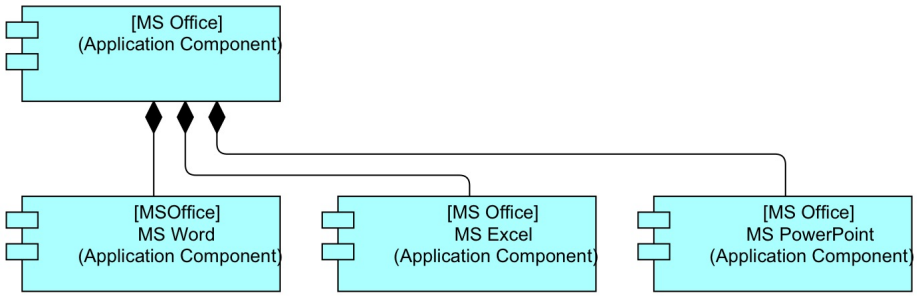
- If the software does not have specific business logic for your enterprise, it is System Software. Examples are database systems, operating systems, scheduling systems, enterprise service buses, Microsoft Excel, but also more business-specific platforms (as long as they need to be tailored for your specific business demands).
- If the software has specific business logic for your enterprise, it is an Application Component. Examples are specific flows configured in a scheduling system or a specific spreadsheet based on Microsoft Excel (Excel is the System Software, the specific spreadsheet is an Application Component), or the explicit tailoring/configuration of a business-specific platform.

System Software generally acts as *platform* for Application Components or for other System Software.

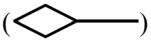
Composition and Aggregation

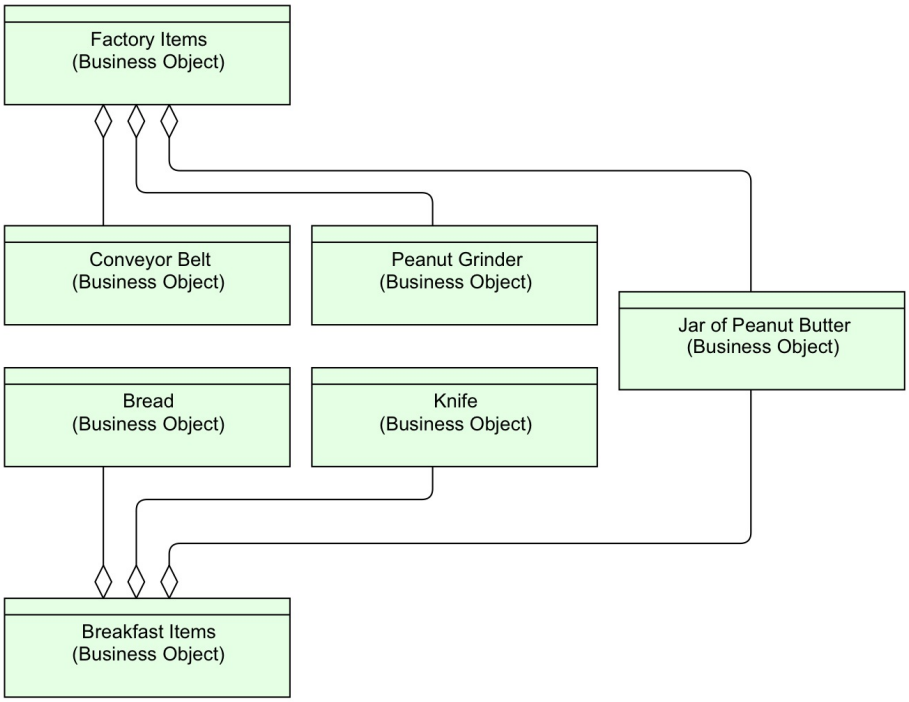
We already met the Composition relation () earlier. The composition represents a whole-part relation. It is best to look at the ArchiMate version of this common relation as the relation between a tree (a real one from your garden, not the computational concept) and its branches. A branch is branch of a single tree and cannot be a branch of multiple trees. If the tree is destroyed, all of its branches are destroyed as well.

Generally, you may always create a Composition relation between two elements of the same type (or "super-type", explained later). View 9 contains an example.



View 9: Composition Example

The Aggregation relation () is a variation on the theme. It looks like View 10.



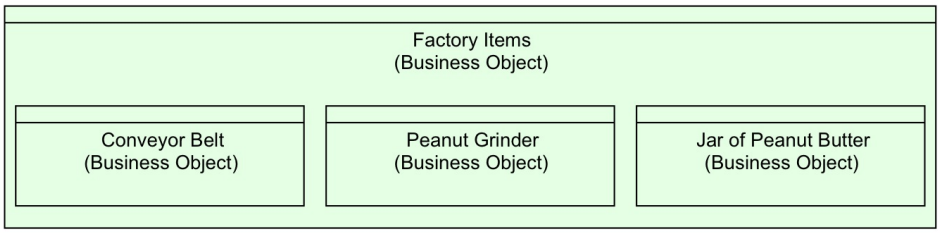
View 10: Aggregation Example

It is best to look at the Aggregation relation as a kind of grouping. (Note: in The Grouping Relation we will see an official "grouping" relation.) The "parent" (on the end with the diamond) represents a "collection" of children. But other than with Composition, the children may be a member of multiple Aggregations, as you can also see in View 10: the "Jar of Peanut Butter" is both part of the "Factory Items" of the peanut butter factory and the "Breakfast Items" of a consumer's home. It's like the number 4 being both part of the collection of squares of integers and the collection of even numbers.

Nesting

There are three relation types that may be drawn by nesting an element inside another element: Composition, Aggregation, and Assignment. Note: tooling often allows more than the language does, especially if you use tooling that is nothing more than a good model-drawing application such as Visio for Windows or

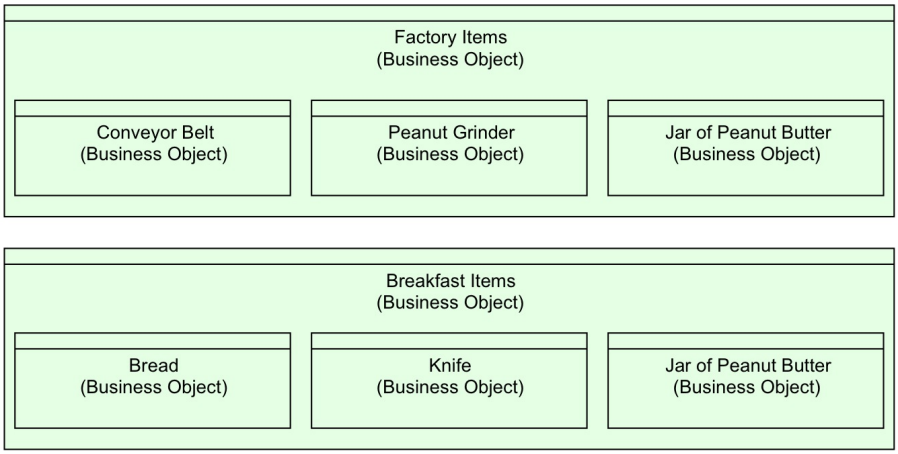
OmniGraffle for Mac OS. Anyway, let's take the "Factory Items" from View 10 as an example. Nested, it looks like View 11.



View 11: Nested Aggregation

This looks a lot cleaner and that is why many modelers like it. But we can already see a disadvantage: you can no longer see what the relation is between parts and whole: Composition? Aggregation?

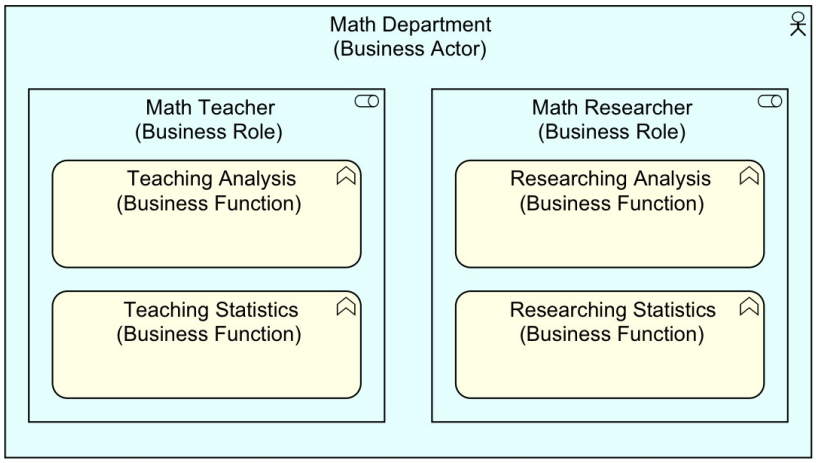
It gets worse, when you want to model both Aggregations from View 10 in nested form as in View 12:



View 12: Two Nested Aggregations with Shared Element

Not only are you unable to see the actual relations, it has now become necessary to include the "Jar of Peanut Butter" element twice. And though the name is the same, there is nothing that will tell you whether "under water" it is the same element. You can have two different elements with the same name in the ArchiMate language; after all, the label has no meaning inside the language (as the language is in fact a grammar) even if it has a meaning in the world of an architect. Besides, even if your modeling guidelines force different names for different elements, think of a very large view with the same element twice. Are you going to spot that the same element occurs twice? Probably not. So are you going to see the dependencies in full? Again: probably not.

View 13 contains an Assignment example of Nesting, two levels deep:

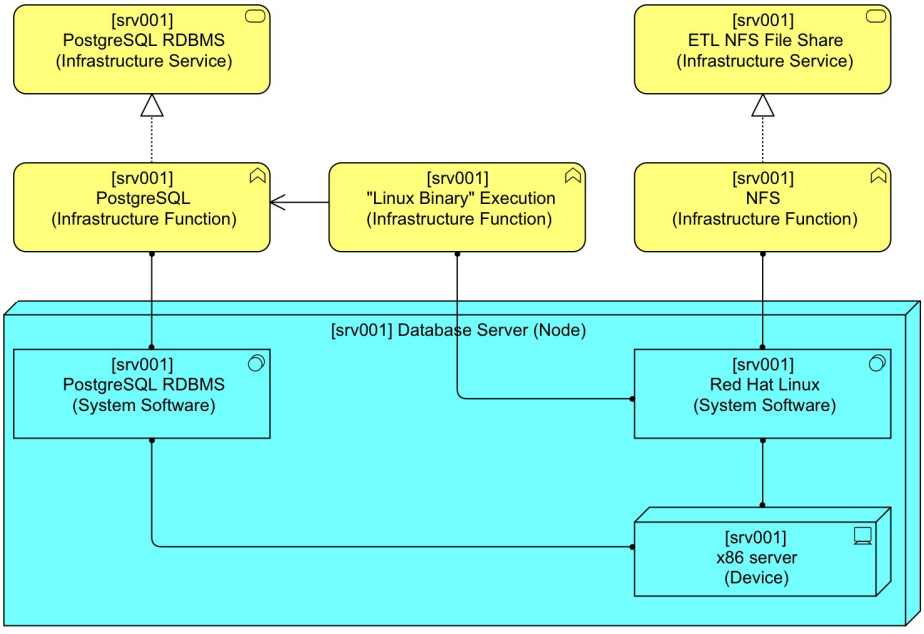


View 13: Assignment Nesting, Two Levels Deep

In summary, Nesting makes for nice views on your model, views that are easy on the eye. But don't think about them too lightly, because constructing your model this way comes with risks. And even bigger risks than you think, because some tools will let you create nestings without actually creating a relation between the nested elements (which is in conflict with the ArchiMate standard), or they may create a default relation which was not the relation you were thinking of.

Using a Node to Encapsulate Infrastructure

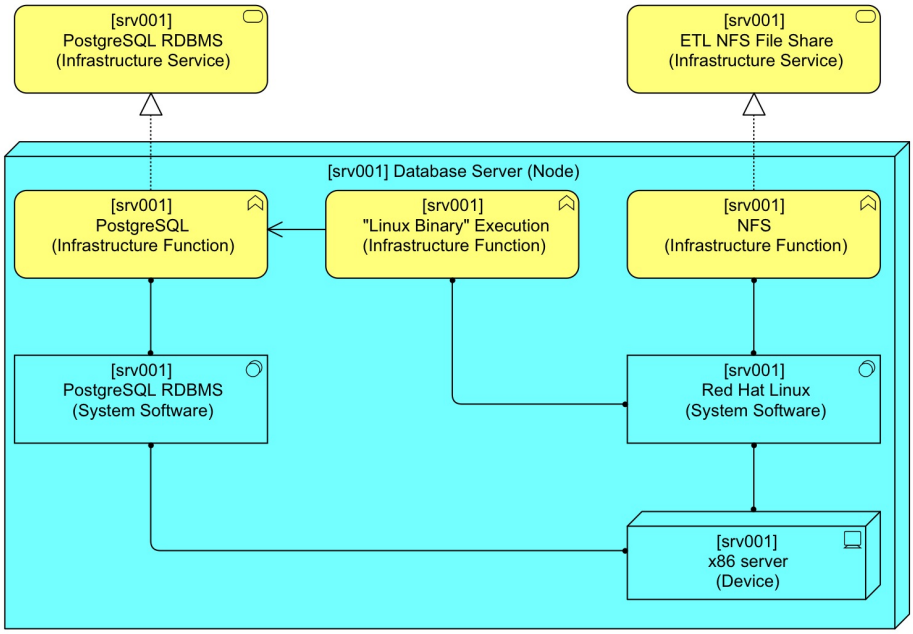
Now that we have seen Composition and Nesting, we can look at View 8 that introduced System Software and Device and show in View 14 how it might be modeled.



View 14: Device and System Software Nested in a Node

In this example, the System Software and Device elements are children (Composition) of an abstract Node element (the Composition relations are not explicitly shown here, but instead modeled as Nesting). This is kind of a nice grouping of an infrastructure element as the composition's children have no independent existence or use. We can also go further: in View 15, the Infrastructure Functions too have been Nested in the Node. This is possible, because we can also Assign the functions to the overall Node and we may Nest an Assignment relation.

In this example, the Node realizes two Infrastructure Services: the PostgreSQL software realizes an RDBMS service that is available to Application Functions, and the operating system also realizes a network file system shared directory. Basically, all structure below the Infrastructure Services is modeled inside the Node.



View 15: Device, System Software, and Infrastructure Function Nested in a Node

In the Style & Patterns chapter of the *Mastering ArchiMate* book we will see that it might be handy to restrict ourselves here, to make our models simpler and analysis of the model easier. You have the choice, of course, if you want to model these details (and you can go as far and deep as you like). It all depends on the use you want to make of your model. Here, it is only shown to explain what the element types are and how they relate to each other.

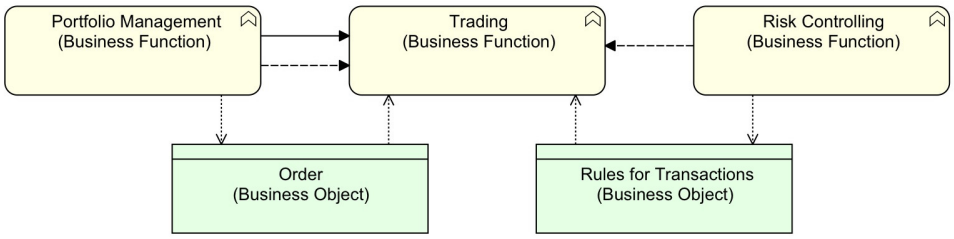
Events, Triggers, and Flows

So far, we have handled the "structure" of your architecture and all relations so far were so-called "structural relations". The ArchiMate standard does not fully describe the dynamics of an architecture, but it does have two "dynamic relations": Trigger and Flow, and one operator to combine these into a dynamic structure (see The Junction Relation).

View 16: Trigger and Flow

In View 16 we find Trigger (——————▶) and Flow (- - - - - - - -▶) relations in a business layer example. We see three Business Functions here from the asset management world: "Portfolio Management" is taking investment decisions, which result in orders for the "Trading" function. So, "Trading" starts trading when it receives an order from "Portfolio Management". Triggering means there is a causal relation between the two functions. The flow between "Portfolio Management" and "Trading" says information flows from one to the other. In this case it is the "Order".

But the Trading function also regularly receives a list of allowed counterparts, countries, and currencies from the "Risk Controlling" function. This information Flows from one to the other, but it does not Trigger a trade. A Flow relation between two Business Functions can also be modeled as a Business Object written (Accessed) by one function and read (Accessed) by another. If we add those, it looks like View 17.

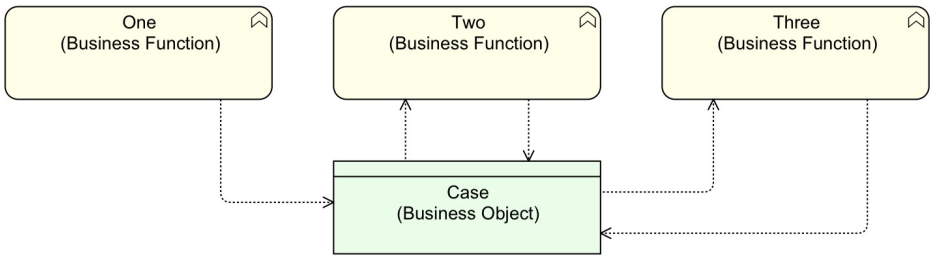


View 17: Trigger and Flow and Access to Elements

So, how useful is the Flow relation if you can also use the Business Objects and the Access relation? Well, it has a few advantages.

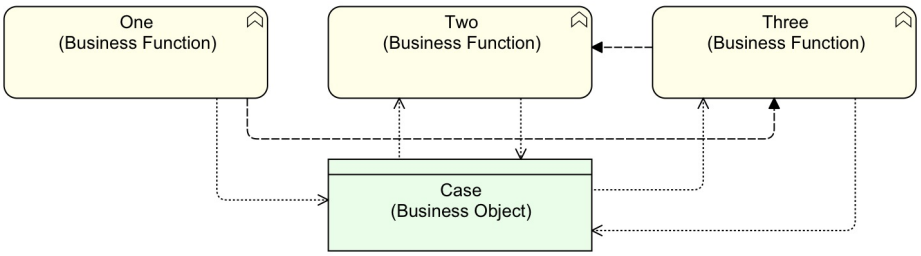
You can make a simpler view by leaving the Business Objects out and, for instance, labeling the Flow relations. The problem, though, is that such a label generally cannot be used to analyze what is on your landscape: watch out for relying too much on labels and properties of elements, they often live outside the "analyzable" structure of your model.

But, most importantly, your dependencies can become clearer with a Flow. Take, for instance, the example of a "Case" flowing though your business from Business Function to Business Function. Having read/write relations from these Business Functions to that single Business Object tells you nothing about how information flows. Take the example in View 18:



View 18: Business Functions sharing Access to a Business Object, without Flows

This does not tell you how the "Case" Flows through your organization. Look at View 19 and it becomes clear what the flow of information is.

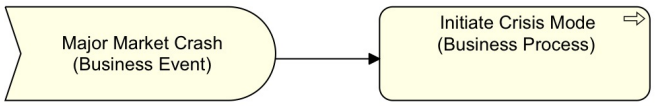


View 19: Business Functions sharing Access to a Business Object, with Flows

Without the Flow relations, would you have known? Could you have drawn the wrong conclusion? Certainly. Is that a bad thing? After all they all depend on that Business Object. Well, take this example: without the Flow relation, you might think that the roles behind function One, Two, and Three may have to agree concurrently on all the contents of the Case Business Object. But, in reality, you might only need to set up talks between One and Two on the one hand and Three and Two on the other, and, depending on the issues at hand, that might be simpler.

Trigger and Flow relations may generally be drawn between behavioral elements within a layer; e.g., from Application Function to Application Function or from Business Process to Business Process.

ArchiMate also has a special element that depicts a Business Event. A Business Event is "something that happens". Events can trigger a Business Process or a Business Function and they can be raised by a Business Process or Business Function, which is also depicted with a Trigger relation. Business Events are normally used for standalone "things that happen". An example can be seen in View 20.



View 20: Business Event Triggers Business Process

The (almost) Complete Picture

We have described 14 of the ArchiMate Standard's 31 element types, so in that sense we are only half way. But, with the elements and relations in this section, you can probably do 95% of an architect's work, if that work is focused on normal business-IT integration and modeling things like Project Architectures.

The title of this section is "Elements and Relations: 3x3x3". It has this title because ArchiMate:

- Divides Enterprise Architecture into a Business and Information layer, an Application and Data layer, and a Technical Infrastructure layer. These are the rows in ArchiMate's meta-model and it is fairly standard in the Enterprise Architecture world.
- Divides architecture in any layer "strictly" into Active Structure, Behavior, and Passive Structure. Put in a sentence: "Who/what does what to what?". The clear separation of actors and their behavior is not common in Enterprise Architecture and it is a main foundational aspect of ArchiMate.
- Has three kinds of relations: Structural Relations, Dynamic Relations, and "Other'" Relations (see Other Elements and Relations). The reason for this division becomes clear in Derived Relations.

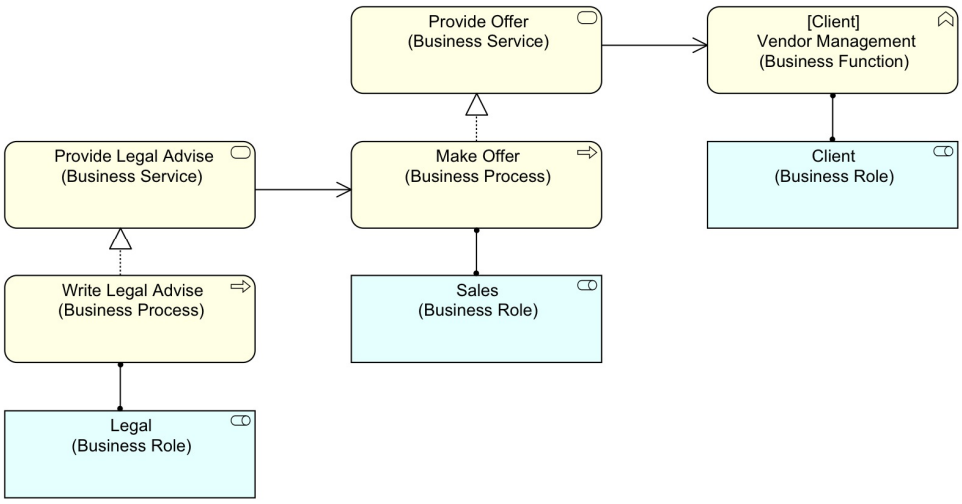
Other Elements and Relations

With the elements and relations of the previous section you can probably do most of an architect's work, if that work is focused on Current State and Project Architectures. Here is the rest.

Collaborations and Interactions

Suppose you have two Business Roles in your organization that need to work together to get some specific work done. For instance, the "sales" and the "legal" Business Roles need to work together to create the offer for the prospective client. Within ArchiMate there are generally two ways of modeling this.

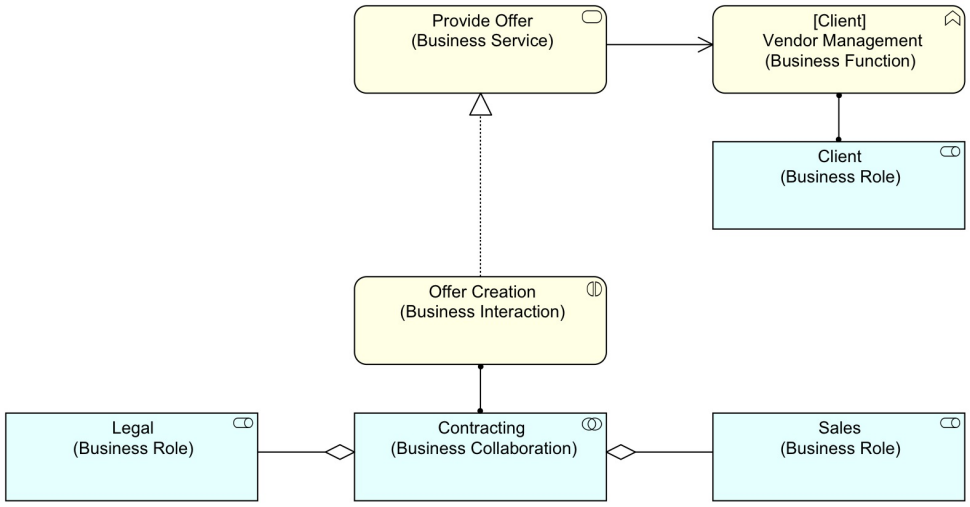
Using the elements and relations we already described, you can put one of them in the lead and let the second provide a service. In our example: "sales" produces the offer, but it uses the (internal) services realized by (the processes of) "legal" in its processes:



View 21: Collaboration: Sales Uses Legal on Order Creation

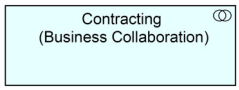
In this set-up, it is clear who is in charge. "Sales" decides to send the offer out, not legal. "Legal" must provide something (and, for instance, withholding approval means "sales" cannot proceed), but the service to the client is realized by the process of "Sales".

ArchiMate offers a second way to model such collaborations. You can create a Business Collaboration that is made up of the Business Roles that are part of that collaboration. Then, you can assign a Business Interaction to that collaboration, the interaction being the behavior of that collaboration. It looks like this:



View 22: Collaboration: Sales and Legal Collaborate on Order Creation

The whole "offer creation" "process" is now modeled as not one of the parties "owning" it, but both. There are a few new elements:



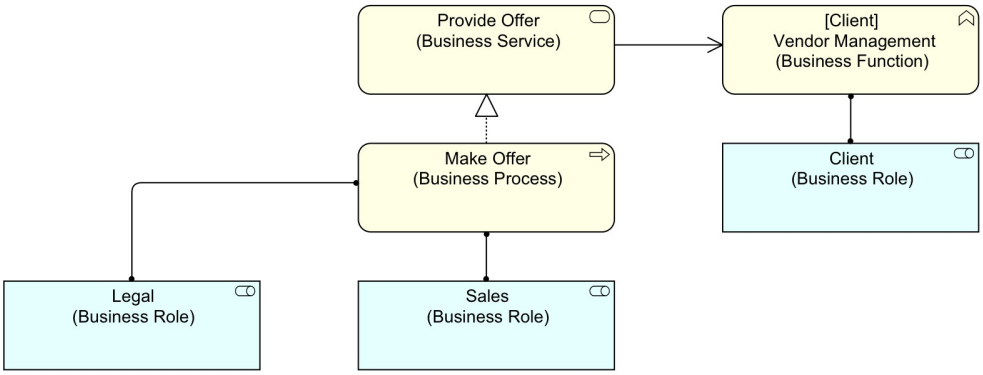
This is the Business Collaboration. It is a special kind of Business Role that Aggregates multiple other Business Roles (or Business Collaborations, of course, as these are also types of roles, see The Specialization Relation). It is the concept that defines multiple roles forming a (specific) single "collective" role to do something together.

This is the Business Interaction. It is the behavior of that Business Collaboration, the activities they do together. Note that, while the Business Collaboration Aggregates two or more Business Roles, the Business Interaction does *not* Aggregate Business Functions or Business Processes. What is not clear in ArchiMate is if we want to see a Business Interaction as functional or process-oriented. Given that it generally produces a result, it is probably best to see this as process-oriented.

One consequence of using a Collaboration is that it is not clear who is in charge. This might be more acceptable to the organization in terms of sensitivities, but sometimes it is just true: nobody is really in charge. In the *Mastering ArchiMate* book I present a "modeling pattern" of the business that uses Collaboration to show the loosely-coupled nature of some of the organization's "end-to-end" processes.

Both methods (using a service and interaction) are correct; it is a matter of style what you want to use and both approaches have some consequences. If you think about it, you could even model the sales-client interaction as a collaboration, after all the transaction requires decisions on both sides.

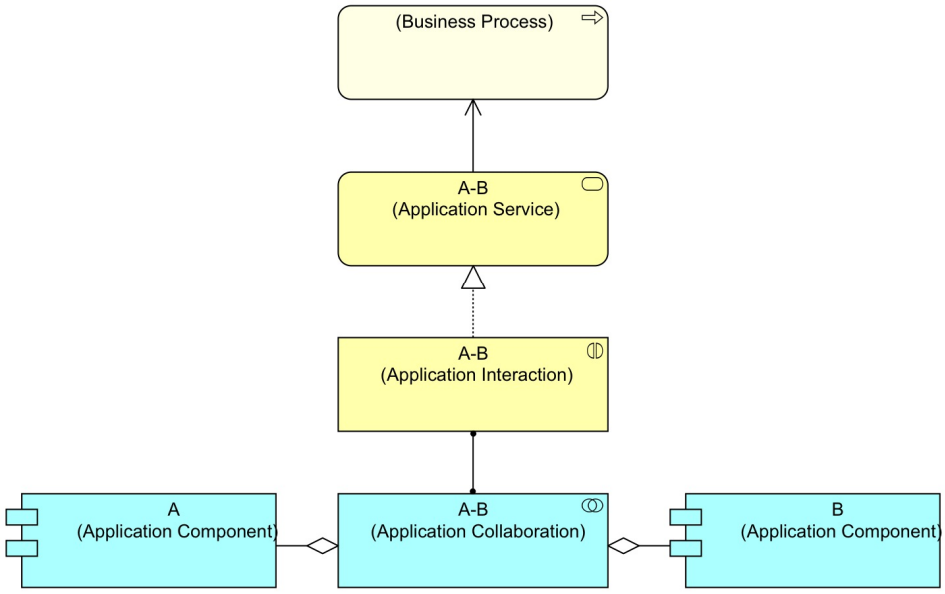
Actually, there is a quick and dirty way to do this too. ArchiMate says that a Business Process is assigned to a single role, but it does not enforce this in the formal definition. Nothing stops you from assigning multiple roles to a single process. So, we can model it quick-and-dirty as follows:



View 23: Collaboration: Quick and Dirty Method of Modeling

This way, we also show that the "Make Offer" process is assigned to two roles, in fact forming a *de facto* collaboration. This is, however, not proper ArchiMate. Besides, why is this a collaboration? Maybe either of the two performs it (which is what I would use this pattern for, if at all).

At the application layer, something likewise exists.



View 24: Application Collaboration

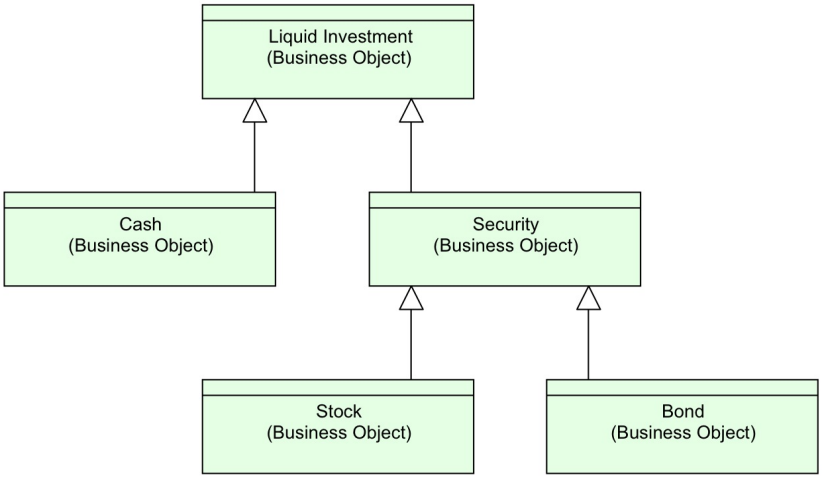
The Application Collaboration stands for the collaboration of two Application Components. The behavior of that collaboration is the Application Interaction and such a behavior may realize an Application Service. Here too, it is not clear who is in charge, but an architect may like it because it is "fairer" to the importance of both. Also, it gives the architect a clear central place to document how applications interact. In the Used-By model, the description gets divided over multiple elements This is a matter of style, and there is more about that in the *Mastering ArchiMate* book.

The Association Relation

The weakest relation, the "catch-all" relation of ArchiMate, is the Association relation which is depicted as a simple line ( ). It has a couple of formal roles in ArchiMate, but it also has the role of "relation of last resort". If you want to model the relation between two elements, if you know they are related somehow but you cannot model how, you can use Association. It is, therefore, often a sign of lack of knowledge or effort, so it is a matter of style if you want to use it. For now, know that it exists and below we will see some official uses.

The Specialization Relation

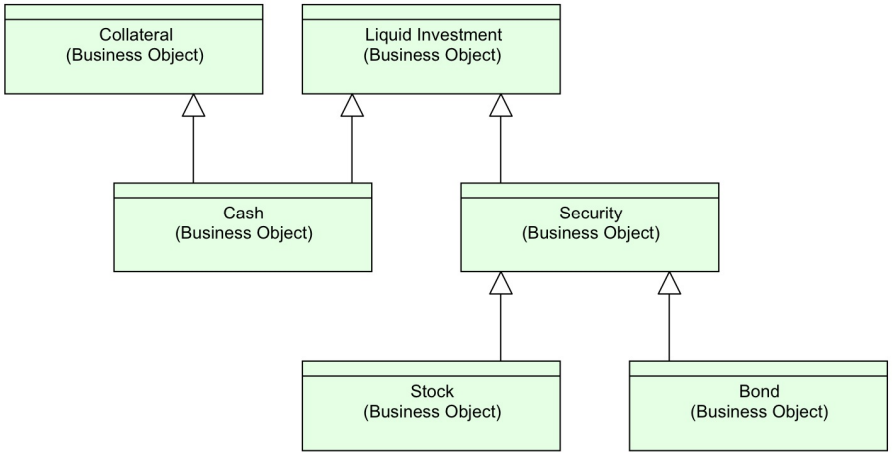
All relations we have seen so far are relations between actual elements in your landscape. The Specialization relation ( ) is different. From a software engineering perspective, all other ArchiMate relations are element-level relations, and this one is a class-level relation. The Specialization relation says that an element is a kind of another element. For instance, a "car insurance" and a "travel insurance" are both a kind of "insurance". Or a "stock" and a "bond" are both a kind of "liquid investment". It may look like this:



View 25: Specialization Example: Investment Types

Here, we say that "Cash" is a kind of "Liquid Investment" (a liquid investment is an investment that you can easily trade) and another kind is "Security" which again can be specialized into "Stock" (a deed to a partial ownership of a company) and "Bond" (a loan to an organization or country). If an element is a specialization of another element, whatever relations are true for the "parent" (the element at the arrowhead) must be true for the child. If a Business Process handles "Liquid Investment" (e.g., a reporting process), it must be able to handle both "Cash" and "Security". On the other hand, if a business process handles "Security", it must be able to handle both "Stock" and "Bond", but it does not need to be able to handle "Cash".

In object-oriented design, the specialization is sometimes called the "Is-A" relation. Its counterpart is the "Has-A" relation, which in ArchiMate is either Composition ( ) or Aggregation ( ). Generally, what you can do with an "Is-A" Sub-classing (which is what you do with the Specialization relation), can have complex consequences. Take, for instance, "Cash" from the current example. Suppose our company says we may only use cash as collateral for securities we lend and not (other) securities. We would have a new Business Object in our landscape called "Collateral", and "Cash" could be a kind of "Collateral". It looks like this:



View 26: Specialization Example: Multiple Inheritance

What we have here is called, in object-oriented terms, "multiple inheritance": Cash is both a kind of collateral and a kind of security. Most object-oriented languages do not support true multiple inheritance (C++ does, more modern Objective-C, Java, and C# do not) and for a reason: it tends to become messy because the "parents" may have conflicting rules of behavior for the "child" to inherit (a bit like parenting in real life – true, but maybe not the best paradigm for software engineering or business architecture modeling).

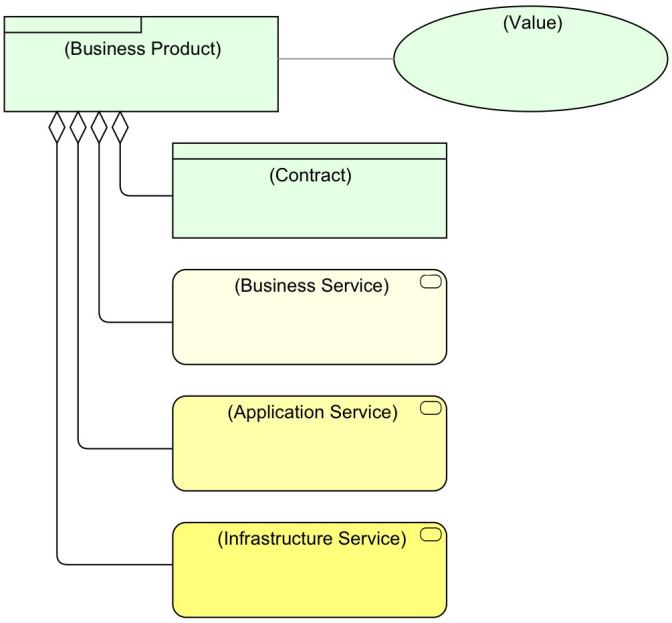
What is interesting to know is that Specialization also plays a role inside the ArchiMate meta-model. For instance, Device and System Software are both Specializations of Node, a Business Collaboration is a Specialization of Business Role, and an Application Collaboration is a Specialization of Application Component. What this means is that as you can have Compositions and Aggregations between elements of the same type, and, for instance, since Device is a subtype of Node, you can have a Device as a Composite part of a Node (as in Using a Node to Encapsulate Infrastructure).

There are also hidden specializations in ArchiMate. In the standard, Business Process, Business Function, and Business Interaction are all specializations of a common, invisible "business behavior" concept. As a result, all these "business behavior" elements may be Composites and Aggregates of each other.

In the *Mastering ArchiMate* book I show an example of Specialization use in a current state or change model, but largely I would advise you to be careful when using this relation type in models that are meant to describe a concrete (project (end) state or current state) reality. In other words, you can forget most of what has been written in this section and still properly create good ArchiMate models.

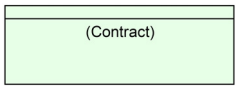
Products, Contracts, and Value

If you want to model the offerings of your organization to the outside world (anyone who uses what you produce, be it clients or regulators), a handy element is Product. A Product is a simple enough concept. It is an Aggregation of one or more Business and/or Application Services and (optionally) a Contract. It is Associated with a Value. It looks like this:

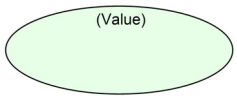


View 27: Product, Contract, Value, and a Product's Constituents

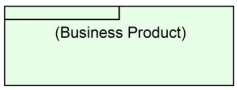
As you can see, the Product element type is a little strange: it is part of ArchiMate's Passive Structure (elements acted upon), but it also Aggregates Behavioral elements (the actions themselves). And it is a business-layer element, but it may also Aggregate an application-layer or infrastructure-layer element. If I was a purist, I would say this element type is a "kludge". Still, it can be useful. The new element types are:



This is the Contract. It represents the formal or informal agreement that covers the delivery of the service provided. This might be a specific Service Level Agreement or General Terms & Conditions.



This is the Value of the Product. A Value is a pretty abstract kind of element in ArchiMate. It has wide-ranging use. It can be the value to the consumer, but also to the producer. It could be described in monetary terms, but it may just as well be described in emotional terms (e.g., an insurance service may be associated with the value of "economic security" or "be able to sleep easy at night". "Be able to" since "sleep at night" is the actual Business Process). In the standard view of the ArchiMate model, it is generally Associated with the Product (and this is also the case in the underlying meta-model as shown in the standard), but the actual ArchiMate definition of Value Associates it with the service and mentions the Association with Product only as "indirect".

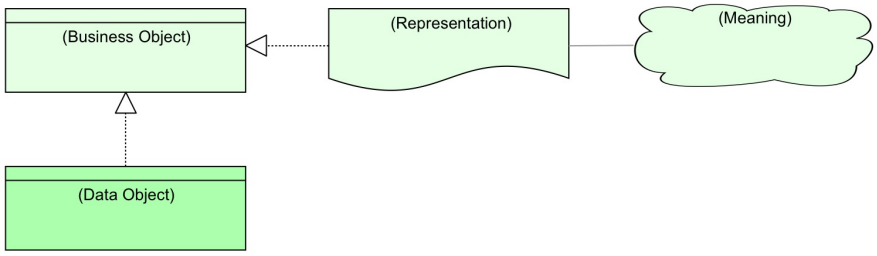


This is the Product. It is what you offer to the outside world. For a department of your organization, the Product may be something offered "internally". It can be handy to make the link between your Enterprise Architecture and how management looks at the organization.

Representations and Meanings

Earlier we encountered the Business Object, the work-horse of passive structure in your business layer architecture. And we saw how this business-level element could be Realized by an application layer Data Object. Take, for instance, the "Answer" Business Object that was Realized by the "Word Document" Data

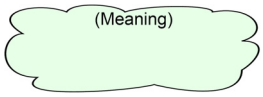
Object in View 7. ArchiMate also has another way to Realize a Business Object: it can be realized by a Representation. A good example would be a print of the answer letter you are sending to your customer. The relation between a Business Object and its direct surroundings looks like this:



View 28: Representation and Data Object Realizing a Business Object

The new element types are:

This is the Representation. While the Data Object is the element in our application layer that represents the Business Object, the Representation is another way of representing the Business Object and it stands for a representation that can be shared with others; e.g., a print, a PDF file you send as an attachment in a mail message, or a web-page.

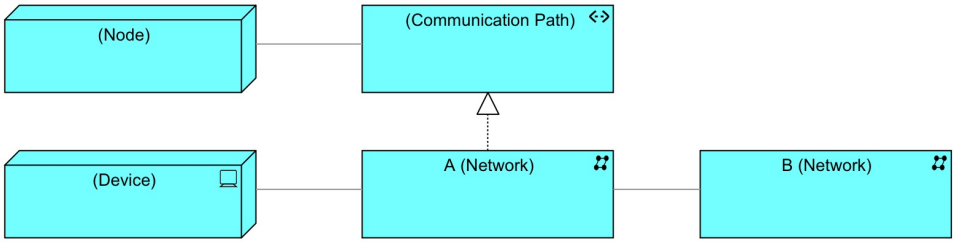


This is the Meaning of the Representation. ArchiMate says it is the "information-related counterpart of a Value" and it "represents the 'intention' of a Business Object or a Representation".

For corporate strategy-level Enterprise Architects, they can be useful (e.g., in the future state architecture), but in most practical circumstances (current state architecture, project architecture) they do not add a lot of value. Specifically, Meaning runs against Uncle Ludwig's idea about what a meaning is, but that is a philosophical issue, not an architectural one, so we will not discuss it here. Representation is sometimes a useful link to the physical world, but the description ArchiMate gives with links to "RTF" or "HTML" overlaps with the IT levels of your architecture.

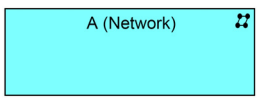
Network and Communication Path

ArchiMate offers two elements to model communication at the infrastructure level: Network and Communication Path. View 29 provides a simple overview of their use:

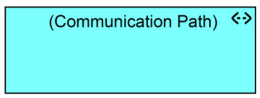


View 29: Network and Communication Path

The new element types are:



This is the Network. It stands for the physical communication medium, just as the Device stands for the physical computer hardware. A typical example could be "1Gbps Ethernet".



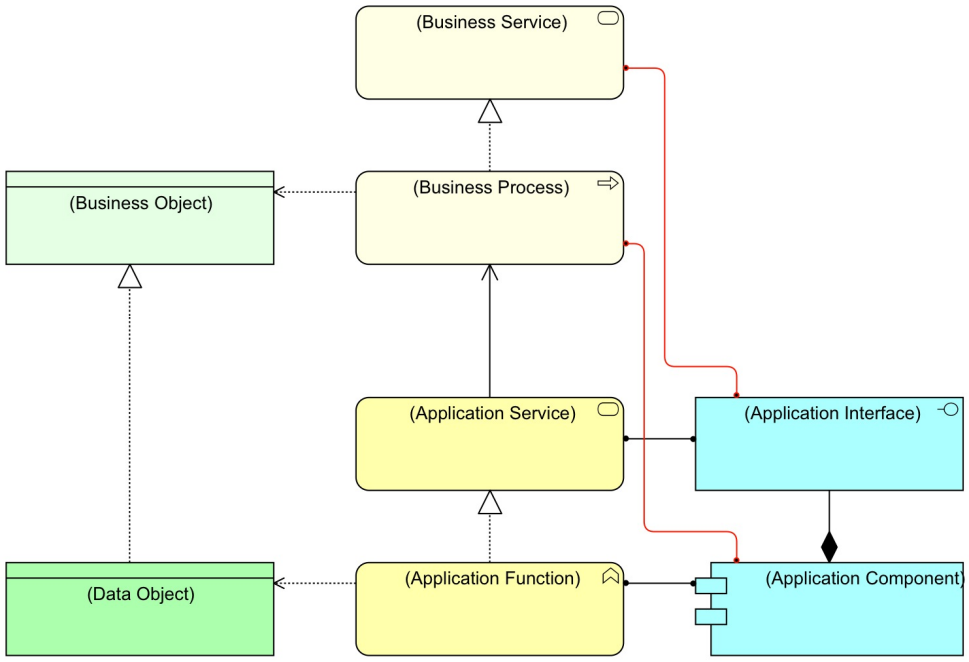
This is the Communication Path. What the Network is at the physical level is the Communication Path at the logical level. It is a way for nodes to exchange data. The example ArchiMate 2.0 gives is "message queuing".

A Network Realizes a Communication Path. The other relations are all Associations. To be honest, I have never seen these used in practice. Maybe I haven't seen enough practice or maybe they were thought of in a time that such aspects were still important in Enterprise Architecture work, whereas they are now more or less a given, like the air we breathe.

Network and Communication Path can both be drawn as "boxes" with the Association relation and also as connections, even though they technically are elements. Basically that would mean that you have to draw a Realization relation between two connections and my tool of choice cannot do that.

Automated Processes

So far, our landscape was based on a business process performed by people (actors fulfilling a role). But what if a process is automated? ArchiMate has the following solution: If a process is run by people, a Business Role is Assigned-To the Business Process and a Business Interface is Assigned-To a Business Service (as can be seen in View 3). The Business Role performs the Business Process. However, if a Business Process is performed by an application, we draw an Assigned-To between the Application Component and the Business Process. And consequently, we also draw an Assigned-To between the Application Interface and the Business Service. Here they are (in red):

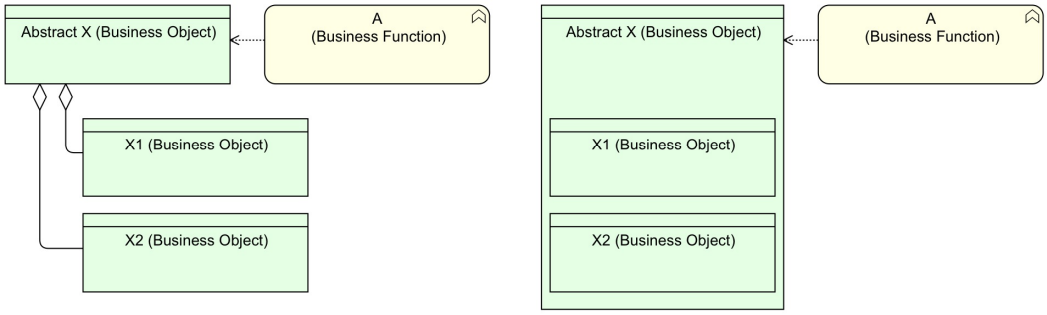


View 30: Automated Process

The Grouping Relation

In ArchiMate you can group elements visually in two ways: Nesting (see Nesting) and Grouping. Nesting is in fact a way to draw *three* types of relations: Composition, Aggregation, and Assignment. According to the standard, you may *only* use Nesting for one of these relations (tools often allow more nestings than

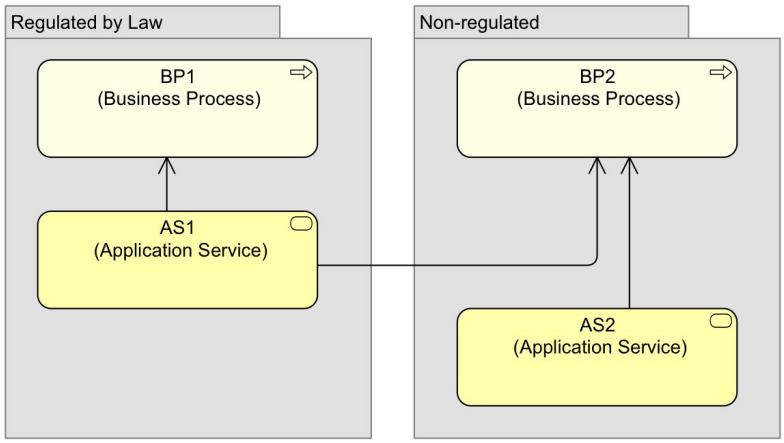
ArchiMate allows). But what if you want to visually group other elements? Suppose you want to group the different Business Objects accessed by a Business Function? You can, for instance, use Aggregation and create an abstract Business Object in your landscape as you can see in View 31.



View 31: Aggregation can be used as Grouping

On the left of the view you see the actual relations, and on the right the Aggregations are drawn using Nesting. It works if you can Aggregate and if you do not mind adding an abstract (non-existing) element in your model.

But what if you cannot aggregate? For this, ArchiMate has the Grouping relation. This is a relation, but it looks like an element with other elements nested, like the gray boxes with labels in View 32.

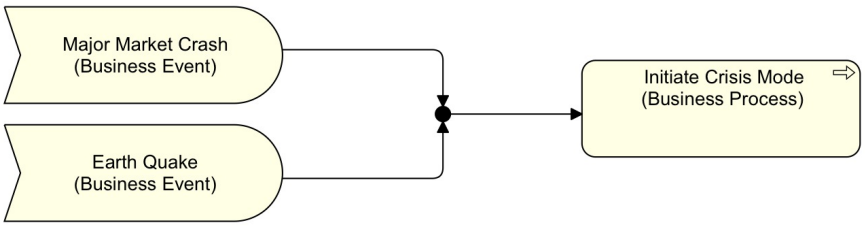


View 32: Grouping Relation

The Grouping relation is a catch-all kind of relation. It says more or less the same as if all elements in the box have something in common; in the example: legal status. So, in the left box, we find a Business Process "BP1" and an Application Service "AS1" that are regulated by law, and in the right box we find a Business Process "BP2" and an Application Service "AS2" that are non-regulated. Given the catch-all nature, you can think of grouping as a visual way to show Associations, but ArchiMate does not define it that way. Though Grouping is formally a relation in ArchiMate, it tends to be used in a purely graphical way to make certain views of your model clearer for the reader.

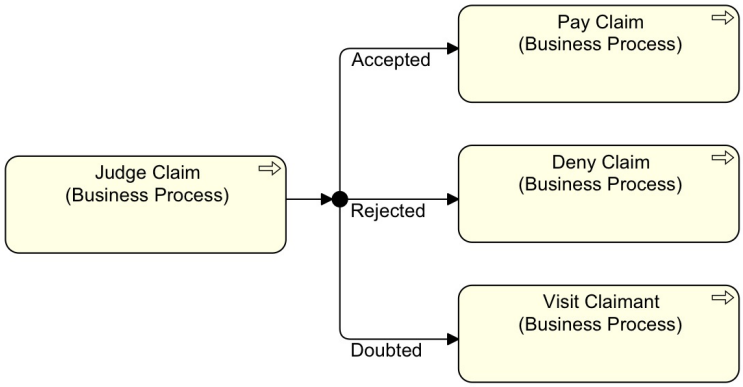
The Junction Relation

Finally, there is the Junction relation. This is very special because it is a *relation* that can relate other (dynamic: Flow and Trigger) relations to each other. Here is an example:



View 33: Junction Relation (join)

The big dot in the middle is the Junction. You can label the incoming or outgoing dynamic relations; e.g., in case of a choice:

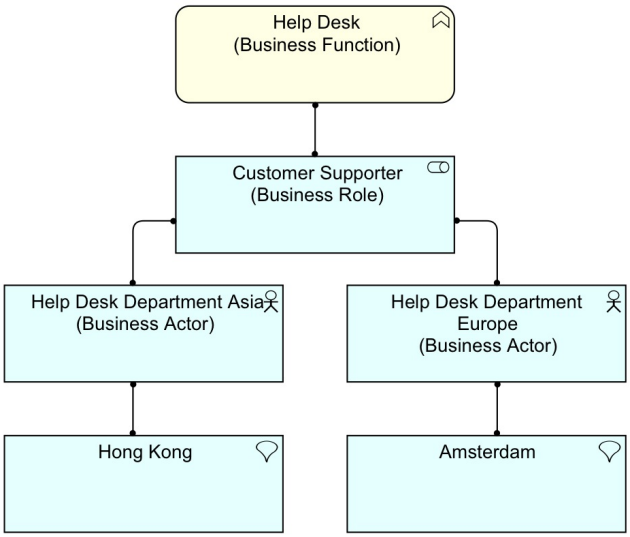


View 34: Junction Relation (split)

The Junction is very generic. The ArchiMate 2.0 standard suggests that it might be handy to extend it (ArchiMate is extensible) to And-Split, Or-Split, And-Join, and Or-Join. The tool I use comes by default with an And-Junction and an Or-Junction (not separated in splits and joins).

Location

The final ArchiMate element to explain is Location. This is new in ArchiMate 2.0 and an example is given in View 35.



View 35: Example use of Location Element

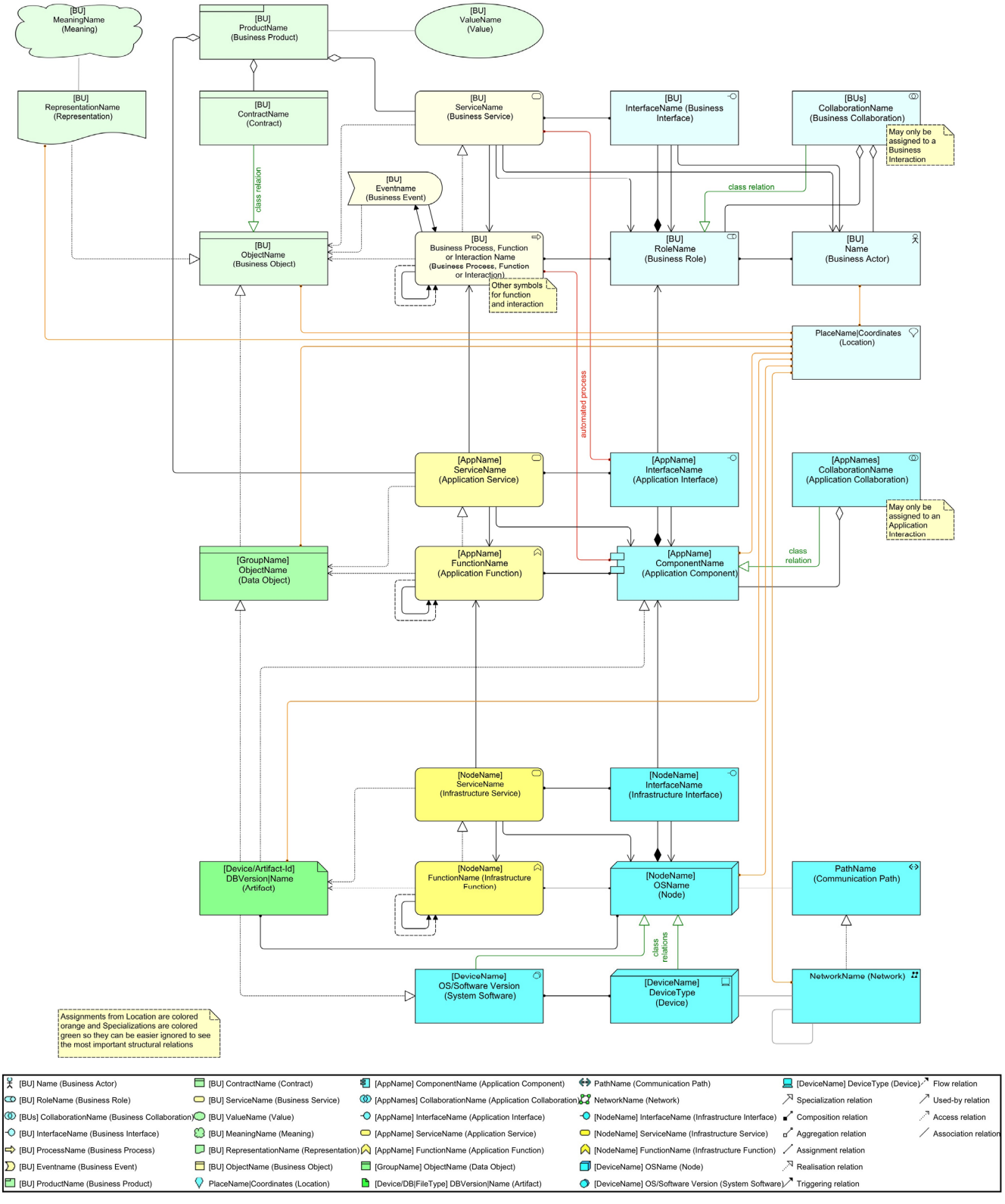
This is the Location element. This element stands for a geographical location where something resides. You can use the Assignment relation to link it to several other element types to model where they are located.

The Complete Picture

The ArchiMate language is reasonably rich. There are 31 element types and 10 relation types in Version 2.0, excluding the Motivation and Implementation extensions.

In View 36 you will find the overall picture of the ArchiMate element types and their relations. Note: the labels used for the elements in View 36 (and in this White Paper) are not required like that; they represent the standard labeling pattern I normally use and I explain in the *Mastering ArchiMate* book why. Labels are not regulated in the ArchiMate standard, but as with natural language, play an important role along with grammar in producing meaning.

This is often referred to as the ArchiMate Meta-Model (note: Business Process, Business Function, and Business Interaction take the same place and are represented by the Business Process icon in this view of the meta-model).



View 36: ArchiMate 2.0 Metamodel

Closing Remark on the Core ArchiMate Language Model

In my experience, ArchiMate is extremely usable. Creating Project Start Architectures based on detailed ArchiMate models has, in my experience, substantially reduced delays and unforeseen issues – having a detailed "Current-State" model has seriously improved our control over and our reporting on our landscape to regulators.

When we started to use ArchiMate, we initially had many doubts about how the ArchiMate designers had set up the language. But, we did follow the rule that we would strictly stick to the meta-model and not change or adapt it in any way (even though our tool supported that). We did this for two reasons:

- Any future tool update might break our models or require a lot of work.
- Understanding something comes only after a lot of experience. Though we did sometimes dislike what we saw, we decided to distrust our (beginner) skills more than the skills of the language designers. This was wise: we learned to appreciate most of the choices the designers made when our experience grew.

If you focus on finding faults (something that comes naturally for architects as they are always on the lookout for something that may break), you will. For instance, we have seen relations that look like elements (Grouping) and elements that may look like relations (Communication Path). But in my experience, that does not affect the usability much. ArchiMate is very, very usable.

Abstraction *versus* Precision

We architects love abstraction. Abstraction makes the nitty gritty and complex details disappear. Abstraction makes the unmanageable manageable. Our business colleagues also love it when we produce abstractions. It makes our end products easier to digest. In the ArchiMate language, you can do both, which means that – apart from the precision-like modeling that you do, for instance, when building "current-state" descriptive models – you can use the ArchiMate language in a more "loose" set-up; e.g., when designing a landscape top-down, even in a rough way with broad strokes.

For instance, when designing, if you have a Business Process that requires the services of an application, you generally start with defining that application service in terms of "what the business needs", and then you design how this service will be realized by one or more applications. The ArchiMate language has enough freedom to be used to make models with a lot of abstraction, and as such it also supports an approach to Enterprise Architecture design that sees services as abstractions defined top-down (from the needs of those that use the service). In fact, such thinking was an important aspect of the design process of the ArchiMate language in the first place. But the ArchiMate language also supports a more specific and precise use that is useful for creating precise "current-state" models. There is much more that can be said about this, but you can rest assured that the power of expression of the ArchiMate language is good for both, even if the concepts are explained here in the "precision approach". Besides, the ArchiMate language has another important way of simplification: the use of derived relations, which are described in the next section. Together, both ways ("more abstract use of the concepts" and "derived relations") create a powerful set that enables very high-level/abstract modeling in the ArchiMate language.

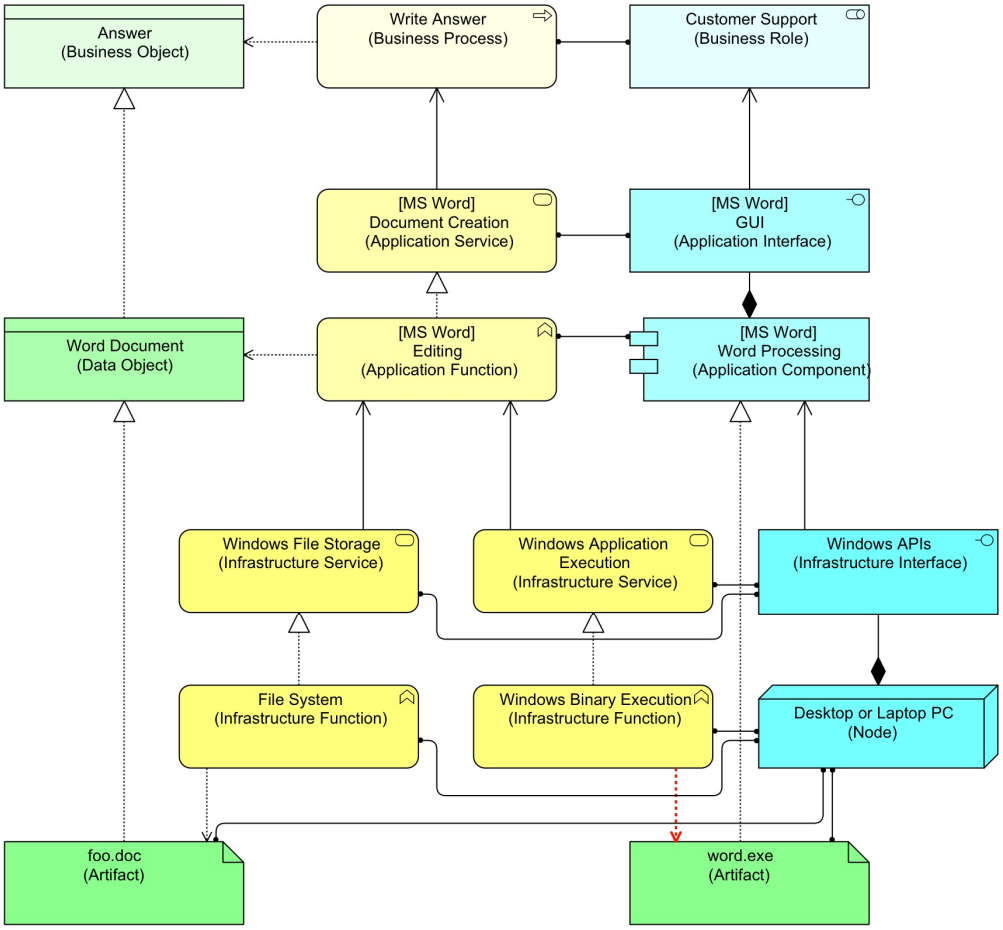
Derived Relations

Derived Structural Relations

So far we have included every concept, every element type, and all connecting relations. ArchiMate, however, offers a mechanism to create "shortcuts". The researchers, business users, and students that created ArchiMate even offered a mathematical proof that their definition of the shortcuts are "correct". Adding these shortcuts to the model was originally seen as extending "ArchiMate proper" to a sort of "ArchiMate+", but these shortcuts are so handy and useful that most practitioners (or teachers) do not really distinguish between the fundamental and shortcut relations anymore (which is a bad thing, as we will see). Add to that, tools generally do not support the actual difference and it is easy to understand that the shortcuts are probably one of the most (implicitly) used but least (explicitly) understood aspects of ArchiMate.

Officially, ArchiMate calls these shortcuts "derived structural relations" and I generally explain them by saying that the derived relations are a summary of a route that lies between two elements in a model. All tools I know allow you to model a summary without modeling the underlying "true" route of elements and relations, but in the end each summary relation more or less assumes that such a route with real elements and relations is there, even if you have not (or need not have) modeled them in your model.

The best way to illustrate them is by using an already familiar example, the earlier used "write answer" example. It is shown again here for easy reference (ignore the redness of one relation this time).



View 37: Write Answer Process, Supported by MS Word and a Standalone PC

Given this example, the questions we may want to ask are:

- What is the relation between the "Desktop or Laptop PC" Node and the "Write Answer" Business Process?
- What is the relation between the "word.exe" Artifact and the "Document Creation" Application Service?
- What is the relation between the "Desktop or Laptop PC" Node and the "Answer" Business Object?

For this, ArchiMate comes with the following procedure:

- Find a route from one element in the model to another, following the structural relations between them. There is a additional requirement: a route is only valid if all relations followed are followed in the same direction. All relations have a direction, except Association which is bidirectional (Assignment was also bidirectional in ArchiMate 1.0).
- Every structural relation has a strength. If you have a valid route, the derived relation between both ends of the route is the weakest relation found on the route, much like the weakest link in a chain. The strengths are (from weak to strong):
    - Association (bidirectional)

- Access (direction: always from behavioral element to passive structural element, independent from arrow which depicts read or write; that the direction may be opposite to the arrow is a pitfall when starting with ArchiMate)
- Used-By
- Realization
- Assignment (in each layer from actor to behavior and in the infrastructure layer from Node to Artifact, bidirectional in ArchiMate 1.0)
- Aggregation (direction: from parent to child)
- Composition (direction: from parent to child)

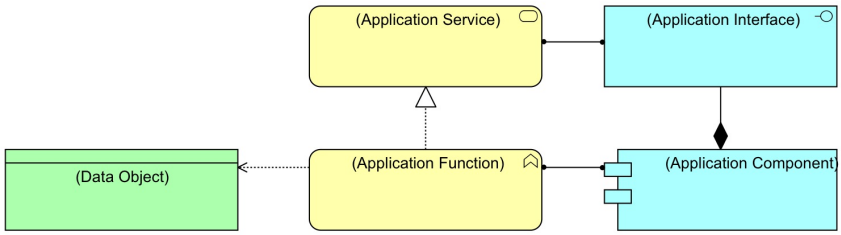
Using this procedure, we can answer the questions above:

- From the "Desktop or Laptop PC" Node, via Assignment (strength: 5) to "Windows Binary Execution" Infrastructure Function, via Realization (strength: 4) to "Windows Application Execution" Infrastructure Service then via Used-By (strength: 3) to the "Editing" Application Function then via Realization (strength: 4) to the "Document Creation" Application Service then via Used-By (strength: 3) to the "Write Answer" Business Process. The weakest relation encountered is Used-By, which is the relation between the PC and the Business Process: the PC is Used-By the Business Process, which makes good sense.
- The "word.exe" Artifact Realizes (4) the "Word Processing" Application Component, which is Assigned-To (5) the "Editing" Application Function which Realizes (4) the "Document Creation" Application Service. The weakest is Realization, so the "word.exe" Artifact Realizes the "Document Creation" Application Service. Which also makes sense.
- If we take the route from (a) we need just one extra step: from the "Write Answer" via Access (strength: 2) to the "Answer" Business Object. The weakest of that route is now Access, so the PC Accesses the "Answer" Business Object.

Sometimes, multiple shortcuts do exist. For instance, there is an alternative answer to the third question: From the PC Node via Assignment (strength: 5) to the "foo.doc" Artifact, then via Realization (strength: 4) to the "Word Document" Data Object and via another Realization to the "Answer" Business Object. The result is: the PC Realizes the "Answer" Business Object.

In other words: the PC Accesses the "Answer" Business Object and it also Realizes that same "Answer" Business Object. Here it is clear that we are looking at two aspects of this PC: it both executes the application and it stores the data. If the data were to reside on another server, we would not have both routes. So, the two routes are in fact a true aspect of the situation: the PC does both and so both relations dutifully appear. No problem.

So, all is well? It gets a little more complicated. Have a look at the Basic Application pattern again in View 38.

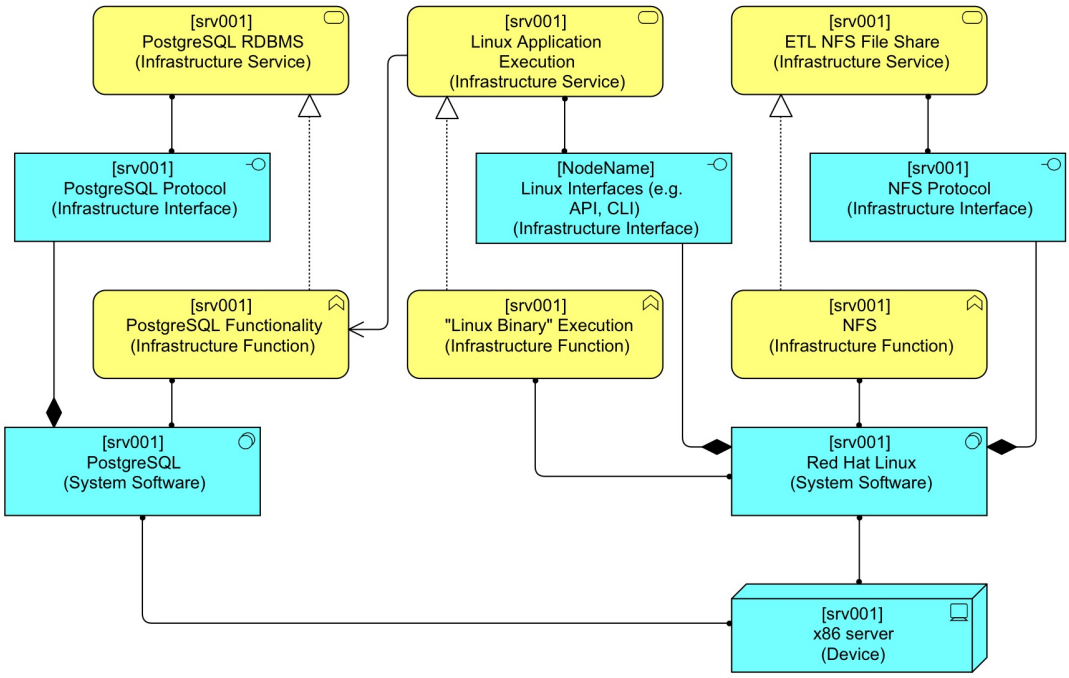


View 38: Basic Application Pattern

If you go from Application Component to Application Service, you can follow two routes. The first route (via the Application Interface) leads to Assigned-To as the (weakest) derived "summary" relation. But the one via the Application Function leads to Realization as the (weakest) derived "summary" relation. ArchiMate does not tell you which one to take if two possible routes exist. That is OK, as we saw in the previous example, when two routes stand for two different aspects. But in this case, either derived relation says something about how you view the concept of Application Component. If you choose Assigned-To, you keep to the strict separation of active component and its behavior. If you choose Realization, you look at the Application Component from a more behavioral point of view.

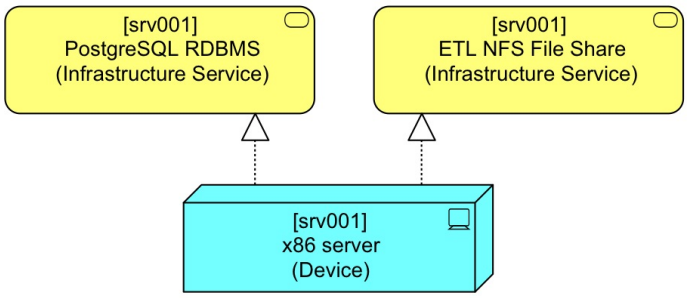
What all of this so far implies is that derived relations are useful to make summary views or abstract representations of a complex landscape, but there are risks involved in using them without (or mixed with) the underlying reality. The risks we have seen so far are still benign. But there are some pitfalls that are outside the scope of this White Paper.

Returning to View 8 in System Software and Devices, if we add a couple of Infrastructure Interfaces to that view we get:



View 39: System Software and Device with Interfaces

You might understand why I did not show this in View 8: very complicated for something as simple as a server that provides a file share and a database. But using derived relations, you can conclude that the following is a correct representation of the same:

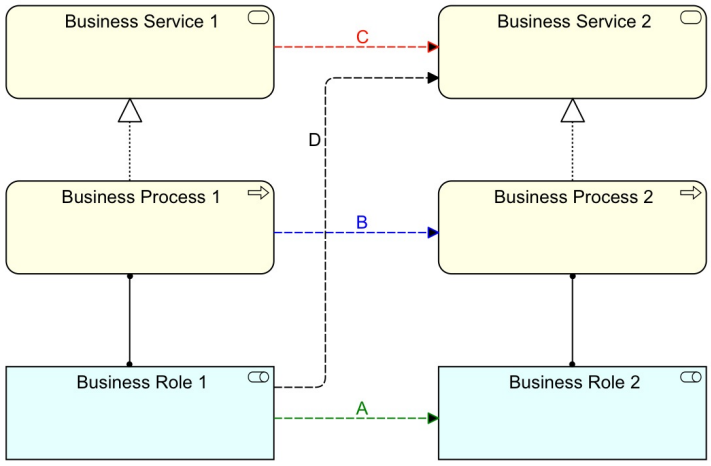


View 40: Simplified Database and File Server

So, if you want to keep it simple: you can just model these. Keeping it simple and using abstractions make your models and views easier for the reader, but there are some caveats. Much more about this topic can be found in the book.

Derived Dynamic Relations

Quite a bit simpler, but there is also the possibility to create derived relations from *dynamic* relations. Take a look at the following example:



View 41: Derived Dynamic Relations

Suppose we start with the Flow relation B, between "Business Process 1" and "Business Process 2". ArchiMate says:

- You may move a begin or end point of a dynamic relation between behavioral elements to an active structural element that is Assigned-To the current element.
- You may move a begin or end point of a dynamic relation between behavioral elements to a service that is Realized by the current element.

Moving both begin and end points simultaneously, we can turn Flow B into Flow A, and turn Flow B into Flow C. Note that we cannot turn Flow C into Flow B or A or turn Flow A into Flow B or C.

You can, of course, move only one of the begin or end points; e.g., from Flow relation B, you can derive a Flow relation D between “Business Role 1” and “Business Service 2”.

The same is true for the Trigger relation and the same derivations can be made on the application layer and on the infrastructure layer.

About the Author

Gerben Wierda (1961) is the Lead Architect of APG Asset Management, the Asset Management subsidiary of APG, one of the largest fiduciary managers in the world with over 300 billion Euro pension fund assets under management (Fall 2012). He has overseen the construction of one of the largest single ArchiMate set-ups in the world to date (approximately 75,000 elements and relations in two coordinated models, maintained by a single FTE).

His experience has led to his book *Mastering ArchiMate*,[2] of which this White Paper is a (slightly amended) excerpt.

He holds an MSc in Physics from the University of Groningen and an MBA of RSM Erasmus University of Rotterdam.

About The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through IT standards. With more than 400 member organizations, The Open Group has a diverse membership that spans all sectors of the IT community – customers, systems and solutions suppliers, tool vendors, integrators, and consultants, as well as academics and researchers – to:

- Capture, understand, and address current and emerging requirements, and establish policies and share best practices
- Facilitate interoperability, develop consensus, and evolve and integrate specifications and open source technologies
- Offer a comprehensive set of services to enhance the operational efficiency of consortia
- Operate the industry's premier certification service

Further information on The Open Group is available at www.opengroup.org.

[2] The full *Mastering ArchiMate* book is available as a downloadable PDF (ISBN 978-90-819840-0-3) and in print (ISBN 978-90-819840-1-0). For more information, please go to the website http://masteringarchmate.com, specifically, the article at http://bit.ly/UYxWNm.