Discussing ArchiMate

35. Proposed Improvements for ArchiMate 3.1

I might rightly be called a fanboy of ArchiMate, but that doesn't mean I am without criticism. I had a whole lot of criticism on the ArchiMate 2.1 standard. Much of that has been fixed in ArchiMate 3 (though not always to my exact liking), but in general the ArchiMate 3 standard is much less sloppy and much better put together than any previous version*. It did mean that it was a lot of work coming up with Edition III, as working around limitations in 2.1 and not having improvements of version 3 turned out to have been influential in the creation of the patterns in Edition II. This has repeated itself with the 'minor' update from version 3.0.1 to version 3.1 and now 3.2.

35.1 Fragmentation of element types

The standard says:

> *The most important design restriction on the language is that it has been explicitly designed to be as small as possible*

And while the standard has remained economical with relations (only the influence relation has been added since the original report published in 2007, and relations are often co-opted for distinct meanings), the same cannot be said about element types. There has been a rather unchecked growth, a doubling in fact, from the original ArchiMate to version 3.2 today. Some of this is unavoidable as new aspects and domains have been added, but not all. The new aspects and domains have all brought their own separate world to ArchiMate, and that is *by definition* questionable. After all, all the elements that have been introduced are part of *the* reality of an organization. Hence, in principle they should in one way or another have to be part of what I have called in Section 6 "An ArchiMate Map" on page 17 'the enterprise itself'. Good examples of such overlaps are:

- The Assessment element from the Motivation aspect represents something that has a real counterpart in the organization, e.g. a report or even a mail message. As such, it is a Business Object that is created (Access) by some sort of business behavior. The same can be said of other elements from Motivation, such as Goal, Requirement/Constraint, Principle, Stakeholder and so forth. They don't exist in a universe outside the organization, they are part and parcel of it;
- The Work Package from the Implementation and Migration layer is almost indistinguishable from a Business Process. But it cannot use applications, nor can it Access data or material;
- Any Business Role is by definition also a Stakeholder. But we cannot model for instance one Business Role Influencing another.

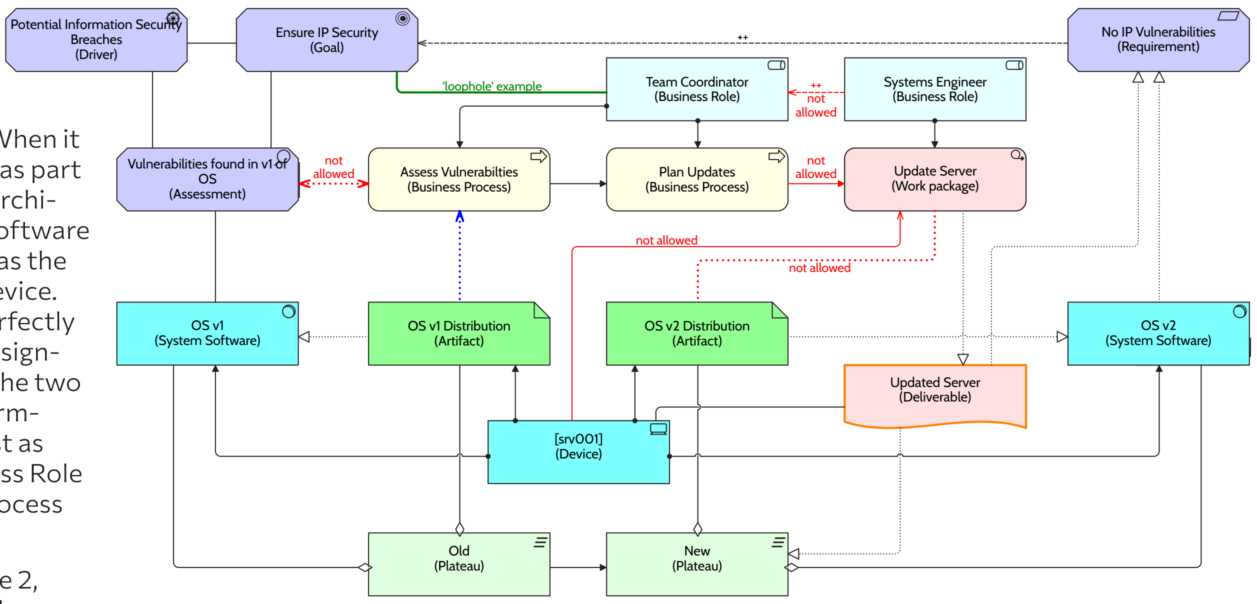
View 356 on page 204 shows a few relations in red that make a lot of sense, but that are not allowed. The blue one is allowed since ArchiMate 3. The situation is actually worse than it looks. The Motivation aspect uses a lot of Associations as formal direct relations. The fact that Association is allowed between all elements hides thus some of the fragmentation because there is a loophole, i.e. I can Associate not just a Stakeholder with a Goal, I can also Associate a Business Role with it, but not because the metamodel is particularly good, but because there is this general rule. This is shown in green.

35.2 Remove historical baggage

Related to this is the growing amount of historical baggage that ArchiMate carries. The designers are apparently very careful to strive for maximum backward compatibility, but the result is a situation that I would label as 'technical debt'.

* With the exception of the somewhat slipshod version 3.0 which contained the better (though not yet fully complete) foundation, but also many contradictions, omissions and glaring errors, such that it was technically unusable as it forbade even many foundational relations.

A good example is the Assignment from Device to System Software. When it was introduced as part of the original ArchiMate, System Software was positioned as the *behavior* of a Device. That made it perfectly logical to use Assignment between the two to model 'performing behavior' just as between Business Role and Business Process for example.



Potential Information Security Breaches (Driver)
Ensure IP Security (Goal)
No IP Vulnerabilities (Requirement)
++
'loophole' example
Team Coordinator (Business Role)
Systems Engineer (Business Role)
++
not allowed
Vulnerabilities found in v1 of OS (Assessment)
not allowed
Assess Vulnerabilities (Business Process)
Plan Updates (Business Process)
not allowed
Update Server (Work package)
not allowed
not allowed
OS v1 (System Software)
OS v1 Distribution (Artifact)
OS v2 Distribution (Artifact)
OS v2 (System Software)
Updated Server (Deliverable)
[srv001] (Device)
Old (Plateau)
New (Plateau)

View 356. *Various examples of real relations we cannot do in ArchiMate*

But in ArchiMate 2, internal Technology behavior (e.g. Technology Function) was introduced to model the infrastructure behavior of infrastructure active elements such as a Device (or any meta-model-Specialization of Node). The old Assignment was kept and (in part implicitly) redefined as 'deployment'. But not completely. For instance, if I add a special GPU card to a PC, isn't that deployment (from Device to Device) too? I can deploy System Software on System Software. Why can't I deploy an Application Component on System Software? Note: if we change something about Realization and/or derivation, this could all be removed and become derivable (see below).

And while I'm at it: a simple improvement would be to rename System Software to *Software Platform*.

Location was added in ArchiMate 2 as a sort of specialized grouping without the generic Grouping we have now. Then in ArchiMate 3 we got Facility and Grouping as real elements. There is serious overlap between these three. Would we have added Location to ArchiMate 3 if it hadn't been already in ArchiMate 2 (as a 'fix' for not having Grouping)?

Representation has been in ArchiMate from the start, mainly as a means to model printed information (the PDF example has always overlapped with Artifact). Given that we now have Material as well as Artifact, Representation can go.

Path and Communication/Distribution Network may be repositioned in a cleaner way as well. Why aren't they (abstract) Nodes which offer Technology Services through Technology Interfaces? Why do they 'serve' via Associations? Do we really need them? We have Flow!

35.3 Issues with abstractions

There is in my view a many-pronged confusion in ArchiMate over abstraction.

Section 7.17 "Abstractions" on page 31 already gave us an overview of the many ways we can model abstractions in ArchiMate. The standard adds to this in Section 3.6, when it (amongst other things) also tells us that we may use behavioral elements as abstractions as they are 'implementation-independent' (which then frankly is ignored in the rest of the standard). Then we have the complicated role of Specialization as described in Section 8.4 "Specialization in the Meta-model" on page 34 and Section 8.5 "The Specialization Relation in an Actual Model" on page 34.

Ignoring the — from the perspective of the rest of the standard — confusing suggestion that the behavior of an active element is an abstraction of the 'implementation' (instead of simply the behavior performed by the 'acting element'), and ignoring the issues surrounding Specialization, we are left with the following abstractions in ArchiMate:

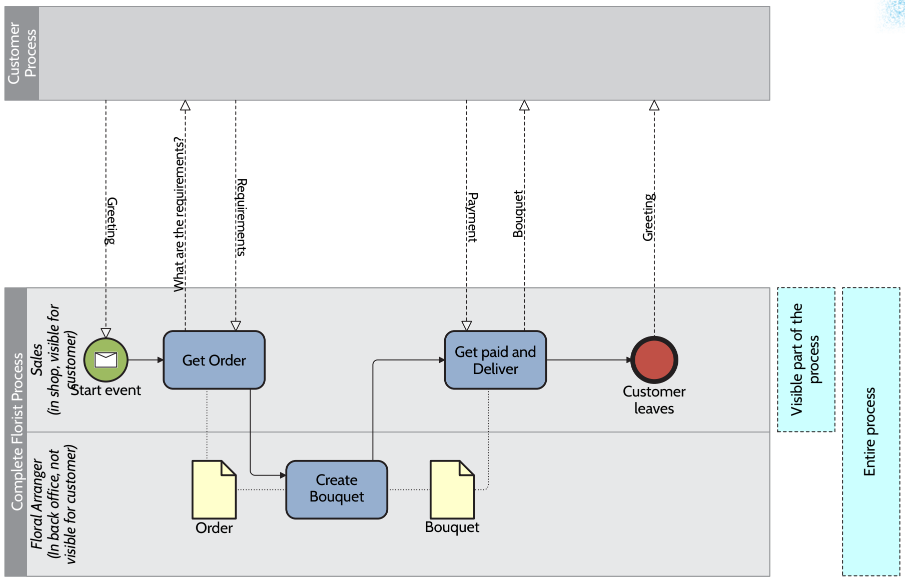
- *Identity*[*]. When one element is an abstraction of another, the two elements are both a representation of *the same thing* at a different 'level'. We see this in ArchiMate as the TOGAF logical/physical Realizations within the application and technology layer. We also see this in the Realizations from the passive elements (the Business Object or the Application Component or the System Software represents *the same thing* as the Artifact, just in a different layer or aspect). We also see this in the Realizations of core interfaces and behavioral elements to other layers, and when a Deliverable Realizes a core element;
- *Creation*. When one element is an abstraction of another it means that one element creates another element that does *not represent the same* thing. We see this when internal behavior (function, process) Realizes external behavior (service — though I disagree too service is an independent abstraction from internal behaviour, see Section 35.4 "Service as Composite Part of Function/

* When multiple elements Realize a single other element, the word 'identity' requires an explanation. All the identities that Realize the target are together the one identity that is Realized. We will not take those subtleties into consideration in the story as they only complicate but not really change this analysis in a fundamental sense.

Process")[*], when a core element Realizes a Requirement, or when a Work Package Realizes a Deliverable or a core element;

- *Collection*. The Realization of Strategy elements is harder to pin down. Close to Identity: it can be experienced as collections of a sort of the elements that Realize them. Note: currently, Aggregation is used to create a network or path from underlying infrastructure. It is quite enticing to see Path and Network as abstractions of that underlying infrastructure.

It is confusing that Realization is used for non-Identity abstractions. If something passive is created, the Access relation is more appropriate. If an active element creates behavior, we have Assignment. If we are collecting/grouping something, Aggregation is more appropriate.



View 358. *Selling Flowers. Exposed behavior is part of the total process.*

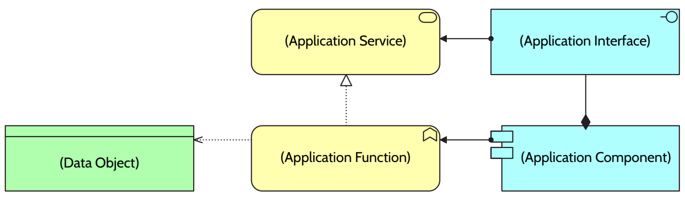
35.4 Service as Composite Part of Function/Process

There are two views of what a service is.

One is that the service is *visible* behavior that is *strongly coupled* to the *invisible* behavior. E.g. if you describe a selling process that creates a service which is used by a buyer, all the behavior that is visible to the buyer is an inseparable part of the process (and its description). It is not only part of the service, it is *at the same time* part of the process and the service is seen as the (visible) Composite part of the process. This, we could call the 'constructionist view' of a service.

The other is that a service provides some value to the environment and that it is an *abstraction* from the nitty-gritty details of the *internal* behavior (e.g. a function). In other words, some people want to see a service as an abstraction that is *weakly coupled* to the (independent) internal behavior. For them, the service is disjunct from and is *independent* behavior, for instance because the actual service delivery requires the behavior of the service consumer to succeed[†]. This we could call the 'abstractionist' view of a service.

In the ArchiMate metamodel, the view is mixed: abstractionist on the side of the service and constructionist on the side of the interface, as can also be seen in the basic pattern of View 357.



View 357. *The Basic Application Pattern*

The constructionist view fits reality better in my opinion. Suppose, for instance, you have a Business Process that Realizes a Service — say a Sell a Luggage Insurance Business Process that Realizes a Luggage Selling Business Service which Serves a Luggage Insurance Customer — the behavior that the Luggage Insurance Customer actually uses must be part of the Sell a Luggage Insurance Business Process. It cannot be otherwise. *There exists no behavior* that Serves the customer that is not performed by the seller as *part* of a Business Process. Suppose at some point in the Sell a Luggage Insurance Business Process the contract is presented to the customer. This is definitely part of what the customer experiences, it is 'exposed' behavior, but it is definitely *also* part of the 'internal' behavior, because it is part of the process. Or have a look at View 358, which shows in BPMN how the visible/exposed behavior cannot be anything *but* part of the service provider's process.

In other words: the insurance seller's 'offer the contract for signing' activity is both part of its process but *also* a part that is externally visible/usable, that is exposed. It is the *same* activity, *not* something independent. Looking at it as something independent introduces an abstraction that makes life more difficult, not easier. After all, we must remember AI's old adagium: the best model of the world is the world itself.

Important to note for this is that I am not talking about modeling the nitty gritty details, even if my argument to make the split different stems from the analysis that the 'external' (detailed) behavior must be part of the 'internal' (detailed) behavior. Both the internal behavior (e.g. process) and the external behavior (service) may be abstract in your model. After all, the fact that, e.g., Business Process and Business Function are hidden from the outside world does not mean they must be detailed. We think in 'hiding the details' but in Enterprise Architecture models you will not want to see some-

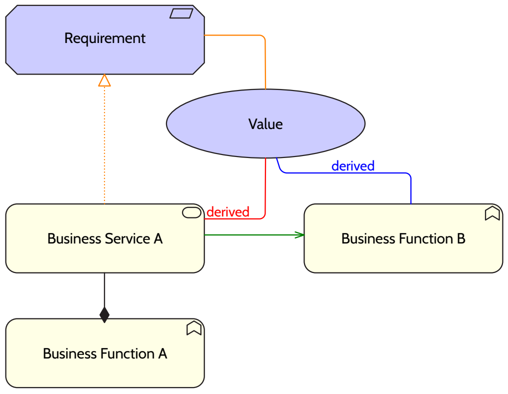
* Though I know from private communication that seeing service as independent from 'internal' behavior (based on the idea that a service is a sort of 'interaction' that cannot be seen independent from the behavior of the service consumer) was the view of (some of) the original ArchiMate designers, it can nowhere be found in the text of ArchiMate, from the original document to today. See also Section 35.4.

† A rather transactional view. There are clearly services that can exist without anyone consuming them, such as broadcasts. See also the footnote on page 205

thing like the process details anyway. For that, we have languages like BPMN. The Business Service is always an abstraction of actual behavior, but so is a Business Process in ArchiMate.

Hence, from a constructionist view, the external behavior must be a *part* of the internal behavior, just as the external interface already is a Composite part of the Node, Business Role or Application Component, offering that interface to the outside world. In other words: from a constructionist view we have 'behavior' and 'external behavior' which is part of 'behavior' and the obvious relation is a Composition.

In the original ArchiMate, there was no Requirement element type. Given the importance of the service *consumer*, it was logical the requirement side ended up (informally) inside the service concept. When thinking along the lines of 'business requirements drive service definition' — hence, outside-in — service also got a role as the 'requirement/value' side, the side that has to do with the consumer. On the interface side, this thinking was mirrored in 'required/provided' interfaces (an aspect already removed from ArchiMate).



View 359. *Constructionist Interpretation of Service, Combined with Motivation Extension Requirement*

But now that ArchiMate has the Motivation Extension to cover the requirement side, I think that we have the means to make a clean cut: the Requirement concept from the Motivation Extension covers the 'requirement/value' side and we can see the service itself — cleanly — as the 'externally usable ('exposed' as ArchiMate itself now puts it) *part*' of the provider's behavior (which has a meaning for the environment). In fact, it has been originally defined that way by the ArchiMate designers. The standard says about the generic 'service' concept:

> *An external behavior element, called a service, represents an explicitly defined exposed behavior.*

The standard explains:

> *Thus, a service is the externally visible behavior of the providing system, from the perspective of systems that use that service; the environment consists of everything outside this providing system. [...] For the users, only this exposed behavior and value, together with non-functional aspects such as the quality of service, costs, etc., are relevant. These can be specified in a contract or Service Level Agreement (SLA). Services are accessible through interfaces.*

The point I am making can be summarized as:

- the service is behavior and *all* behavior of the performer (the process) is an integral whole of which the 'exposed' is a *part*;
- the fact that you can (and probably should) *design* a service independently does not mean it should *exist* independently.

Interestingly. The definitions in ArchiMate 3 have changed from ArchiMate 2 and now support a 'service as exposed *part* of behavior' very well. E.g. for Business Service:

- *ArchiMate 2*: A business service is defined as a service[*] that fulfills a business need for a customer (internal or external to the organization).
- *ArchiMate 3*: A business service represents an explicitly defined exposed business behavior.

Instead of the need and the customer, we now only have the explicit mention that it is 'exposed' behavior Note: 'exposed' does imply that it is behavior solely by the 'exposer'.

The structure surrounding the service concept then becomes like View 359 (example at business layer level), where the service is a Composite part of the function. Using the Motivation Extension, the service Realizes a Requirement which is Associated with a Value. This more or less states that the Value is only there when the Requirement has been Realized, which is kind of nice.

ArchiMate 3 has dropped Association from the set of relations for which derivation rules exist. It would however be nice if some sort of derivation with Association would remain possible as long as the relation also plays a role in the actual metamodel. Hence, I've kept it in the example in View 359 to show what would be nice derived relations. Together the two orange relations would allow the derivation of the red Association linking Business Service to Value. And as Business Service A can be used by Business Function B, the red Association and the green Serving could be used to derive the blue Association. In other words: Business Function B has an association with the Value (because that Value is there when Business Function B uses Business Service A. I am not yet certain whether I want that derivation both with structural and dependency relations.

I too suspect that in the behavioral column, it has been more natural to talk about a service as being 'created' (the 'second' type of realization in Section 35.3 "Issues with abstractions" on page 204) because the behavioral column is all about 'doing' and 'creating' is a verb. In the active structure column, it could only be sensibly seen as an interface being 'part of' an Application Component or Node or Business Role. But it works as well (even better) if we just see the service as a (usable) part of the function, just like the interface is a (usable) part of the role/component/node. Not external/internal division but exposed/visible versus 'everything'.

Changing the Realization relation (between function/process and the service it provides) to a Composition relation also removes the unnecessary difference between the

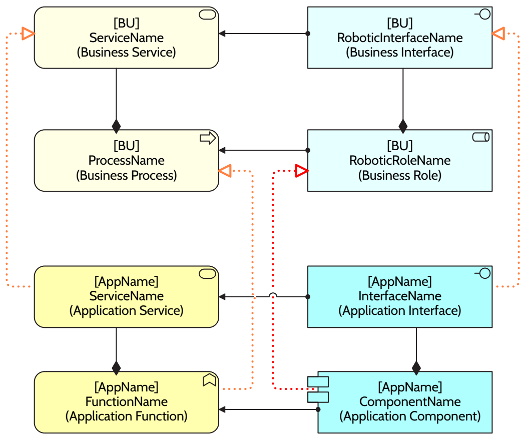
[*] Which before has been defined as: "A service is defined as a unit of functionality that a system exposes to its environment, while hiding internal operations, which provides a certain value (monetary or otherwise)".

behavioral column and the active structure column. The result looks like View 360.

This also has the effect that — using the current derivation rules — the derived relation between an active element and its service is always Assigned-To, it no longer depends on the route taken, which is also kind of nice.

35.5 Automated Processes

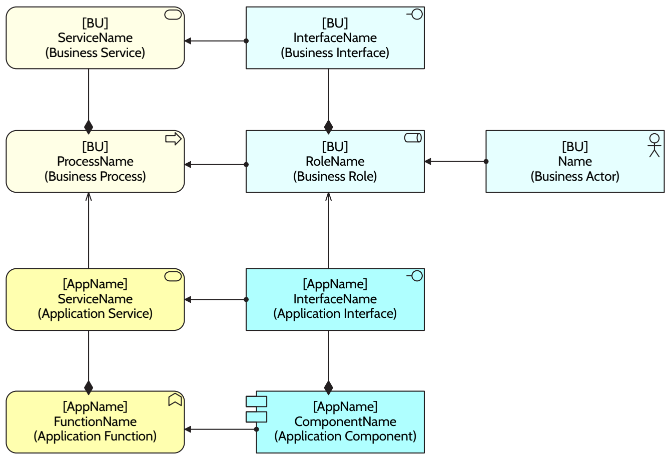
In Edition II of Mastering ArchiMate, I proposed a few changes. One of these was to use (identity-)Realizations to model automated processes. The idea was that application (and technology) layer would be able to Realize representations of themselves in higher layers. This would make it possible to better combine automated and non-automated elements of your landscape in a single structure. By having an Application Component Realize a (robotic) Business Role, the (robotic) Business Role could be modeled as a full-fledged business entity together with the humans. The proposal can be seen in View 361. This


[BU] ServiceName (Business Service)
[BU] RoboticInterfaceName (Business Interface)
[BU] ProcessName (Business Process)
[BU] RoboticRoleName (Business Role)
[AppName] ServiceName (Application Service)
[AppName] InterfaceName (Application Interface)
[AppName] FunctionName (Application Function)
[AppName] ComponentName (Application Component)

View 361. *Proposed meta-model Realization relations for automated processes*

is the 'automated process' version of View 360. They are the 'service-is-part-of-behavior' metamodel suggestions of ArchiMate's two ways of looking at layering (See 12.10 "Layering Ambiguity" on page 53).

Now, ArchiMate 3 comes with the three orange Realizations, but not with the red one. It also has added the

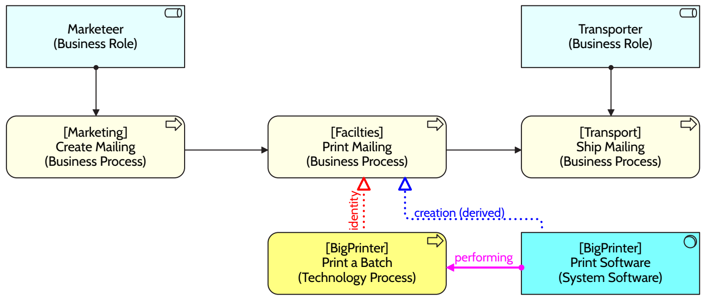

[BU] ServiceName (Business Service)
[BU] InterfaceName (Business Interface)
[BU] ProcessName (Business Process)
[BU] RoleName (Business Role)
[BU] Name (Business Actor)
[AppName] ServiceName (Application Service)
[AppName] InterfaceName (Application Interface)
[AppName] FunctionName (Application Function)
[AppName] ComponentName (Application Component)

View 360. *Proposal: A service is the Composite usable part of a Function*

same pattern to the metamodel between Technology on the one hand and Application and Business layers on the other. Suddenly, the 'identity-Relations' (see Section 12.10 "Layering Ambiguity" on page 53) have become center stage.

I'm not yet quite happy with this. What I like is the flexibility it gives you to have lower layer elements play a first class citizen's role in the business layer. What I don't like is the fact that the derivation of an Assignment (e.g. from Node to Technology Process) followed by an *identity*-Realization (e.g. from Technology Function to Business Function) derives into a Realization. There are two major problems with this.

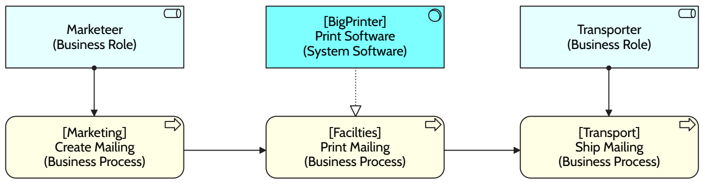
The first is that the Realization from Node to business behavior misleadingly suggests that the Business Process is an abstraction of the Node. The second is related to this but more practical. Take the semi-automated process in View 362. The marketeers prepare a mailing, the big


Marketeer (Business Role)
Transporter (Business Role)
[Marketing] Create Mailing (Business Process)
[Facilties] Print Mailing (Business Process)
[Transport] Ship Mailing (Business Process)
identity
creation (derived)
[BigPrinter] Print a Batch (Technology Process)
performing
[BigPrinter] Print Software (System Software)

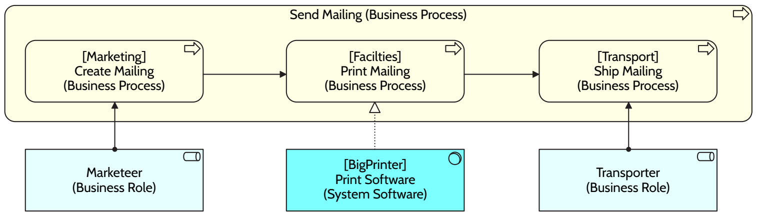
View 362. *Technology performing business behavior*

printer system prints it and the transport people ship it. In ArchiMate 3, we can add the Print Mailing (Business Process) and let it be (identity-)Realized by the Technology Process that is performed by BigPrinter's software. Now, you might not want the intermediate clutter and the nice thing is of course that you don't need it. It is perfectly all right to model the System Software to perform that automated business process. The derivation of the (violet) Assignment and the (red) Realization is the (blue) Realization. So, our diagram becomes View 363:

So far so good, but these three Business Processes are not full end-to-end processes. They are mere steps, sub-processes, in the whole. If we add the whole we get View 364.

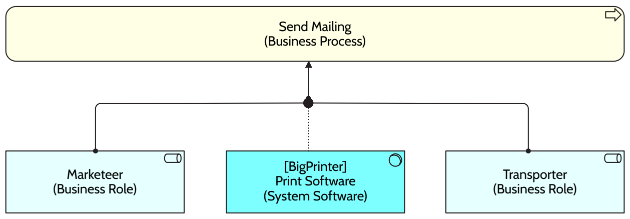

Marketeer (Business Role)
[BigPrinter] Print Software (System Software)
Transporter (Business Role)
[Marketing] Create Mailing (Business Process)
[Facilties] Print Mailing (Business Process)
[Transport] Ship Mailing (Business Process)

View 363. *View 362 with the technology behavior left out*


Send Mailing (Business Process)
[Marketing] Create Mailing (Business Process)
[Facilties] Print Mailing (Business Process)
[Transport] Ship Mailing (Business Process)
Marketeer (Business Role)
[BigPrinter] Print Software (System Role)
Transporter (Business Role)

View 364. *The process steps of View 363 Aggregated in an overall parent process*

Now, suppose we want to get an ever simpler, more abstracted, diagram. We *are* architects, after all. We just want to show the overall Business Process and who performs it. We can do that as shown in View 365:



View 365. *Modeling how people and automation together perform the semi-automated process of View 364*

Sadly though, we can't. Because we are not allowed to have different types of relations on a Junction. And that is a wise constraint of ArchiMate because it would be totally undefined what a Junction means if it was allowed.

The problem can be easily solved as per the suggestion in the next section:

35.6 Changing the Strength of Assignment and Realization

The strength order of structural relations was decided upon, but I have no documentation of why that particular order was chosen and privately I have been told it was more intuitively decided than reasoned. Part of it may have been the result of the 'abstractionist' (weakly coupled) view on the service concept.

What happens if we switch Assignment and Realization in the strength table for deriving structural relations? If we start in the middle of the ArchiMate meta model, the basic Application Pattern (see View 357 on page 205), the *derived* relation of the route from Application Component via Application Function to Application Service changes from Realization to Assignment. Incidentally, that is the same result that we get if we follow the route from Application Component via Application Interface to Application Service.

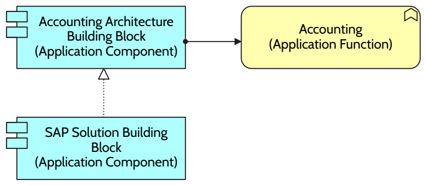
This is kind of nice in two ways: first, because the derived result does not depend on which route you take. But secondly, because it is kind of nice to have Assignment as the resulting relation between an active component and a behavioral component. It means that you never break the pattern that an active element is Assigned-To a behavioral element. And that is a much more direct statement about your landscape (who does what) than the fuzz that the architect's abstractions bring.

In the real ArchiMate meta-model, the derived relation from Node, via Assigned-To to Artifact, via Realization to Data Object and via Realization to Business Object is Realization. If we switch strengths of Assignment and Realization, the resulting relation would become Assigned-To in its meaning of 'resides on'. In other words the Node is Assigned-To the Business Object, or, the Business Object resides on the Node, which is I think a slightly cleaner way of looking at it than that the Business Object is 'an abstraction' of the Node.

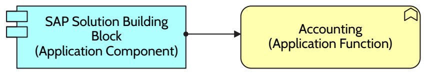
When we change these strengths, the derived relation of "Device Assigned-To Artifact Realizes System Software", becomes Assignment, which is also the direct relation that still exists in the metamodel, which can then safely be removed cleaning up the baggage of the past.

Another effect is that System Software that is Assigned-To Artifact that Realizes an Application Component becomes a derived Assignment. We nicely get that Application Component — just like System Software — can be deployed on System Software. As is the fact in the real world.

In fact, I would like to see the (identity-)Realization relation become the strongest relation of them all. This makes a lot of sense. After all, what the identity-abstraction says is that both ends of the relation are the same thing, just differently represented. So for me, when someone wants to model TOGAF's logical Architecture Building Block and Solution Building Block, it makes sense that View 366 can turn into View 367.



View 366. *TOGAF ABB and SBB*



View 367. *Derivation from View 366*

35.7 Make Access multi-directional

ArchiMate 2.0 removed the bi-directionality of the Assignment relation that existed in ArchiMate 1, and that was a good move. That bi-directionality in ArchiMate 1 led to all kinds of senseless derived relations. ArchiMate 2.0 removed all of those and added the ones that were no longer derivable and that made sense explicitly to the core meta-model.

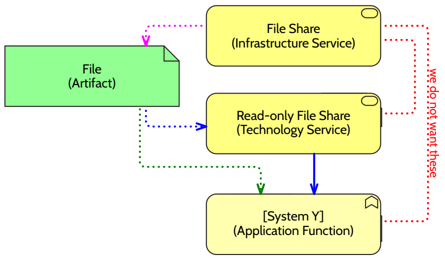
So, why propose now to make another relation bidirectional? Well, what drives this is a reality, namely that Behavior may depend on passive elements, and not only the other way around. A good example is application maintenance from 18.6 "Secondary Architecture: Application Maintenance" on page 106. Here, the application maintenance process edits a file, say an '.ini" file, that influences an application's functionality. The application's functionality is dependent on the settings in the file (on the 'Settings' Data Object the Artifact realizes). Though the Artifact is shared, the Data Object isn't, which shows up when you make errors in that ini and the application crashes.

Actually, I think it is best to make the direction of the Access relation depend on its 'read/write' status:

- Read Access: direction from passive structure to behavior;
- Write Access (including Create and Delete): direction from behavior to passive structure;
- Both read and write Access, or undefined: bidirectional.

To illustrate what derived relations we can have when Access becomes bidirectional, have a look at View 368.

The violet and blue Access relations are the original ones. Under the ArchiMate 3.0.1 rules, none of the other Access relations are derivable, because the Access relations runs from behavior to passive. However, if we use the multi-directional approach, the green Access relation becomes derivable from the blue Access and Serving relations. But now we also can derive the red Access relations and that is what we do not want. Luckily, the standard limits (in Appendix B) the derivation of Access to end-situations where one side is a passive element. This could be made bidirectional. Note that if we limit Realization to the 'identity' role and we start using the Access relation for the 'creation' role, the situation becomes more complex and the ArchiMate designers need to solve a more complex puzzle.


File Share (Infrastructure Service)
File (Artifact)
Read-only File Share (Technology Service)
[System Y] (Application Function)
we do not want these

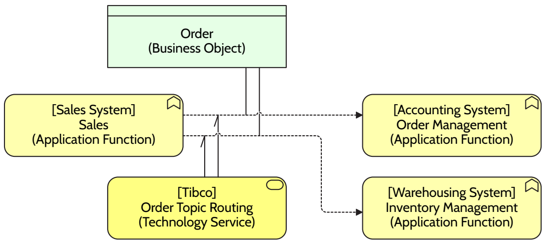
View 368. *Multi-directional Access derivation*

35.8 Allow two-way Flows and Triggers

This one is easy. Two way Flows and Triggers would be a very easy and efficient way to model two way dynamic interactions. Especially for Flows — and certainly at a higher abstraction level — communication is often two-way. Having to model two separate Flow relations for that is a bore. This could even be solved by allowing in the standard the use of a two-way form to represent under water two separate relations.

35.9 Allow Serving to other relations

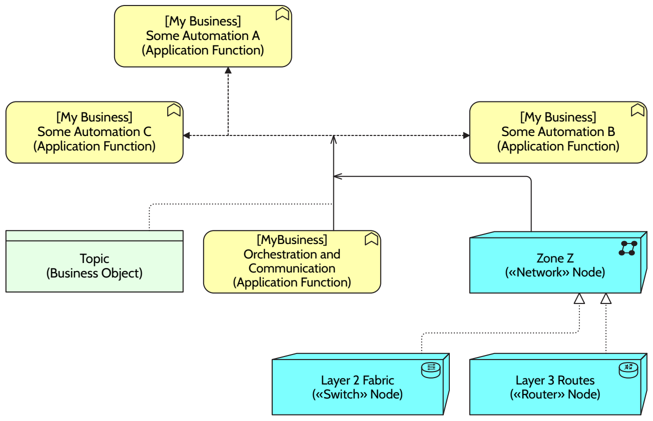
In section24.4 "Routing and Orchestrating Middleware (ESB and APIs)" on page 129 we encountered the diagram repeated here in View 369. I modeled that some service — in this case a Technology Service — was instrumental in making a Flow possible. What I wanted to do is use *Serving* for that, but that is not allowed. So I used a directed Association. But if you think about it, the Flow is a kind of a shorthand for behavior. Structurally, we would let the Technology Service Serve the Sales and Inventory Management functions. But Serving the Flow is much more precise.


Order (Business Object)
[Sales System] Sales (Application Function)
[Accounting System] Order Management (Application Function)
[Tibco] Order Topic Routing (Technology Service)
[Warehousing System] Inventory Management (Application Function)

View 369. *The ESB as infrastructure 'Realizing' the Flows*

There are many more changes that I think could be made, but these are more radical. I even have some radical enough ideas that — were they implemented — we would have to say that it is really a new language and a new approach to modeling.

Having said that, even within the context of ArchiMate, we could do some pretty radical things that increase the power of the language, while enabling options to simplify other parts. I'm not going to write this all down here in detail, but I would like to illustrate some by providing you with a single example. Take a look at View 370. This view, which — let's remind people who are leafing through the book and starting at random somewhere — is absolutely not valid ArchiMate at the moment contains the following patterns:


[My Business] Some Automation A (Application Function)
[My Business] Some Automation C (Application Function)
[My Business] Some Automation B (Application Function)
Topic (Business Object)
[MyBusiness] Orchestration and Communication (Application Function)
Zone Z («Network» Node)
Layer 2 Fabric («Switch» Node)
Layer 3 Routes («Router» Node)

View 370. *Some rather radical ideas*

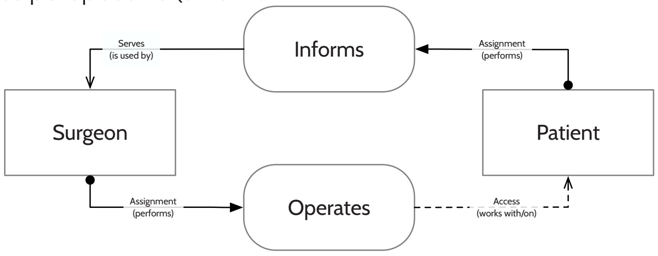
- The two-way Flow as mentioned in section 35.8 "Allow two-way Flows and Triggers";
- Letting external behavior Serve Flows and Triggers as mentioned in 35.9 "Allow Serving to other relations". If we could do that, the one shown in the diagram would be derivable;
- Attaching Flows to other Flows to build actual communication patterns. E.g., by extending this, one could model buses or hubs. Here shown as just attaching flows to flows, but in current ArchiMate we could already do this by having a central hub (a Junction) and let all the Flows be spokes. In that case, however, we should be able to Serve the Junctions instead;
- Let a Serving relation Access passive structure. The reasoning is this: by looking at Serving as being an extension of what external behavior/structure is we could connect the payloads involved. Note, we can already do this with Association;
- Let Serving Relations be Served in turn. This could also be derivable by making an outgoing Serving relation of an element be able to inherit an incoming Serving relation to that same element. But a direct relation is more useful: it is clear that this structural relation depends on another structural relation.

You could make the same arguments about for instance Serving being allowed to Serve all structural relations. E.g. a certain service enables that a certain active element is Assigned to a behavior element. That would be modeled by a Serving relation from the enabler to the Assignment in question. It would also make the Communication and Path elements prime candidates for a more abstract role in the Business Layer (as Specializations of Actor?) as we could do the technical side with normal technical elements).

35.10 Why change ArchiMate?

I think ArchiMate is great. It is the best thing since sliced bread for Enterprise Architecture modeling. The language is not strictly formal, but its concepts and relations have been selected for usability, and as Uncle Ludwig explained to us, that it one of the best tests of meaningfulness.

The world of Enterprise Architecture stretches from the strictly logical world of bits and bytes to the not-always-so-logical world of human behavior. It is unavoidable that such a stretching exercise leaves its marks. So, it is easy to find (logical) fault with the language, as I have shown as well. But from a business perspective (a human perspective) ArchiMate is very good at enabling you to model to the extremes of Enterprise Architecture. And the fact that — even without all these improvements — we still were able to use the grammar to the extent we did shows how powerful the language already is.



View 371. *Active/Passive is not a property, it is a consequence of behavior*

With the right use of patterns and the right discipline and a good knowledge of the powerful underlying ideas, you can take this language far. Even without the improvements proposed by me in this chapter.

But I do think cleaning a few things up and improving the language here and there would make it greater still. It is up to the — by nature (and rightly so) conservative— standards body to take that step. Standards bodies *should* be conservative and slow, or the standard would be too volatile to be a real standard. On the other hand: when 'backwards compatibility' becomes your main worry, and you cannot innovate, the standard will probably die.

Having said that, I must say that if you write a book like this (with all that detail and hundreds of diagram, and all those small side remarks that are so instrumental in giving you a feel for the language) every update of the standard, — however `minor' — tends to become a lot of work. Because when the standard changes, every detail has to be checked if it is still correct. That is hard. Hence, I think that there probably still are details in this book that are incorrect. There must be. If you find them, mail me.

In the meantime, I have come to the conclusion that I both like improvements and hate them for the inordinate amount of work they result in. Some changes would mean I would have to write an entirely new book as with new possibilities the existing patterns shown would become too suboptimal. You would still learn the grammar, but the patterns would not be necessarily good ones.

So, making a decision about these changes would be a nightmare.

Luckily, it is not up to me but up to The Open Group.

35.11 ArchiMate's two most serious (and unsolvable) issues

In my opinion, the most fundamental problem with ArchiMate is the split between active and passive in structure. They weren't thinking properly twenty-odd years ago, principled (logical) 'grammar thinking' won out over 'realistic thinking'.

The problem is that active structure is often also passive structure. If a human actor can act, it can also be acted-upon. In fact, the right definition for 'active structure' might be 'structure that acts'. It is not a fundamental property at all, it is a matter of being assigned to behavior that turns structure into active structure.

In View 371 there is an example:

Assignment from structure to behavior makes structure active. Being acted upon ('Accessed'), makes structure passive. The relations define what is the case and that may be both.

ArchiMate is not a language, it is a grammar. And grammatical thinking (Uncle Ludwig again) simply cannot catch reality (as an enterprise architecture notation must).

This one, together with the BDAT 'layering' (I invite you to read https://ea.rna.nl/2022/08/20/layering-is-it-really-a-useful-approach-in-business-it-enterprise-architecture/), are so fundamental to ArchiMate that fixing them isn't really possible. You will have to define a new grammar.

35.12 Does ArchiMate have a future?

Everything goes extinct sometime. Even ArchiMate will. ArchiMate could be better, but for the coming years the market forces will make it difficult for something to establish itself.

The biggest issue I see for ArchiMate is that with the world's (digital) landscapes becoming more complex and more volatile (elements appearing and disappearing within seconds) over time, having a language that is visual — as it is intended for *human* consumption — is becoming more and more a problem. If we design — and we will, AI won't do it for us for the forseeable future — we will design patterns. And it it will be necessary to link these patterns fleetingly to instances of those patterns. Automation of our knowledge about the landscape will be inevitable. Those kind of systems are already available and they are not ArchiMate-based, nor can they really be. A more fundamental grammar, based for instance on elements of this discussion chapter, and potentially generative AI that is able to create 'good enough' human-aesthetics-pleasing representations. This is far off, I guess. And until then, ArchiMate will have enough value in our design and documentation processes to not yet go extinct.