

## Using ArchiMate with an Architecture Method

### *A conversation*

Graham Berrisford and Marc Lankhorst

### Introduction

The ArchiMate language for enterprise architecture [1] has recently been adopted by The Open Group as a technical standard. This may result in a steep rise in its popularity. However, ArchiMate is only a language and does not prescribe a way of working. Hence, it will be used in combination with some other method to guide the process of architecting. Some assume it will be easy to use ArchiMate when following the process of their preferred architecture method. Indeed, it will be easy to use the ArchiMate box symbols in drawing artifacts/diagrams. Perhaps many will do this, thinking they have thus used the ArchiMate language with their method.

But ArchiMate is not just a set of box shapes and line styles. If you only use ArchiMate symbols to draw the artifacts of an architecture method, you can miss the point of ArchiMate, and possibly undermine its qualities. ArchiMate is a component-based development approach, with its own meta model, entity definitions, and philosophy.

In this paper, the first of a series edited from our conversation on the topic over recent months, we intend to shed light on the possibilities of using the ArchiMate language with an architecture method and inform you on the pitfalls you may face.

The motivation for this series of papers is threefold:

1. We are not so worried if people use methods like TOGAF in ways that deviate from what is defined as "core" TOGAF. We are more concerned that people rarely use a modelling language as its authors intend. The truth is that most people (other than programmers) *do not use* UML. They use UML *tools*. They use UML boxes and lines to mean whatever they want. They do not even realise they are not using UML. That undermines the notion of UML as a language. If we can help ArchiMate being abused in the same way, then we would like to.
2. Consultants may well combine methods like TOGAF and ArchiMate in ways their customers accept. But if nobody notices they are not using either "properly", then we will never correct the weaknesses in the methods. We want to expose the discrepancies between TOGAF and ArchiMate, and their weaknesses, sufficiently clearly that someone can and will address them. That requires a properly detailed analysis of inconsistencies - between the methods and within them.
3. Teachers of architecture processes and notations need explanations that work from first principles to form a coherent overall explanation. We are not satisfied with informal alignments of methods that depart from what their authors intended - since students are then left without understanding the basic principles and ask embarrassing questions that cannot be answered.

Naturally, it is never possible to “enforce” correct usage of any language or method. No method is perfect. They are all products of inevitable compromises. In the end, the skills of the architect are much more decisive than the methods he or she uses. Nevertheless, we should explain our methods in ways that make them easy to understand, minimise confusions and maximise the quality of the end result.

In this paper, we first outline the basic principles on which ArchiMate is founded. Next, we will address a series of questions that will help you decide how readily you can use ArchiMate with your preferred architecture method.

Our conversation is continuing and we will edit that into more specific papers on using ArchiMate with The Open Group Architecture Framework (TOGAF) [2] and on some fundamental differences and similarities between the two. We have added brief references to this paper. This conversation also intends to extend and improve the ideas on combining ArchiMate and TOGAF outlined in [3] and it applies various thoughts on modelling and abstraction as described in [4]. These ideas may also serve The Open Group’s working group on the convergence of ArchiMate and TOGAF as input.

## ArchiMate basics

Within many of the different domains of expertise that are present in an enterprise, some sort of architectural practice exists, with varying degrees of maturity. All kinds of frameworks try to map these domains, such as the Zachman framework, TOGAF, and many more. However, due to the heterogeneity of the methods and techniques used to document the architectures, it is very difficult to determine how the different domains are interrelated. Still, it is clear that there are strong dependencies between the domains. For example: the goal of the (primary) business processes of an organisation is to realise their products; software applications support business processes, while the technical infrastructure is needed to run the applications; information is used in the business processes and processed by the applications. For optimal communication between domain architects, needed to align designs in the different domains, a clear picture of the domain interdependencies is indispensable.

Hence a language for modelling *enterprise architectures* should focus on inter-domain relations. ArchiMate therefore provides concepts to model both the global structure *within* each domain, showing the main elements and their dependencies, and the relations *between* the domains, in a way that is easy to understand for non-experts. To facilitate learning and understanding ArchiMate, it has a limited set of concepts and is built on a number of basic elements that are visible throughout the various layers of the language.

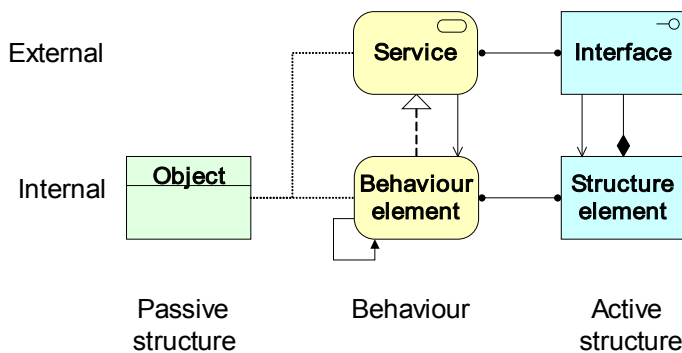
First, its basic structure draws on the deep grammatical structure of natural language. All human languages have sentences consisting of subjects, verbs and objects; only their order differs. So we have languages with an SVO grammatical structure like English, SOV like Hindi, or VSO like Arabic. Similarly, ArchiMate distinguishes between *active structure* elements (the business actors, application components and devices that display actual behaviour, i.e., the ‘subjects’ of activity), *passive structure* elements, i.e., the ‘objects’ on which behaviour is performed, and *behavioural* elements, the activities that are carried out, i.e., the ‘verbs’. Behavioural concepts are assigned to active structural concepts, to show who or what carries out these activities.

This subdivision also has implications for the design process. In developing a greenfield architecture, we often work from behaviour to structure, i.e., first design the necessary functionality and next assign it to the constructional units that have to provide this functionality. In accommodating pre-existing elements into the architecture, a bottom-up (or perhaps middle-out) way of working is often called for: first describing the existing components, actors, etc., and next analysing their behaviour and extracting the services they provide.

Second, ArchiMate distinguishes between an *external view* and an *internal view* on systems. When looking at the behavioural aspect, these views reflect the basic principles of encapsulation that are also prevalent in service orientation. The *service* concept represents a unit of essential functionality that a system exposes to its environment. For the users of a system, only this external functionality, together with non-functional aspects such as the quality of service,

costs etc., are relevant. Services are accessible through *interfaces*, which constitute the external view on the structural aspect.

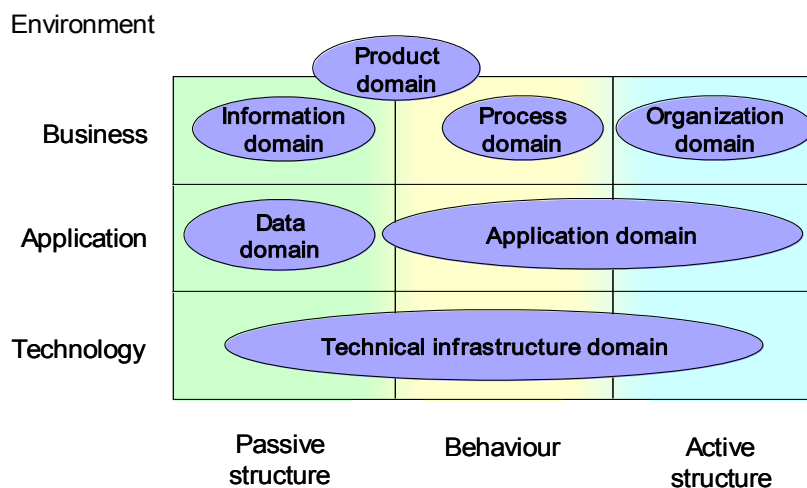
Again, this is also reflected in the design process when we use ArchiMate: usually, we first design the desired external properties of a system, and next its internals.



Although, at an abstract level, the concepts that are used throughout EA models in ArchiMate are similar, ArchiMate defines more concrete concepts that are specific for a certain layer of the architecture. In this context, we distinguish three main layers:

1. The *Business layer* offers products and services to external customers, which are realised in the organisation by business processes (performed by business actors or roles).
2. The *Application layer* supports the business layer with application services which are realised by (software) application components.
3. The *Technology layer* offers infrastructural services (e.g., processing, storage and communication services) needed to run applications, realised by computer and communication devices and system software.

This results in the language framework shown below. In this framework, we have projected commonly occurring architectural domains.



## Core questions

To assess whether an architecture method matches the ArchiMate language sufficiently to use the two in combination, we need to investigate whether the core ideas of ArchiMate as outlined in the previous section are in accordance with the method's fundamentals. This leads us to the following set of questions:

1. Does the method separate process from artifacts? If a method's process (way of working) is too tightly coupled to the artifacts it delivers, using a language like ArchiMate for these artifacts may prove problematic.

2. Does the method separate structure from behaviour? This separation is an important element of ArchiMate with impact on the design process, as described above.
3. Does the method separate external from internal? As explained, the concept of encapsulation is another fundamental tenet of ArchiMate (and many other modelling languages).
4. Does the method separate logical from physical? Many methods use a progression from more abstract (or logical) to more concrete (physical) things. ArchiMate's realisation relationship is an example of such a progression, but the terms 'logical' and 'physical' are used in various ways in different methods. The concepts for abstraction used by the method need to match ArchiMate's sufficiently.
5. Does the method separate the general from the specific? This is another kind of abstraction that features in some architecture methods. ArchiMate has the generalisation relationship but doesn't really emphasise this dimension. Some methods do, however, and if so, how does this match with ArchiMate?
6. Does the method present domains as layers? As outlined before, ArchiMate provides a three-layer structure, Does your preferred method feature the same layering?
7. Does the method's meta model align with ArchiMate's? Some methods come with their own explicit and/or implicit metamodel. Do these match sufficiently, i.e., can we provide a workable mapping of concepts and relations?
8. Does the method rely on UML? Since ArchiMate uses some concepts derived from UML, but differs in some important aspects, some confusion may arise if your chosen method is based on UML (or on a UML profile).
9. How abstract is the method? ArchiMate targets the enterprise level of abstraction and the level directly below that (domain architectures). Its concepts possess just enough 'meaning' for that level of abstraction, which should match with the level targeted by a method used in conjunction with it.

The subsequent sections address these questions one by one.

## Does the method separate process from artifacts?

Marc: Wherever a method separates architecture process from architecture descriptions (entities, artifacts and outputs) we should be able to use the process to produce an ArchiMate-style architecture. f

Graham: We'll expect an architecture method to come with its own architecture entities and artifacts. Moreover, the bigger challenge for ArchiMate is that its process steps will produce outputs that imply a meta model relating those architecture entities.

Marc: Yes. We don't want following the process of an architecture method to water down the essential ideas of ArchiMate.

Marc: Does TOGAF separate process from artifacts?

Graham: Inevitably, the ADM process refers to artifacts and deliverables. And strictly speaking, TOGAF defines the process steps and their outputs as "core", meaning that if you don't produce the outputs, then you are not following TOGAF.

Marc: Still, I have seen the ADM process used to construct an ArchiMate-style architecture. E.g. at a life insurance company here in the Netherlands.

Graham: Using ArchiMate to construct ADM artifacts and outputs is more challenging, because we have to address the underlying meta models.

## Does the method separate structure from behaviour?

Marc: Yes. We don't want following an architecture method to water down ArchiMate's separations between internal and external, and between structure (active & passive) and behaviour.

Graham: The table below shows these two essential dimensions as rows and columns, and contains the five entities in ArchiMate's meta meta model.

ArchiMate dimensions	Structure (passive)	Behaviour	Structure (active)
External		Service	Interface
	Object		
Internal		Behaviour element	Structure element

Marc: You're right about the essential nature of this two-way classification. The separations between internal and external, and between structure (active & passive) and behaviour are central to ArchiMate.

Graham: To make the same distinctions explicit throughout any given architecture process could require significant reorganisation and rewriting of the process. (By the way, the structure-behaviour distinction does feature in the ISEB reference model for Enterprise and Solution Architecture [5])

Marc: Does the structure-behaviour distinction appear in TOGAF?

Graham: Not explicitly, but it isn't hard to find.

## Does the method separate external from internal?

Marc: ArchiMate's external-internal dimension separates services and interfaces from their realisation or implementation by process and components.

Graham: This interface-component distinction is the common thread in all component-based development methods. It is a feature of SOA. It is related to distinctions we might call requirement-design, and logical-physical.

Marc: Does the external-internal distinction appear in TOGAF?

Graham: Yes, though the distinction is not drawn clearly and consistently throughout ADM, and it isn't the core idea that it is in ArchiMate.

## Does the method separate logical from physical?

Marc: Object-oriented software architects model the implementation of interfaces by components using realisation relationships. And in ArchiMate, such relationships are used to model the progression from 'more logical' to 'more physical'.

Graham: Ah! You may reasonably say that services are more logical, and the components that provide them are more physical. But a component may be either logical or physical in the sense we need to address under this heading.

Marc: I gather that in TOGAF, an information system service is realised by a logical application component, which in turn is realised by a physical application component.

Graham: Only very loosely speaking. Both those relationships are many-to-many. And your two "realised" have two different meanings. The first describes the external-internal relationship that is strongly emphasised in component-based development methods. The second describes the relationship between vendor-neutral (logical) and vendor-specific (physical) specifications that is strongly emphasised in public sector methods and frameworks.

Marc: I know Zachman, TOGAF and other frameworks put considerable emphasis on this logical-physical distinction. However, people do get confused in this area. That is why we didn't put

the logical-physical distinction into the ArchiMate language as a separate dimension.

Graham: I understand people’s confusions, and your reluctance to cloud their understanding of ArchiMate. But this logical-physical dimension is fundamental to many methods. So we must clarify it and address it.

We have discussed two kinds of logical-physical distinction. ArchiMate is centred on the idea that required services (logical) are realised by designed components (physical). Whereas many architecture methods are centred on the idea that vendor-neutral component specifications (logical) are realised by vendor-specific component specifications (physical).

An architecture method typically follows a process of logical-to-physical realisation in which requirements are realised as logical components, which are realised as physical components, which are realised as deployed solutions.

Marc: Does this kind of logical-to-physical dimension appear in TOGAF?  
 Graham: Yes, very much so. It appears as in the content classification scheme known as the Enterprise Continuum. And it appears in the progress of the ADM process.

### Does the method separate the specific from the general?

Graham: Enterprise architecture is cross-organisational and strategic. It is about generalities: general principles, general standards, industry reference models, common systems, design patterns, reusable components. Enterprise architects deal largely in generalisations, be they at the logical or physical level.

Marc: Generalisation is not a big deal in ArchiMate philosophy, though you can model it of course using the generalisation relationship.

Graham: Yes. It is another kind of abstraction. This table below shows idealisation and generalisation as orthogonal dimensions.

Generalisation	Universal	Fairly generic	Fairly specific	Uniquely configured
Idealisation				
Ideal				
Logical				
Physical				
Real				

Marc: Does the general-to-specific dimension appear in TOGAF?  
 Graham: Yes. It appears as the horizontal axis in the four-by-four content classification scheme called the Enterprise Continuum.

### Does the method present domains as layers?

Marc: ArchiMate presents the three architecture domains (business, application and technology) as layers. Might we map the application and technology layers to the logical and physical levels/rows in the Enterprise Continuum, or the Zachman framework?

Graham: No, though the Zachman Framework is widely misinterpreted this way. John has restated his basic ideas (on his web site) as these. “The Zachman Framework is a schema [based on] classifications that have been in use for literally thousands of years.

- The first is the fundamentals of communication found in the primitive interrogatives: What,

How, When, Who, Where, and Why.

- The second is derived from reification, the transformation of an abstract idea into an instantiation.”

So, the purest form of John’s schema is a table that maps the levels of idealisation (in rows) to six interrogatives (in columns).

The Zachman Framework		What	How	Where	Who	When	Why
Ideal to Real	Identification						
	Definition						
	Representation						
	Specification						
	Configuration						
	Instantiation						

However, John has always applied this schema to the world of information systems. I’d summarise the levels thus. Level 6 is run-time systems. Level 5 is machine-interpretable code, data definitions and configurations (compile-time specifications). Level 4 is more abstract specification of the compile-time specifications. Level 3 is more idealised vendor-neutral specifications. Level 2 is application-independent specifications of the business. Level 1 is the sketchiest outline.

TOGAF focuses on level 2 more than any other. The table below maps the four levels of the TOGAF 9’s Enterprise Continuum to the six levels of Zachman’s schema - as they are variously interpreted.

Levels in TOGAF’s Enterprise Continuum	Zachman levels v1 As John means them	Zachman levels v2 As stakeholders or viewpoints	Zachman levels v3 As what? architecture domains or layers?
Context & requirements	Identification	Planner	Scope
Architecture continuum	Definition (conceptual)	Owner	Enterprise
	Representation (logical)	Designer	System
Solutions continuum	Specification (physical)	Builder	Technology
Not addressed	Configuration	Subcontractor	Detailed
Deployed solutions	Instantiation	Operations	Operations

The 2<sup>nd</sup> and 3<sup>rd</sup> interpretations of the Zachman levels are somewhat at odds with the first. The 3<sup>rd</sup> has been bent by Jaap Schekkerman and others (Cap Gemini?) to fit layered architecture domains. Jaap’s schema turns levels 2, 3 and 4 into architecture layers. But that is to radically change the meaning of the levels. You might as well map architecture domains to John’s columns as to his rows. A much better match is to be found between the levels/rows of the Zachman Framework and the Enterprise Continuum.

The truth is, no single two-dimensional table is adequate as an architecture framework. We need to separate six, seven or eight dimensions. See also [4].

Marc: Does TOGAF present architecture domains as layers?  
 Graham: There is one over-complex nested box diagram that can be read to do this, and you could say the idea is implicit, but it is not thoroughly worked through.

## Does the method's meta model align with ArchiMate's?

Graham: There are bound to be mismatches between the meta model of ArchiMate and that of any architecture method we examine (if it has its own meta model of course). They may well be serious enough to confuse people, and they could damage understanding of ArchiMate.

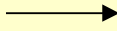
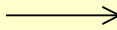
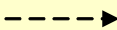
Marc: Does the TOGAF meta model align with ArchiMate's?  
 Graham: Hmm... You cannot properly understand TOGAF by studying its explicit meta model (derived from the meta model in SAP's EAF) because it does not fully match the implicit meta model in the TOGAF text.  
 Marc: Convergence of explicit and implicit meta models is a challenge the Architecture Forum needs to resolve, ideally before convergence with ArchiMate. However, the explicit TOGAF meta model doesn't look too far removed from ArchiMate's.  
 Graham: Hmm...I am not convinced. We need to do lot more work at the level of detailed entity-by-entity comparisons.

## Does the method rely on UML?

Marc: ArchiMate is not intended as a competitor to UML for software architecture.

Graham: Nevertheless, they may be used in the same organisation, or even the same team. ArchiMate is both similar to and different from UML. I'm not bothered about differences between box shapes. I am more concerned about differences between relationship lines. The following table of gaps and differences shows the four kinds of abstraction relationships are the same in ArchiMate and UML, but the remaining relationships are different.

Comparison	UML	Line symbol	ArchiMate
Gaps	Dependency		???
	???		Assignment
	???		Access
Matches (Abstraction)	Association		Association
	Aggregation		Aggregation
	Composition		Composition
	Specialisation		Specialisation
	Realisation		Realisation

Differences	Data flow		Trigger
	Trigger		Use
	???		Flow

Marc: The table is a reasonable summary. We might include a dependency-like relation in a future version of ArchiMate, if sufficient demand from practice arises. The reason we didn't include it originally is that we found that all dependencies in practice turned out to be one of the other kinds of relations (e.g. use or access).

Graham: Yes. That is because a dependency is an abstraction (by composition) of other kinds of relationship, for example several invocations or data flows.

Marc: Does TOGAF use UML in any way?  
 Graham: No. TOGAF is concerned with the application portfolio rather than application design. Class diagrams are mentioned in different context from software design. Interaction diagrams are not mentioned in the definition of a process-system realisation diagram.

## How abstract is the method?

Graham: Enterprise architecture is cross-organisational and strategic. Enterprise architecture diagrams are usually more abstract than the solution architecture diagrams. Enterprise architects draw dependency relationships that are abstractions from the more atomic relationships drawn by system, solution and software architects.

Marc: True, but in ArchiMate the "use" relation (missing from UML) largely serves this role. The other major kind of dependency is a realisation. Even at a high abstraction level, it is usually clear if one architecture element uses another or is realised by another.

Graham: Yes, an enterprise architect might draw a line to represent not only "uses" and "realises" relationships, but also "replaces" and "migrates to" and "traces to" relationships.

Marc: Relationships covering the change of an architecture between states, like your examples, are useful as well (and are currently lacking in ArchiMate), but I wouldn't group them in the same category as the other types of dependency. We could add a dependency as a generic super class of all directed relations in ArchiMate (pointing in the opposite direction to the UML dependency), but that would result in a relationship with a very weak meaning - basically a sort of 'directed association'.

Graham: Enterprise architects often document relationships for use in impact analysis, gap analysis, cluster analysis and traceability analysis. In such models - a general dependency relationship may well be enough.

Marc: But you can also model this using the slightly less generic relationships like "use" and "realisation". We have tried to avoid weak relations to make ArchiMate models more 'meaningful'. Otherwise we might end up with models consisting of nothing more than boxes and lines (since the same argument holds not only for the relations, but also for the other concepts of the language). Nevertheless, if there is a real need for a dependency-like relation, this could be added in the future.

Graham: Experience suggests that, at the most abstract level of enterprise architecture description, we do indeed end up with diagrams that are simply boxes and lines. Enterprise architecture is abstract – cross-organisational – strategic. The diagrams are a long way from the realisation of the architecture as a physical system. As you reverse engineer, you abstract from, the bottom-level and detailed diagrams to higher-level more abstract views, the meaning of the lines between the boxes evaporates.

Marc: But then you need to assign more meaning to these boxes and lines to understand or

explain these diagrams, either implicitly (by the name in the box) or explicitly (by using specific types of boxes and lines like ArchiMate does). In the first case, you then need to explain your naming scheme, and implicitly you are then constructing a kind of modelling vocabulary. Eventually, you'll run into the same discussion. To have a meaningful and workable architecture, you can't do without some meaning. Of course, we can debate how generic or specific these concepts can be, and a dependency might well be useful in ArchiMate too, but using concepts that are too generic leaves you with a pretty picture that isn't very useful in practice.

Graham: Meaning can be added when we elaborate from enterprise architecture diagrams to solution or software architecture diagrams. The main purpose of the higher level enterprise architecture diagrams is to help impact analysis in change management, rather than to help designers.

Marc: True, but I wouldn't want to abstract away all meaning. I think it's a matter of balance.

Graham: In my view, ArchiMate is probably best used at the level of system or solution architecture. Its use at the level of cross-organisational strategic enterprise architecture will likely be more limited. But I am sure there is a great deal more to be said about abstraction in general, and levelling of architecture specifications in particular.

Marc: ArchiMate extensions towards goal/requirements modelling are envisaged; these should make it more applicable at the "level of cross-organisational strategic enterprise architecture".

Marc: How abstract is TOGAF?

Graham: Very. It tells us "physical elements in an enterprise architecture may still be considerably abstracted from solution architecture, design, or implementation views".

## Conclusion

Although it would be easy to think that ArchiMate can be used with 'any' method, there are some serious considerations to be addressed, as we have shown in the previous sections. In the next instalments of this series, we will specifically look into the use of ArchiMate together with TOGAF, and some fundamental differences and similarities between the two.

## References

- [1] The Open Group, *ArchiMate® 1.0 Specification*. Van Haren Publishing, 2009. <http://www.opengroup.org/archimate>, <http://www.opengroup.org/bookstore/catalog/c091.htm>,
- [2] The Open Group, *TOGAF™ Version 9*. Van Haren Publishing, 2009. <http://www.opengroup.org/togaf>
- [3] Marc Lankhorst, Hans van Drunen, "Enterprise Architecture Development and Modelling – Combining TOGAF and ArchiMate", *Via Nova Architectura*, 21 March 2007. <http://www.via-nova-architectura.org/magazine/magazine/enterprise-architecture-development-and-mode.html>
- [4] Graham Berrisford, Papers on abstraction etc. in the "Library" at <http://avancier.co.uk>. In particular <http://avancier.co.uk/12%20Abstraction/01%20Abstraction.htm>
- [5] British Computer Society, *Reference model for ISEB certificates in enterprise and solution architecture (v11.3)*, BCS, 2009. <http://www.bcs.org/upload/pdf/reference-model-enterprise-solution-architecture.pdf>

## Copyright

This paper has been edited from conversations between Marc Lankhorst (group leader Service Architectures at Novay and a founder and teacher of ArchiMate® [1]) and Graham Berrisford (a TOGAF™ 9 [2] instructor for *Architecting-the-Enterprise*).

## Creative Commons Attribution-No Derivative Works Licence 2.0

No Derivative Works: You may copy, distribute, display only complete and verbatim copies of this document, not derivative works based upon it.

Attribution: You may copy, distribute and display this copyrighted work only if you clearly credit the authors Graham Berrisford of [Avancier Limited](#) and Marc Lankhorst of [Novay](#) before the start and include this footnote at the end.

For more information about the licence, see <http://creativecommons.org>

ArchiMate® and TOGAF™ are registered trademarks of The Open Group.

Graham Berrisford

[graham.berrisford@architecting-the-enterprise.com](mailto:graham.berrisford@architecting-the-enterprise.com)

Marc Lankhorst

[marc.lankhorst@novay.nl](mailto:marc.lankhorst@novay.nl)