Chapter 21
Next-Generation Enterprise Modeling

Bas van Gils and Hend*erik* A. Proper

Abstract In the Western world, digital has become the new normal, both in our daily lives and at our work. Additionally, Western countries have seen a transition from a goods-oriented economy to a services-oriented economy. Whereas, in the recent past, it was already the case that change was *the only constant*, these intertwined, and mutually amplifying, trends even further increase the pace of change. As a result, enterprises are confronted with a need to transform (continuously) accordingly.

During any enterprise transformation, *coordination* among the key stakeholders and the projects that drive the transformations is essential. A shared understanding, agreement, and commitment are needed on (1) what the overall mission/vision of the enterprise is, (2) the current affairs of the enterprise and any ongoing changes, (3) the current affairs of the context of the enterprise, and (4) what (given the latter) the ideal future affairs of the enterprise are.

Models, and ultimately enterprise (architecture) modeling languages and framework, are generally considered as an effective way to support such (informed) coordination. In the past, different frameworks and languages have been developed to this end, including the ArchiMate language. The latter has evolved to become a widely accepted industry standard.

The objective of this chapter is threefold: (1) we intend to illustrate some of the key challenges which the digital transformation, and the two intertwined trends that drive it, puts on enterprise (architecture) modeling languages, (2) assess to what extent ArchiMate meets these challenges, and (3) draft the outline of a next-generation enterprise (architecture) modeling language (framework) that may be more suited to meet the challenges of these trends.

B. van Gils
Strategy Alliance, Amersfoort, the Netherlands
e-mail: bas.vangils@strategy-alliance.com

H. A. Proper (✉)
Institute of Information Systems Engineering, TU Wien, Vienna, Austria
e-mail: henderik.proper@tuwien.ac.at

H. A. Proper et al. (eds.), *Digital Enterprises*, The Enterprise Engineering Series,
https://doi.org/10.1007/978-3-031-30214-5_21

21.1 Introduction

In the Western world, digital has become the new normal, both in our daily lives and at our work. Computers continue to shrink in size, while their computing powers increase. Furthermore, all these computers have become increasingly interconnected (e.g., the Internet of Things Weber & Weber, 2010). Even the smallest light bulbs have become connected.

It seems as if every aspect of our lives is being impacted on by this trend. Letters are all but replaced by email, and books are digitized, while we track our health through digital/wearable technology (leading to the so-called quantified self Swan, 2012). With the increasing popularity of dating sites, it seems that even our love life is increasingly becoming digital. The same holds for organizations. Whereas IT originally was a mere supportive tool for administrative purposes, it is safe to say that nowadays IT has become an integral part of an organization's primary processes. If the people working in an organization go on strike, then this is likely to lead the organization to come to a *grinding* halt. However, when IT systems fail, most organizations will come to an *abrupt* halt.

According to a recent publication (Brown, 2017), we should even prepare for new forms of diversity in the workforce, where humans should learn to collaborate better with non-humans (e.g., agents, robots, etc.). Human actors, and digital actors, will increasingly work close together.

From a management perspective, this means that considering only the alignment between *business* and *IT* (Henderson & Venkatraman, 1993) is no longer sufficient. The difference between business and IT is increasingly fading; they have been "fused" into one. This has given rise to a wide range of management approaches that are considered to be more "holistic" and consider all aspects of the enterprise; this includes, e.g., information systems architecture (Scheer, 1992; Zachman, 1987) and enterprise architecture (EA) approaches (Pereira & Sousa, 2005; Op 't Land et al., 2008a; Dietz, 2008; van Gils & van Dijk, 2014). Companies, such as Amazon, Airbnb, Uber, Netflix, Spotify, Bitcoin, etc., illustrate how IT and business have indeed become fused. The CEO of a major bank, such as the ING Bank, can even be quoted as stating "We want to be a tech company with a banking license" (Hamers, 2017).

In parallel, Western countries have seen a transition from a goods-oriented economy to a services-oriented economy. Marketing sciences (Vargo & Lusch, 2008; Grönroos & Ravald, 2011; Lusch & Nambisan, 2015; Vargo & Lusch, 2016) suggest that the notion of economic exchange, core to the economy, has shifted from following a goods-dominant logic to a service-dominant logic. While the former focuses on tangible resources to produce goods and embeds value in the transactions of goods, the latter concentrates on intangible resources and the creation of value in relation with customers. It should be noted that a goods-dominant logic is not only adhered to when selling goods. For example, when buying a train ticket, one (might think to) buy a service to get from A to B. At the same time, numerous travelers

have experienced that a train ticket is not a guarantee to get from *A* to *B* at all, let alone on time to make it to an important face-to-face meeting.

Service dominance puts the continuous *value co-creation* between providers and consumers at the core. For instance, in the airline industry, jet turbine manufacturers used to follow a classical goods-dominant logic by selling turbines to airlines. However, since airlines are not interested in *owning* turbines, but rather in the realization of *airtime*, manufacturers nowadays sell airtime to airlines instead of jet turbines. As a result, *value co-creation* is shaping up as a key design concern for modern-day enterprises.

Both of these trends are highly intertwined while also amplifying each other. The digital transformation enables new ways of doing business that also enables more value co-creation, resulting in the development of a plethora of new *digital services*. Conversely, the desire of enterprises to co-create value results in a need for more integrated IT solutions, to, e.g., better understand the precise needs of customers and better integrate them in the design/delivery process (Grönroos & Ravald, 2011).

Whereas, in the recent past, it was already the case that change was *the only constant*, the combination of these trends even further increases the pace of change. Ever since the Industrial Revolution, change has often been driven by the introduction of new technology. It seems that the organization that is best at levering technology wins in the marketplace—meaning that keeping up (or even ahead) of developments has become a crucial capability for modern organizations. The plethora of changes that the digital transformation has brought about, and the many more that we are not even aware of yet or have not even been thought of yet, provides organizations with deep and fundamental challenges. How to excel as an organization, while everything is changing constantly? There are hardly any securities left; traditional business models are continuously challenged by digitally inspired and empowered startups.

We consider the trends of *business-IT fusion* and the *shift to value co-creation*, as being the key challenges to enterprises (be they companies, governmental agencies, or organizations) which aim to thrive (or at least survive) in the digital transformation of society.

There are no simple answers to these challenges: a truly wicked problem (Camillus, 2008). We observe how approaches for *digital transformation* increasingly gain popularity, in order to manage the complexity that arises around digitization and the increased speed of change (Gouillart & Kelly, 1995; Rouse & Baba, 2006; Berman, 2012; Westerman et al., 2014). In line with the definitions provided in Chap. 1, we consider transformation to be "the coordinated effort to change the architecture of an enterprise" and digital transformation to be "a transformation of the enterprise with a major impact on its digital resources."

Key in the latter definition is the phrase *coordinated effort*: the actors that make up the enterprise coordinate their efforts not only to fulfil the goals of the enterprise but also coordinate their efforts in ongoing, deliberate change initiatives. During an enterprise transformation, *coordination* (Proper et al., 2018c) among the key stakeholders and the projects that drive the transformations is indeed essential. A shared understanding, agreement, and commitment are needed on (1) what the

overall mission/vision of the enterprise is, (2) the current affairs of the enterprise and any ongoing changes, (3) the current affairs of the context of the enterprise, and (4) what (given the latter) the ideal future affairs of the enterprise are. Borrowing the terminology from architecture frameworks such as TOGAF (The Open Group, 2011), this refers to the development of a shared vision, a baseline architecture, and a target architecture, respectively.

Given the speed of change, it remains to be seen whether a true "target architecture" can be developed in light of the fact that it is hard to make predictions, especially about the future. As such, it may be wiser to depart from using the term "target architecture" and refer to it in more open terms, such as "directional architecture," to clarify that it expresses a desired direction of development, rather than a specific target. This would also require a departure from the traditional style of defining a target architecture in terms of a rather "instructive" style in terms of typical "boxes and lines" diagrams toward a more "directional"/"regulative" approach using, e.g., (normative) architecture principles (Greefhorst & Proper, 2011a).

Models, and ultimately enterprise (architecture) modeling languages and framework, are generally considered as an effective way to support such (informed) coordination. Many languages and frameworks have indeed been suggested as a way to create and capture a shared understanding of the desired future affairs. Examples include DEMO (van Reijswoud et al., 1999; Dietz & Hoogervorst, 2007), BPMN (Freund & Rücker, 2012), UML (Object Management Group, 2010), ArchiMate (Band et al., 2016; Lankhorst et al., 2017), 4EM (Sandkuhl et al., 2014), and MERODE (Snoeck, 2014). The latter approaches are applicable in the context of capturing an enterprise's current affairs in terms of its baseline architecture. However, as argued above, for an (open!) future-oriented "directional architecture," these "boxes and lines"-based approaches may have to be complemented with a "directional"/"regulative" approach using, e.g., (normative) architecture principles (Greefhorst & Proper, 2011a).

It appears that ArchiMate is rapidly becoming the industry standard for enterprise architecture modeling[1] and has, as such, a key role to play in the coordination of (Proper et al., 2018c) enterprise transformations. Based on our experience from research and practical work in the field, however, we hypothesize that there are serious challenges with the existing ArchiMate language in light of the needs to support digital transformation efforts (van Gils & Proper, 2018; Proper et al., 2020; Proper, 2020; Proper & van Gils, 2019). As we will see below, these challenges are not only limited to the "instructive" vs. "directional" issue.

The objective of this chapter is therefore threefold: (1) we intend to illustrate some of the challenges that the digital transformation puts on enterprise architecture modeling languages, (2) assess to what extent ArchiMate meets these challenges,

[1] The support for this claim lies in the steady growth of the number of certified professionals http://archimate-cert.opengroup.org/certified-individuals as well as the popularity of the ArchiMate topic on Google trends https://trends.google.com/trends/explore?date=all&q=archimate.

and (3) draft the outline of a next-generation architecture modeling language (infrastructure) that may be more suited to meet the challenges of the digital transformation. In line with this, the remainder of this chapter is structured as follows. We start, in Sect. 21.2, by more closely investigating the challenges that the digital transformation may have on enterprise architecture and the modeling languages used. Based on this, we will then, in Sect. 21.3, present a critical reflection of the suitability of the ArchiMate language in light of these findings. This is followed, in Sect. 21.4, by an outline of a possible "digital transformation-ready" next-generation architecture modeling language (infrastructure). We end this chapter, in Sect. 21.5, with conclusions and directions for future research.

Throughout this chapter, we will use small examples to illustrate key points. Most of these examples derive from real-world projects that have been conducted over the last few years.

21.2 Challenges for Enterprise Modeling

The aim of this section is to identify some of the key challenges on enterprise (architecture) modeling in the context of digital transformation and the increasing focus on value co-creation. The resulting challenges will be used, in Sect. 21.3, as a base to critically reflect on the extent to which ArchiMate already meets the challenges brought forward by these fundamental trends, as well as reflect on possible modifications to ArchiMate to make it better meet the challenges (see Sect. 21.4).

We have grouped the challenges in three classes. First, we discuss challenges pertaining to the expressiveness of the modeling language used in light of the digital transformation and value co-creation. The digital transformation and value co-creation trends push for further specialization and domain specificity of modeling languages. Therefore, the second class of challenges zooms in on the need to be able to manage the resulting spectrum of modeling concepts. The final class of challenges concerns the earlier made observation that the digital transformation fuels the speed of change in organizations and their enterprises.

21.2.1 Expressiveness of the Modeling Language

21.2.1.1 Objects Can Be Operand and Operant

Objects (including humans) in the world around us can play different roles. Sometimes, they play an active role, in the sense that they become the operant actor, which (co-)enacts a certain activity. They may even become the actor bearing the social responsibility for the enactment of such an activity (Dietz et al., 2013). Objects may also play a passive role, in which case they can actually be the

operand/subject of an activity. Key is that it is natural for the same objects to play different roles in the course of time or even in parallel.

In traditional views on enterprise architecture, it was more or less assumed that objects were either passive (operand) or active (operant) for their entire life. One would certainly not mix these types of roles. This simplification might have indeed worked in former times. However, in the context of the digital transformation, this simplification becomes increasingly difficult to uphold.

Objects, in particular the digital ones, are created and manipulated by other (human and/or digital) objects. In the digital world, objects can be both operand and operant, even at the same time. An enterprise architecture modeling language used in digital transformations should therefore be able support this plurality of the roles played by objects:

Challenge 1 *Objects should be allowed to play operand and operant roles.*

21.2.1.2 Information Versus Reality

Digital transformations also result in an increased reliance on the quality of (digitally represented) information in terms of the correctness at which it represents the world around us. As a result, it becomes increasingly important to remain aware of, and thus explicitly capture, the distinction between elements in the real world and the information that *stands model* for those real-world elements.

Enterprise (architecture) modeling languages should, therefore, also clearly reflect such a distinction: for example, in terms of a clear distinction between *business objects* as they exist in the real world and *business information objects* that represent information *about* the former objects. An example of an architecture framework which already supports such a distinction is the Integrated Architecture Framework (Wout et al., 2010). This results in the following challenge for enterprise modeling languages:

Challenge 2 *Clear separation between objects that represent"things" in the real world and objects representing information about the real world.*

21.2.1.3 Natural Duality of Human and Digital Actors

As also discussed in the introduction, according to a recent publication (Brown, 2017), a consequence of the digital transformation is that we should prepare for new forms of diversity in the workforce, where humans should learn to collaborate closely with digital actors (e.g., agents, robots, etc.). In line with (Dietz et al., 2013), we take the position that the social responsibility of activities should remain with human and/or organizational entities. We are, for example, not (yet) expecting that robots can be taken to court, to account for their actions, and possibly be punished when breaking societal rules. Underlying this is the well-known question: *When an autonomous car causes an accident, who is responsible?*

Modern-day enterprise modeling languages should, therefore, be able to deal more naturally with the duality of human and digital actors while making explicitly clear where the ultimate social responsibility and accountability of the actions by these actors lie:

Challenge 3 *Ability to deal naturally with the duality of human and digital actors.*

21.2.1.4 Identification Management

A key aspect in traditional (conceptual) data modeling is the notion of *unique identification*: in other words, the ability to specify how objects in the real world can be distinguished from one another.

The ability to uniquely identify the (passive and/or active, operand/operant) objects around is indeed quite convenient, even though not all applications will need it. Depending on the application context, we may need unique identification. In some cases, it might even be illegal, e.g., due to privacy considerations, to have such a unique identification. For example: Is there a need to uniquely identify all water molecules in the stream of water coming from a well with mineral water? Probably not. Is there a need to identify each individual bottle of water filled with water from this well? Probably yes, as well as the date when it was filled. Is there a need to identify each individual traveler on a public transport system? Would probably be useful for optimization purposes, as well as monitoring possible "terrorist" activities. Is it allowed/desirable from a privacy perspective? Probably not.

At the same time, even when a unique identification mechanism is available and is allowed to be used, there may be limits regarding its completeness and uniqueness. In a business network involving multiple partners, one may have to use multiple, partially overlapping, identification mechanisms. Even more, one may not have control over the creation of objects, which may (accidentally or maliciously) end up having the same properties as used in the identification.

For enterprise modeling languages, this makes it important to be able to specify if objects can, should, and/or are allowed to be uniquely identified and, if so, to what extent this unique identification can indeed be assumed to cover the entire (possible) population of such objects:

Challenge 4 *Ability to specify if objects can, should, and/or are allowed to be uniquely identified.*

21.2.1.5 Optional Modalities on Relationships

Most enterprise modeling languages do not allow for detailed modalities (mandatory, optional, one-to-one, one-to-many, etc.) on relationships. In general, this has been a deliberate choice by the language designers. In practice, however, this decision is challenged. It has been debated extensively—for example, in the

LinkedIn group for ArchiMate as well as during training and coaching sessions—how useful it would be to be able to specify modalities, in particular, in the context of privacy and security, two concerns that become even more important in digital transformation(s).

One may argue that such modality rules are "too detailed" to be included at an architecture level. At the same time, there are many cases where there is a need to specify (even at an architecture level) the rules governing relationships in more detail. A typical example would be the four-eyes principle, where two roles must be fulfilled when performing a certain task. It is expected that such modeling constructs will be needed frequently, in the context of privacy and security. We suggest that, although one should not categorically require architecture models to use modalities on relationships, this should be addable when needed:

Challenge 5 *Ability to specify modalities on relationships.*

21.2.1.6 Orientation Toward Value Co-creation

Western countries have witnessed a transition from a goods-oriented economy to a services-oriented economy. Digital transformation triggers the development of a plethora of new *digital services*, even further boosting the dominance of services in Western economies.

Several studies (Vargo & Lusch, 2004; Lusch & Vargo, 2006; Vargo & Lusch, 2008; Maglio et al., 2009; Grönroos & Ravald, 2011) observe a fundamental paradigm shift from, what they call, a goods-dominant logic to a service-dominant logic. While the former focuses on the production of goods, the latter concentrates on the delivery of services using resources and/or goods in doing so. These studies motivate this shift by observing that it is ultimately the customer who attributes value to a good or a service. Goods and services, "at rest," only have a potential value to a customer. The actual value is experienced when the resources/goods are actually *used* by the customer to some purpose.

In parallel to the shift from a goods-dominant logic to a service-dominant logic, one can observe a growing awareness that a conventional enterprise-centric (inside-out) view of value creation is now being challenged by a newer customer-centric (outside-in) view of value creation (Prahalad & Ramaswamy, 2000; Priem, 2007; Lepak et al., 2007; Priem et al., 2013). This leads to the perspective that value results by way of a process of *co*-creation between producer and consumer, involving the integration of their resources (Vargo & Lusch, 2008).

To achieve strategic advantage, service-providing enterprises must be able to co-create value for their customers, at a higher level of quality than the competition does (Bettencourt et al., 2014). This also entails a need for enterprises to broaden the scope of their enterprise architecture, more specifically, from a focus on the design of efficient, reliable, and flexible (IT-supported) business processes to a broadened one, with a more prominent place for the design of value co-creation with partners and customers.

The *digital transformation* not only brings about a new wave of digital services, but it also acts as an enabler that allows providers of goods and service to better optimize the co-creation of value with their customers: for example, by being able to (1) more swiftly create, and manage, *on-the-fly* business processes and (2) tune/customize their products and services to the needs of specific users (in their context of use) and (3) based on detailed (digital) profiles of the needs, preferences, and habits of the users.

As a consequence, enterprise architecture modeling languages need to include constructs to explicitly express the (potential) value(s) of products and services to customers, in particular in terms of *value in use* and *resource integration*, and how this results in *value co-creation* between providers and consumers of services.

Challenge 6 *Ability to capture (potential) value(s) of products and services and how this results in value co-creation between providers and consumers of services by way of resource integration.*

21.2.1.7 Implementation and Design Choice Awareness

Given the speed of technological developments that drive the digital transformation, it is increasingly important for organizations to be aware of the essential design choices shaping the essence of their business[2] activities, as well as choices with regard to their implementation by means of different platforms and technologies. The latter includes choices such as the use of (business process) outsourcing, software platforms, hardware platforms, cloud computing, division of labor between human and computer-based actors (also see Challenge 3), etc.

For enterprise (architecture) modeling languages, this means that one should be able to express the design of the enterprise (including its use of information technology) at different levels of specificity with regard to implementation decisions, as well as enable the capturing of the associated design decisions and their motivation (Plataniotis et al., 2015b, 2014b).

Challenge 7 *Express the design of the enterprise at different levels of specificity with regard to implementation decisions.*

Challenge 8 *Capture design decisions and their motivation.*

[2] When using the word "business," we do so in the sense of "a particular field of endeavour" (Meriam–Webster, 2003); i.e., we are specifically not only referring to "commercial businesses."

21.2.2 Managing the Spectrum of Modeling Concepts

21.2.2.1 Managing the Set of Modeling Concepts

An enterprise (architecture) modeling language typically features a rich set of modeling concepts. As a natural consequence of the use of such a language, and as a corollary to the law of entropy, there is a tendency to continue adding concepts to modeling languages (Bjeković et al., 2014), in particular when such a language has the status of being a standard.

Digital transformation, due to its deep impact and multifacetedness, is likely to further fuel the entropic forces, likely leading to a further increase in the number of modeling concepts. Some of the challenges listed above actually also point toward a desire to extend existing modeling languages. Next to that, specific concerns, such as security, privacy, value co-creation, etc., are likely to play a stronger role in digital transformation and thus also trigger a need for dedicated modeling concepts (Bjeković et al., 2014).

At the same time, an ever-increasing set of modeling concepts will lead to a modeling language that will be hard to learn (Moody, 2009; Krogstie et al., 1995) while also endangering the overall consistency of the set of modeling concepts.

This leads to the following challenge on enterprise (architecture) modeling language (frameworks):

Challenge 9 *A way to manage the set of modeling concepts, balancing the needs of domain, and purpose, specificity, the need for standardization, and comprehensibility of the modeling language.*

21.2.2.2 Consistent Abstraction Layer Structures

Enterprise architecture modeling languages typically involve different abstraction layers. Examples include the business, application, and technology layer as used in ArchiMate (Lankhorst et al., 2017); the essential and implementation layer as suggested by Enterprise Ontology (Dietz & Hoogervorst, 2007); the function and construction perspective as suggested by the same; the business, information systems, and technology layer from TOGAF (The Open Group, 2011); the business, information, information systems, and technology infrastructure columns from IAF (Wout et al., 2010); as well as the conceptual, logical, and physical layers of the same.

In line with the earlier discussion on implementation and design choice awareness (Challenges 7 and 8), using such abstraction layers for digital transformations is indeed wise. At the same time, we observe in practice (both in using such frameworks and teaching about them) that confusion about the precise scoping of the used abstractions exists. In this regard, one can even distinguish changes in the interpretation of the business, application, and technology layer from ArchiMate as intended originally (Lankhorst et al., 2017), where the technology layer was

purely intended as the (IT) technological *infrastructure*, to the current interpretation, where it has evolved to include the entire (IT) technological implementation (Band et al., 2016). As we will discuss in Sect. 21.3, this also leads to further challenges regarding relationships between layers in the case of ArchiMate.

In general, one could say that abstraction layers (even in multiple dimensions, as suggested by the IAF (Wout et al., 2010) and Zachman (Zachman, 1987) frameworks) result from the *design philosophy* underlying the specific framework. In this chapter, we do not aim to take a specific position with regard the question of which *design philosophy* would be best. However, we do argue, in particular when considering the challenges of digital transformations, that it is important that the layer structure must use clear and consistent abstractions.

For enterprise (architecture) modeling language (frameworks), this leads to the following challenge:

Challenge 10 *Provide a structure that allows to consistently use abstractions across relevant aspects of the enterprise.*

21.2.2.3 Grounding Modeling

Enterprise (architecture) models play an increasingly important role. When developing/evolving an enterprise, models are used to capture the current affairs, as well as articulate different possible future affairs. Even more, nowadays, it is quite common that models are even part of the "running system," in the sense that they are an artifact that drives/guides day-to-day activities. This includes workflow models, business rule sets, etc.

This makes it important that enterprise models also capture their meaning[3] in a way that is understandable to the model's audience. We therefore posit that a conceptual model should be grounded in the terminology as it is actually *used* (naturally) by the people involved in/with the modeled domain. We see this as a key enabler for the transferability of models across time and among people, in particular in situations where the model needs to act as a *boundary object* (Abraham et al., 2013a).

Most existing enterprise modeling languages (e.g., process models, goal models, actor models, value models, architectural models, etc.) only offer a "boxes and lines"-based representation that only provide a limited linkage to the (natural) language as used by the model's audience. In general, the only link in this regard are the names used to label the "boxes." Relationships are replaced by generic graphical representations in terms of arrows and lines capturing relations such as "assigned to," "part of," "realizes," "aggregates," and "triggers."

[3] In principle, we would prefer to use the word "semantics" here. However, since the word "semantics," in our computer science-oriented community, tends to be equated to only mean "formal semantics," we will use the word *meaning*.

While these abstract, and more compact, notations of purpose/domain-specific modeling languages enable a more compact representation of models, they offer no means to provide a "drill down" to an underlying grounding in terms of, e.g., well-verbalized fact types that capture, and honor, the original natural (language) nuances (Hoppenbrouwers et al., 2019). They leave no room for situation-specific nuance or more explicit capturing of the meaning of the models in a way that is understandable to the model's audience (beyond engineers). The challenge therefore is:

Challenge 11 *How to ground enterprise models in terms of natural language like verbalizations, without losing the advantages of having compact notations (as well).*

21.2.3 Enabling a Regulative Perspective

As mentioned in the introduction of this chapter, an often used idiom is that *change is the only constant*, while the digital transformation results in a further increase in the *rate* of this "constant change." In light of such rapid changes, the notion of the traditional baseline architecture has its difficulties.

As a result of these rapid changes, the enterprise is in a constant motion, which means that the baseline is not simply a "state," but rather a "vector" (Proper & Lankhorst, 2014). Hence, it is better to speak about capturing "current affairs," which includes past, and present, change trends, of the enterprise and its environment. As a consequence, the traditional concept of a "target architecture" needs to be reconsidered as well. Of course, in terms of TOGAF and ArchiMate, this concept has been extended toward a multi-stage version in terms of "plateaus" toward the future. Nevertheless, it remains to be seen how specific such plateaus/target architectures can be developed in light of the fact that it is hard to make predications, especially about the future.

As such, it may be wiser to depart from using the term "target" and refer to it in more open terms, such as "directional," to clarify that it expresses a desired direction of development, rather than a specific target. This would also require a departure from the traditional style of defining a target architecture (or plateaus) in terms of a rather "instructive" style in terms of typical "boxes and lines" diagrams toward a more "directional/"regulative" approach using, e.g., (normative) architecture principles (Greefhorst & Proper, 2011a).

When considering the motivation extension of the latest ArchiMate (Band et al., 2016) version, and the increasing awareness of the role of architecture principles, it seems sensible to identify three levels of enterprise architecture modeling:

Desires-oriented dealing with goals of stakeholders and their ensuing requirements. Model artifacts from this perspective should be owned (content-wise) by

the stakeholders and should be formatted in terms of what the stakeholders want to do and achieve.

Constraints-oriented dealing with (normative) architecture principles, regulations, constraints, etc., limiting the design space. Model artifacts from this perspective should be owned by both stakeholders and architects/designers and form a translation from the stakeholders' desires to consequences/constraints toward the actual design, without making concrete/specific design decisions yet.

Construction-oriented dealing with specific "instructions" on how (parts of) the enterprise should actually be constructed (and implemented). This involves the typical boxes and lines diagrams. Ownership lies with the architects/designers, and design decisions should of course comply to what has been stated from the constraints- and desires-oriented perspectives.

The resulting challenge for modeling languages is:

Challenge 12 *How to balance a desires-, a constraints-, and a construction-oriented perspectives on an enterprise, in light of constant change.*

21.3 ArchiMate's Readiness for the New Enterprise Modeling Challenges

In this section, we start by providing a high-level introduction to the current version of the ArchiMate language, including its development history.[4] We then continue with a discussion to what extent the current version of ArchiMate meets the challenges of digital transformations, as identified in Sect. 21.2. This provides the context for the discussion in the next section, where we propose improvements of the language.

21.3.1 The Development of the ArchiMate Language

In line with (Hoppenbrouwers, 2000, 2003; Hoppenbrouwers et al., 2005b; Frank, 2013, 2011), we argue that a modeling language, and designed languages in general, should reflect the actual (intended) use of the language.

A purposely developed language, such as an enterprise architecture modeling language, is fundamentally an artifact in the design science research (Hevner et al., 2004; van Aken, 2004; Peffers et al., 2007) sense. A design science process typically follows an (iterative) design process in terms of requirements elicitation, design, and development, followed by some form of testing/evaluation, while also allowing for

[4] At the time of writing, ArchiMate 3.0.1 is available online.

possible iterations. The process for the development of domain-specific modeling languages as suggested by, e.g., (Frank, 2013) follows a similar pattern.

In line with this, it would be appropriate to use the design science research process as suggested in, e.g., (Peffers et al., 2007) in the development of a language such as ArchiMate. Even though at the time of the development of the initial versions of the ArchiMate language (early 2000s), design science research had not yet fully emerged (Hevner et al., 2004), the development of ArchiMate did follow a basic design process. ArchiMate's development started with the establishment of a set of initial requirements (Bosma et al., 2002; Jonkers et al., 2003). Using further input from enterprise architects from industrial partners involved in the research project, the architecture of the ArchiMate language was then developed (Lankhorst et al., 2010), and the final design of (the initial version of) the ArchiMate language was created.

Since then, the ArchiMate language has gone through several iterations (Lankhorst et al., 2017; Band et al., 2016). Based on real-world use of the language, several refinements and improvements were made. In addition, the tighter integration into TOGAF (The Open Group, 2011) also resulted in additional extensions. As a result, the language has also grown considerably in terms of the included concepts.

21.3.2 Overview of the ArchiMate Language

ArchiMate is a dedicated language for representing (enterprise) architecture models that was originally developed by a consortium of organizations in the Netherlands after which it was adopted by The Open Group as an (open) standard (Lankhorst et al., 2017; Band et al., 2016). Its adoption has grown rapidly, both in terms of the *users* of the language and the *vendors* that deliver software solutions based on this language. A full discussion of ArchiMate (Lankhorst et al., 2017) is beyond the scope of this chapter. However, for purposes of our analysis, we will present a rough outline of the structure of the language.

The current version of the language supports five layers (strategy, business, application, technology, and physical) and four aspects (active structure, passive structure, behavior, and motivation). The core of the framework—and focus of this discussion—consists of the business/application/technology layer, and all aspects save the motivation aspect. The rationale for leaving out motivation and implementation aspects lies in the fact that these are crucial for the architecture process, but are not used to describe the actual architecture of the enterprise. The layers in the core have the same generic meta-model which is shown in Fig. 21.1. The later meta-model is also contained in the specification of the ArchiMate standard (Band et al., 2016), albeit with some additional details.

Services are used as a decoupling mechanism. They are used to specify what an active structure element exposes to its environment and hide the complexity of how the services are realized. Services can be used both within a layer (e.g., a department

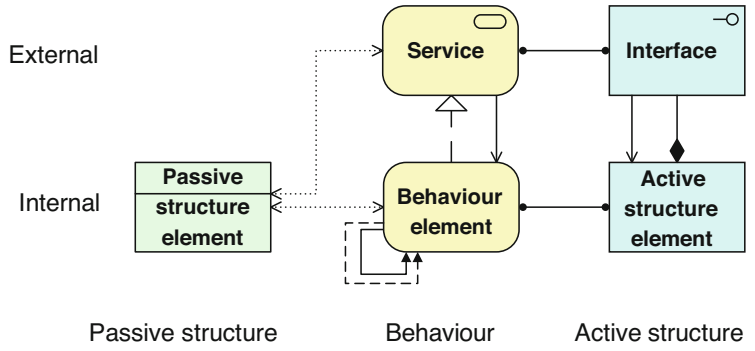

Fig. 21.1 Generic ArchiMate meta-model, adapted from Lankhorst et al. (2017); ©2017 Springer-Verlag Berlin Heidelberg; reprinted with permission

offering services to another or an application offering services to another) and across layers (e.g., which processes are served by an application service).

A second abstraction mechanism in the language is the *specialization relation*, which is to be interpreted as "is a kind off." Some languages, such as ORM and UML, distinguish between (a) specialization, (b) generalization, and (c) type/instance relations (Halpin, 2001; ter Hofstede & van der Weide, 1993; Hay, 2011; Fowler, 2004). In ArchiMate, these are all captured by the same specialization relation. Using this relation, it is possible to relate generic architecture constructs (e.g., a process pattern) into more specific manifestations (e.g., distinguishing between the regular manifestation of the process or the manifestation that is followed during times of crisis).

An abstraction mechanism that was introduced in version 3 of the ArchiMate language is the use of *grouping*. Previously, the grouping was a visual construct only, which was intended to show on a view which concepts "belong together" for some reason. Since ArchiMate version 3, the intended meaning is more rigorous: the grouping is said to aggregate the concepts that are in it and thus functions as a semantic whole. Groupings may be related to other concepts (including other groupings). This makes it particularly well suited to use the grouping as a form of *building blocks* along the lines of the TOGAF standard (e.g., The Open Group, 2011, Chapter 37).

The last mechanism that is relevant to our discussion here is the notion of *cross-layer dependencies* (Band et al., 2016, Chapter 12). The general idea is that elements from one layer can be connected to elements of other layers using the *serves* (previously: used-by) relation or the *realization* relation. Through this mechanism, we can specify, for example, that a business process is realized by an application process. Along the same lines, it allows us to specify that a group of elements (i.e., a building block) is realized by another group of elements (another building block).

Putting this all together leads to the example of Fig. 21.2 that illustrates the concepts explained in this section. Starting at the top left, we see an architecture

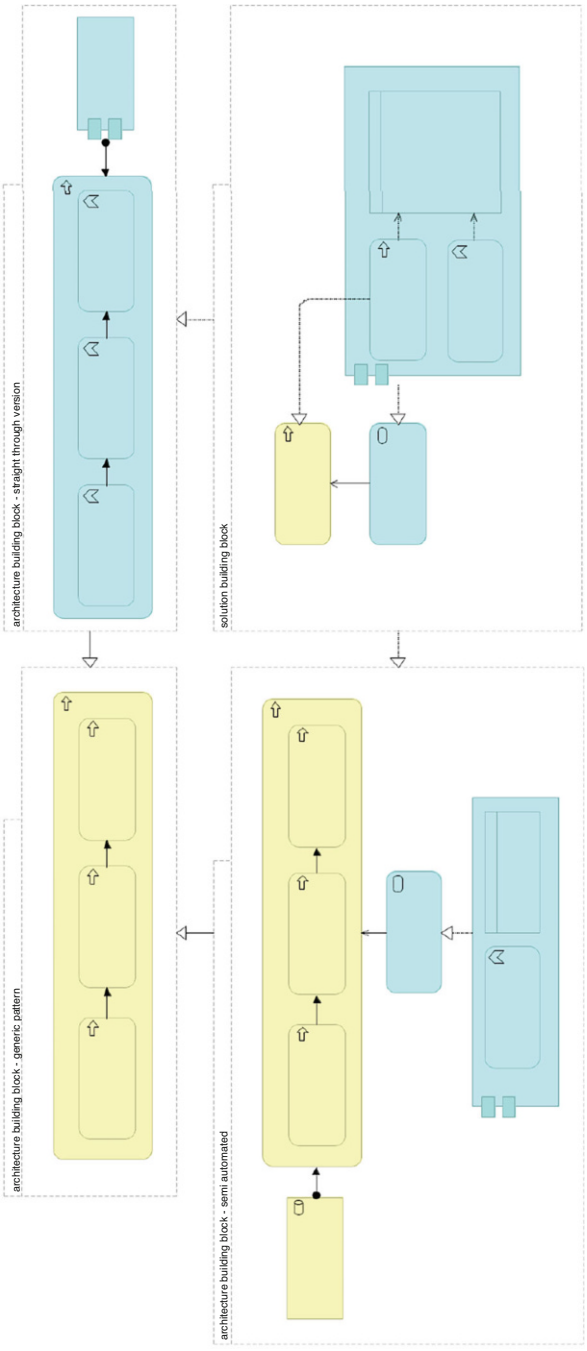

Fig. 21.2 Abstraction mechanisms in ArchiMate, version 3

building block that specifies a process pattern of three steps. Using the specialization relation, we see two more specific building blocks at the bottom left (semi-automated) and top right (straight through). Using the realization relation, we see that these two together are implemented through a solution building block that ties objects together.

21.3.3 Analysis in Light of the Identified Challenges

In this section, we briefly touch upon each of the identified challenges before presenting a short reflection on the current "digital transformation readiness" of ArchiMate.

Challenge 1 *Objects should be allowed to play operand and operant roles.*

The current ArchiMate language does not deal with this well due to the strict distinction between *active* and *passive* structure elements. This challenge lies at the heart of the ArchiMate language.

Challenge 2 *Clear separation between objects that represent "things" in the real world and objects representing information about the real world.*

This challenge refers to the passive structure elements in ArchiMate. Currently, there is no clear distinction between the two types of objects, other than the observation that data objects/artifacts presumably are about the bits and bytes that represent information. The ArchiMate specification does suggest that the *business object* concept can be specialized but in the default language this has not been done. What the specification does not mention is that additional relations may also be required in order to present that informational business object *A is about* real-world business object *B*.

Challenge 3 *Ability to deal naturally with the duality of human and digital actors.*

Here, the ArchiMate language, through its layering, does provide a fair attempt at tackling this challenge since there are different concepts for, e.g., actor, information system, and node. Some interesting challenges remain, however. First of all, only (business) actors can be assigned a role in behavior; other structure elements cannot. A second mismatch lies in the fact that *collaborations* in ArchiMate can only be composed of structure elements from the same layer. This prevents us from specifying that a human actor and computer actor collaborate to achieve a certain task.

Challenge 4 *Ability to specify if objects can, should, and/or are allowed to be uniquely identified.*

In ArchiMate, concepts are essentially "types," representing the "instances" in the real world. The ArchiMate concepts have a name to tell one apart from the other. There is no mechanism to specify how the "instances" should be told apart.

Challenge 5 *Ability to specify modalities on relationships.*

For this challenge, we can be short: ArchiMate has no support for this. Objects are either related, or they are not.

Challenge 6 *Ability to capture (potential) value(s) of products and services and how this results in value co-creation between providers and consumers of services by way of resource integration.*

Evaluation of the current ArchiMate language against this challenge is somewhat tricky. This is because the language does have the *value* concept, and it seems possible to model value *co*-creation by using the collaboration/interaction concepts. However, as discussed in, e.g., (Razo-Zapata et al., 2017, 2018), representing value co-creation (Lusch & Vargo, 2006; Lusch & Nambisan, 2015), scenarios require more dedicated modeling constructs.

Challenge 7 *Express the design of the enterprise at different levels of specificity with regard to implementation decisions.*

Challenge 8 *Provide a structure that allows to consistently use abstractions across relevant aspects of the enterprise.*

These challenges are closely related. The latest version of ArchiMate does indeed provide some rudimentary support to tackle these challenges through the *grouping* mechanism. It is now possible to express the fact that one group of concepts (together) realizes another group of concepts. This allows the modeler to work from a big picture level to a more detailed level, as well as from a functional level to a more construction-oriented level.

Challenge 9 *Capture design decisions and their motivation.*

There is limited support in ArchiMate to address this challenge. We would argue that using (a specialization of) the requirement concept could potentially work, but is far from elegant. As an example of a more elaborate approach to the motivation of design decisions, consider the work reported in (Plataniotis et al., 2014a, 2015a).

Challenge 10 *A way to manage the set of modeling concepts, balancing the needs of domain, and purpose, specificity, the need for standardization, and comprehensibility of the modeling language.*

Potentially, this challenge is addressed partially by means of the *extension mechanisms* to tailor the language to local needs while keeping the core of the language compact. This can be done by specializing existing concepts or by adding properties to existing concepts. While it is good that the language indeed supports this, being able to reuse extensions across toolsets of different vendors is not straightforward. Even more, the extension mechanism is not really positioned as a key feature in the standard either.

In addition, recent extensions of the language have been captured as so-called extensions, such as the *motivation* extension and the *implementation and migration* extension.

Challenge 11 *How to ground enterprise models in terms of natural language like verbalizations, without losing the advantages of having compact notations (as well).*

ArchiMate has no support for this, neither in the language nor the modeling process. Even more, there is no predefined modeling procedure such as ORM's CSDP (Halpin & Morgan, 2008), leaving (in particular novice modelers) to guess how to master ArchiMate's elaborate set of modeling concepts (Proper et al., 2018b).

Challenge 12 *How to balance a desires-, a constraints-, and a construction-oriented perspectives on an enterprise, in light of constant change.*

Support for this challenge is limited. ArchiMate does have the ability to model different plateaus—which relates to different points in time—and it allows the modeler to link concepts to motivational elements of key stakeholders. Full support for balancing the different perspectives, however, is lacking.

21.3.4 Reflection

Building a modeling language that supports modelers to consistently solve challenges, and solve them well, is a difficult task indeed. After listing modeling challenges and evaluating the current version of ArchiMate against these challenges, we conclude that—even though ArchiMate has been around for a while and has a strong conceptual framework—its support for the challenges of digital transformation is fair at best. At first glance, it appears that several of the constructs in the language need reconsideration in order to meet the listed challenges. How this could play out is the topic of the next section.

21.4 Next-Generation Architecture Modeling Language

A full (re)design of the ArchiMate language is certainly beyond the scope of this chapter. Instead, we provide (motivated) recommendations that could overcome the challenges as discussed above.

21.4.1 Modular Language Design

modular As discussed in Sect. 21.3.1, the set of modeling constructs within the ArchiMate language has grown considerably. Furthermore, as discussed in Sect. 21.3.3, the use of ArchiMate's *extension mechanism* indeed provides a good starting point to better manage the resulting set of concepts. The positioning of

recent additions to the language as *extensions*, such as the *motivation* extension and the *implementation and migration* extension, indeed underlines this.

In general, we suggest that modeling language standards should focus primarily on providing a generic core of well-defined, and possibly even formalized (ter Hofstede & Proper, 1998), modeling concepts. On top of this core, one could then define refinement mechanisms that can be used to extend/tailor the core to the needs at hand. This may involve both specializations of the core concepts and the introduction of different abstraction layers.

In addition, a library of (meta-model) *modules* can be defined, which could potentially even be (re)used across different language cores. For example, a generic *motivation* module could be shared between ArchiMate, DEMO (Dietz, 2006), and BPMN (OMG, 2011). At the same time, a modular approach would also enable more flexibility in terms of, e.g., the layering of abstractions. For instance, ArchiMate (and TOGAF) have a "hard-wired" layering of the so-called business-to-IT stack involving *business*, *application*, and *technology*. Other frameworks, such as Capgemini's IAF (Wout et al., 2010), have a more refined layering "hard-wired" into their structure, involving *business*, *(business) information (systems)*, *(computerized) information systems*, and *technology*.

When looking at the original architecture of the ArchiMate language as reported in (Lankhorst et al., 2010), there are indeed ample opportunities for further modularization of the ArchiMate language. For example, following (Lankhorst et al., 2010), and as also confirmed by (Band et al., 2016), the core of the language is formed by five key generic "active systems" modeling concepts: *objects*, *service*, *internal behavior*, *interface*, and *internal structure*. All other concepts are explicitly derived from these in terms of specializations (Lankhorst et al., 2010).

We argue that this specialization hierarchy has been left too implicit for far too long and that an explicit re-factoring of the current ArchiMate language based on this hierarchy is long overdue, more specifically, using language construction mechanisms such as:

Meta-model modules: that allows for the expression of specific language functionalities, such as *motivation* and *migration planning*. Each of such modules should include the identification of an *interface* by which it can be connected to other language modules.

For example, a *motivation* module could feature a generic *design element* as a placeholder for the elements of design for which a motivation needs to be provided. The motivation module can then be used to motivate designs in different languages.

Layering mechanisms: involving a set of *meta-model modules* used to connect multiple layers. For instance, ArchiMate's business/application/technology layers are typically connected by means of services-calls and realizations-relations. Making these into more explicit modules allows users to adapt the layering to the needs of their organization. For instance, as mentioned above, IAF (Wout et al., 2010) suggests a more refined layering in terms of business/information/(computerized) information systems/technology.

Concept specialization: in terms of, e.g., the existing extension mechanism.

21.4.2 Grounding Enterprise Modeling

Challenge 5 suggests that enterprise models should (unless they only serve a temporary "throw away" purpose) include a precise definition of the meaning[5] of the concepts used in the model. We see this as a key enabler for the transferability of models across time and among people (Proper et al., 2004; Hoppenbrouwers et al., 2005a), in particular in situations where the model needs to act as a *boundary object* (Abraham et al., 2013a).

In line with this, we posit that, to ensure that a model is understandable to its audience, it should be grounded on an (underlying) fact-based model involving verbalizations using the terminology as it is actually *used* (naturally) by the people involved in/with the modeled domain (Hoppenbrouwers et al., 2019).

As exemplified in (van Bommel et al., 2007a,b; Tulinayo et al., 2013; Proper et al., 2018b), fact-based models can be used to ground enterprise models that are expressed in languages, such as ArchiMate, DEMO (Dietz, 2006), system dynamics (Rouwette & Vennix, 2006), and BPMN (OMG, 2011), and architecture principles (Greefhorst & Proper, 2011a), in terms of underlying fact models. In doing so, the basic idea is to:

1. Consider an enterprise (be it an existing one or an imagined future one), including all its aspects (in particular, the "business-to-IT stack"), as an active system (of systems)
2. describe the structures and behavior of this active system in terms of (observable) facts

The latter is fully aligned to ArchiMate's roots on natural language structures involving agens (*active structure*), patiens (*passive structure*), and verb (*behavior*).

When indeed observing an active system in terms of fact (types), one essentially creates the (structure of a) fact-based logbook of what "happens" in the active system (van Bommel et al., 1996).

Even though we strongly suggest to remain close to the terminology as it is actually *used* (naturally) by the people involved in/with the modeled domain, we do see the potential benefits of providing guidance in structuring/refining this terminology based on, e.g., foundational ontologies (Guizzardi, 2006).

Grounding ArchiMate on fact-based models would also lead to a natural way to deal with Challenge 1, i.e., the challenge that objects should be allowed to play operand and operant roles. Indeed, when observing objects and expressing their engagements in activities in terms of fact types, one can easily observe object to play different roles in different facts (types), mixing between *passive structure/active structure/behavior* roles. For example, a computer may be a passive element in the

[5] In principle, we would prefer to use the word "semantics" here. However, since the word "semantics," in our computer science-oriented community, tends to be equated to only mean "formal semantics," we will use the word *meaning*.

context of it being manufactured, but it may be an active element in a context where it processes key processes at some company.

Based on (Proper et al., 2018b), the suggested solution would be to treat the ArchiMate concepts as *roles* which an object may enact. This allows for a natural way for an object, say the computer in the above example, to enact both the role of an active and a passive element in the same ArchiMate model.

21.4.3 Adding More Semantic Precision

Both Challenge 4 and Challenge 5 require the ability to specify more semantic specificity regarding objects and relations. Such properties, e.g., identification mechanisms and cardinality constraints, have always been part of modeling languages such as ER (Chen, 1976), ORM (Halpin & Morgan, 2008), and UML (OMG, 2007). As such, it would be logical to "import" such mechanisms from these existing languages into ArchiMate.

Needless to say, it is not required for architects to specify such constraints in all situations. The key is to provide the ability to do so when required.

As an example, consider the situation as shown in Fig. 21.3. This is not a full example, yet it illustrates the main line of thinking. The setting is risk management and (quality) control in a production process. Suppose that we have a manual production process that is tightly controlled with production guidelines, metrics, controls, etc. In such a situation, there is a need to be able to represent the fact that:

- Supervisor roles must be executed by a human actor who may, in some other setting, also perform other tasks. The supervisor role may be played in one or more processes. However, a production process must have one and only one supervisor.
- Production roles must be fulfilled by a human actor also. The production role may be played in many processes. Even more, a production process may have more than one production role.
- We want to avoid that the supervisor role is played by the same human actor as the production role.

In Fig. 21.3, we have chosen to use the UML-style notation of adding cardinality at the association ends.

21.4.4 Abstraction Layers

Challenges 2, 4, and 10 are essentially all concerned with different ways to "separate concerns." As argued in Sect. 21.2.2, it is important to ensure a clear and consistent

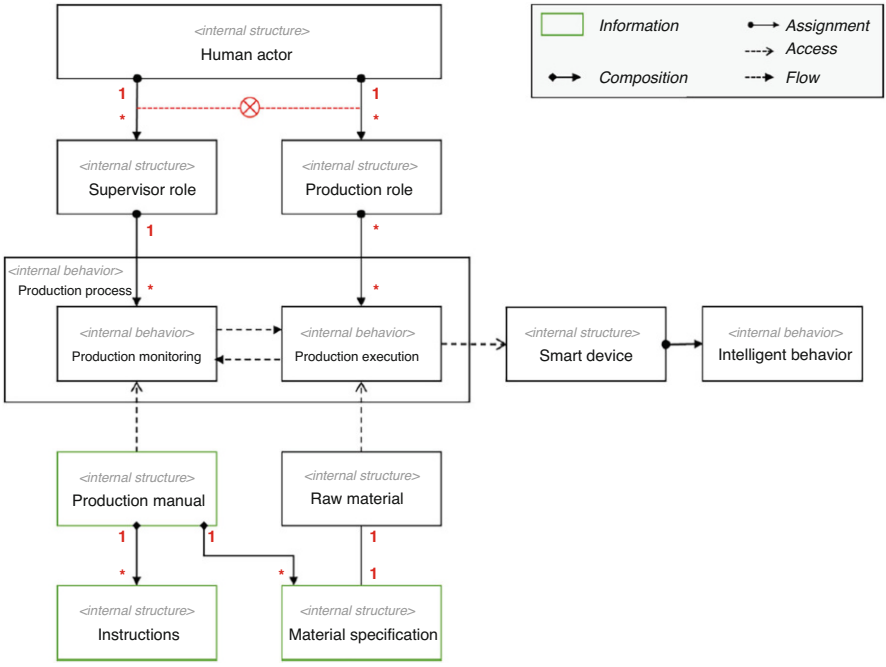

Fig. 21.3 Example of the use of with modality and cardinality constraints

structure of abstraction layers. In this chapter, we do not aim to take a specific position as regards the question of which *design philosophy* would be best.

When looking "across" different frameworks (ArchiMate Lankhorst et al., 2017, Enterprise Ontology Dietz & Hoogervorst, 2007, TOGAF The Open Group, 2011, IAF Wout et al., 2010, and Zachman, 1987), we posit that the following key constructs for the creation of abstractions (in different dimensions) are used:

Function-construction—This involves a distinction between:

1. Function refers to how a system is intended to function in light of what users, clients, and other stakeholders might deem useful.
2. Construction refers to how a system actually functions/is constructed to realize the provided functions.

Note that there may be good reasons for a constructed system to deviate from how it was specified from a functional perspective. For example, it may be more cost-effective to purchase a system that provides additional functionality—which was not originally specified—than to construct a system in line with what was specified.

Informational functioning—This dimension concerns different levels (of *aspect* systems (of systems)) that describe different levels of *functioning* of an enterprise

in terms of informational support, leading to a *business*, an *informational*, and a *documental* level.

Infrastructure usage—This concerns the fact that one system (of systems) can *use* the functions of another system (of systems), where the actual construction of the latter is of no interest to the (designers) of the former, except to the extent of defining service-level agreements.

Implementation abstraction—This concerns the gradual/stepwise introduction of details of the socio-technical implementation. For example, in IAF (Wout et al., 2010), this corresponds to the distinction between a conceptual, logical, and physical level, while in TOGAF (The Open Group, 2011), this corresponds to the level of architectural and logical building blocks.

Making a clear implementation abstraction also provides a natural way to deal with Challenge 3 pertaining to deal with the duality of human and digital actors. At the highest level of implementation abstraction, one would need to describe the workings of the enterprise independent of the question if it will be implemented by means of human actors or computerized actors. The immediate next level of implementation abstraction might then to make choices with regard to human/computerized actors explicitly, even allowing for mixed scenarios, while, e.g., also identifying which actors are ultimately responsible and accountable.

Each of the above-discussed abstraction mechanisms has a potential added value, also in the context of digital transformation. It is important to note that these abstraction mechanisms should not be thought of as a set of orthogonal dimensions. On the contrary, the *function-construction* mechanism and *information functioning* or *function-construction* and *infrastructural usage* can easily be mixed. We also do not want to suggest to "prescribe" a specific set of dimensions. We do, however, argue that an enterprise modeling language (framework) should ensure a consistent use of the above mechanisms within one dimension.

As discussed in Sect. 21.3, ArchiMate seems to have been mixing some of these dimensions in an inconsistent way.

21.4.5 Value Co-creation

The increasing focus on value co-creation, resulting from the shift from a goods-dominant logic to a service-dominant logic (Vargo & Lusch, 2008; Grönroos & Ravald, 2011; Lusch & Nambisan, 2015; Vargo & Lusch, 2016), results in Challenge 6, i.e., how to capture (potential) value(s) of products and services and how this results in value co-creation between providers and consumers of services by way of resource integration.

ArchiMate already provides *value* concept, and it seems possible to model value *co*-creation by using the collaboration/interaction concepts. However, as mentioned before, value, or even a value stream, is not the same as value *co-creation*. How

to best express this is still largely an open question. Some initial work/suggestions have been presented in Razo-Zapata et al., 2016; Feltus & Proper, 2017a, 2017b; Razo-Zapata et al., 2017).

The very nature of value co-creation also requires a shift from (only) architecting the "internals" in an enterprise to co-architecting the collaboration (including needed inter-organizational IT platforms) between multiple partners in the co-creation network (Chew, 2016).

In further elaborating the set of needed concepts for value co-creation, our recommendation (Proper et al., 2018a) is to (1) use the provider/customer roles as identified in (Grönroos & Voima, 2013), specialized to more specific co-creation activities taking place within the *provider sphere*, the *joint sphere*, or the *customer sphere*, as a reference model, while (2) using the foundational premises as articulated in (Vargo & Lusch, 2016) as design/architecture principles (Greefhorst & Proper, 2011a) that will guide the design of service systems for value co-creation, and (3) apply this in the context of real-world cases, to gain insight into the actually needed modeling concepts.

21.4.6 Capturing Design Motivations

The current version of ArchiMate does provide a motivation extension. However, as discussed in the previous section, it does not meet Challenge 8 in a satisfactory way. Separate from the fact that, as suggested above, it would be good if such an extension could be shared between, e.g., ArchiMate, 4EM (Sandkuhl et al., 2014), and BPMN (OMG, 2011), the actual level at which design decisions remain rather crude.

The work as reported in, e.g., Plataniotis et al. (2014a, 2015a) provides suggestions on how to remedy this. This includes the ability to, e.g., capture trade-offs between design alternatives and the actual decision-making process and the criteria used to make decisions (including the identification of compensatory and/or non-compensatory (Rothrock & Yin, 2008) criteria).

21.4.7 Managing Constant Change

As discussed in Sect. 21.2, the digital transformation requires enterprises to change constantly. This makes it less realistic to capture an enterprise's *current affairs* and/or *desired affairs* in terms of traditional notions such as "baseline" architecture and "target" architecture or even *plateaus*/*transition* architectures. Even though we observe some ingredients toward solutions for this challenge, we would argue that more research is certainly needed.

In an ideal world, the description of the *current affairs* would be maintained continuously, preferably in an automated way (Proper, 2014). Approaches such as

process mining (van der Aalst, 2011) and enterprise cartography (Tribolet et al., 2014b) indeed provide good starting points.

Architectures capturing the *desired affairs* also tend to be specified using a rather "instructive" of typical "boxes and lines" diagrams. This does not really invite architects to reflect on what the more *endurable* elements and assumptions and what the less stable elements and assumptions are. This has also triggered the development of the concept of multi-speed enterprise (IT) architectures (Abraham et al., 2012). It also resulted in a stronger positioning of, e.g., (normative) architecture principles (Greefhorst & Proper, 2011a) as a way to complement the "instructive" style (the "boxes and lines" diagrams) by a more "directional"/"regulative" perspective.

21.4.8 Consequences for the Meta-Modal

Modeling is a key aspect of how we, as humans, attempt to get to grips with reality. Whether models are—as much as possible—an accurate representation of what happens/should happen in the real world or whether they are "merely a hypothesis" of what we believe to be true about the real world does not change the fact that the modeling *language* must be precise enough to express what we want. In the previous section, we explored limitations of the current predominant architecture modeling language: ArchiMate. In this chapter, we explored how some of these limitations can be alleviated. This is deliberately positioned as an *exploration*.

The main contribution of our approach is twofold. First of all, we believe that the meta-model for representing all aspects of the digital enterprise should be (a) greatly simplified and (b) made more flexible. The biggest change is to remove the active/passive structure dichotomy, which provides a more natural way of expressing the state of affairs in the real world. Another is only apply layering on the structure side, which avoids duplication of modeling decision. Even more, when combined with a decision to have less predefined (structure) concepts, this allows users of the language to adapt the language more to their individual needs while still retaining the integrity rules of the overall framework. Last but not least, adding the notion of constraints and modalities will give modelers the option to add more precision to their models where needed.

It is a well-known fact that the proof of the pudding is in the eating: it would make sense to use our approach in practice to see if, indeed, it lives up to its promise.

21.5 Conclusion and Further Research

In this chapter, we presented key challenges which the digital transformation puts on enterprise (architecture) modeling languages. These challenges are based on practical experiences and insights from the field of enterprise architecture.

We then assessed the extent to which the current version of ArchiMate meets these challenges. The conclusion was that ArchiMate does not yet fully cover all of the identified challenges. This can be explained by the fact that ArchiMate was developed at a time when the digital transformation was not yet that dominant.

We then provided suggestions on how to possibly improve ArchiMate to better meet the challenges of digital transformations. In further research, we intend to further elaborate these suggestions, in particular with the aim of finding strategies that work in real-world practice.