Effectively Modeling Your Architecture

12

Gerben Wierda

Abstract

Enterprise architecture management is to a large extent about managing complexity; this holds for both the business and the IT architecture level. As a discipline, it has tended to focus on creating simplicity, such as embodied by principles, guidelines, and simplified models of what is or what is to be. This chapter is about the why and how of setting up a broader and deeper role of modeling in enterprise architecture management. In particular, it presents ways (specifically languages) to model the business architecture in its current state at a reasonable level of detail. It also discusses the role of modeling in future state planning.

12.1 Introduction: Managing Complexity in Enterprise Architecture Management

Though enterprise architecture management as a discipline has originally been established, in particular, to fight the intractable complexity of the information technology (IT) used by the business, the reason for any enterprise is its business goal and thus its business processes, and not the IT.

Enterprise architecture management has traditionally moved between a rather IT-oriented approach and a "start with the business" approach, e.g., the "business-IT-alignment" approach. In the extreme application of the first approach, the business is often ignored in practice. In the extreme application of the second approach, a (flawed) "waterfall" mechanism ensues, which generally has a limited effect on the problem that was to be solved: the intractable complexity of IT used by the business.

G. Wierda (✉)
APG, Heerlen, The Netherlands
e-mail: gerben.wierda@rna.nl

D. Simon, C. Schmidt (eds.), *Business Architecture Management*, Management for Professionals, DOI 10.1007/978-3-319-14571-6_12

Both approaches—the "IT-centric" and the "business-first" approach—are attempts to find a simplified starting point to manage the complexity of all the relations between IT and IT, business and IT, and business and business. The business-first approach, for instance, often at the start tries to establish a clear division of the business into business functions, so that these can be used to divide the landscape into—from the enterprise architecture perspective—manageable semi-isolated parts.

The problems with these approaches—starting in a specific "layer" (e.g., business, applications, infrastructure) leads to problem addressing the other layers—has led to another type of simplification: create a basic set of guidelines, often separately for each layer. This is the "principles approach." The Open Group Architecture Framework (TOGAF), for instance, starts with principles: clear and simple guidelines for design decisions that are taken later. TOGAF gives a series of examples (cf. The Open Group 2011), as exemplified in Table 12.1.

This sounds fine from a business perspective and it also is a truism: if there is no business case for a change, do not change. Now, given that changes in the IT landscape come by definition with continuity risks, as errors are all too human in these complex endeavors, business is generally not too keen on changes in the IT landscape such as infrastructure life cycle events,[1] such events being a detail that also escapes the attention of the abstraction-oriented enterprise architects. The

Table 12.1 Exemplary principle [adopted from The Open Group (2011)]

Principle	Requirements-based change
Statement	Only in response to business needs are changes to applications and technology made
Rationale	This principle will foster an atmosphere where the information environment changes in response to the needs of the business, rather than having the business change in response to IT changes. This is to ensure that the purpose of the information support—the transaction of business—is the basis for any proposed change. Unintended effects on business due to IT changes will be minimized. A change in technology may provide an opportunity to improve the business process and, hence, change business needs
Implications	Changes in implementation will follow full examination of the proposed changes using the enterprise architecture We don't fund a technical improvement or system development unless a documented business need exists Change management processes conforming to this principle will be developed and implemented This principle may bump up against the responsive change principle. We must ensure the requirements documentation process does not hinder responsive change to meet legitimate business needs. The purpose of this principle is to keep us focused on business, not technology needs—responsive change is also a business need

[1] Life cycle events are changes in the architecture that are driven by the evolution of used components. Good examples are operating systems that go out of maintenance, or updates of any type of software.

business may understand keeping up to date with security patches to operating system and updates to virus scanners, as this can be directly related to business demands with respect to security and continuity, but why update a SAN[2]-driver on a server (a server that uses the SAN, not one that provides it) if the old one works? The old adagium says: "If it ain't broken, don't fix it." Changes always carry risks and these risks are not welcome, especially when that change is happening in a period in which the business performs critical processes, like year-end reporting. So, all too easily, a business will ask for "freezes" in the IT landscape because of its continuity (business) need. Continuously, however, parts of the landscape change. That SAN environment at some point needs to be upgraded and at that time there is a clear business need. And then, unexpectedly, a while later, the business-critical SAN-using server fails, because the old driver and the new SAN turn out to have some hidden incompatibility. The SAN provider of course never tested its system with all the older driver versions of all operating systems. The data is corrupted and the IT department is blamed for the lack of continuity, while everybody forgets that it was the nasty little detail of the business demanding the freeze because of its own business needs that was the root cause of the problem. As the proverb says: "People trip over molehills, not over mountains."

Most existing approaches of enterprise architecture management have in common that they start out with trying to find a position where the world is clear and simple, so it can be managed from there. The example of the lowly SAN-driver (a small detail in the overall landscape, if there ever was one) is to show that principles and other simplification instruments may sound fine (especially when they are truisms), but that does not turn them into sufficiently reliable approaches to design/change decisions. And that is true for all the simplifications enterprise architects use to make their work as enterprise architects manageable and easier to communicate.

The bottom line in enterprise architecture is that there are a multitude of complex relations, both within the layers we recognize, but also between those layers. Looking at any layer in isolation carries large risks and so does ignoring details. Enterprise architecture is a holistic type of subject with a rather unlimited amount of complexity. There are approaches such as SIP (cf. Sessions 2008) that promise ways to minimize your architectural complexity by partitioning your business and IT. These may help a bit in fighting the "accidental" complexity of your architecture; they will, however, have no effect on its "fundamental" complexity. Such "object-oriented" approaches also generally tend to ignore the limits of the physical world: a nice layered architecture approach, with services and messaging between "functional partitions," or an architecture developed for flexibility, tends to suffer from performance issues and the limitations of the physical world. That is another part of the complexity that is often ignored in enterprise architecture management approaches: it is often assumed that everything is possible, technically, and that the

[2] SAN stands for "Storage Area Network"—a technology for offering network-based storage to systems.

world is not physical but logical. But when the whole architecturally beautiful setup leads to the risk department getting its numbers long after they are needed, pragmatic fixes (read: breaches of the ideal architectural picture) will be implemented. Business comes first, after all.

In other words: approaches to enterprise architecture management that try to establish a simplified model of reality to work from, be it functional divisions, principles, or other approaches, tend to operate in a world that lacks so much relevant detail that its decisions are brittle and of limited effect on the actual problem. Practicing enterprise architecture management, being a method to manage complexity, should not be based on ignoring that complexity, but on confronting it. This means three things:

1. If enterprise architecture management is to be successful, it has to be a collaborative effort in the company. The main reason for this is that nobody, not even an enterprise architecture department, has enough knowledge about relevant details to go at it alone. Setting up good enterprise architecture processes and organizing this collaborative use of the fragmented knowledge on all aspects of the enterprise is a key for success.
2. Enterprise architecture management needs a mechanism to assess the need for details. A good candidate is using the risk aspect. For example, instead of being based on ignoring details, abstraction in enterprise architecture must be based on "consciously leaving out irrelevant details." which assumes actually analyzing the potential effect of ignoring details before they are left out of the analysis.
3. Knowing your "current state architecture" (CSA) in enough detail is essential to improve the change-design-and-assessment process. The business architecture is an important aspect of that.[3]

12.2 Current State: Modeling Your Business

12.2.1 Why Model Your Business?

Businesses, especially complex ones, have a need to document themselves. Partly, this is for their own use: setting up standardized, reproducible behavior is essential for many an organization's success. There are, however, more reasons to document your business. Some of these are:

- Regulators demanding well-documented and auditable business processes and IT support. Note, this is the same one as the one mentioned above, just from another stakeholder.

[3] For more complete architectures, often the term "landscape" is a good replacement for the term "architecture" to make people aware that it is not about the guidelines but about the (to be created) reality.

- Less work finding out your start position at the beginning of a change initiative
- Identifying weak spots in your landscape and other uses for portfolio management

Such demands generally lead to initiatives to document the landscape. For primary business processes, many (but certainly not all) organizations have this as an established practice and they will have "business process manuals" that describe the processes that the business follows (though often not always at a reasonable quality). For IT, this is also an established practice, though certainly not always of a high quality either. Most larger organizations will have a "Configuration Management Database" (CMDB[4]) that is above all directed to IT infrastructure. Often, companies will have an IT service management system (e.g., to log incidents and calls) which implicitly also has an administration of available systems, sometimes roles (such a system owner), sometimes processes, the latter seldom coupled to the actual process documentation. Various business functions (service management, business continuity management, security, database management, operational risk management, etc.) generally keep their own administration, sometimes in dedicated systems, sometimes just in drawings, spreadsheets, and documents. Not surprisingly, these various administrations do not form a consistent whole. There might be simple differences, like an application named differently in different administrations and documentations, up to complete mismatches between the overlap of functions. The business continuity administration might be based on completely different process descriptions than those in the official business process manuals, for instance.

An essential property of these (fragmented) administrations is that they are very difficult to maintain. Look below the surface, and even those business process manuals will often have internal and external inconsistencies. The attempt to model some landscapes in drawings leads to impressive looking large posters on the wall, that, even if they are correct, are out of date within a few months.

There is only a single solution for this mess: modeling. A model is structured (as opposed to unstructured) information that allows automated coupling between models as well as automated analysis. There are many modeling environments for both domain-specific models (such as process models) and (integrated) enterprise architecture models, and there are a lot of tools available. Many of these are based on a proprietary modeling language. For the modeling to be robust under tool change (and given the fact that these models should exist for a long time), and for other reasons, using standard modeling languages is generally preferred. For business process modeling, the leading standard is *BPMN*® (OMG 2011), the "Business Process Model and Notation" (or, depending on where you read in the documentation, "Business Process Modeling Notation"), a standard managed by the Object

[4] A CMDB is a system to document the IT that is being maintained (at an instance level). It is often limited to hardware items such as computers, keyboards, monitors, and a rough sketch of software installs.

Management Group. For enterprise architecture modeling, the leading standard is *ArchiMate*® (The Open Group 2013), managed by The Open Group. Both standards overlap in several ways, e.g., processes are concepts that are modeled in both.

12.2.2 A Single Logical Model

Before describing the key parts of a current state modeling setup, we should clarify an important aspect of business modeling: it is unavoidable that there are multiple models of your organization in use in your organization. They are seldom recognized as models, but in practice they are. Examples, in line with the previously mentioned documentation efforts, are:

- A CMDB will effectively contain a "model" of your infrastructure, possibly with links to applications, business actors and business roles such as owners, etc.
- An operational risk management tool will have a "model" of your processes, roles and actors, maybe business functions.
- The IT service management ("help desk") tool will have a "model" of applications, maybe processes, owners, platforms.
- The business continuity management administration will have a "model" of processes, applications, maybe data.
- The security function will have a "model" of applications, maybe processes.
- The business will have a "model" of its processes (flow charts with documentation), applications used, roles, actors, data, etc.
- Information management will have a "model" of all applications, platforms, maybe business processes or business functions, etc.
- "Run" managers (those responsible for keeping IT running in day-to-day operations) may have their own "model" of what they are managing, e.g., an overview of databases, servers, software.
- Management may keep "balanced score cards," and strategists may work with business model canvases, both based on a structuring of the organization.
- HR maintains the "management structure" of the organization, generally in HR systems.

All these models describe a single reality (your business) from a particular point of view. The problem is that in most organizations, these models are separate and they may not tell the same story and sometimes they even contradict each other.

A good modeling approach therefore requires setting up the different physical models in such a way that:

- They form a single logical model.
- That single logical model can be maintained with limited effort.

12.2.3 Business Process Models: BPMN

Though many organizations still use unstructured approaches to modeling their business (word processing documents, including graphical representations such as flow charts), a standard process modeling grammar has established itself over the last decennium: BPMN, an open standard from the Object Management Group (OMG). BPMN looks like flow-charting (and as such is easily accepted by the business), but is based on a structured definition of the grammar (itself written in UML—the Universal Modeling Language, also from the OMG). It is also a grammar for which there is ample tool support, including free/open source solutions.

BPMN is not perfect. Even Bruce Silver—its leading teacher and author of a very good introduction to its use (Silver 2011)—admits that the grammar has its troublesome aspects.[5] These stem mainly from the fact that the language has been designed above all to model "executable" processes, that is, processes that can be directly executed by a computer. It is, in other words, a language that shows its technical heritage and that it was never initially intended as a documentation language for human processes to be understood by humans. Examples of peculiar aspects of BPMN are for instance the absence of a graphical representation of its core concept "Process," the ambiguity of a core element like "Pool/Participant," and the "bolted-on" nature of the (for humans important) "Lane" concept.

Though that sounds like a list of reasons not to use BPMN, this is not what is intended. BPMN is eminently usable for the purpose of structurally documenting your processes. What a list like the one above illustrates is that perfection is not necessary for usability.

BPMN is almost completely focused on modeling the behavior of an enterprise. It has structures for activities of all kinds, and has trigger- and flow-relations that can be used to create complex behavioral descriptions.

12.2.4 Enterprise Architecture Models: ArchiMate

Roughly of the same age as BPMN is ArchiMate, but it has grown in popularity beyond its initial following after its adoption by The Open Group as its standard for enterprise architecture modeling in 2009. ArchiMate is now a de facto "open standard" enterprise architecture modeling language, though it is not yet as widespread as BPMN.

ArchiMate was developed by a university-business collaboration in the Netherlands in the early 2000s. ArchiMate is fundamentally different from BPMN in a number of aspects, the main one being that it was not designed from a formal perspective but from a pragmatic perspective: the collaborators designed a language that fit their use of modeling. As such it is not based on a formal definition (as BPMN is), but on practical considerations. Where BPMN is more "early Wittgenstein"

[5] See Business Process Watch (2014) for several blog posts that address these issues.

(Wittgenstein 1984), ArchiMate is more "late Wittgenstein" (Wittgenstein 1958), and those who are familiar with the philosopher's work will understand that, while there is a common theme, the difference is rather fundamental.

ArchiMate is based on splitting the architecture in the usual enterprise architecture layers: business and information architecture, application and data architecture, and infrastructure architecture. Its internal structure in each layer is based on dividing into "active structure," that performs "behavior" which affects "passive structure," as in simple natural language like subject-verb-object patterns. Another special aspect is that relations between different elements in a model that are not directly connected (e.g., a business process and a server) may be derived from all the intermediate relations between the elements involved, thus offering standardized ways to summarize detailed models into simplified ones.[6]

12.2.5 Combining BPMN and ArchiMate to Create Models of Your Business

Table 12.2 summarizes the differences between BPMN and ArchiMate. These differences make the obvious idea of modeling process details in BPMN and the overall enterprise architecture in ArchiMate not straightforward.

Table 12.2 Comparison of ArchiMate and BPMN

ArchiMate	BPMN
Model the enterprise	Model (executable) processes
Split in layers: business, application, and infrastructure architecture	Not split in layers
Pragmatic metamodel	Formal metamodel
Fully graphical grammar	Graphical representations added to parts of the formal grammar
Strong on the structure, weak on the dynamics of the enterprise (i.e., triggers, flows)	Strong on the dynamics of processes, weak on the structure of the enterprise
Split in active structure, behavior (of active structure), and passive structure	Mostly behavior
Derived relations	No derived relations
Model/view separation common in tooling	Model/view separation not common in tooling

[6] This mechanism has various restrictions. For example, there are non-derivable relations that are clearly valid and derived relations that may not be valid. See Wierda (2014) for more information and, in general, a possible introduction into the language.

12.2.6 Patterns and Links

To call BPMN or ArchiMate a language is a bit of a misnomer, stemming from the IT adoption of the word "language" for constructs like programming languages. The elements that make up BPMN and ArchiMate are more elements of a *grammar* than a *language*. What is said in the language depends not just on the grammar, but what the elements are given as "name" (i.e., label). To make this clear with an example, it is easy in ArchiMate to model that the Large Hadron Collider is being managed using an Excel spreadsheet. Such a model is correct ArchiMate (correct grammar), but the meaning is nonsense.

What ArchiMate and BPMN have in common with real languages is that there are many ways to say the same thing. Not just in level of detail, but there is also a certain freedom of choice on how to use the grammar and what elements to include. Different modelers of a process or a piece of enterprise architecture will come up with correct, but different models. This effect is stronger in ArchiMate than in BPMN (as ArchiMate slightly more resembles the non-logical pragmatic nature of ordinary language), but both grammars have this aspect.

The advantages of structured documentation of your business come from the possibilities of manageable coupling of those different "models of reality" and manageable maintenance of and reliable analysis based on those models. All these advantages melt away when there is a total freedom of choice with respect to the patterns[7] used. Because the languages themselves are limited in the structure they force upon the modeler, using them effectively requires the disciplined use of fixed patterns. Setting up and strictly following these patterns is a "conditio sine qua non" for the effective and successful use of structured modeling of your business.

If these patterns are designed well and followed strictly, it enables the linking of models in various grammars, creating that single logical model of your enterprise. An example of a linkage between BPMN and ArchiMate models can be found in Wierda (2014).

12.3 Future State and Change: Planning Your Changes

12.3.1 From Broad Strokes to Fine Detail: The Architecture of a Change

From Jan Hoogervorst comes the term "Columbus Management":

[7] As there are many ways to model each aspect (from business processes/functions to database servers) each way can be considered a pattern (or a style). Using patterns means that one will model the same aspect always exactly in the same structure, similar to using a very limited set of grammatical constructs when writing text.

> When we left, we did not know where we were going. When we arrived, we did not know where we were. And everything was paid for with other people's money.[8]

Interestingly enough, if you relate this to people, while most will understand that this means that starting a project without enough design is asking for trouble, some will argue that Columbus was successful. In reality, of course, Columbus was an exception. Most projects that were based on flawed or missing plans, then and now, failed (in Columbus' time often with deadly results). Going forward with a project, and hoping it will "strike lucky" like Columbus, does not seem like a rational strategy. That is why the idea of a "project architecture" to establish some guidance that makes the project more predictable is generally accepted.

Changes to our existing landscapes, certainly the more substantial ones, are generally the result of projects. These days, when working "under architecture," projects are generally required to create some sort of "*project start architecture*" (PSA). This deliverable is generally intended to guide the design work of the project.

The question of course is: what constitutes a good project architecture? Can we go ahead with allocating large sums of time, money, and people when we have only a vague idea what the outcome will be? Strangely enough, this is still what often happens. Projects often start with a dreamed up budget and plan, or they start in a phased approach that in the end eats up a multiple of the resources originally planned. The reason is that businesses are generally convinced that it is impossible to design everything in advance. They are right, of course, as the ultimate complexity overwhelms such an attempt.

The conviction that having all details in advance is impossible generally leads to an approach where there are no details in advance. Such an approach generally fails, as it are those details that derail the projects in the first place. What is needed, therefore, is a smarter approach to those details, and here modeling can help.

Without going into much detail: instead of trying to establish a well-defined level of detail to adhere to, it is much better to have a dynamic approach to details. It is clear that one has to end with all the details in place, at which time these details become part of your current state landscape. But the question which level of detail is appropriate for the start of your project requires a bit more intelligence.

From the field of "industrial safety" comes an approach that can be adapted to enterprise architecture management. In that field, risk is not seen as something to be avoided, as there is no such thing as "not taking risks." The key to safety is not "avoiding risks," but "consciously taking acceptable risks." This insight can be translated to enterprise architecture management's handling of—potentially risky—details. In our practice, we have adopted the definition (in enterprise architecture management): "Abstraction is consciously leaving out irrelevant details."

[8] Jan Hoogervorst is former Vice President Corporate Information Strategy of KLM Royal Dutch Airlines. He used this in a speech once, and told the author in private communication that he did not invent the term, but had also picked it up from someone else somewhere. The source is unknown.

It is clear that at the start of a project, not all details of the target state can be known. But leaving them out requires an assessment of the risk of leaving them out. There are two kinds of risk to contemplate: risk for the project's success (scope, cost, time) and risk for the ensuing landscape of the organization.

This translates to modeling as follows. A project start architecture should contain a model of what is to be delivered by the project: the target state. Details may be left out if it is estimated that leaving them out does not carry too much risk for neither the project nor the overall landscape. During the project, more and more details will be added until at the end the model is complete at the level of detail that is required for the current state architecture.

Next to modeling the current state architecture and target states for projects, planning of the overall future state can also be supported by modeling, though generally in a less detailed manner.

12.3.2 Future State: Multiple Models of a Single Reality and the BITMAP

A good "*future state architecture*" is more than just a sketch of the future and a roadmap to get there. This is not the place to delve into it deeply, but part of it can be supported by modeling.

And while current state models and project end state models need to be sufficiently detailed, future state models (such as domain target states, or plateaus[9]) may be more abstract and less detailed. Here too, the rule about details can help to decide on the required level of details.

When thinking about the future of the organization, architects need some sort of map of that organization. Enterprise architects often use a pattern where they divide the business into "business functions" and model part of their design patterns on the basis of such divisions. Now, the concepts of enterprise architecture—and "business function" is not an exception—are not universally defined. In some approaches, the function is an abstraction that looks like an "acting element" as in "the payments function makes sure that all payments are executed before they are late." In other approaches, such as in ArchiMate, the business function is behavior *of* an "acting element," so the active element may be "payments department" and its behavior the "paying invoices" business function. Such different uses of words lead to a lot of confusion in the enterprise architecture world. In this chapter, we will follow ArchiMate's definitions of these concepts.

[9] A plateau is a situation in the development of your landscape that will be in production. Projects deliver results, and when these results are used, the actual landscape changes. As there are many changes and many projects, or even phases of delivery in projects, working with the concept of a plateau gives you the possibility to view the interrelations between various change initiatives to make sure beforehand that the landscape in production will be acceptable and catch conflicts between change initiatives before they can become a problem.

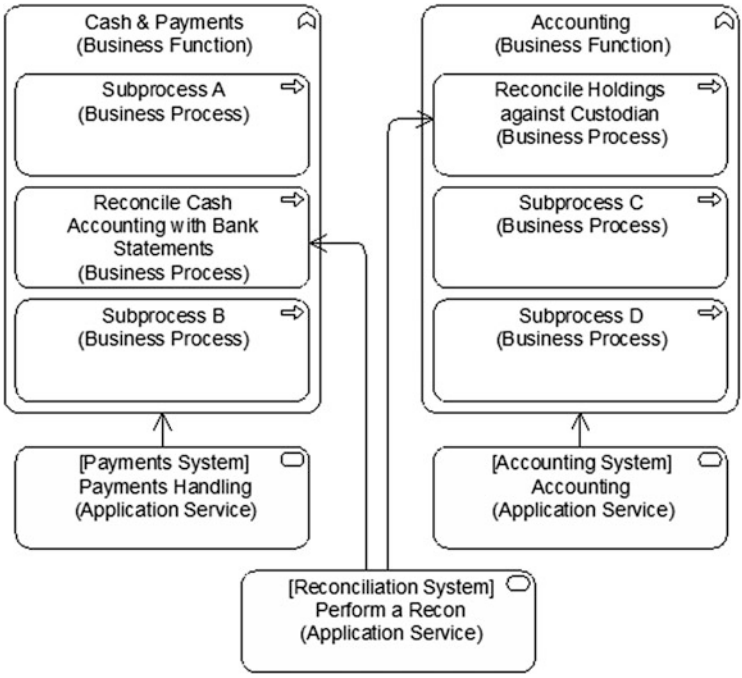

Fig. 12.1 The "Cash & Payments" business function and the "Accounting" business function with the applications they use (Wierda 2012)

Now, suppose we want to make a functional division of our company. As an example, suppose we have two aspects: handling our holdings (stock, bonds, etc.), and handling our cash. Both types of activities require a different set of skills, work with different external parties, use (partly) different administrations, etc. An example ArchiMate view can be seen in Fig. 12.1.

Here we see two ArchiMate business functions with some internal processes modeled. The arrows represent the ArchiMate "Used-By" relation, which tells us that the business uses a couple of systems (the "service" the application provides for the business). These services may be provided by one or more systems; in our labeling standard we label the application service generally with the name of the system that provides it, so in this example there is a 1:1 relationship between application and realized service.

Both functions need to reconcile their internal administration with the outside world. For "Cash & Payments" this means reconciling what is in the payments system with what is in the bank statements. For "Accounting" this means reconciling what is in the accounting system with the statements from the custodians (parties that keep assets for the actual owners, just like a bank keeps cash for its real owner).

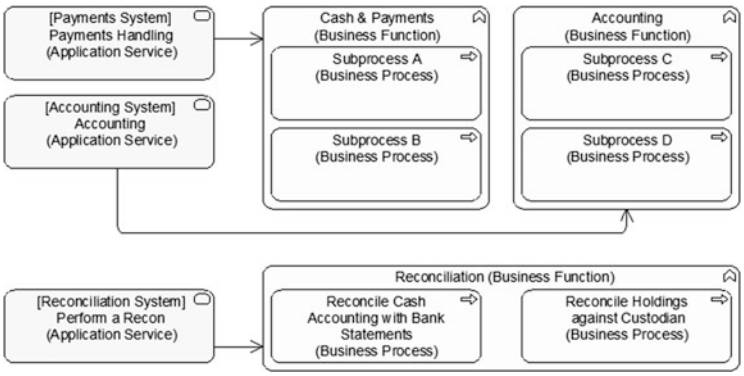

Fig. 12.2 Introducing the "Reconciliation" function (Wierda 2012)

Enterprise architects may end up in heated discussions about dividing the business into functions. This is because, from a different perspective, we may not have two functions, but three.

Figure 12.2 shows this different way of looking at the business. Enterprise (IT) architects are drawn to this way of looking, because they know that one of the problems they have to address is that the same kind of IT support may be implemented twice if you look purely at the business process. If you do not watch out, "Cash & Payments" sets up quite a different reconciliation system than "Accounting" and suddenly our landscape has two different systems for what is in effect the same functionality. Exactly this is something that working with enterprise architecture management is meant to prevent. In other words, simply dividing the business into functions and working from there towards the IT landscape may lead to suboptimal results from an enterprise (IT) architecture point of view. This example may be easily recognized, but when such divisions are coupled with design principles based on that division (e.g., "each function 'owns' its data" or "data communication between functions is message-based") it may quickly lead to unintended complexities or problems (e.g., performance of the eventual resulting setup).

ArchiMate's definition of a business function says it is a grouping of activities—a grouping that is based on aspects such as skills, resources, location, etc. The architect is in fact free to decide on which basis the division is to be made. But how should we solve the heated argument? The answer may be to "let go." If the choice of aspect leads to a different division, and both aspects make sense, then we can accept that there are multiple concurrent valid divisions. From a business perspective, a "Reconciliation" function may not make sense, but from an IT perspective it does, and such a perspective is also useful.

The idea is not to have an endless number of different functional landscapes; this would defeat the object of creating structure in these discussions in the first place. But there is nothing wrong with having more than one perspective. From a management perspective, one division may be needed; from a business-IT-

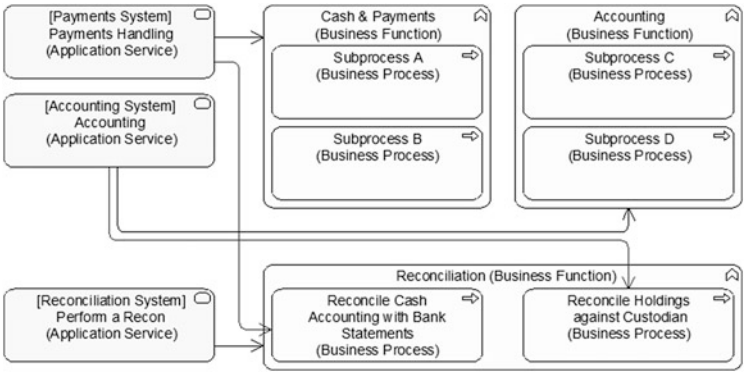

Fig. 12.3 Adding "Break Resolution" to the mix (Wierda 2012)

alignment perspective, another may be useful. The name we use for the one that is used for enterprise IT architecture (business-IT-alignment) planning is "BITMAP," which stands for "Business-IT Mapping."

The division in Fig. 12.2 above looks cleaner from the business-IT-alignment perspective: every function uses its own IT; a popular architectural principle is being followed. But our picture above hides details that show the opposite.

As Fig. 12.3 displays, if we add "Break Resolution"[10] as part of the reconciliation processes to the mix, the situation becomes more realistic and instead of a single reconciliation system being used by two business functions, the "Reconciliation" function now uses three systems (as shown by the additional "Used-By" arrows). It is still useful to have a BITMAP though, as it helps in keeping the IT landscape as simple as possible.

12.4 Introducing and Sustaining a "Modeling-Supported" Architecture Management Approach

Introducing a modeling-based approach in an organization is not easy. There are a couple of obvious requirements that need to be fulfilled:

- The appropriate skills (e.g., ArchiMate and BPMN modeling) need to be established in the organization.
- A serious initial investment needs to be made in setting up the current state models of (at least) the enterprise architecture and (more detailed) business processes.
- There must be an effective governance to keep the models up to date.

[10] Break Resolution is solving the found discrepancies between internal and external administrations, hence handling the exceptions that come out of the (automated) reconciliation process.

This is not all. To fulfill these requirements, other prerequisites are necessary. The most important is that the organization understands both the need and the feasibility of the approach. This is not easy, because many people in the enterprise architecture field do not believe the approach is possible. There is a widespread and deeply felt belief that doing this is either impossible (because of the overwhelming real complexity of organizations) or unnecessary (because other approaches like setting up design principles and abstract future state landscapes are expected to be sufficiently effective). Not having the shared belief makes the approach very difficult (if not impossible) to realize. It will probably take quite some time before using models more extensively in enterprise architecture management has established itself.

The fact that you cannot model everything in enterprise architecture has led to a practice where almost nothing is really modeled. But a smarter, risk-based approach to modeling may change that in the coming years.

References

Business Process Watch (2014) Blog. http://brsilver.com/blog/
OMG (2011) Business process model and notation (BPMN) v2.0 specification. Object Management Group, Needham, MA
Sessions R (2008) Simple architectures for complex enterprises. Microsoft Press, Redmond, WA
Silver B (2011) BPMN method & style, 2nd edn. Cody-Cassidy Press, Aptos, CA
The Open Group (2011) TOGAF® version 9.1. Van Haren Publishing, Zaltbommel
The Open Group (2013) ArchiMate® 2.1 specification. Van Haren Publishing, Zaltbommel
Wierda G (2012) Mastering ArchiMate, Ist edn. R&A, Heerlen
Wierda G (2014) Mastering ArchiMate, IInd edn. R&A, Heerlen
Wittgenstein L (1958) Philosophical investigations. Blackwell, Oxford
Wittgenstein L (1984) Tractatus Logico-Philosophicus. Suhrkamp, Frankfurt