Open Group Standard

ArchiMate® 3.0 Specification

Open Group Standard

ArchiMate® 3.0 Specification

ISBN: 1-937218-74-4

Document Number: C162

Published by The Open Group, June 2016.

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

The Open Group, Apex Plaza, Forbury Road, Reading, Berkshire, RG1 1AX, United Kingdom

or by electronic mail to:

ogspecs@opengroup.org

Contents

List of Figures

List of Examples

List of Tables

Preface

The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through IT standards. With more than 500 member organizations, The Open Group has a diverse membership that spans all sectors of the IT community – customers, systems and solutions suppliers, tool vendors, integrators, and consultants, as well as academics and researchers – to:

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

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

The Open Group publishes a wide range of technical documentation, most of which is focused on development of Open Group Standards and Guides, but which also includes white papers, technical studies, certification and testing documentation, and business titles. Full details and a catalog are available at www.opengroup.org/bookstore.

Readers should note that updates – in the form of Corrigenda – may apply to any publication. This information is published at www.opengroup.org/corrigenda.

This Document

This document is the ArchiMate® 3.0 Specification, an Open Group standard. It has been developed and approved by The Open Group.

This edition of the standard includes a number of corrections, clarifications, and improvements to the previous edition, as well as several additions.

Intended Audience

The intended audience of this standard is threefold:

- All those working to shape and implement complex organization change. Typical job titles include Enterprise Architecture practitioners, business architects, IT architects, application architects, data architects, information architects, process architects, infrastructure architects, software architects, systems architects, solutions architects, product/service managers, senior and operational management, project leaders, and anyone working within the reference framework defined by an Enterprise Architecture.

- Those who intend to implement the ArchiMate language in a software tool. They will find a complete and detailed description of the language in this document.
- The academic community, on which we rely for amending and improving the language based on state-of-the-art research in the architecture field.

Structure

The structure of this standard is as follows:

- Chapter 1, Introduction, provides the introduction to this standard, including the objectives, a brief overview, conformance requirements, and terminology.
- Chapter 2, Definitions, defines the general terms used in this standard.
- Chapter 3, Language Structure, describes the structure of the ArchiMate modeling language, including the top-level structure, layering, the ArchiMate Core Framework, and the full Framework.
- Chapter 4, Generic Metamodel, describes the structure and elements of the ArchiMate generic metamodel.
- Chapter 5, Relationships, describes the relationships in the language.
- Chapter 6, Motivation Elements, describes the concepts for expressing the motivation for an architecture, together with examples.
- Chapter 7, Strategy Elements, provides elements for modeling the enterprise at a strategic level, together with examples.
- Chapter 8, Business Layer, covers the definition and usage of the Business Layer elements, together with examples.
- Chapter 9, Application Layer, covers the definition and usage of the Application Layer elements, together with examples.
- Chapter 10, Technology Layer, covers the definition and usage of the Technology Layer elements, together with examples.
- Chapter 11, Physical Elements, describes the language elements for modeling the physical world, together with examples.
- Chapter 12, Cross-Layer Dependencies, covers the relationships between different layers of the language.
- Chapter 13, Implementation and Migration Elements, describes the language elements for expressing the implementation and migration aspects of an architecture (e.g., projects, programs, plateaus, and gaps).
- Chapter 14, Stakeholders, Viewpoints, and Views, describes the ArchiMate viewpoint mechanism.
- Chapter 15, Language Customization Mechanisms, describes how to customize the ArchiMate language for specialized or domain-specific purposes.
- Appendix A, Summary of Language Notation, is an informative appendix.

- Appendix B, Relationship Tables, is a normative appendix detailing the required relationships between elements of the language.
- Appendix C, Example Viewpoints (Informative), presents a set of architecture viewpoints, developed in ArchiMate notation based on practical experience. All viewpoints are described in detail. The appendix specifies the elements, relationships, usage guidelines, goals, and target groups for each viewpoint.
- Appendix D, Relationship to Other Standards (Informative), describes the relationships of the ArchiMate language to other standards, including the TOGAF framework, BPMN, UML, and BMM.
- Appendix E, Changes from ArchiMate 2.1 to ArchiMate 3.0 (Informative), is an informative appendix outlining the changes in the standard between Version 2.1 and Version 3.0.

Trademarks

ArchiMate®, DirecNet®, Making Standards Work®, OpenPegasus®, The Open Group®, TOGAF®, UNIX®, UNIXWARE®, X/Open®, and the Open Brand X® logo are registered trademarks and Boundaryless Information Flow™, Build with Integrity Buy with Confidence™, Dependability Through Assuredness™, FACE™, the FACE™ logo, IT4IT™, the IT4IT™ logo, O-DEF™, Open FAIR™, Open Platform 3.0™, Open Trusted Technology Provider™, Platform 3.0™, the Open O™ logo, and The Open Group Certification logo (Open O and check™) are trademarks of The Open Group.

Java® is a registered trademark of Oracle and/or its affiliates.

OMG®, UML®, and Unified Modeling Language® are registered trademarks and BPMN™ and Business Process Modeling Notation™ are trademarks of the Object Management Group.

All other brands, company, and product names are used for identification purposes only and may be trademarks that are the sole property of their respective owners.

Acknowledgements

The Open Group gratefully acknowledges the ArchiMate Forum, a forum of The Open Group, for developing this standard.

The Open Group gratefully acknowledges the contribution of the following people in the development of this and earlier versions of this standard:

- Iver Band, EA Principals & Cambia Health Solutions
- Thorbjørn Ellefsen, Capgemini
- William Estrem, Metaplexity Associates
- Maria-Eugenia Iacob, University of Twente
- Henk Jonkers, BiZZdesign
- Marc M. Lankhorst, BiZZdesign
- Dag Nilsen, Biner
- Erik (H.A.) Proper, Luxembourg Institute for Science and Technology & Radboud University Nijmegen
- Dick A.C. Quartel, BiZZdesign
- Serge Thorn, Metaplexity Fellow

The Open Group and ArchiMate project team would like to thank in particular the following individuals for their support and review of this and earlier versions of this standard:

- Adina Aldea
- Mary Beijleveld
- Alexander Bielowski
- Remco de Boer
- Adrian Campbell
- John Coleshaw
- Jörgen Dahlberg
- Garry Doherty
- Ingvar Elmér
- Wilco Engelsman
- Roland Ettema

- Henry M. Franken
- Mats Gejnevall
- Sonia González
- Kirk Hansen
- Jos van Hillegersberg
- Andrew Josey
- Ryan Kennedy
- Louw Labuschagne
- Antoine Lonjon
- Veer Muchandi
- Michelle Nieuwoudt
- Erwin Oord
- Carlo Poli
- G. Edward Roberts
- Jean-Baptiste Sarrodie
- Daniel Simon
- Gerben Wierda
- Egon Willemsz

The first version of this Open Group standard was largely produced by the ArchiMate project. The Open Group gratefully acknowledges the contribution of the many people – former members of the project team – who have contributed to it.

The ArchiMate project comprised the following organizations:

- ABN AMRO
- Centrum voor Wiskunde en Informatica
- Dutch Tax and Customs Administration
- Leiden Institute of Advanced Computer Science
- Novay
- Ordina
- Radboud Universiteit Nijmegen
- Stichting Pensioenfonds ABP

Referenced Documents

The following documents are referenced in this standard. These references are informative.

(Please note that the links below are good at the time of writing but cannot be guaranteed for the future.)

[1] Enterprise Architecture at Work: Modeling, Communication, and Analysis, Third Edition, M.M. Lankhorst et al., Springer, 2013.

[2] The Anatomy of the ArchiMate® Language, M.M. Lankhorst, H.A. Proper, H. Jonkers, International Journal of Information Systems Modeling and Design (IJISMD), 1(1):1-32, January-March 2010.

[3] Extending Enterprise Architecture Modeling with Business Goals and Requirements, W. Engelsman, D.A.C. Quartel, H. Jonkers, M.J. van Sinderen, Enterprise Information Systems, 5(1):9-36, 2011.

[4] TOGAF® Version 9.1, an Open Group Standard (G116), December 2011, published by The Open Group; refer to: www.opengroup.org/bookstore/catalog/g116.htm.

[5] Extending and Formalizing the Framework for Information Systems Architecture, J.F. Sowa, J.A. Zachman, IBM Systems Journal, Volume 31, No. 3, pp.590-616, 1992.

[6] TOGAF® Framework and ArchiMate® Modeling Language Harmonization: A Practitioner's Guide to Using the TOGAF® Framework and the ArchiMate® Language, White Paper (W14C), December 2014, published by The Open Group; refer to: www.opengroup.org/bookstore/catalog/w14c.htm.

[7] Unified Modeling Language®: Superstructure, Version 2.0 (formal/05-07-04), Object Management Group, August 2005.

[8] Unified Modeling Language®: Infrastructure, Version 2.4.1 (formal/201-08-05), Object Management Group, August 2011.

[9] A Business Process Design Language, H. Eertink, W. Janssen, P. Oude Luttighuis, W. Teeuw, C. Vissers, in Proceedings of the First World Congress on Formal Methods, Toulouse, France, September 1999.

[10] Enterprise Business Architecture: The Formal Link between Strategy and Results, R. Whittle, C.B. Myrick, CRC Press, 2004.

[11] Composition of Relations in Enterprise Architecture, R. van Buuren, H. Jonkers, M.E. Iacob, P. Strating, in Proceedings of the Second International Conference on Graph Transformation, pp.39-53, Edited by H. Ehrig et al., Rome, Italy, 2004.

[12] Business Process Modeling Notation™ (BPMN™), Version 2.0 (formal/2011-01-03), Object Management Group, 2011.

[13] Performance and Cost Analysis of Service-Oriented Enterprise Architectures, H. Jonkers, M.E. Iacob, in Global Implications of Modern Enterprise Information Systems: Technologies and Applications, Edited by A. Gunasekaran, IGI Global, 2009.

[14] ISO/IEC 42010:2011, Systems and Software Engineering – Recommended Practice for Architectural Description of Software-Intensive Systems, Edition 1.

[15] Business Motivation Model (BMM), Version 1.1 (formal/2010-05-01), Object Management Group, 2010.

[16] Using the ArchiMate® Language with UML®, White Paper (W134), September 2013, published by The Open Group; refer to: www.opengroup.org/bookstore/catalog/w134.htm.

1 Introduction

1.1 Objective

This standard is the specification of the ArchiMate Enterprise Architecture modeling language, a visual language with a set of default iconography for describing, analyzing, and communicating many concerns of Enterprise Architectures as they change over time. The standard provides a set of entities and relationships with their corresponding iconography for the representation of Architecture Descriptions.

1.2 Overview

An Enterprise Architecture is typically developed because key people have concerns that need to be addressed by the business and IT systems within an organization. Such people are commonly referred to as the "stakeholders" of the Enterprise Architecture. The role of the architect is to address these concerns by identifying and refining the motivation and strategy expressed by stakeholders, developing an architecture, and creating views of the architecture that show how it addresses and balances stakeholder concerns. Without an Enterprise Architecture, it is unlikely that all concerns and requirements are considered and addressed.

The ArchiMate Enterprise Architecture modeling language provides a uniform representation for diagrams that describe Enterprise Architectures. It includes concepts for specifying inter-related architectures, specific viewpoints for selected stakeholders, and language customization mechanisms. It offers an integrated architectural approach that describes and visualizes different architecture domains and their underlying relations and dependencies. Its language framework provides a structuring mechanism for architecture domains, layers, and aspects. It distinguishes between the model elements and their notation, to allow for varied, stakeholder-oriented depictions of architecture information. The language uses service-orientation to distinguish and relate the Business, Application, and Technology Layers of Enterprise Architectures, and uses realization relationships to relate concrete elements to more abstract elements across these layers.

1.3 Conformance

The ArchiMate language may be implemented in software used for Enterprise Architecture modeling. For the purposes of this standard, the conformance requirements for implementations of the language given in this section apply. A conforming implementation:

1. Shall support the language structure, generic metamodel, relationships, layers, cross-layer dependencies, and other elements as specified in Chapter 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, and 13

2. Shall support the standard iconography as specified in Chapters 5, 6, 7, 8, 9, 10, 11, and 13, and summarized in Appendix A

3. Shall support the viewpoint mechanism as specified in Chapter 14
4. Shall support the language customization mechanisms specified in Chapter 15 in an implementation-defined manner
5. Shall support at least the relationships between elements as specified in Appendix B
6. May support the example viewpoints described in Appendix C

Readers are advised to check The Open Group website for additional conformance and certification requirements referencing this standard.

1.4 Normative References

None.

1.5 Terminology

For the purposes of this standard, the following terminology definitions apply:

Can — Describes a possible feature or behavior available to the user.

Deprecated — Items identified as deprecated may be removed in the next version of this standard.

Implementation-defined — Describes a value or behavior that is not defined by this standard but is selected by an implementor of a software tool. The value or behavior may vary among implementations that conform to this standard. A user should not rely on the existence of the value or behavior. The implementor shall document such a value or behavior so that it can be used correctly by a user.

May — Describes a feature or behavior that is optional. To avoid ambiguity, the opposite of "may" is expressed as "need not", instead of "may not".

Obsolescent — Certain features are obsolescent, which means that they may be considered for withdrawal in future versions of this standard. They are retained because of their widespread use, but their use is discouraged.

Shall — Describes a feature or behavior that is a requirement. To avoid ambiguity, do not use "must" as an alternative to "shall".

Shall not — Describes a feature or behavior that is an absolute prohibition.

Should — Describes a feature or behavior that is recommended but not required.

Will — Same meaning as "shall"; "shall" is the preferred term.

1.6 Future Directions

None.

2 Definitions

For the purposes of this standard, the following terms and definitions apply. The TOGAF framework [4] should be referenced for Enterprise Architecture-related terms not defined in this section. Merriam-Webster's Collegiate Dictionary (11th Edition) should be referenced for all other terms not defined in this section.

Any conflict between definitions described here and the TOGAF framework is unintentional. If the definition of a term is specific to the ArchiMate modeling language, and a general definition is defined by the TOGAF framework, then this is noted in the definition.

2.1 ArchiMate Core Framework

A reference structure used to classify elements of the ArchiMate core language. It consists of three layers and three aspects,

Note: The ArchiMate Core Framework is defined in detail in Section 3.4.

2.2 ArchiMate Core Language

The central part of the ArchiMate language that defines the concepts and relationships to model Enterprise Architectures. It includes three layers: Business, Application, and Technology.

2.3 Aspect

Classification of elements based on layer-independent characteristics related to the concerns of different stakeholders. Used for positioning elements in the ArchiMate metamodel. See also Section 2.6.

Note: Aspects are described in Section 3.4.

2.4 Attribute

A property associated with an ArchiMate language element or relationship.

2.5 Concept

Either an element or a relationship. See also Section 2.10 and Section 2.12.

Note: The top-level language structure is defined in detail in Section 3.2.

2.6 Conformance

Fulfillment of specified requirements.

2.7 Conforming Implementation

An implementation which satisfies the conformance requirements defined by the conformance clause of this standard. See Section 1.3.

2.8 Core Element

A structure or behavior element in one of the core layers of the ArchiMate language.

Note: Core elements are described in detail in Section 4.1.

2.9 Composite Element

An element consisting of other elements from multiple aspects or layers of the language.

2.10 Element

Basic unit in the ArchiMate metamodel. Used to define and describe the constituent parts of Enterprise Architectures and their unique set of characteristics.

2.11 Layer

An abstraction of the ArchiMate framework at which an enterprise can be modeled.

2.12 Model

A collection of concepts in the context of the ArchiMate language structure.

Note: The top-level language structure is defined in detail in Section 3.2.

For a general definition of model, see the TOGAF framework [4].

2.13 Relationship

A connection between a source and target concept. Classified as structural, dependency, dynamic, or other.

Note: Relationships are defined in detail in Chapter 5.

3 Language Structure

This chapter describes the structure of the ArchiMate Enterprise Architecture modeling language. The detailed definition and examples of its standard set of elements and relationships follow in Chapter 4 to Chapter 13.

3.1 Language Design Considerations

A key challenge in the development of a general metamodel for Enterprise Architecture is to strike a balance between the specificity of languages for individual architecture domains, and a very general set of architecture concepts, which reflects a view of systems as a mere set of inter-related entities.

The design of the ArchiMate language started from a set of relatively generic concepts. These have been specialized towards application at different architectural layers, as explained in the following sections. The most important design restriction on the language is that it has been explicitly designed to be as small as possible, but still usable for most Enterprise Architecture modeling tasks. Many other languages try to accommodate all needs of all possible users. In the interest of simplicity of learning and use, the ArchiMate language has been limited to the concepts that suffice for modeling the proverbial 80% of practical cases.

This standard does not describe the detailed rationale behind the design of the ArchiMate language. The interested reader is referred to [1], [2], and [3], which provide a detailed description of the language construction and design considerations.

3.2 Top-Level Language Structure

Figure 1 outlines the top-level hierarchical structure of the language:

- A model is a collection of *concepts*. A concept is either an *element* or a *relationship*.
- An element is either a behavior element, a structure element, a motivation element, or a composite element.

Note that these are *abstract* concepts; they are not intended to be used directly in models. To signify this, they are depicted in white with labels in italics. Further note that implementation and migration elements (Chapter 13) are instances of core elements.

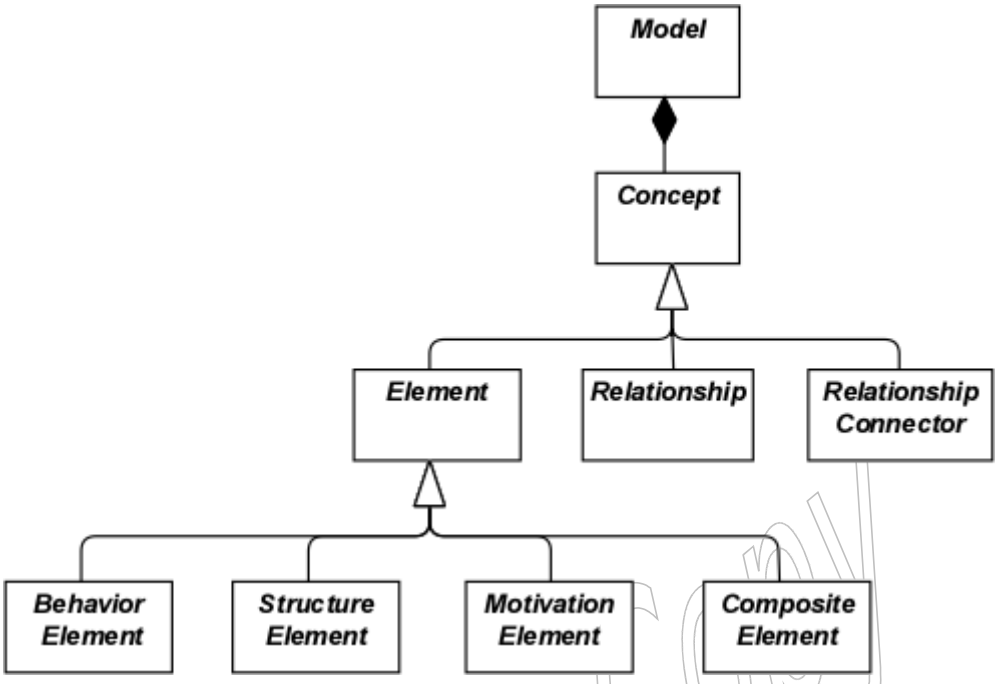


Figure 1: Top-Level Hierarchy of ArchiMate Concepts

3.3 Layering of the ArchiMate Language

The ArchiMate core language defines a structure of generic elements and their relationships, which can be specialized in different layers. Three layers are defined within the ArchiMate core language as follows:

1. The *Business Layer* depicts business services offered to customers, which are realized in the organization by business processes performed by business actors.
2. The *Application Layer* depicts application services that support the business, and the applications that realize them.
3. The *Technology Layer* depicts technology services such as processing, storage, and communication services needed to run the applications, and the computer and communication hardware and system software that realize those services. Physical elements are added for modeling physical equipment, materials, and distribution networks to this layer.

The general structure of models within the different layers is similar. The same types of elements and relationships are used, although their exact nature and granularity differ. In the next chapter, the structure of the generic metamodel is presented. In Chapter 6, Chapter 9, and Chapter 10, these elements are specialized to obtain elements specific to a particular layer.

In alignment with service-orientation, the most important relationship between layers is formed by "serving"[1] relationships, which show how the elements in one layer are served by the services of other layers. (Note, however, that services need not only serve elements in another layer, but

[1] Note that this was called 'used by' in previous versions of the standard. For the sake of clarity, this name has been changed to 'serving'.

also can serve elements in the same layer.) A second type of link is formed by realization relationships: elements in lower layers may realize comparable elements in higher layers; e.g., a "data object" (Application Layer) may realize a "business object" (Business Layer); or an "artifact" (Technology Layer) may realize either a "data object" or an "application component" (Application Layer).

3.4 The ArchiMate Core Framework

The *aspects* of the core, as defined by the three types of element at the bottom of Figure 1, combined with the layers identified in the previous section, make up a framework of nine cells, as illustrated in Figure 2. This is known as the ArchiMate Core Framework.

It is important to understand that the classification of elements based on aspects and layers is only a global one. It is impossible to define a strict boundary between the aspects and layers, because elements that link the different aspects and layers play a central role in a coherent architectural description. For example, running somewhat ahead of the later conceptual discussions, (business) functions and (business) roles serve as intermediary elements between "purely behavioral" elements and "purely structural" elements, and it may depend on the context whether a certain piece of software is considered to be part of the Application Layer or the Technology Layer.

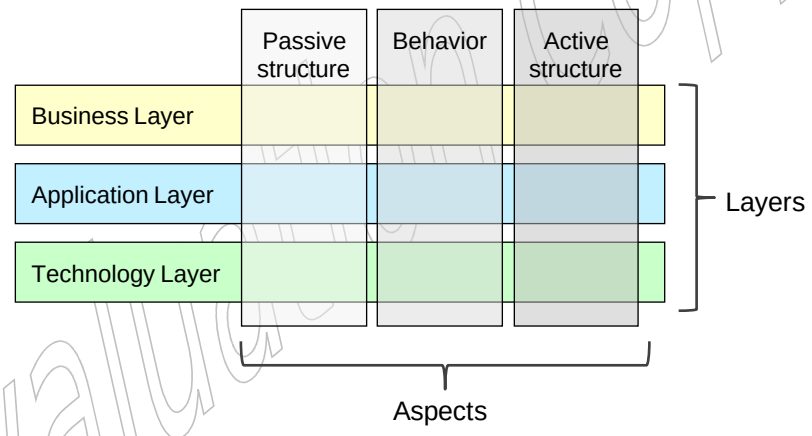


Figure 2: ArchiMate Core Framework

The structure of the framework allows for modeling of the enterprise from different viewpoints, where the position within the cells highlights the concerns of the stakeholder. A stakeholder typically can have concerns that cover multiple cells.

The dimensions of the framework are as follows:

- Layers: The three levels at which an enterprise can be modeled in ArchiMate – Business, Application, and Technology (as described in Section 3.3).
- Aspects:
  - The *Active Structure Aspect*, which represents the structural elements (the business actors, application components, and devices that display actual behavior; i.e., the "subjects" of activity).

— The *Behavior Aspect*, which represents the behavior (processes, functions, events, and services) performed by the actors. Structural elements are assigned to behavioral elements, to show who or what displays the behavior.

— The *Passive Structure Aspect*, which represents the objects on which behavior is performed. These are usually information objects in the Business Layer and data objects in the Application Layer, but they may also be used to represent physical objects.

A composite element, as shown in Figure 1, is an element that does not necessarily fit in a single aspect (column) of the framework, but may combine two or more aspects.

Note that the ArchiMate language does not require the modeler to use any particular layout such as the structure of this framework; it is merely a categorization of the language elements.

3.5 Full Framework

The full ArchiMate language, as described in this version of the standard, adds a number of layers and an aspect to the framework. The physical elements build upon the Technology Layer and add elements for modeling physical facilities and equipment, distribution networks, and materials. It is described in Chapter 11. The motivation aspect is introduced at a generic level in the next chapter and described in detail in Chapter 6. The implementation and migration elements are described in Chapter 13. The resulting full ArchiMate framework is shown in Figure 3.

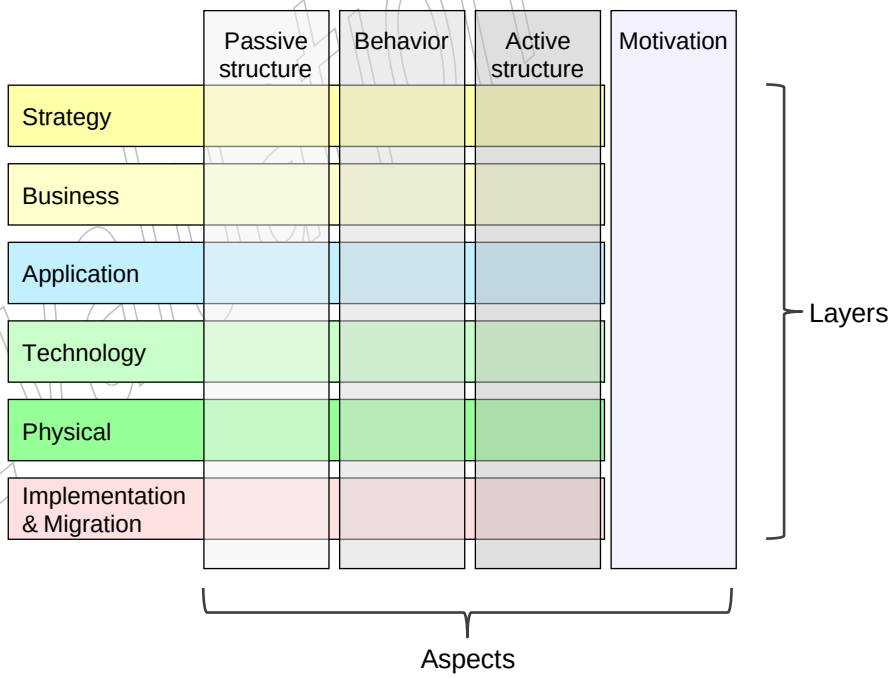


Figure 3: Full ArchiMate Framework

The ArchiMate language does not define a specific layer for information; however, elements from the passive structure aspect such as business objects, data objects, and technology objects are used to represent information entities. Information modeling is supported across the different ArchiMate layers.

3.6 Abstraction in the ArchiMate Language

The structure of the ArchiMate language accommodates several familiar forms of abstraction and refinement. First of all, the distinction between an external (black-box, abstracting from the contents of the box) and internal (white-box) view is common in systems design. The external view depicts what the system has to do for its environment, while the internal view depicts how it does this.

Second, the distinction between behavior and active structure is commonly used to separate what the system must do and how the system does it from the system constituents (people, applications, and infrastructure) that do it. In modeling new systems, it is often useful to start with the behaviors that the system must perform, while in modeling existing systems, it is often useful to start with the people, applications, and infrastructure that comprise the system, and then analyze in detail the behaviors performed by these active structures.

A third distinction is between conceptual, logical, and physical abstraction levels. This has its roots in data modeling: conceptual elements represent the information the business finds relevant; logical elements provide logical structure to this information for manipulation by information systems; physical elements describe the storage of this information; for example, in the form of files or database tables. In the ArchiMate language, this corresponds with business objects, data objects, and artifacts, and the realization relationships between them.

The distinction between logical and physical elements has also been carried over to the description of applications. The TOGAF Content Metamodel [4] describes logical and physical data, application, and technology components. Logical components are implementation or product-independent encapsulations of data or functionality, whereas physical components are tangible software components, devices, etc. The distinction within the TOGAF framework between Architecture Building Blocks (ABBs) and Solution Building Blocks (SBBs) is very similar. This distinction is again useful in progressing Enterprise Architectures from high-level, abstract descriptions to tangible, implementation-level designs. Note that building blocks may contain multiple elements, which are typically modeled using the grouping notation in the ArchiMate language.

The ArchiMate language has three ways of modeling such abstractions. First, as described in [6], behavior elements such as application and technology functions can be used to model logical components, since they represent implementation-independent encapsulations of functionality. The corresponding physical components can then be modeled using active structure elements such as application components and nodes, assigned to the behavior elements. Second, the ArchiMate language supports the concept of realization. This can best be described by working with the Technology Layer upwards. The Technology Layer defines the physical artifacts and software that realize an application component. It also provides a mapping to other physical concepts such as devices, networks, etc., needed for the realization of an information system. The realization relationship is also used to model more abstract kinds of realization, such as that between a (more specific) requirement and a (more generic) principle, where fulfillment of the requirement implies adherence to the principle. Third, logical and physical application components can be defined as specializations of the application component element, as described in Chapter 15 (see also the examples in Section 15.2.2). The same holds for the logical and physical technology components of the TOGAF Content Metamodel, which can be defined as specializations of the node element (see Section 15.2.3).

The ArchiMate language intentionally does not support a difference between types and instances. At the Enterprise Architecture abstraction level, it is more common to model types

and/or exemplars rather than instances. Similarly, a business process in the ArchiMate language does not describe an individual instance (i.e., one execution of that process). In most cases, a business object is therefore used to model an object type (*cf.* a UML class), of which several instances may exist within the organization. For instance, each execution of an insurance application process may result in a specific instance of the insurance policy business object, but that is not modeled in the Enterprise Architecture.

3.7 Concepts and their Notation

The ArchiMate language separates the language concepts (i.e., the constituents of the metamodel) from their notation. Different stakeholder groups may require different notations in order to understand an architecture model or view. In this respect, the ArchiMate language differs from languages such as UML or BPMN, which have only one standardized notation. The viewpoint mechanism explained in Chapter 14 provides the means for defining such stakeholder-oriented visualizations.

Although the notation of the ArchiMate concepts can (and should) be stakeholder-specific, the standard provides one common graphical notation, which can be used by architects and others who develop ArchiMate models. This notation is targeted towards an audience used to existing technical modeling techniques such as ERD, UML, or BPMN, and therefore resembles them. In the remainder of this document, unless otherwise noted, the symbols used to depict the language concepts represent the ArchiMate standard notation. This standard notation for most elements consists of a box with an icon in the upper-right corner. In several cases, this icon by itself may also be used as an alternative notation. This standard iconography should be preferred whenever possible so that anyone knowing the ArchiMate language can read the diagrams produced in the language.

3.8 Use of Nesting

Nesting elements inside other elements can be used as an alternative graphical notation to express structural relationships. This is explained in more detail in Section 5.1 and in the definition of each of these relationships.

3.9 Use of Colors and Notational Cues

In the metamodel pictures within this standard, shades of grey are used to distinguish elements belonging to the different aspects of the ArchiMate framework, as follows:

- White for abstract (i.e., non-instantiable) concepts
- Light grey for passive structures
- Medium grey for behavior
- Dark grey for active structures

In ArchiMate models, there are no formal semantics assigned to colors and the use of color is left to the modeler. However, they can be used freely to stress certain aspects in models. For instance, in many of the example models presented in this standard, colors are used to distinguish between the layers of the ArchiMate Core Framework, as follows:

- Yellow for the Business Layer
- Blue for the Application Layer
- Green for the Technology Layer

They can also be used for visual emphasis. A recommended text providing guidelines is Chapter 6 of [1].

In addition to the colors, other notational cues can be used to distinguish between the layers of the framework. A letter 'M', 'S', 'B', 'A', 'T', 'P', or 'I' in the top-left corner of an element can be used to denote a Motivation, Strategy, Business, Application, Technology, Physical, or Implementation & Migration element, respectively. An example of this notation is depicted in Example 33.

The standard notation also uses a convention with the shape of the corners of its symbols for different element types, as follows:

- Square corners are used to denote structure elements.
- Round corners are used to denote behavior elements.
- Diagonal corners are used to denote motivation elements.

4 Generic Metamodel

4.1 Behavior and Structure Elements

The main hierarchy of behavior and structure elements of the ArchiMate language is presented in the metamodel fragment of Figure 4. It defines these elements in a generic, layer-independent way. Note that most of these elements (the white boxes) are *abstract* metamodel elements; i.e., these are not instantiated in models but only serve to structure the metamodel. The notation presented in this chapter is therefore the generic way in which the specializations of these elements (i.e., the elements of the different architecture layers) are depicted. The concrete elements (the gray boxes), which can be used to model the Enterprise Architecture at a strategic level, are described in more detail in Chapter 7.

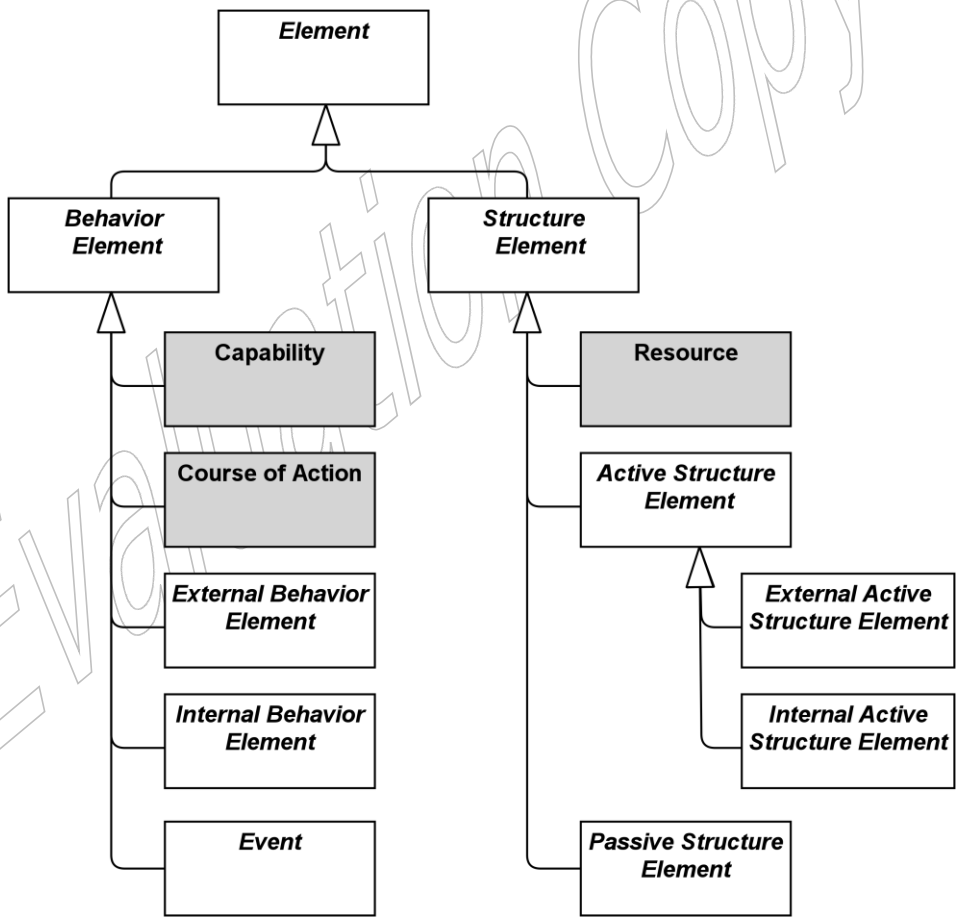


Figure 4: Hierarchy of Behavior and Structure Elements

This generic metamodel fragment consists of two main types of elements: *structure* ('nouns') and *behavior* elements ('verbs').

Structure elements are the strategic element *resource*, and structural elements, which can be subdivided into *active structure* elements and *passive structure* elements. Active structure elements can be further subdivided into *external* active structure elements (also called *interfaces*) and *internal* active structure elements.

Behavior elements are the strategic elements *course of action* and *capability*, and behavioral elements, which can be subdivided into *internal behavior* elements, *external behavior* elements (also called *services*), and *events*.

These three aspects – active structure, behavior, and passive structure – have been inspired by natural language, where a sentence has a subject (active structure), a verb (behavior), and an object (passive structure).

Figure 5 specifies the main relationships between the behavior and structure elements defined above. For an explanation of the different types of relationships, see Chapter 5. In this and other metamodel figures, the label of a relationship signifies the role of the source element in the relationship; e.g., a service serves an internal behavior element.

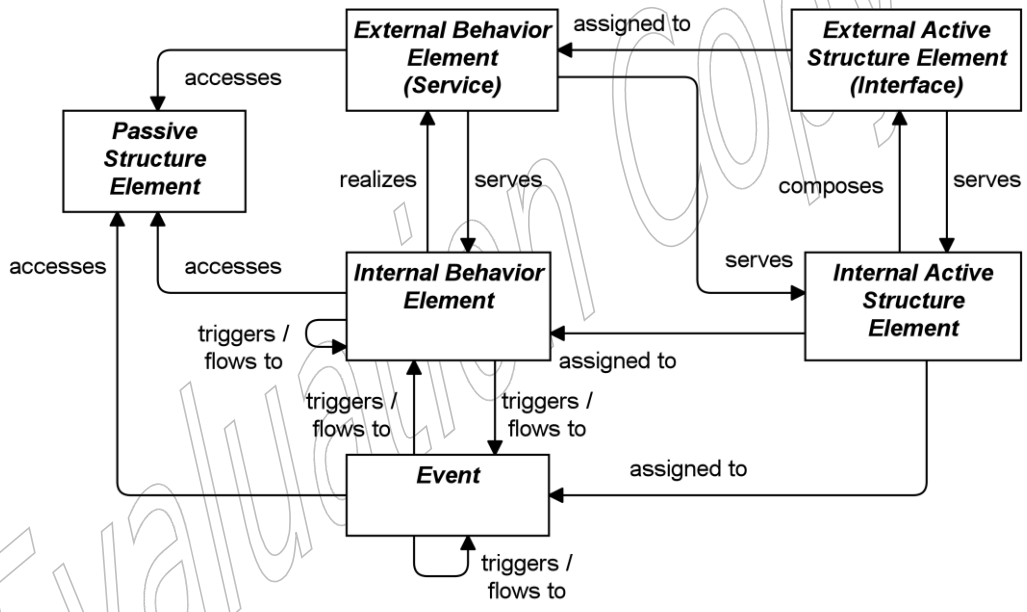


Figure 5: Behavior and Structure Elements Metamodel

Note: This figure does not show all permitted relationships: every element in the language can have composition, aggregation, and specialization relationships to elements of the same type. Furthermore, there are indirect relationships that can be derived, as explained in Section 5.6. The full specification of permitted relationships can be found in Appendix B.

4.1.1 Active Structure Elements

Active structure elements are the subjects that can perform behavior. These can be subdivided into internal active structure elements; i.e., the business actors, application components, nodes, etc., that realize this behavior, and external active structure elements; i.e., the interfaces that expose this behavior to the environment. An interface provides an external view on the service provider and hides its internal structure.

An internal active structure element represents an entity that is capable of performing behavior.

An external active structure element, called an interface, represents a point of access where one or more services are provided to the environment.

Active structure elements are denoted using boxes with square corners and an icon in the upper-right corner, or by the icon on its own.

Figure 6: Generic Active Structure Elements Notation

4.1.2 Behavior Elements

Behavior elements represent the dynamic aspects of the enterprise. Similar to active structure elements, behavior elements can be subdivided into *internal* behavior elements and *external* behavior elements; i.e., the services that are exposed to the environment.

An internal behavior element represents a unit of activity performed by one or more active structure elements.

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

Behavior elements are denoted in the standard iconography using boxes with round corners and an icon in the upper-right corner, or by the icon on its own.

Figure 7: Generic Behavior Elements Notation

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

In addition to this, a third type of behavior element is defined to denote an event that can occur; for example, to signal a state change.

An event is a behavior element that denotes a state change.

An event may have a time attribute that indicates the moment or moments at which the event happens. For example, this can be used to model time schedules.

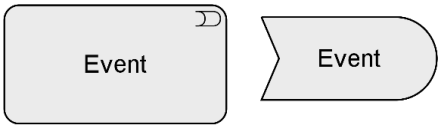


Figure 8: Generic Event Notation

4.1.3 Passive Structure Elements

Passive structure elements can be accessed by behavior elements.

A passive structure element is a structural element that cannot perform behavior. Active structure elements can perform behavior on passive structure elements.

Passive structure elements are often information or data objects, but they can also represent physical objects.

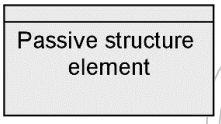


Figure 9: Generic Passive Structure Element Notation

4.2 Specializations of Structure and Behavior Elements

Going one level deeper in the structure of the language, the collective nature of a behavior can be made either implicit (several active structure elements assigned to the same internal behavior) or explicit through the use of a collective internal behavior (interaction) that is performed by (a collaboration of) multiple active structure elements.

A collaboration is an aggregate of two or more active structure elements, working together to perform some collective behavior.

This collective internal behavior can be modeled as an interaction.

An interaction is a unit of collective behavior performed by (a collaboration of) two or more active structure elements.

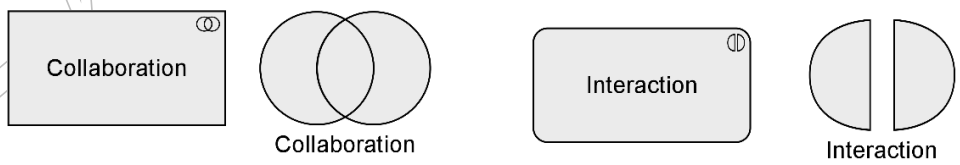


Figure 10: Generic Collaboration and Interaction Notation

Furthermore, for individual internal behavior elements, a distinction is made between processes and functions.

A process represents a sequence of behaviors that achieves a specific outcome.

A function represents a collection of behavior based on specific criteria, such as required resources, competences, or location.

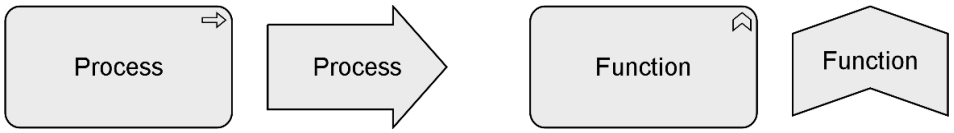


Figure 11: Generic Process and Function Notation

The specializations of core elements are summarized in Figure 12. Within each layer, it is permitted to use composition and aggregation relationships between processes, functions, and interactions; e.g., a process can be composed of other processes, functions, and/or interactions.

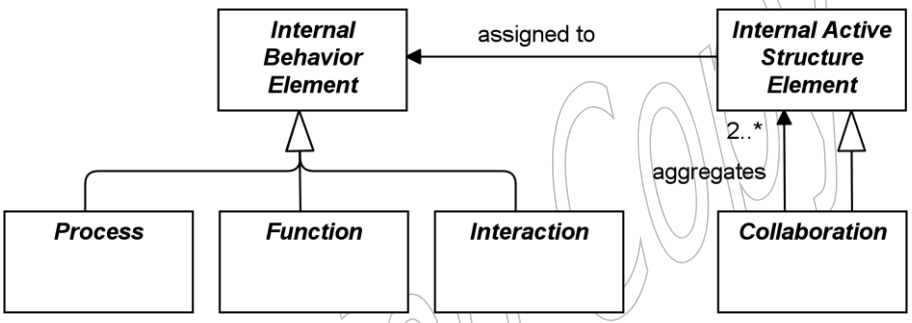
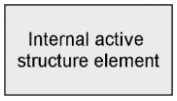
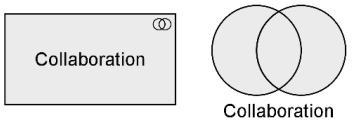
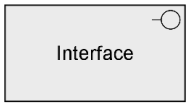
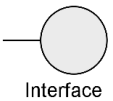


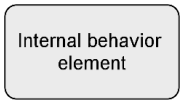
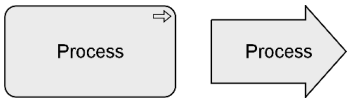
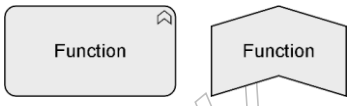
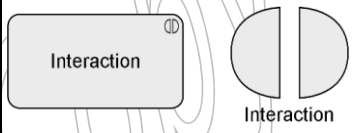
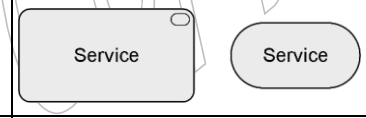
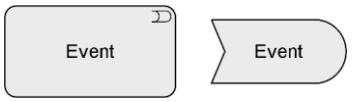
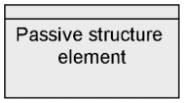
Figure 12: Specializations of Core Elements

4.3 Summary of Structure and Behavior Elements

Table 1 gives an overview of the core elements, their definitions, and their default graphical notation. But note that most of these elements are abstract; they are not used in models but only their descendants in the different layers of the ArchiMate language.

Table 1: Core Elements

Element	Specializations	Definition	Notation
Active Structure			
Internal active structure element		An entity that is capable of performing behavior.	
	Collaboration	An aggregate of two or more active structure elements, working together to perform some collective behavior.	
Interface		A point of access where one or more services are exposed available to the environment.	 

Element	Specializations	Definition	Notation
Behavior			
Internal behavior element		A unit of activity performed by one or more active structure elements.	
	Process	A sequence of behaviors that achieves a specific outcome.	
	Function	A collection of behavior based on specific criteria, such as required resources, competences, or location.	
	Interaction	A unit of collective behavior performed by (a collaboration of) two or more structure elements.	
Service		An explicitly defined exposed behavior.	
Event		A state change.	
Passive Structure			
Passive structure element		An element on which behavior is performed.	

4.4 Motivation Elements

The core elements of the ArchiMate language focus on describing the architecture of systems that support the enterprise. They do not cover the elements which, in different ways, *drive* the design and operation of the enterprise. These motivation aspects correspond to the "Why" column of the Zachman framework [5].

Several *motivation elements* are included in the language: stakeholder, value, meaning, driver, assessment, goal, outcome, principle, and requirement, which in turn has constraint as a subtype. In this section, the generic motivation element is introduced. The more specific motivation elements are described in Chapter 6.

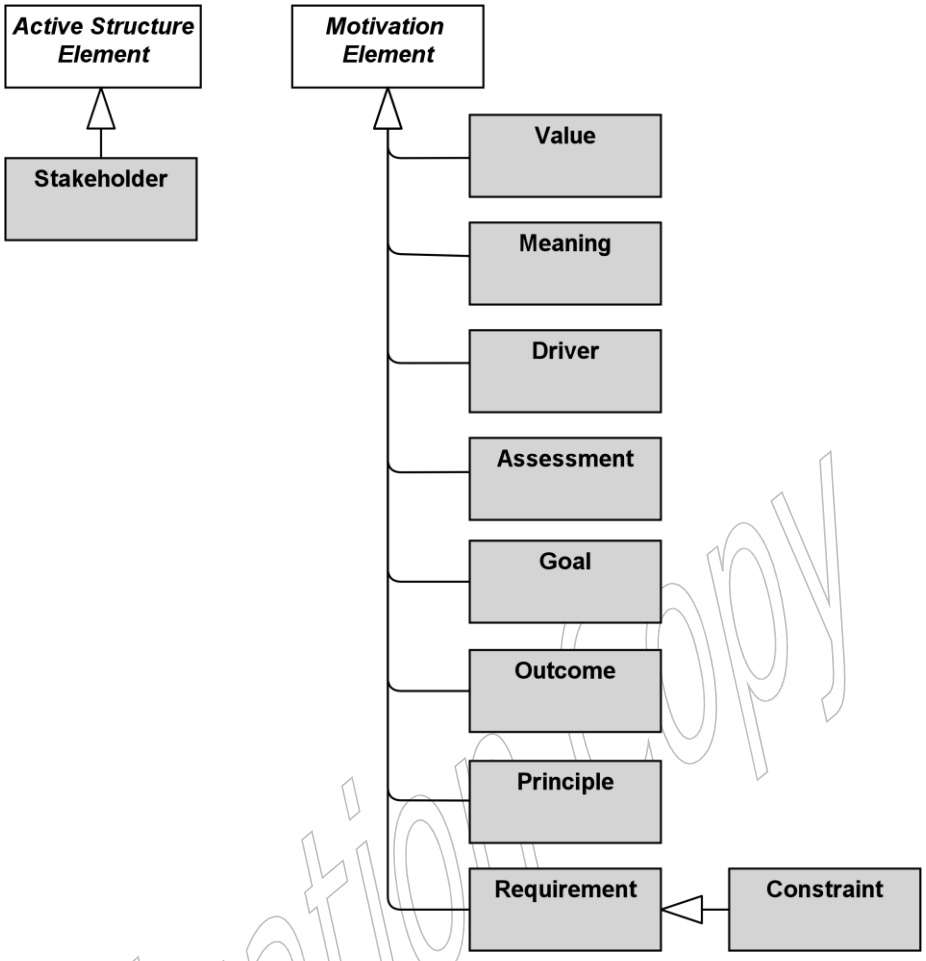


Figure 13: Overview of Motivation Elements

The motivation elements address the way the Enterprise Architecture is aligned to its context, as described by these intentions.

A motivation element is an element that provides the context of or reason behind the architecture of an enterprise.

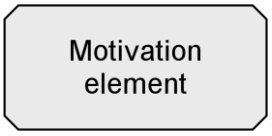
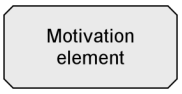


Figure 14: Generic Motivation Element Notation

Motivation elements are usually denoted using boxes with diagonal corners.

Table 2: Motivation Element

Element	Definition	Notation
Motivation element	An element that provides the context of or reason behind the architecture of an enterprise.	

4.5 Strategy Elements

Next to the motivation elements described in the previous section, the language also includes a number of *strategy elements*, notably capability, resource, and course of action, as exhibited in Figure 4. These are defined as specializations of the generic behavior and structure elements and are defined in more detail in Chapter 7.

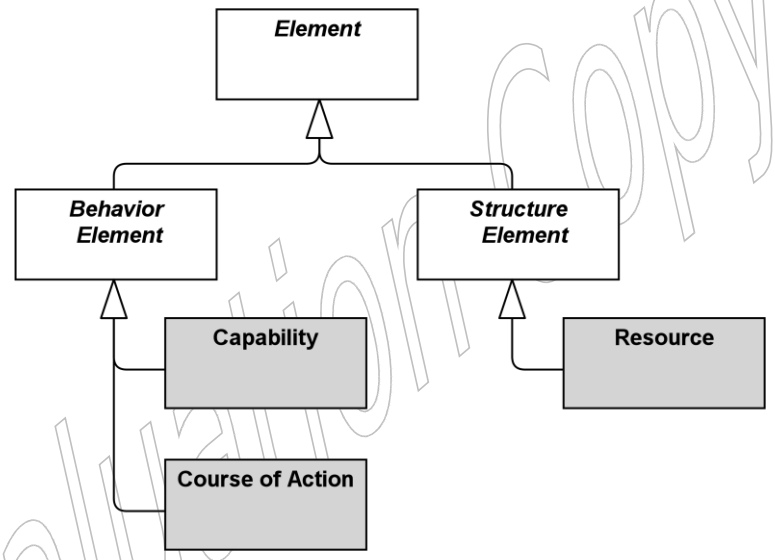


Figure 15: Strategy Elements

4.6 Composite Elements

Composite elements consist of other concepts, possibly from multiple aspects or layers of the language. Grouping and location are generic composite elements (see Figure 16). Composite elements can themselves aggregate or compose other composite elements.

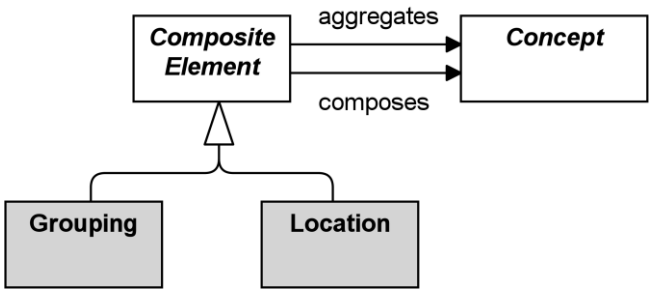


Figure 16: Composite Elements

4.6.1 Grouping

The grouping element aggregates or composes concepts that belong together based on some common characteristic.

The grouping element is used to aggregate or compose an arbitrary group of concepts, which can be elements and/or relationships of the same or of different types. An aggregation or composition relationship is used to link the grouping element to the grouped concepts.

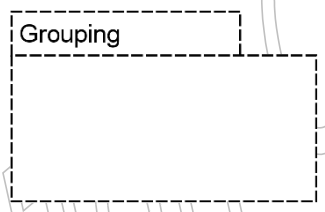


Figure 17: Grouping Notation

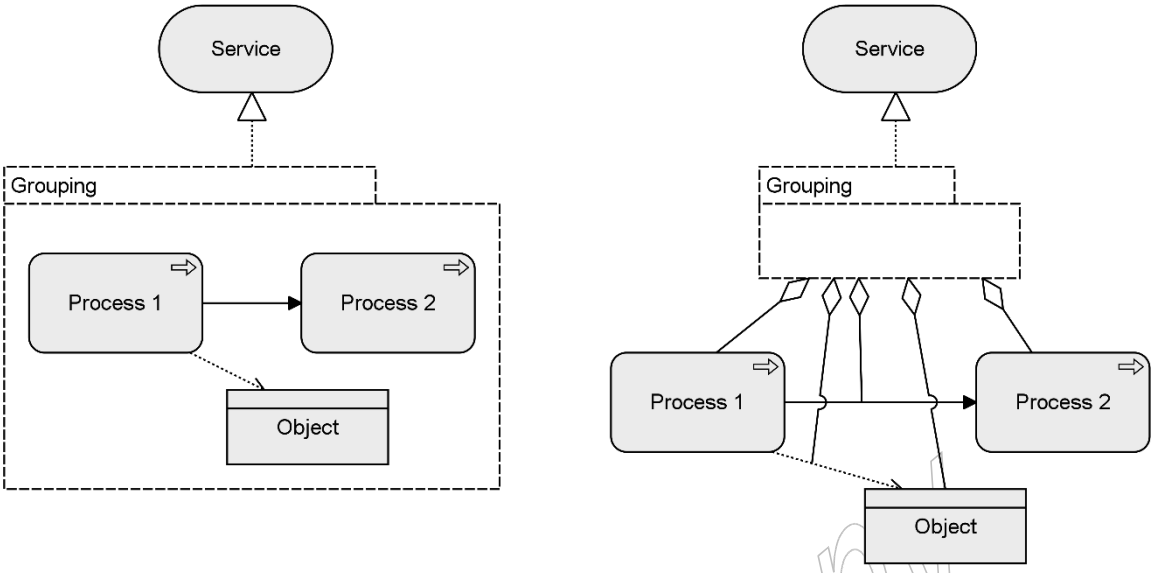
Concepts may be aggregated by multiple (overlapping) groups.

One useful way of employing grouping is for modeling Architecture and Solution Building Blocks (ABBs and SBBs), as described in the TOGAF framework [4].

Another useful application of grouping is for modeling domains. For example, the TOGAF framework Glossary of Supplementary Definition (Section A.40) defines Information Domain as: *"grouping of information (or data entities) by a set of criteria such as security classification, ownership, location, etc. In the context of security, Information Domains are defined as a set of users, their information objects, and a security policy"*.

Example

In the model below, the Grouping element is used to aggregate a conglomerate of two processes and an object that together realize a service (both with nesting and explicitly drawn aggregation relationships).



Example 1: Grouping

4.6.2 Location

A location is a place or position where structure elements can be located or behavior can be performed.

The location element is used to model the places where (active and passive) structure elements such as business actors, application components, and devices are located. This is modeled by means of an aggregation relationship from a location to structure element. A location can also aggregate a behavior element, to indicate where the behavior is performed. This element corresponds to the "Where" column of the Zachman framework [5].

Figure 18: Location Notation

5 Relationships

In addition to the generic elements outlined in Chapter 4, the ArchiMate language defines a core set of generic relationships, each of which can connect a predefined set of source and target concepts (in most cases elements, but in a few cases also other relationships). Many of these relationships are 'overloaded'; i.e., their exact meaning differs depending on the source and destination concepts that they connect.

The relationships are classified as follows (see Figure 19):

- *Structural* relationships, which model the static construction or composition of concepts of the same or different types
- *Dependency* relationships, which model how elements are used to support other elements
- *Dynamic* relationships, which are used to model behavioral dependencies between elements
- *Other* relationships, which do not fall into one of the above categories

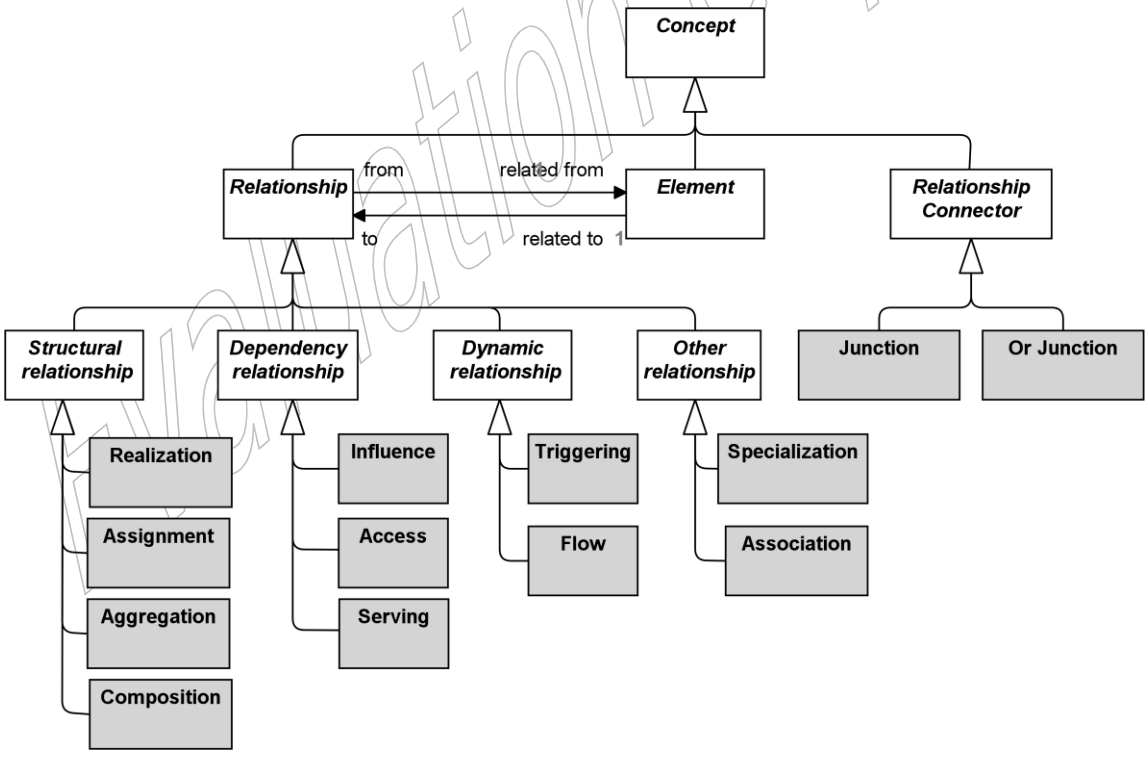


Figure 19: Overview of Relationships

Figure 19 shows that each relationship has exactly one 'from' and one 'to' concept (element or relationship) as endpoints. It does not show which types of concepts are permitted as endpoints. For the sake of readability, the metamodel figures throughout this document do not show all

possible relationships in the language. Section 5.6 describes a set of derivation rules to derive indirect relationships between elements in a model. Aggregation, composition, and specialization relationships are always permitted between two elements of the same type, and association is always allowed between any two elements, and between any element and relationship. The exact specification of permitted relationships is given in Appendix B.

5.1 Structural Relationships

Structural relationships represent the 'static' coherence within an architecture. The composing concept (the 'from' side of the relationship) is always an element; the composed concept (the 'to' side of the relationship) may in some cases also be another relationship.

As an alternative to the graphical notations proposed in this section, structural relationships may also be expressed by means of nesting of the composed concept within the composing element. Note, however, that this can lead to ambiguous models, in case multiple structural relationships are allowed between these elements.

5.1.1 Composition Relationship

The composition relationship indicates that an element consists of one or more other elements.

The composition relationship has been inspired by the composition relationship in UML class diagrams. In contrast to the aggregation relationship, the composed concept can be part of only one composition.

A composition relationship is always allowed between two instances of the same element type.

In addition to this, the metamodel explicitly defines other source and target elements that may be connected by a composition relationship.

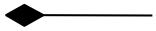
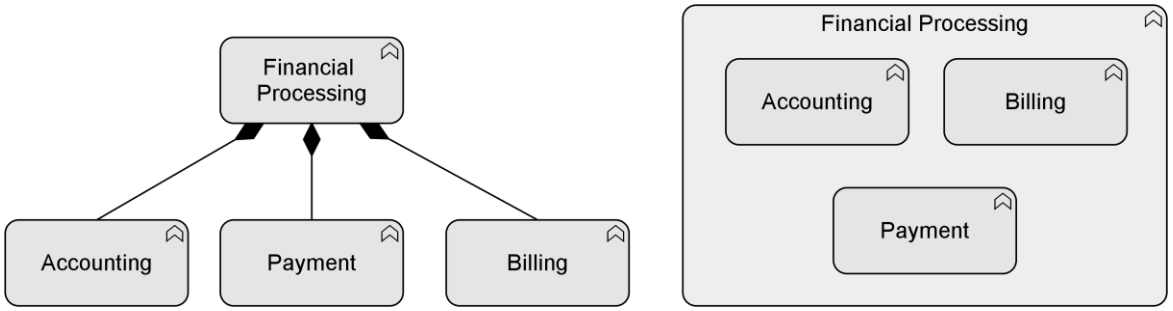


Figure 20: Composition Notation

The usual interpretation of a composition relationship is that *whole or part* of the source element is composed of the *whole of* the target element.

Example

The models below show the two ways to express that the Financial Processing function is composed of three sub-functions.



Example 2: Composition

5.1.2 Aggregation Relationship

The aggregation relationship indicates that an element groups a number of other elements.

The aggregation relationship has been inspired by the aggregation relationship in UML class diagrams. In contrast to the composition relationship, an object can be part of more than one aggregation.

An aggregation relationship is always allowed between two instances of the same element type.

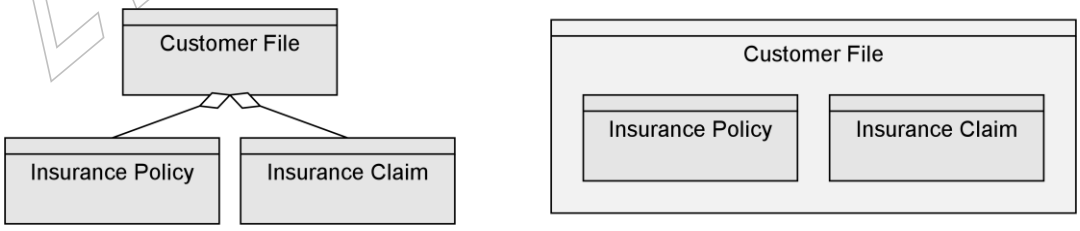
In addition to this, the metamodel explicitly defines other source and target elements that may be connected by an aggregation relationship.

Figure 21: Aggregation Notation

The usual interpretation of an aggregation relationship is that *whole or part* of the source element aggregates the *whole of* the target element.

Example

The models below show two ways to express that the Customer File aggregates an Insurance Policy and Insurance Claim.



Example 3: Aggregation

5.1.3 Assignment Relationship

The assignment relationship expresses the allocation of responsibility, performance of behavior, or execution.

The assignment relationship links active structure elements with units of behavior that are performed by them, business actors with business roles that are fulfilled by them, and nodes with technology objects. It can, for example, relate an internal active structure element with an internal behavior element, an interface with a service, or a node with a technology object. The full set of permitted relationships is listed in Appendix B.

Figure 22: Assignment Notation

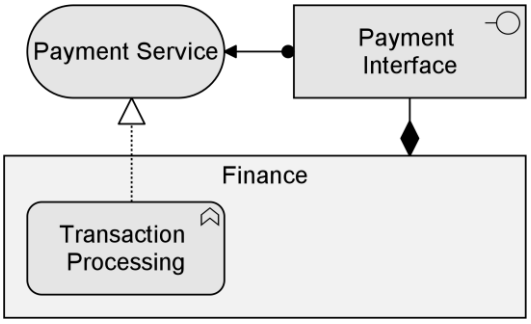
In the ArchiMate framework described in Section 3.4, it always points from active structure to behavior, and from behavior to passive structure. The non-directional notation from the ArchiMate 2.1 specification and before, which shows the black ball at both ends of the relationship, is still allowed but deprecated.

As with all structural relationships, an assignment relationship can also be expressed by nesting the model elements. The direction mentioned above is also the direction of nesting; for example, a business role inside the business actor performing that role, an application function inside an application component executing that function, or an artifact inside a node that stores it.

The usual interpretation of an assignment relationship is that *whole or part* of the source element is assigned the *whole of* the target element. This means that if, for example, two active structure elements are assigned to the same behavior element, either of them can perform the complete behavior. If both active structure elements are needed to perform the behavior, the grouping element or a junction (see Section 5.4.3) can be used, and if the combination of these elements has a more substantive and independent character, a collaboration would be the right way to express this.

Example

The model in the example below includes the two ways to express the assignment relationship. The Finance active structure element is assigned to the Transaction Processing function, and the Payment Interface is assigned to the Payment Service.



Example 4: Assignment

5.1.4 Realization Relationship

The realization relationship indicates that an entity plays a critical role in the creation, achievement, sustenance, or operation of a more abstract entity.

The realization relationship indicates that more abstract entities ("what" or "logical") are realized by means of more tangible entities ("how" or "physical"). The realization relationship is used to model run-time realization; for example, that a business process realizes a business service, and that a data object realizes a business object, an artifact realizes an application component, or a core element realizes a motivation element.

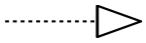
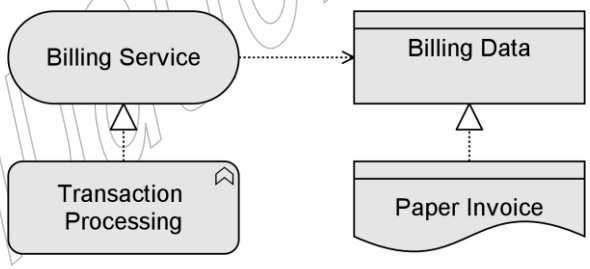


Figure 23: Realization Notation

The usual interpretation of a realization relationship is that the *whole or part* of the source element realizes the *whole of* the target element. This means that if, for example, two internal behavior elements have a realization relationship to the same service, either of them can realize the complete service. If both internal behavior elements are needed to realize, the grouping element or a junction (see Section 5.4.3) can be used. For weaker types of contribution to the realization of an element, the influence relationship (Section 5.2.3) should be used.

Example

The model below illustrates two ways to use the realization relationship. A Transaction Processing function realizes a Billing Service; the Billing Data object is realized by the representation Paper Invoice.



Example 5: Realization

5.2 Dependency Relationships

Dependency relationships describe how elements support or are used by other elements. Three types of dependency relationship are distinguished:

- The *serving* relationship represents a *control* dependency, denoted by a solid line.
- The *access* relationship represents a *data* dependency, denoted by a dashed line.
- The *influence* relationship is the weakest type of dependency, used to model how motivation elements are influenced by other elements.

Note that, although the notation of these relationships resembles the notation of the dependency relationship in UML, these relationships have distinct meanings in ArchiMate notation and

(usually) point in the opposite direction. One advantage of this is that it yields models with directionality, where most[2] of the arrows that represent such supporting, influencing, serving, or realizing dependencies point 'upwards' towards the client/user/business, as you can see in the layered viewpoint example in Section C.1.11. Another reason for this direction, in particular for the serving relationship, is that it abstracts from the 'caller' or 'initiator', since a service may be delivered proactively or reactively. The direction of delivery is always the same, but the starting point for the interaction can be on either end. UML's dependency is often used to denote the latter, showing that the caller depends on some operation that is called. However, for modeling this type of initiative, the ArchiMate language provides the triggering relationship (Section 5.3.1), which can be interpreted as a dynamic (i.e., temporal) dependency. Similarly, the flow relationship is used to model how something (usually information) is transferred from one element to another, which is also a dynamic kind of dependency.

5.2.1 Serving Relationship

The serving relationship models that an element provides its functionality to another element.

The serving relationship describes how the services or interfaces offered by a behavior or active structure element serve entities in their environment. This relationship is applied for both the behavior aspect and the active structure aspect.

Compared to the earlier versions of this standard, the name of this relationship has been changed from 'used by' to 'serving', to better reflect its direction with an active verb: a service serves a user. The meaning of the relationship has not been altered. The 'used by' designation is still allowed but deprecated, and will be removed in a future version of the standard.

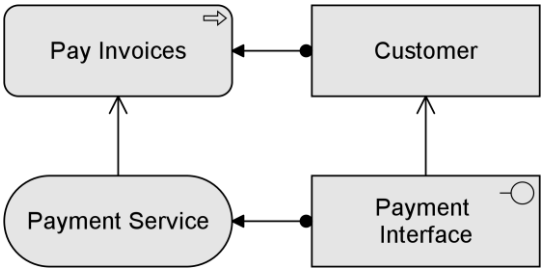
Figure 24: Serving Notation

The usual interpretation of a serving relationship is that *the whole* of the source element serves (is used by) the target element. This means that if, for example, two services serve the same internal behavior element, both of these services are needed. If two services are alternative solutions and only one of them is needed by the internal behavior element, an or junction (see Section 5.4.3) can be used.

Example

The model below illustrates the serving relationship. The Payment Interface serves the Customer, while the Payment Service serves the Pay Invoices process of that customer.

[2] Note that the direction of access depends on the type of access (read *versus* write access) as described in Section 5.2.2.



Example 6: Serving

5.2.2 Access Relationship

The access relationship models the ability of behavior and active structure elements to observe or act upon passive structure elements.

The access relationship indicates that a process, function, interaction, service, or event "does something" with a passive structure element; e.g., create a new object, read data from the object, write or modify the object data, or delete the object. The relationship can also be used to indicate that the object is just associated with the behavior; e.g., it models the information that comes with an event, or the information that is made available as part of a service. The arrow head, if present, indicates the direction of the flow of information. (The access relationship should not be confused with the UML dependency relationship, which uses a similar notation.)

Note that, at the metamodel level, the direction of the relationship is always from an active structure element or a behavior element to a passive structure element, although the notation may point in the other direction to denote 'read' access, and in both directions to denote read-write access. Care must be taken when using access with derived relationships because the arrow on the relationship has no bearing to its directionality.

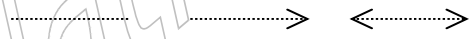


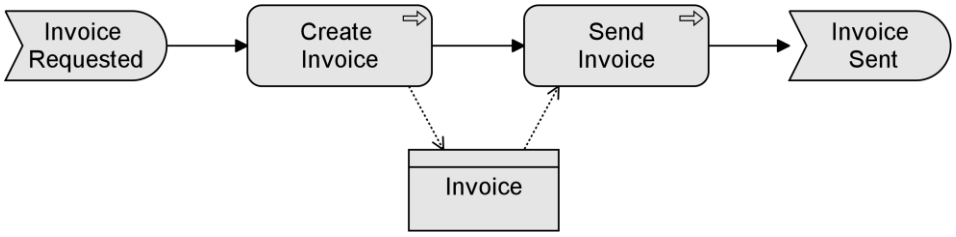
Figure 25: Access Notation

Alternatively, an access relationship can be expressed by nesting the passive structure element inside the behavior or active structure element that accesses it; for example, nesting a data object inside an application component.

The usual interpretation of an access relationship is that *the whole* of the target element is accessed by the source element. This means that if, for example, the same internal behavior element accesses two passive structure elements, both of these passive structure elements are needed. If two passive structure elements are, for example, alternative information sources and only one of them is needed by the internal behavior element, an or junction (see Section 5.4.3) can be used.

Example

The model below illustrates the access relationship. The Create Invoice sub-process writes/creates the Invoice object; the Send Invoice sub-process reads that object.



Example 7: Access

5.2.3 Influence Relationship

The influence relationship models that an element affects the implementation or achievement of some motivation element.

The influence relationship is used to describe that some architectural element influences achievement or implementation of a motivation element, such as a goal or a principle. In general, a motivation element is realized to a certain degree. For example, consistently satisfying the principle 'serve customers wherever they are' will help making the goal 'increase market share' come true. In other words, the principle contributes to the goal. In turn, to implement the principle 'serve customers wherever they are', it may be useful to impose a requirement of '24x7 web availability' on some customer-facing application component. This can be modeled as a requirement that has an influence on that principle, and as an application component that in turn influences the requirement. Consistently modeling these dependencies with an influence relationship yields a traceable motivational path that explains why (in this example) a certain application component contributes to the corporate goal to 'increase market share'. This kind of traceability supports measuring the results of Enterprise Architecture, and provides valuable information to, for example, change impact assessments.

Additional to this 'vertical' use of contribution, from core elements upwards to requirements and goals, the relationship can also be used to model 'horizontal' contributions between motivation elements. The influence relationship in that case describes that some motivation element may influence (the achievement or implementation of) another motivation element. In general, a motivation element is achieved to a certain degree. An influence by some other element may affect this degree, depending on the degree in which this other element is satisfied itself. For example, the degree in which the goal to increase customer satisfaction is realized may be represented by the percentage of satisfied customers that participate in a market interview. This percentage may be influenced by, for example, the goal to improve the reputation of the company; i.e., a higher degree of improvement results in a higher increase in customer satisfaction. On the other hand, the goal to lay off employees may influence the company reputation negatively; i.e., more lay-offs could result in a lower increase (or even decrease) in the company reputation. And thus (indirectly), the goal to increase customer satisfaction may also be influenced negatively.

The realization relationship should be used to represent relationships that are critical to the existence or realization of the target, while the influence relationship should be used to represent relationships that are not critical to the target object's existence or realization. For example, a business actor representing a construction crew might realize the goal of constructing a building, and a requirement to add additional skilled construction workers to an already adequate crew might influence the goal of constructing the building, but also realize an additional goal of opening the building by a particular date. Moreover, an influence relationship can be used to model either:

- The fact that an element positively contributes to the achievement or implementation of some motivation element, or
- The fact that an element negatively influences – i.e., prevents or counteracts – such achievement

Attributes can be used to indicate the sign and/or strength of the influence. The choice of possible attribute values is left to the modeler; e.g., {++, +, 0, -, --} or [0..10]. By default, the influence relationship models a contribution with unspecified sign and strength.

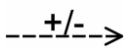
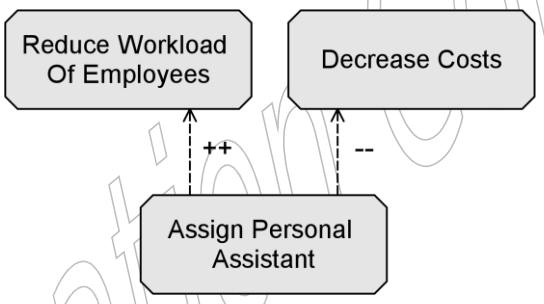


Figure 26: Influence Notation

Example

The model below illustrates the use of the influence relationship to model the different effects of the same motivation element, Assign Personal Assistant. This has a strongly positive influence on Reduce Workload Of Employees, but a strongly negative influence on Decrease Costs.



Example 8: Influence

5.3 Dynamic Relationships

The dynamic relationships describe temporal dependencies between elements within the architecture. Two types of dynamic relationships are distinguished:

- The *triggering* relationship represents a *control* flow between elements, denoted by a solid line.
- The *flow* relationship represents a *data* (or *value*) flow between elements, denoted by a dashed line.

5.3.1 Triggering Relationship

The triggering relationship describes a temporal or causal relationship between elements.

The triggering relationship is used to model the temporal or causal precedence of behavior elements in a process. The usual interpretation of a triggering relationship is that the source element should be completed before the target element can start, although weaker interpretations are also permitted. Note that this does not necessarily represent that one behavior element

actively starts another; a traffic light turning green also triggers the cars to go through the intersection.

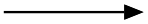
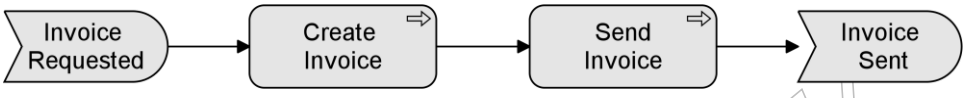


Figure 27: Triggering Notation

Example

The model below illustrates that triggering relationships are mostly used to model causal dependencies between (sub-)processes and/or events.



Example 9: Triggering

5.3.2 Flow Relationship

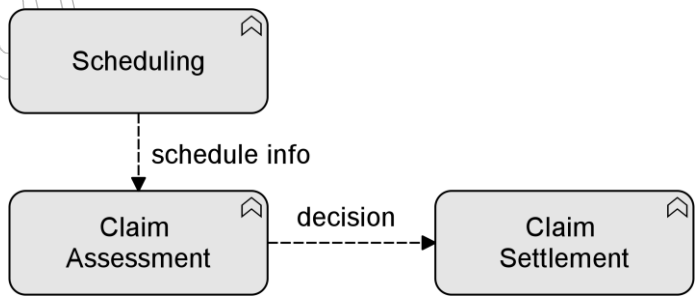
The flow relationship represents transfer from one element to another.

The flow relationship is used to model the flow of, for example, information, goods, or money between behavior elements. A flow relationship does not imply a causal relationship.

Figure 28: Flow Notation

Example

The model below shows a Claim Assessment function, which forwards decisions about the claims to the Claim Settlement function. In order to determine the order in which the claims should be assessed, Claim Assessment makes use of schedule information received from the Scheduling function.



Example 10: Flow

5.4 Other Relationships

5.4.1 Specialization Relationship

The specialization relationship indicates that an element is a particular kind of another element.

The specialization relationship has been inspired by the generalization relationship in UML class diagrams, but is applicable to specialize a wider range of concepts. The specialization relationship can relate any instance of a concept with another instance of the same concept.

A specialization relationship is always allowed between two instances of the same element.

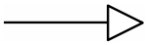
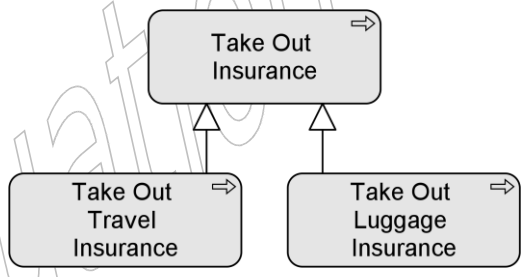


Figure 29: Specialization Notation

Alternatively, a specialization relationship can be expressed by nesting the specialized element inside the generic element.

Example

The model below illustrates the use of the specialization relationship for a process. In this case the Take Out Travel Insurance and Take Out Luggage Insurance processes are a specialization of a more generic Take Out Insurance process.



Example 11: Specialization

5.4.2 Association Relationship

An association models an unspecified relationship, or one that is not represented by another ArchiMate relationship.

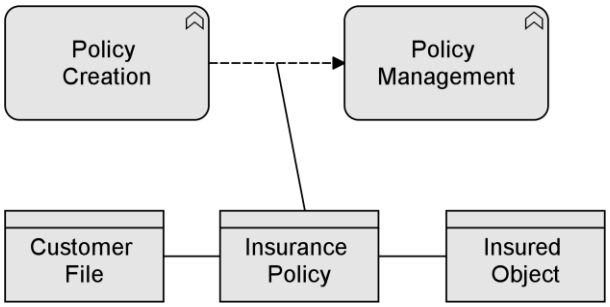
An association relationship is always allowed between two elements, or between a relationship and an element.

The association relationship can be used when drawing a first high-level model where relationships are initially denoted in a generic way, and later refined to show more specific relationship types. In the metamodel pictures, some specific uses of the association relationship are explicitly shown.

Figure 30: Association Notation

Example

The model illustrates a number of uses of the association relationship. It also shows an example of an association between a flow relationship and a passive structure element, to indicate the kind of information that is communicated between the two functions.



Example 12: Association

5.4.3 Junction

A junction is not an actual relationship in the same sense as the other relationships described in this chapter, but rather a relationship connector.

A junction is used to connect relationships of the same type.

A junction is used in a number of situations to connect relationships of the same type. A junction may have multiple incoming relationships and one outgoing relationship, one incoming relationship and multiple outgoing relationships, or multiple incoming and outgoing relationships (the latter can be considered a shorthand of two subsequent junctions).

The relationships that can be used in combination with a junction are all the dynamic relationships, as well as assignment, realization, and association. A junction is used to explicitly express that several elements *together* participate in the relationship (*and* junction) or that *one of* the elements participates in the relationship (*or* junction). A junction should either have one incoming and more than one outgoing relationships, or more than one incoming and one outgoing. It is allowed to omit arrowheads of relationships leading into a junction.

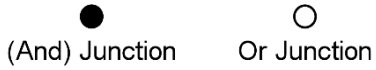
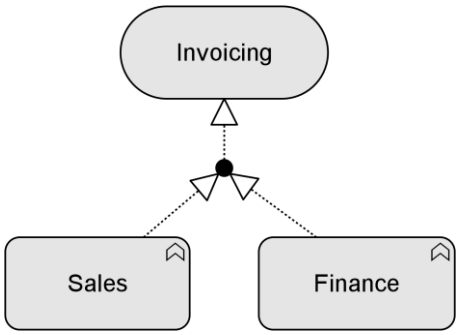


Figure 31: Junction Notation

Junctions used on triggering relationships are similar to gateways in BPMN and forks and joins in UML activity diagrams. They can be used to model high-level process flow. A label may be added to outgoing triggering relationships of a junction to indicate a choice, condition, or guard that applies to that relationship. Such a label is only an informal indication. No formal, operational semantics has been defined for these relationships, because implementation-level languages such as BPMN and UML differ in their execution semantics and the ArchiMate language does not want to unduly constrain mappings to such languages.

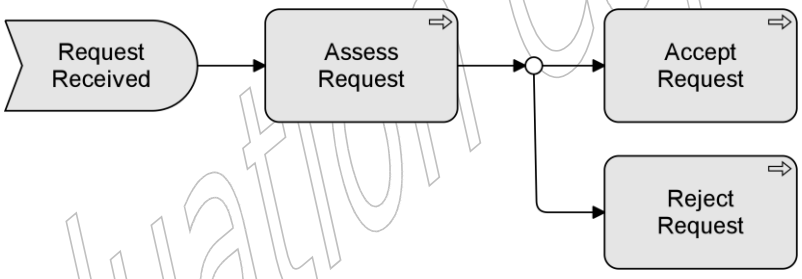
Examples

In the model below, the junction in the model is used to denote that the Sales and Finance functions together realize the Invoicing service.



Example 13: (And) Junction

In the model below, the or junction is used to denote a choice: process Assess Request triggers either Accept Request or Reject Request. (The usual interpretation of two separate triggering relations, one from Assess Request to Accept Request and one from Assess Request to Reject Request, is that Assess Request triggers both of the other processes.)



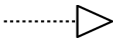
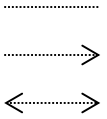
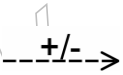
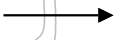
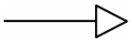
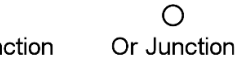
Example 14: Or Junction

5.5 Summary of Relationships

Table 3 gives an overview of the ArchiMate relationships with their definitions.

Table 3: Relationships

Structural Relationships		**Notation**
Composition	Indicates that an element consists of one or more other elements.	
Aggregation	Indicates that an element groups a number of other elements.	
Assignment	Expresses the allocation of responsibility, performance of behavior, or execution.	

Realization	Indicates that an entity plays a critical role in the creation, achievement, sustenance, or operation of a more abstract entity.	
Dependency Relationships		**Notation**
Serving	Models that an element provides its functionality to another element.	
Access	Models the ability of behavior and active structure elements to observe or act upon passive structure elements.	
Influence	Models that an element affects the implementation or achievement of some motivation element.	
Dynamic Relationships		**Notation**
Triggering	Describes a temporal or causal relationship between elements.	
Flow	Transfer from one element to another.	
Other Relationships		**Notation**
Specialization	Indicates that an element is a particular kind of another element.	
Association	Models an unspecified relationship, or one that is not represented by another ArchiMate relationship.	
Junction	Used to connect relationships of the same type.	

5.6 Derivation Rules

This section presents a number of rules to derive indirect relationships between elements in a model, based on the modeled relationships. This makes it possible to abstract from intermediary elements that are not relevant to show in a certain model or view of the architecture, and supports impact analysis.

It is important to note that all these derived relationships are also valid in the ArchiMate language. These are not shown in the metamodel diagrams included in this standard because this would reduce the legibility of them. However, the tables in Appendix B show all permitted relationships between two elements in the language.

Note that these derivation rules do not work on relationships with grouping, or between core elements and other elements such as motivation, strategy, or implementation and migration elements, with the exception of the realization and influence relationships.

5.6.1 Derivation Rule for Structural and Dependency Relationships

The structural and dependency relationships can be ordered by 'strength'. Structural relationships are 'stronger' than dependency relationships, and the relationships within these categories can also be ordered by strength:

- Influence (weakest)
- Access
- Serving
- Realization
- Assignment
- Aggregation
- Composition (strongest)

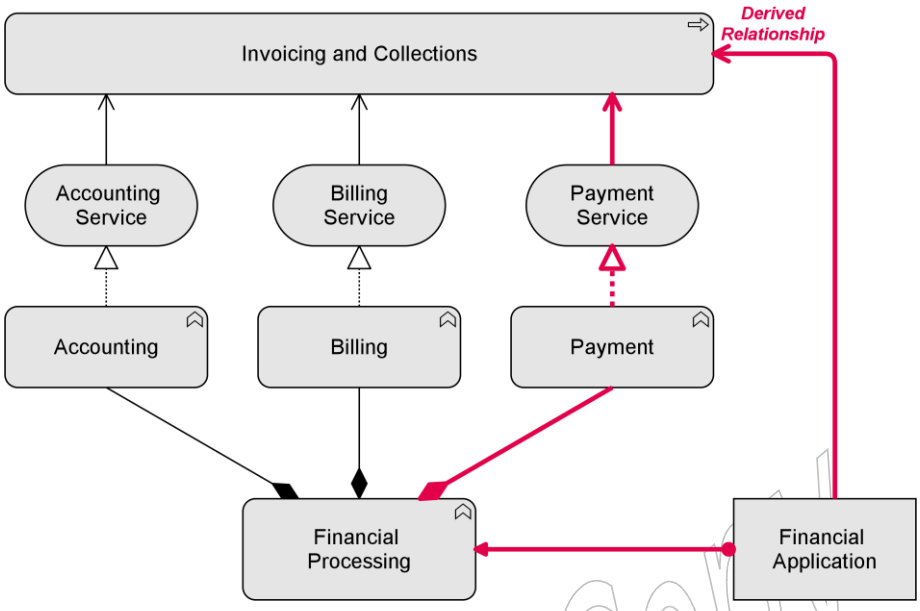
Part of the language definition is an abstraction rule that states that two relationships that join at an intermediate element can be combined and replaced by the weaker of the two.

If two structural or dependency relationships *r*:*R* and *s*:*S* are permitted between elements *a*, *b*, and *c* such that *r*(*a*,*b*) and *s*(*b*,*c*), then a structural relationship *t*:*T* is also permitted, with *t*(*a*,*c*) and type *T* being the weakest of *R* and *S*.

For the application of this rule, it is assumed that the assignment relationship has a direction (as indicated by the role names in the metamodel figures), as defined in Section 5.1.3.

Transitively applying this property allows us to replace a 'chain' of structural relationships (with intermediate model elements) by the weakest structural relationship in the chain. For a more formal description and derivation of this rule the reader is referred to [11]. Note that the resulting derived relationship is a *potential* relationship, which is what is needed for impact analysis. To derive *certain* relationships, other derivation rules are needed, which are beyond the scope of this standard.

An example is shown in the figure below: assume that the goal is to omit the functions, sub-functions, and services from the model. In this case, an indirect serving relationship (the relationship labeled Derived Relationship (thick arrow on the right) can be derived from Financial application to the Invoicing and collections process (from the chain assignment – composition – realization – serving).



Example 15: Derived Structural and Dependency Relationship

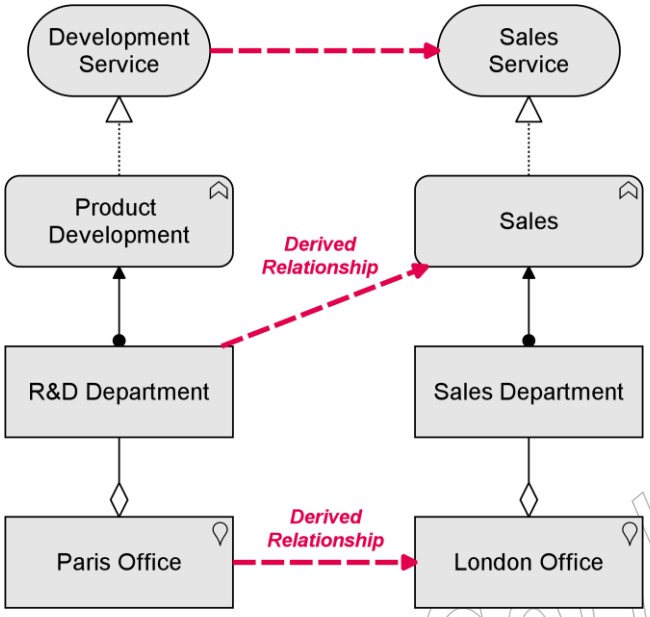
Note: In deriving the relationship tables in the appendix, a distinction has been made between artifacts realizing active structure elements and those realizing passive structure elements, to avoid deriving undesirable relationships.

5.6.2 Derivation Rules for Dynamic Relationships

For the two dynamic relationships, the following rules apply:

- If there is a flow relationship *r* from element *a* to element *b*, and a structural relationship from element *c* to element *a*, a flow relationship *r* can be derived from element *c* to element *b*.
- If there is a flow relationship *r* from element *a* to element *b*, and a structural relationship from element *d* to element *b*, a flow relationship *r* can be derived from element *a* to element *d*.

These rules can be applied repeatedly. Informally, this means that the begin and/or endpoint of a flow relationship can be transferred 'backward' in a chain of elements connected by structural relationships. The example below shows two of the possible flow relationships that can be derived with these rules, given a flow relationship between the two services.



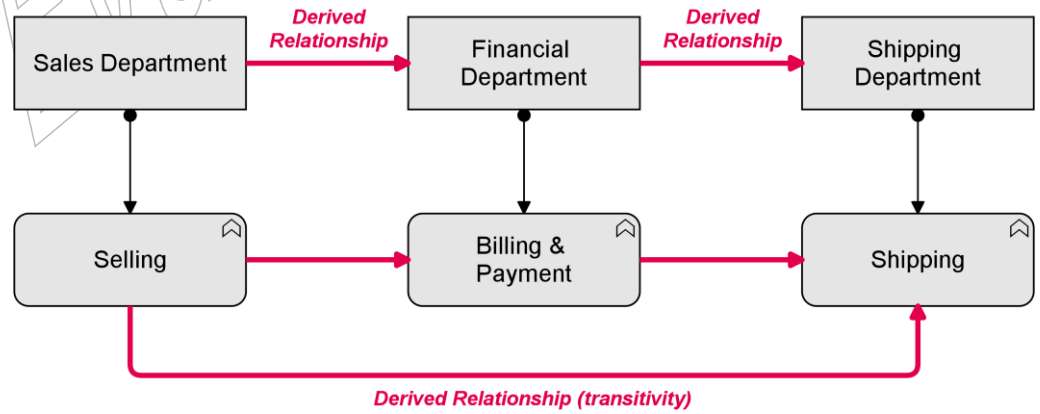
Example 16: Derived Flow Relationships

This rule also applies for a triggering relationship, but only in combination with an assignment relationship (not with other structural relationships):

- If there is a triggering relationship *r* from element *a* to element *b*, and an assignment relationship from element *c* to element *a*, a triggering relationship *r* can be derived from element *c* to element *b*.
- If there is a triggering relationship *r* from element *a* to element *b*, and an assignment relationship from element *d* to element *b*, a triggering relationship *r* can be derived from element *a* to element *d*.

Moreover, triggering relationships are *transitive*:

- If there is a triggering relationship from element *a* to element *b*, and a triggering relationship from element *b* to element *c*, a triggering relationship can be derived from element *a* to element *c*.



Example 17: Derived Triggering Relationships

6 Motivation Elements

6.1 Motivation Elements Metamodel

Figure 32 gives an overview of the motivation elements and their relationships.

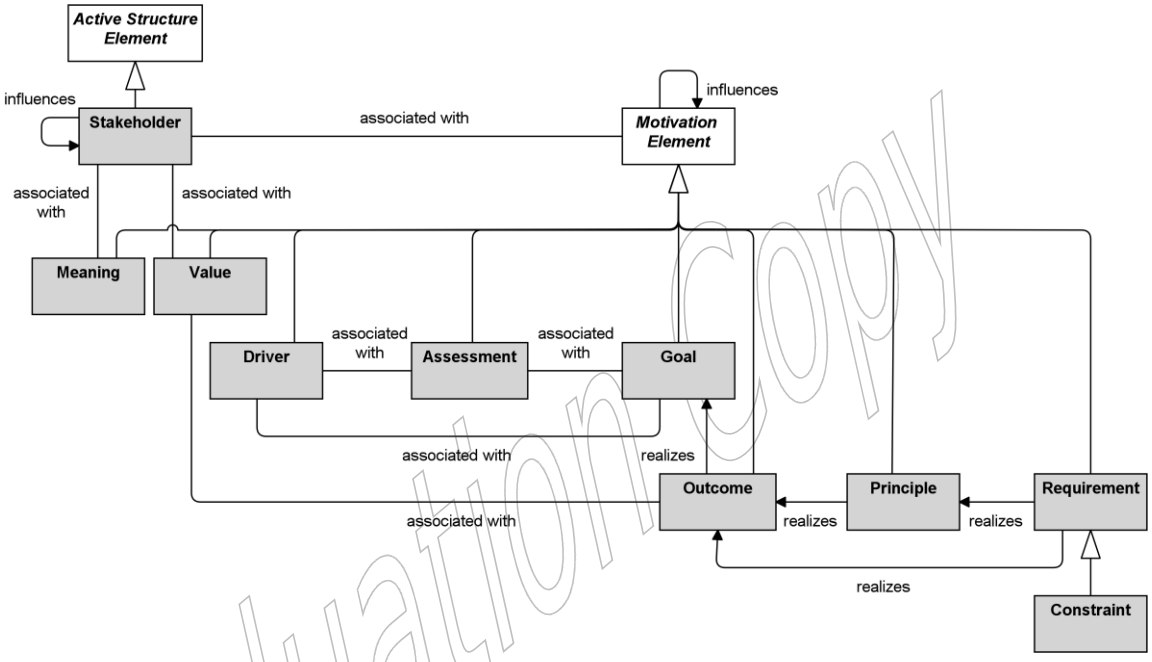


Figure 32: Motivation Elements Metamodel

Note: This figure does not show all permitted relationships: every non-abstract motivation element can have aggregation and specialization relationships with elements of the same type.

6.2 Stakeholder, Driver, and Assessment

Motivation elements are used to model the motivations, or reasons, that guide the design or change of an Enterprise Architecture.

It is essential to understand the factors, often referred to as *drivers*, which influence other motivation elements. They can originate from either inside or outside the enterprise. Internal drivers, also called concerns, are associated with *stakeholders*, which can be some individual human being or some group of human beings, such as a project team, enterprise, or society. Examples of such internal drivers are customer satisfaction, compliance to legislation, or profitability. It is common for enterprises to undertake an *assessment* of these drivers; e.g., using a SWOT analysis, in order to respond in the best way.

6.2.1 Stakeholder

A stakeholder is the role of an individual, team, or organization (or classes thereof) that represents their interests in the outcome of the architecture.

This definition is based on the definition in the TOGAF framework [4]. A stakeholder has one or more interests in, or concerns about, the organization and its Enterprise Architecture. In order to direct efforts to these interests and concerns, stakeholders change, set, and emphasize goals. Stakeholders may also influence each other. Examples of stakeholders are the CEO, the board of directors, shareholders, customers, business and application architects, but also legislative authorities. The name of a stakeholder should preferably be a noun.

Figure 33: Stakeholder Notation

6.2.2 Driver

A driver represents an external or internal condition that motivates an organization to define its goals and implement the changes necessary to achieve them.

Drivers may be internal, in which case they are usually associated with a stakeholder, and are often called "concerns". Stakeholder concerns are defined in the TOGAF framework [4] as "*the key interests that are crucially important to the stakeholders in a system, and determine the acceptability of the system. Concerns may pertain to any aspect of the function, development, or operation of the system, including considerations such as performance, reliability, security, distribution, and evolvability.*" Examples of internal drivers are Customer satisfaction and Profitability. Drivers of change may also be external; e.g., economic changes or changing legislation. The name of a driver should preferably be a noun.

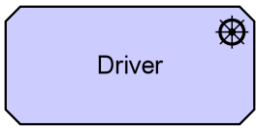


Figure 34: Driver Notation

6.2.3 Assessment

An assessment represents the result of an analysis of the state of affairs of the enterprise with respect to some driver.

An assessment may reveal strengths, weaknesses, opportunities, or threats for some area of interest. These need to be addressed by adjusting existing goals or setting new ones, which may trigger changes to the Enterprise Architecture.

Strengths and weaknesses are internal to the organization. Opportunities and threats are external to the organization. Weaknesses and threats can be considered as problems that need to be addressed by goals that "negate" the weaknesses and threats. Strengths and opportunities may be translated directly into goals. For example, the weakness "Customers complain about the

helpdesk" can be addressed by defining the goal "Improve helpdesk". Or, the opportunity "Customers favor insurances that can be managed online" can be addressed by the goal "Introduce online portfolio management". The name of an assessment should preferably be a noun or a (very) short sentence.

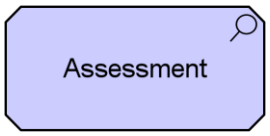
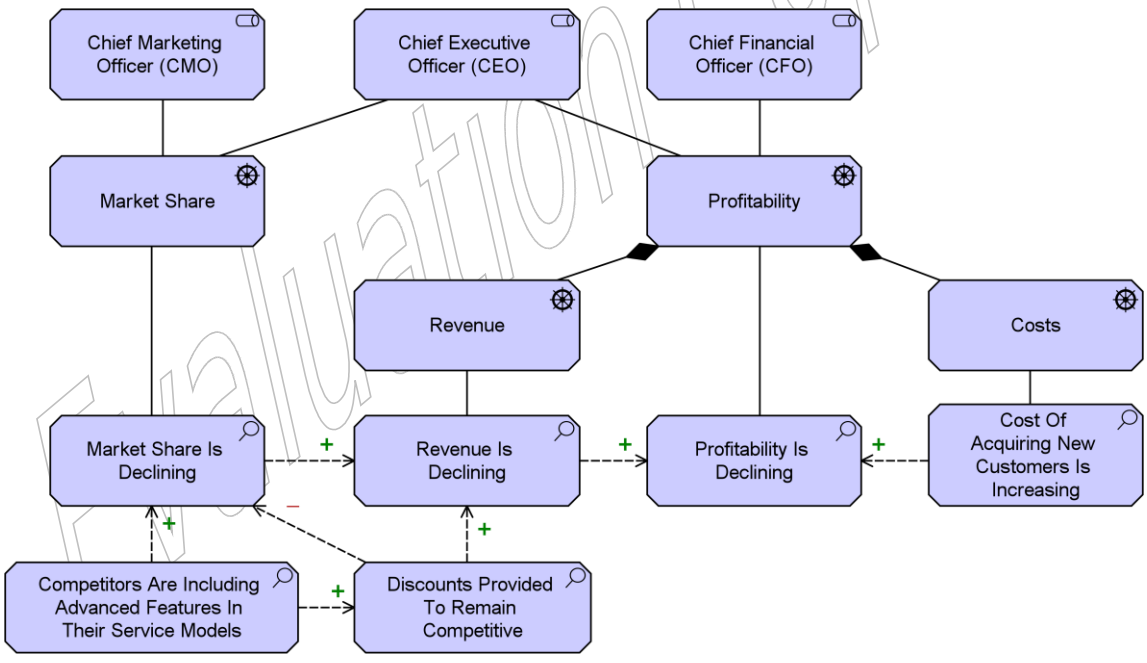


Figure 35: Assessment Notation

6.2.4 Example

The stakeholder Chief Marketing Officer (CMO) is concerned with the driver Market Share, the stakeholder Chief Executive Officer (CEO) is concerned with the drivers Market Share and Profitability, and the stakeholder Chief Financial Officer (CFO) is concerned with the driver Profitability. The driver Profitability is composed of two other drivers: Revenue and Costs. Several assessments are associated with these drivers (e.g., the assessment Market Share Is Declining is associated with driver Market Share), and assessments may influence each other in a positive or negative way (e.g., Market Share Is Declining results in Revenue Is Declining, which in turn results in Profitability Is Declining).



Example 18: Stakeholder, Driver, and Assessment

6.3 Goal, Outcome, Principle, Requirement, and Constraint

The motivation of an organization or individual to achieve certain results is represented by goals, principles, requirements, and constraints. *Goals* represent that a stakeholder wants to realize a certain outcome; e.g., "Increase customer satisfaction by 10%". The end results realized by capabilities that realize these goals are *outcomes.* Principles and requirements represent desired properties of solutions – or means – to realize the goals. *Principles* are normative guidelines that

guide the design of all possible solutions in a given context. For example, the principle "Data should be stored only once" represents a means to achieve the goal of "Data consistency" and applies to all possible designs of the organization's architecture. *Requirements* represent formal statements of need, expressed by stakeholders, which must be met by the architecture or solutions. For example, the requirement "Use a single CRM system" conforms to the aforementioned principle by applying it to the current organization's architecture in the context of the management of customer data.

6.3.1 Goal

A goal represents a high-level statement of intent, direction, or desired end state for an organization and its stakeholders.

In principle, a goal can represent anything a stakeholder may desire, such as a state of affairs, or a produced value. Examples of goals are: to increase profit, to reduce waiting times at the helpdesk, or to introduce online portfolio management. Goals are typically used to measure success of an organization.

Goals are generally expressed using qualitative words; e.g., "increase", "improve", or "easier". Goals can also be decomposed; e.g., Increase profit can be decomposed into the goals Reduce cost and Increase sales. However, it is also very common to associate concrete outcomes with goals, which can be used to describe both the quantitative and time-related results that are essential to describe the desired state, and when it should be achieved.

Figure 36: Goal Notation

6.3.2 Outcome

An outcome represents an end result that has been achieved.

Outcomes are high-level, business-oriented results produced by capabilities of an organization, and by inference by the core elements of its architecture that realize these capabilities. Outcomes are tangible, possibly quantitative, and time-related, and can be associated with assessments. An outcome may have a different value for different stakeholders.

The notion of outcome is important in business outcome-driven approaches to Enterprise Architecture and in capability-based planning. Outcomes are closely related to requirements, goals, and other intentions. Outcomes are the end results, and goals or requirements are often formulated in terms of outcomes that should be realized. Capabilities are designed to achieve such outcomes.

Outcome names should unambiguously identify end results that have been achieved in order to avoid confusion with actions or goals. At a minimum, outcome names should consist of a noun identifying the end result followed by a past-tense verb or adjective indicating that the result has been achieved. Examples include "First-place ranking achieved" and "Key supplier partnerships in place". Outcome names can also be more specific; e.g., "2015 quarterly profits rose 10% year over year beginning in Q3".

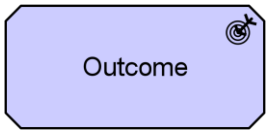


Figure 37: Outcome Notation

6.3.3 Principle

A principle represents a qualitative statement of intent that should be met by the architecture.

Principles are strongly related to goals and requirements. Similar to requirements, principles define intended properties of systems. However, in contrast to requirements, principles are broader in scope and more abstract than requirements. A principle defines a general property that applies to any system in a certain context. A requirement defines a property that applies to a specific system as described by an architecture.

A principle needs to be made specific for a given system by means of one or more requirements, in order to enforce that the system conforms to the principle. For example, the principle "Information management processes comply with all relevant laws, policies, and regulations" is realized by the requirements that are imposed by the actual laws, policies, and regulations that apply to the specific system under design.

A principle is motivated by some goal or driver. For example, the aforementioned principle may be motivated by the goal to maintain a good reputation and/or the goal to avoid penalties. The principle provides a means to realize its motivating goal, which is generally formulated as a guideline. This guideline constrains the design of all systems in a given context by stating the general properties that are required from any system in this context to realize the goal. Principles are intended to be more stable than requirements in the sense that they do not change as quickly as requirements may do. Organizational values, best practices, and design knowledge may be reflected and made applicable in terms of principles.

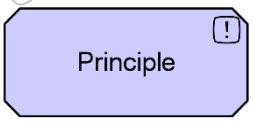


Figure 38: Principle Notation

6.3.4 Requirement

A requirement represents a statement of need that must be met by the architecture.

In the end, a business goal must be realized by a plan or concrete change goal, which may or may not require a new system or changes to an existing system.

The term "system" is used in its general meaning; i.e., as a group of (functionally) related elements, where each element may be considered as a system again. Therefore, a system may refer to any active structural element, behavior element, or passive structural element of some organization, such as a business actor, application component, business process, application service, business object, or data object.

Requirements model the properties of these elements that are needed to achieve the "ends" that are modeled by the goals. In this respect, requirements represent the "means" to realize goals.

During the design process, goals may be decomposed until the resulting sub-goals are sufficiently detailed to enable their realization by properties that can be exhibited by systems. At this point, goals can be realized by requirements that demand these properties from the systems.

For example, one may identify two alternative requirements to realize the goal to improve portfolio management:

- By assigning a personal assistant to each customer, or
- By introducing online portfolio management

The former requirement can be realized by a human actor and the latter by a software application. These requirements can be decomposed further to define the requirements on the human actor and the software application in more detail.

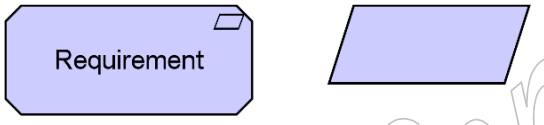


Figure 39: Requirement Notation

6.3.5 Constraint

A constraint represents a factor that prevents or obstructs the realization of goals.

In contrast to a requirement, a constraint does not prescribe some intended functionality of the system to be realized, but imposes a restriction on the way it operates or may be realized. This may be a restriction on the implementation of the system (e.g., specific technology that is to be used), a restriction on the implementation process (e.g., time or budget constraints), or a restriction on the functioning of the system (e.g., legal constraints).

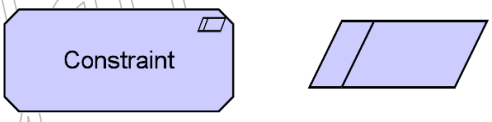
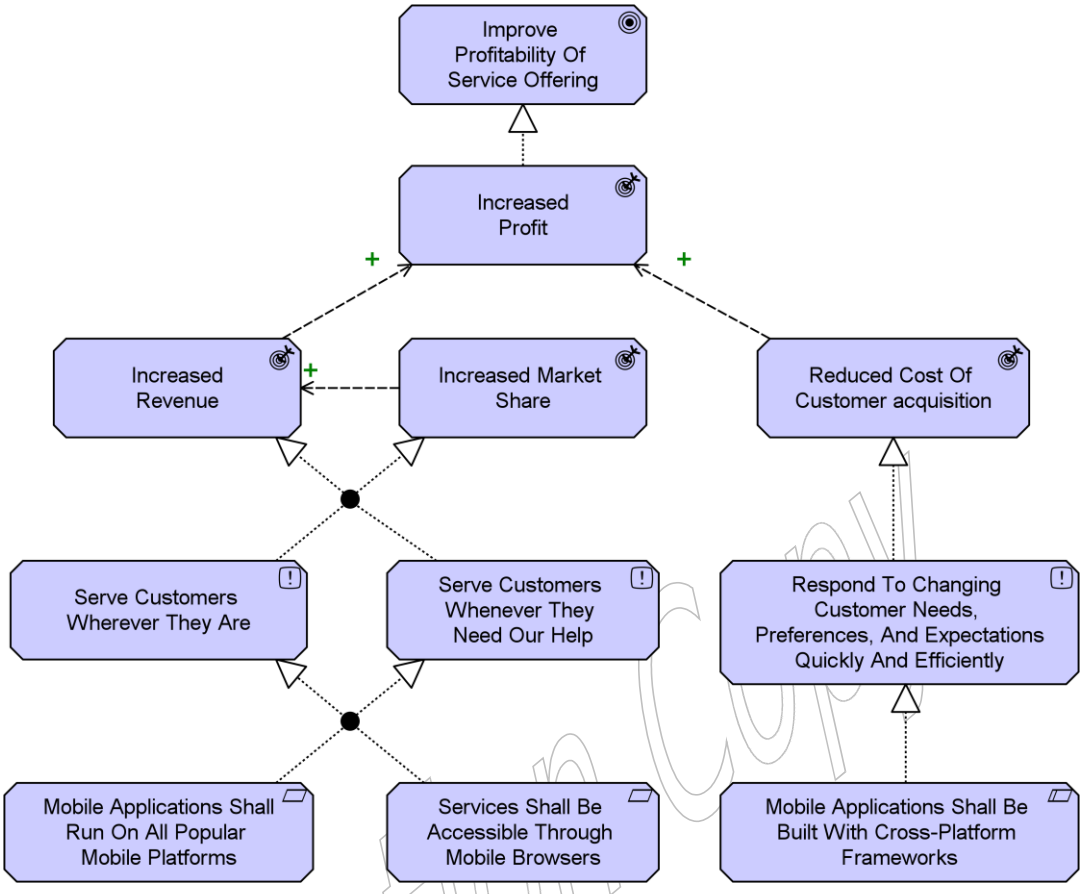


Figure 40: Constraint Notation

6.3.6 Example

The goal Improve Profitability of Service Offering is realized by the outcome Increased Profit. This outcome is influenced positively by the outcomes Increased Revenue and Reduced Cost of Customer Acquisition. The outcome Increased Revenue is influenced positively by an outcome Increased Market Share. Both of these outcomes are realized by a combination of two principles: Serve Customers Wherever They Are and Serve Customers Whenever They Need Our Help. Both of these principles are realized by a combination of two requirements: Mobile Applications Shall Run On All Popular Mobile Platforms and Services Shall Be Accessible Through Mobile Browsers. The goal Reduced Cost Of Customer Acquisition is realized by a principle Respond To Changing Customer Needs, Preferences, And Expectations Quickly And Efficiently, which in turn is realized by a constraint Mobile Applications Shall Be Built With Cross-Platform Frameworks.



Example 19: Goal, Outcome, Principle, Requirement, and Constraint

6.4 Meaning and Value

Different stakeholders may attach a different *value* to outcomes, since they may have different interests. Similarly, they may give their own *meaning* or interpretation to core elements of the architecture.

6.4.1 Meaning

Meaning represents the knowledge or expertise present in, or the interpretation given to, a core element in a particular context.

A meaning represents the interpretation of an element of the architecture. In particular, this is used to describe the meaning of passive structure elements (for example, a document, message). It is a description that expresses the *intent* of that element; i.e., how it informs the *external user*.

It is possible that different users view the informative functionality of an element differently. For example, what may be a "registration confirmation" for a client could be a "client mutation" for a CRM department (assuming for the sake of argument that it is modeled as an external user). Also, various different representations may carry essentially the same meaning. For example, various different documents (a web document, a filled-in paper form, a "client contact" report from the call center) may essentially carry the same meaning.

A meaning can be associated with any core element. To denote that a meaning is specific to a particular stakeholder, this stakeholder can also be associated to the meaning. The name of a meaning should preferably be a noun or noun phrase.

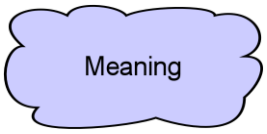


Figure 41: Meaning Notation

6.4.2 Value

Value represents the relative worth, utility, or importance of a core element or an outcome.

Value may apply to what a party gets by selling or making available some product or service, or it may apply to what a party gets by buying or obtaining access to it. Value is often expressed in terms of money, but it has long since been recognized that non-monetary value is also essential to business; for example, practical/functional value (including the *right* to use a service), and the value of information or knowledge. Though value can hold internally for some system or organizational unit, it is most typically applied to *external* appreciation of goods, services, information, knowledge, or money, normally as part of some sort of customer-provider relationship.

A value can be associated with all core elements of an architecture as well as with outcomes. To model the stakeholder for whom this value applies, this stakeholder can also be associated with that value. Although the name of a value can be expressed in many different ways (including amounts, objects), where the "functional" value of an architecture element is concerned it is recommended to try and express it as an action or state that can be performed or reached as a result of the corresponding element being available.

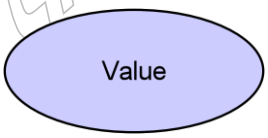
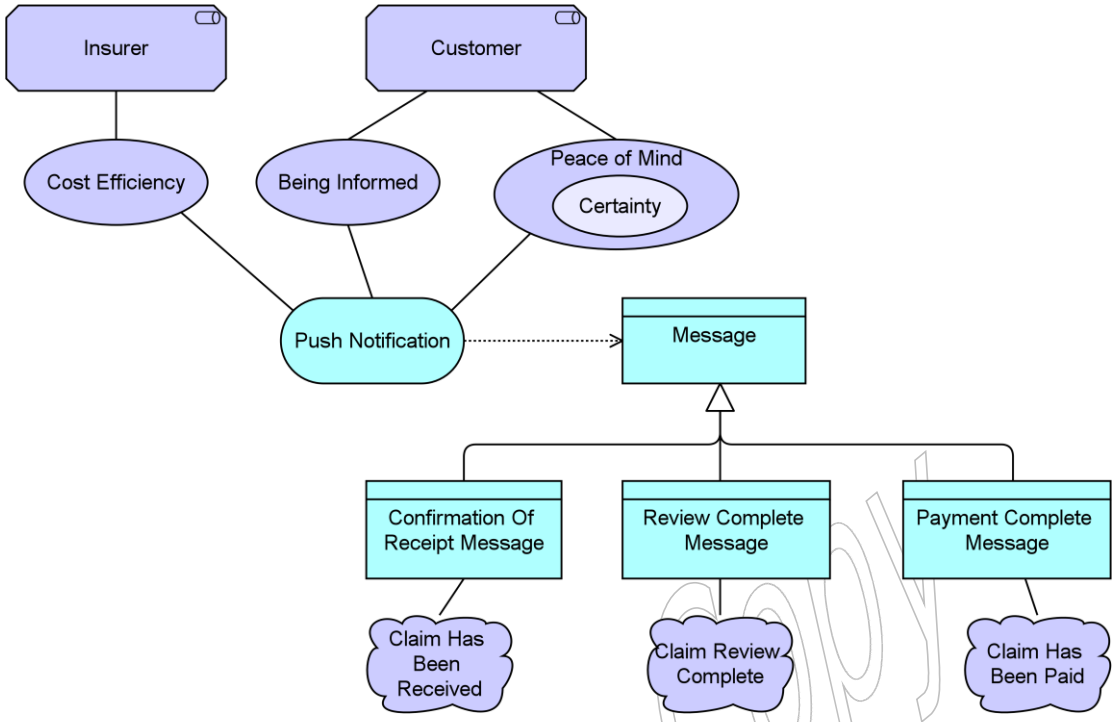


Figure 42: Value Notation

6.4.3 Example

Sending push notifications has a value of Cost Efficiency for the stakeholder Insurer, and a value of Being Informed and Peace of Mind (which is partly due to a value of Certainty) for the stakeholder Customer. Different meanings can be assigned to the different specific types of notification messages. A Confirmation Of Receipt Message has the meaning Claim Has Been Received, a Review Complete Message has the meaning Claim Review Complete, and a Payment Complete Message has the meaning Claim Has Been Paid.

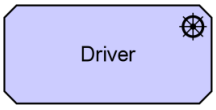
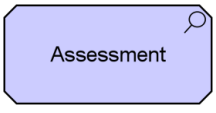


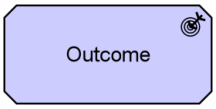
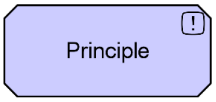
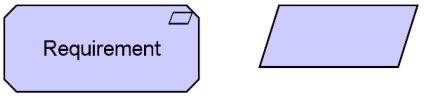
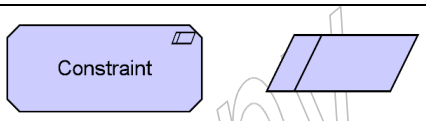
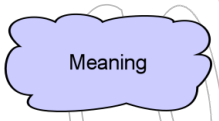
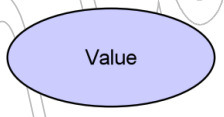
Example 20: Meaning and Value

6.5 Summary of Motivation Elements

Table 4 gives an overview of the motivation elements, with their definitions.

Table 4: Motivation Elements

Element	Definition	Notation
Stakeholder	The role of an individual, team, or organization (or classes thereof) that represents their interests in the outcome of the architecture.	
Driver	An external or internal condition that motivates an organization to define its goals and implement the changes necessary to achieve them.	
Assessment	The result of an analysis of the state of affairs of the enterprise with respect to some driver.	
Goal	A high-level statement of intent, direction, or desired end state for an organization and its stakeholders.	

Element	Definition	Notation
Outcome	An end result that has been achieved.	
Principle	A qualitative statement of intent that should be met by the architecture.	
Requirement	A statement of need that must be met by the architecture.	
Constraint	A factor that prevents or obstructs the realization of goals.	
Meaning	The knowledge or expertise present in, or the interpretation given to, a core element in a particular context.	
Value	The relative worth, utility, or importance of a core element or an outcome.	

6.6 Relationships with Core Elements

The purpose of the motivation elements is to model the motivation behind the core elements in an Enterprise Architecture. Therefore, it should be possible to relate motivation elements to core elements.

As shown in Figure 43, a requirement (and, indirectly, also a principle, outcome, and goal) can be related directly to a structure or behavior element by means of a realization relationship. Also, the weaker influence relationship is allowed between these elements. Meaning and value can be associated with any structure or behavior element.

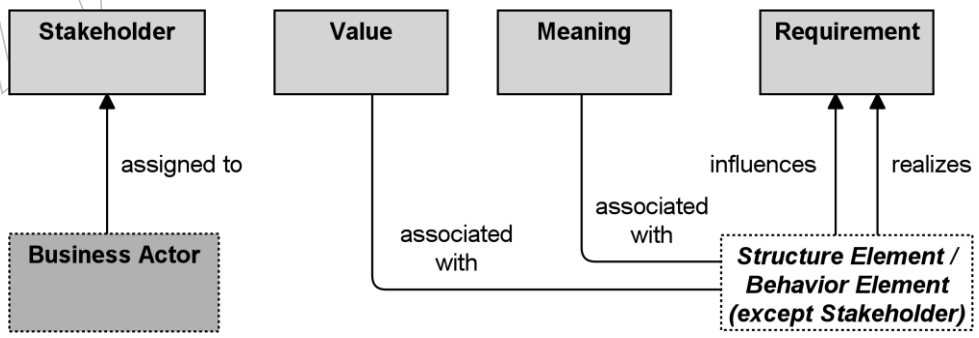


Figure 43: Relationships between Motivation Elements and Core Elements

Also, a business actor may be assigned to a stakeholder, which can be seen as a motivation role (as opposed to an operational business role) that an actor may fulfill.

7 Strategy Elements

7.1 Strategy Elements Metamodel

Figure 44 gives an overview of the strategy elements and their relationships.

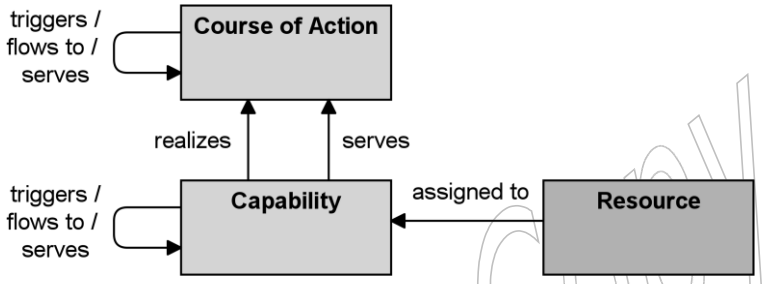


Figure 44: Strategy Elements Metamodel

Note: This figure does not show all permitted relationships; every element in the language can have composition, aggregation, and specialization relationships with elements of the same type; furthermore, there are indirect relationships that can be derived as explained in Section 5.6. The full specification of permitted relationships can be found in Appendix B.

7.2 Structure Elements

7.2.1 Resource

A resource represents an asset owned or controlled by an individual or organization.

Resources are a central concept in the field of strategic management, economics, computer science, portfolio management, and more. They are often considered, together with capabilities, to be sources of competitive advantage for organizations. Resources are analyzed in terms of strengths and weaknesses, and they are considered when implementing strategies. Due to resources being limited, they can often be a deciding factor for choosing which strategy, goal, and project to implement and in which order. Resources can be classified into tangible assets – financial assets (e.g., cash, securities, borrowing capacity), physical assets (e.g., plant, equipment, land, mineral reserves), intangible assets (technology; e.g., patents, copyrights, trade secrets; reputation; e.g., brand, relationships; culture), and human assets (skills/know-how, capacity for communication and collaboration, motivation).

Resources are realized by active and passive structure elements. The name of a resource should preferably be a noun.

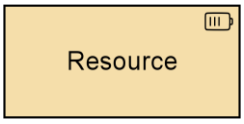


Figure 45: Resource Notation

7.3 Behavior Elements

7.3.1 Capability

A capability represents an ability that an active structure element, such as an organization, person, or system, possesses.

In the field of business, strategic thinking and planning delivers strategies and high-level goals that are often not directly implementable in the architecture of an organization. These long-term or generic plans need to be specified and made actionable in a way that both business leaders and Enterprise Architects can relate to and at a relatively high abstraction level.

Capabilities help to reduce this gap by focusing on business outcomes. On the one hand, they provide a high-level view of the current and desired abilities of an organization, in relation to its strategy and its environment. On the other hand, they are realized by various elements (people, processes, systems, and so on) that can be described, designed, and implemented using Enterprise Architecture approaches. Capabilities may also have serving relationships; for example, to denote that one capability contributes to another.

Capabilities are expressed in general and high-level terms and are typically realized by a combination of organization, people, processes, information, and technology. For example, marketing, customer contact, or outbound telemarketing [4].

Capabilities are typically aimed at achieving some goal or delivering value by realizing an outcome. Capabilities are themselves realized by core elements. To denote that a set of core elements together realizes a capability, grouping can be used.

Capabilities are often used for capability-based planning, to describe their evolution over time. To model such so-called capability increments, the specialization relationship can be used to denote that a certain capability increment is a specific version of that capability. Aggregating those increments and the core elements that realize them in plateaus (see Section 13.2.4) can be used to model the evolution of the capabilities.

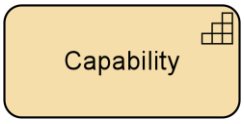


Figure 46: Capability Notation

7.3.2 Course of Action

A course of action is an approach or plan for configuring some capabilities and resources of the enterprise, undertaken to achieve a goal.

A course of action represents what an enterprise has decided to do. Courses of action can be categorized as strategies and tactics. It is not possible to make a hard distinction between the two, but strategies tend to be long-term and fairly broad in scope, while tactics tend to be shorter-term and narrower in scope.

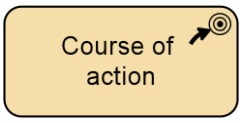
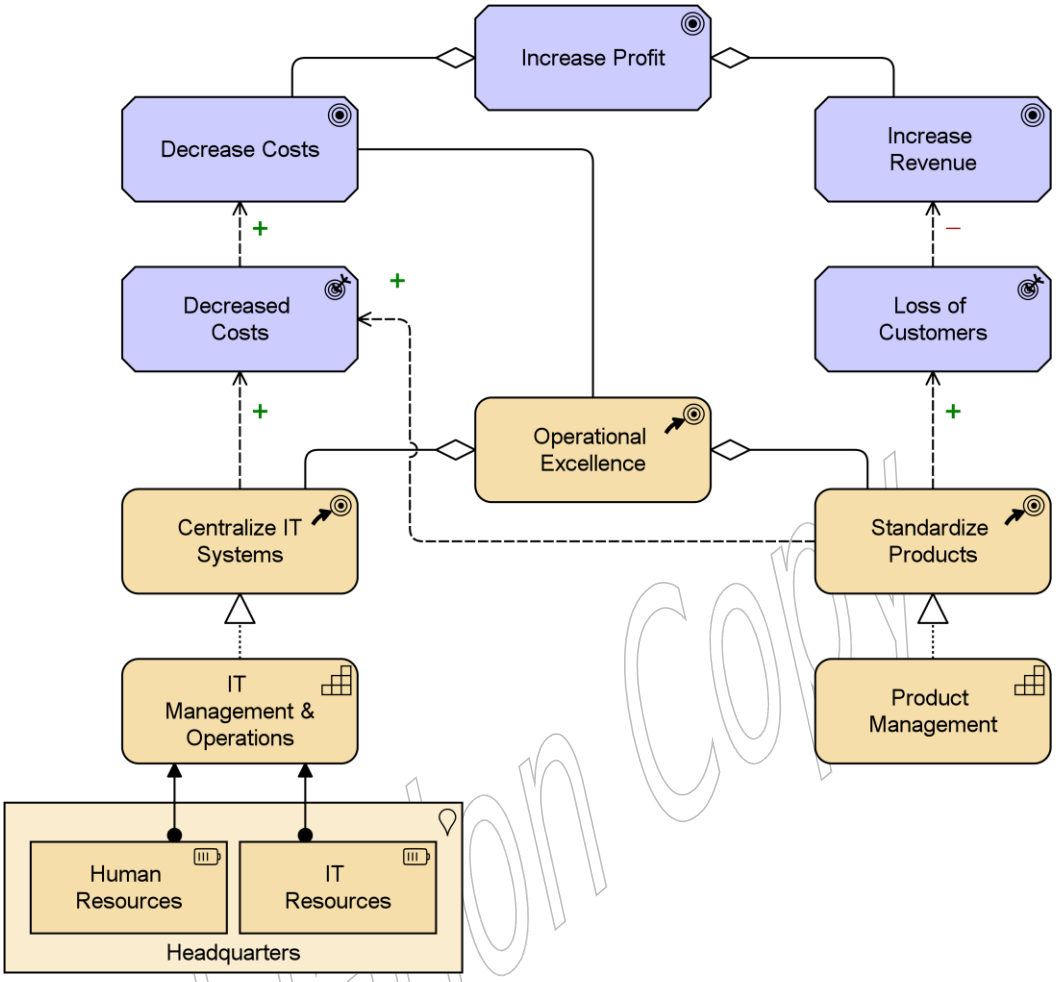


Figure 47: Course of Action Notation

7.4 Example

Increase Profit is a goal that can be decomposed into a number of other goals: Decrease Costs and Increase Revenue. The former is related to the Operation Excellence strategy of the company, modeled as a course of action. This is decomposed into two other courses of action: Centralize IT Systems and Standardize Products. These result in two outcomes: Decreased Costs and Loss of Customers, which influence the goals in positive and negative ways. This shows an important difference between goals and outcomes: not all outcomes lead to the intended results.

The courses of action are realized by a number of capabilities: IT Management & Operations and Product Management, and appropriate resources Human Resources and IT Resources are assigned to the former. The model fragment also shows that these resources are located in the Headquarters of the organization, in line with the Centralize IT Systems course of action.

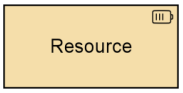
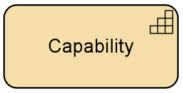
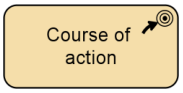


Example 21: Strategy Elements

7.5 Summary of Strategy Elements

Table 5 gives an overview of the strategy elements, with their definitions.

Table 5: Strategy Elements

Element	Description	Notation
Resource	An asset owned or controlled by an individual or organization.	
Capability	An ability that an active structure element, such as an organization, person, or system, possesses.	
Course of action	An approach or plan for configuring some capabilities and resources of the enterprise, undertaken to achieve a goal.	

7.6 Relationships with Motivation and Core Elements

Figure 48 shows how the strategy elements are related to core elements and motivation elements. Internal and external behavior elements may realize capabilities, while an active or passive structure element may realize a resource. Capabilities, courses of action, and resources may realize or influence requirements (and, indirectly, as described in Section 5.6, also principles or goals), and a course of action may also realize or influence an outcome (and, indirectly, also a goal).

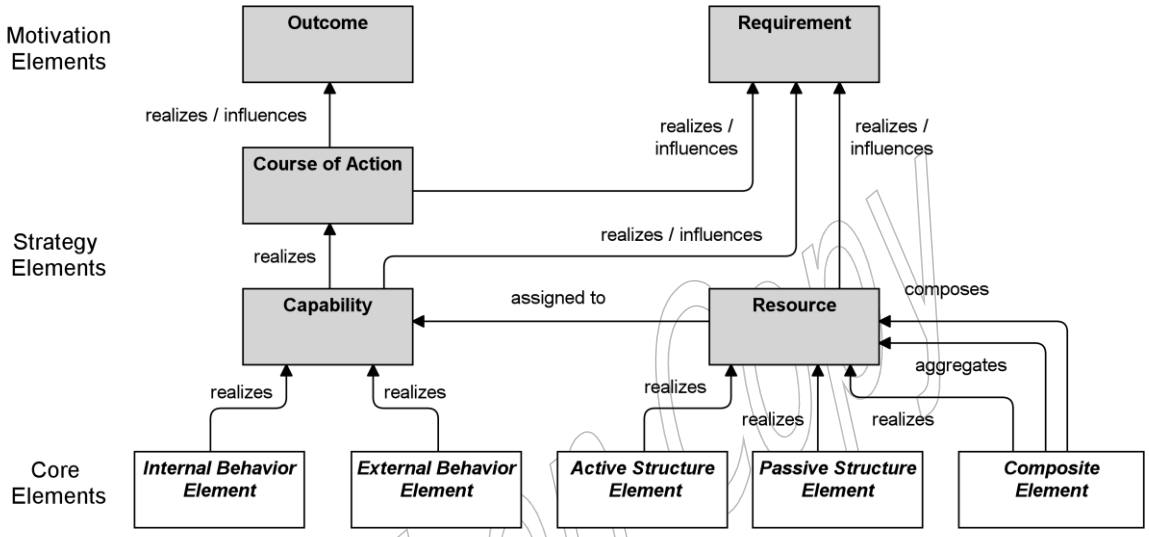


Figure 48: Relationships between Strategy Elements and Motivation and Core Elements

8 Business Layer

8.1 Business Layer Metamodel

Figure 49 gives an overview of the Business Layer elements and their relationships. Business internal active structure element, business internal behavior element, and business passive structure element are abstract elements; only their specializations (as defined in the following sections) are instantiated in models.

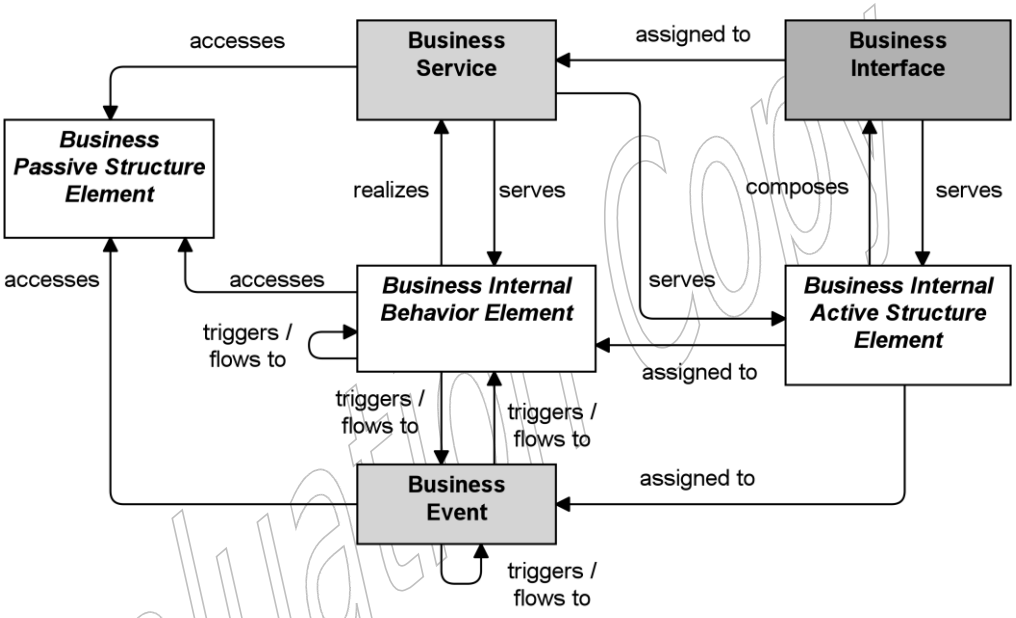


Figure 49: Business Layer Metamodel

Note: This figure does not show all permitted relationships; every element in the language can have composition, aggregation, and specialization relationships with elements of the same type; furthermore, there are indirect relationships that can be derived as explained in Section 5.6.

The Business Layer is typically used (often in conjunction with the strategy elements described in Chapter 7) to model the business architecture of an enterprise, defined by the TOGAF framework [4] as a description of the structure and interaction between the business strategy, organization, functions, business processes, and information needs.

8.2 Active Structure Elements

The active structure aspect of the Business Layer refers to the static structure of an organization, in terms of the entities that make up the organization and their relationships. The *active entities* are the subjects (e.g., business actors or business roles) that perform behavior such as business processes or functions (capabilities). Business actors may be individual persons (e.g., customers

or employees), but also groups of people (organization units) and resources that have a permanent (or at least long-term) status within the organizations. Typical examples of the latter are a department and a business unit.

Architectural descriptions focus on structure, which means that the inter-relationships of entities within an organization play an important role. To make this explicit, the element of business collaboration has been introduced.

The element of business interface is introduced to explicitly model the (logical or physical) places or channels where the services that a role offers to the environment can be accessed. The same service may be offered on a number of different interfaces; e.g., by mail, by telephone, or through the Internet. In contrast to application modeling, it is uncommon in current Business Layer modeling approaches to recognize the business interface element.

In the Business Layer, three types of internal active structure element are defined: *business actor*, *business role*, and *business collaboration*.

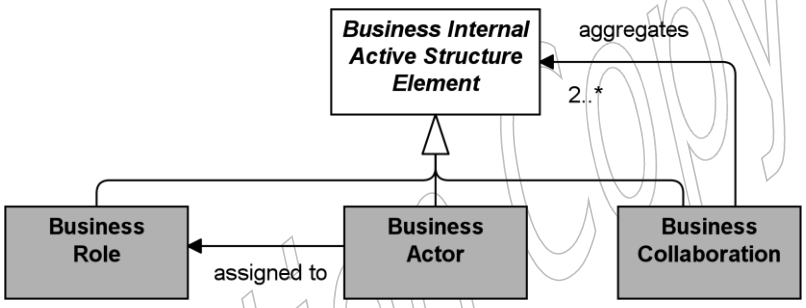


Figure 50: Business Internal Active Structure Elements

8.2.1 Business Actor

A business actor is a business entity that is capable of performing behavior.

A business actor is a business entity as opposed to a technical entity; i.e., it belongs to the Business Layer. Actors may, however, include entities outside the actual organization; e.g., customers and partners. A business actor can represent such business entities at different levels of detail, and may correspond to both an actor and an organizational unit in the TOGAF framework [4]. Examples of business actors are humans, departments, and business units.

A business actor may be assigned to one or more business roles. It can then perform the behavior to which these business roles are assigned. A business actor can be aggregated in a location. The name of a business actor should preferably be a noun. Business actors may be specific individuals or organizations; e.g., "John Smith" or "ABC Corporation", or they may be generic; e.g., "Customer" or "Supplier".

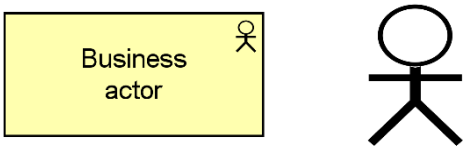


Figure 51: Business Actor Notation

8.2.2 Business Role

A business role is the responsibility for performing specific behavior, to which an actor can be assigned, or the part an actor plays in a particular action or event.

Business roles with certain responsibilities or skills are assigned to business processes or business functions. A business actor that is assigned to a business role is responsible that the corresponding behavior is carried out, either by performing it or by delegating and managing its performance. In addition to the relation of a business role with behavior, a business role is also useful in a (structural) organizational sense; for instance, in the division of labor within an organization.

A business role may be assigned to one or more business processes or business functions, while a business actor may be assigned to one or more business roles. A business interface or an application interface may serve a business role, while a business interface may be part of a business role (through a composition relationship, which is not shown explicitly in the interface notation). The name of a business role should preferably be a noun.

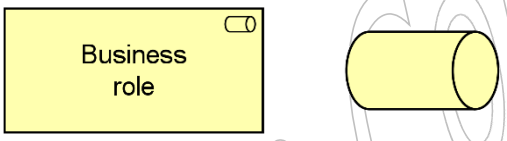


Figure 52: Business Role Notation

ArchiMate modelers may represent generic organizational entities that perform behavior as either business actors or business roles. For example, the business actor Supplier depicts an organizational entity, while the business role Supplier depicts a responsibility. Specific or generic business actors can be assigned to carry responsibilities depicted as business roles. For example, the specific business actor ABC Corporation or the generic business actor Business Partner can be assigned to the Supplier business role.

8.2.3 Business Collaboration

A business collaboration is an aggregate of two or more business internal active structure elements that work together to perform collective behavior.

A business process or function may be interpreted as the internal behavior of a single business role. In some cases, behavior is the collective effort of more than one business role; in fact, a collaboration of two or more business roles results in collective behavior which may be more than simply the sum of the behavior of the separate roles. Business collaborations represent this collective effort. Business interactions are used to describe the internal behavior that takes place within business collaboration. A collaboration is a (possibly temporary) collection of business roles or actors within an organization, which perform collaborative behavior (interactions). Unlike a department, which may also group roles, a business collaboration need not have an official (permanent) status within the organization; it is specifically aimed at a specific interaction or set of interactions between roles. It is especially useful in modeling B2B interactions between different organizations such as provider networks, and also for describing social networks.

A business collaboration may aggregate a number of business roles or actors, and may be assigned to one or more business interactions or other business internal behavior elements. A

business interface or an application interface may serve a business collaboration, while a business collaboration may have business interfaces (through composition, and also through aggregation via derived relationships). The name of a business collaboration should preferably be a noun. It is also rather common to leave a business collaboration unnamed.

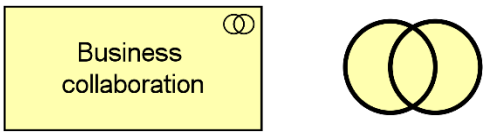


Figure 53: Business Collaboration Notation

8.2.4 Business Interface

A business interface is a point of access where a business service is made available to the environment.

A business interface exposes the functionality of a business service to other business roles or actors. It is often referred to as a channel (telephone, Internet, local office, etc.). The same business service may be exposed through different interfaces.

A business interface may be part of a business role or actor through a composition relationship, which is not shown in the standard notation, and a business interface may serve a business role. A business interface may be assigned to one or more business services, which means that these services are exposed by the interface. The name of a business interface should preferably be a noun.

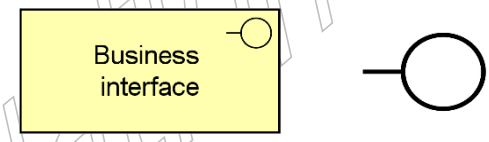
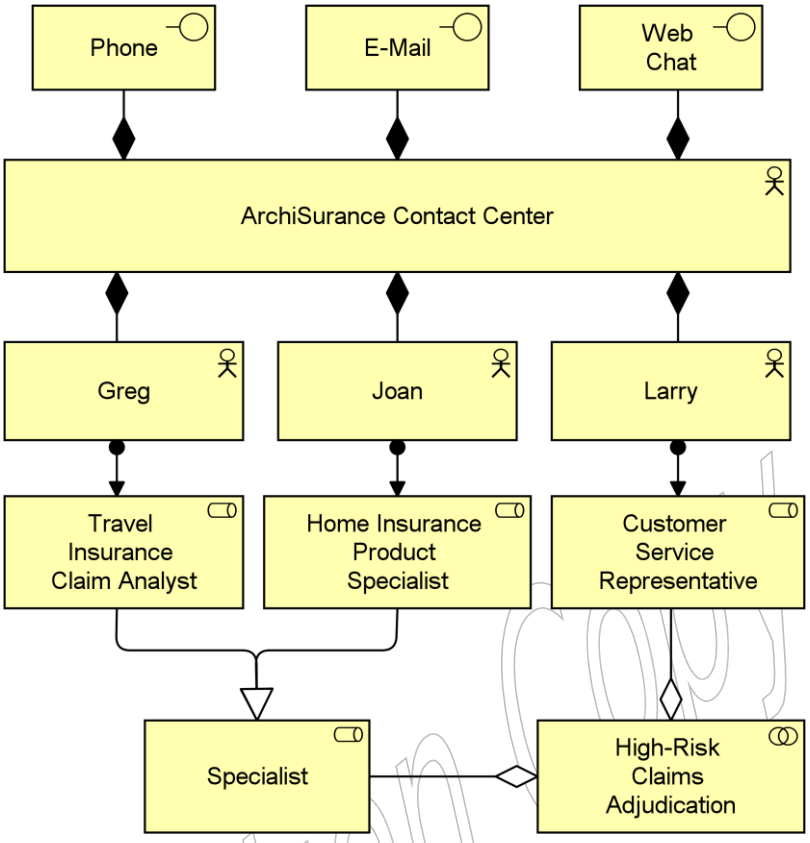


Figure 54: Business Interface Notation

8.2.5 Example

The ArchiSurance Contact Center, modeled as a business actor, is composed of three employees, also modeled as business actors: Greg, Joan, and Larry. The Contact Center has three business interfaces to serve customers: Phone, E-mail, and Web Chat. Greg fulfills the business role of Travel Insurance Claim Analyst, Joan fulfills the business role of Home Insurance Product Specialist, and Larry fulfills the business role of Customer Service Representative. The former two business roles are specializations of a business role Specialist. High-Risk Claims Adjudication is a business collaboration of two business roles: Specialist and Customer Service Representative.



Example 22: Business Active Structure Elements

8.3 Behavior Elements

Based on service-orientation, a crucial design decision for the behavioral part of the ArchiMate metamodel is the distinction between "external" and "internal" behavior of an organization.

The externally visible behavior is modeled by the element *business service*. A business service represents a coherent piece of functionality that offers added value to the environment, independent of the way this functionality is realized internally. A distinction can be made between "external" business services, offered to external customers, and "internal" business services, offering supporting functionality to processes or functions within the organization.

Several types of internal behavior elements that can realize a service are distinguished. Although the distinction between the two is not always sharp, it is often useful to distinguish a *process view* and a *function view* on behavior; two elements associated with these views, *business process* and *business function*, are defined. Both elements can be used to group more detailed business processes/functions, but based on different grouping criteria. A *business process* represents a workflow or value stream consisting of smaller processes/functions, with one or more clear starting points and leading to some result. It is sometimes described as "customer to customer", where this customer may also be an internal customer, in the case of sub-processes within an organization. The goal of such a business process is to "satisfy or delight the customer" [10]. A *business function* offers functionality that may be useful for one or more business processes. It groups behavior based on, for example, required skills, resources, (application) support, etc. Typically, the business processes of an organization are defined based

on the *products* and *services* that the organization offers, while the business functions are the basis for, for example, the assignment of resources to tasks and the application support.

A *business interaction* is a unit of behavior similar to a business process or function, but which is performed in a collaboration of two or more roles within the organization. Unlike the interaction concept in Amber [9], which is an *atomic* unit of collaborative behavior, the ArchiMate business interaction can be decomposed into smaller interactions. Although interactions are external behavior from the perspective of the roles participating in the collaboration, the behavior is internal to the collaboration as a whole. Similar to processes or functions, the result of a business interaction can be made available to the environment through a business service.

A *business event* is something that happens (externally) and may influence business processes, functions, or interactions. The business event element is similar to BPMN event elements, to the trigger element in Amber [9], and the initial state and final state elements in UML activity diagrams. However, the ArchiMate business event is more generally applicable in the sense that it can also be used to model other types of events, in addition to triggers.

In the Business Layer, three types of internal behavior element are defined: business process, business function, and business interaction.

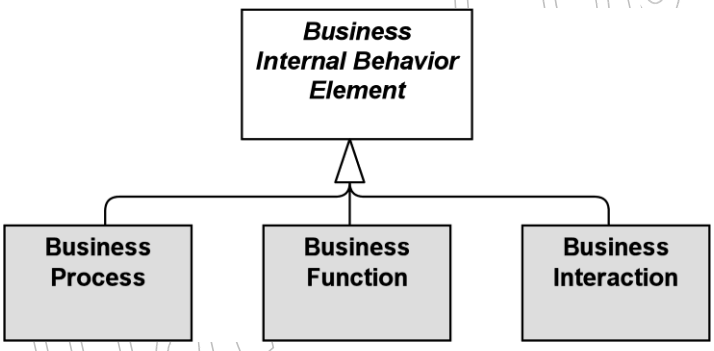


Figure 55: Business Internal Behavior Elements

8.3.1 Business Process

A business process represents a sequence of business behaviors that achieves a specific outcome such as a defined set of products or business services.

A business process describes the internal behavior performed by a business role that is required to produce a set of products and services. For a consumer, the products and services are relevant and the required behavior is merely a black box, hence the designation “internal”.

A complex business process may be an aggregation of other, finer-grained processes. To each of these, finer-grained roles may be assigned.

There is a potential many-to-many relationship between business processes and business functions. Informally speaking, processes describe some kind of “flow” of activities, whereas functions group activities according to required skills, knowledge, resources, etc.

A business process may be triggered by, or trigger, any other business behavior element (e.g., business event, business process, business function, or business interaction). A business process may access business objects. A business process may realize one or more business services and

may use (internal) business services or application services. A business role may be assigned to a business process to perform this process manually or automated, respectively. The name of a business process should clearly indicate a predefined sequence of actions, and may include the word "process". Examples are "adjudicate claim", "employee on-boarding", "approval process", or "financial reporting".

In an ArchiMate model, the existence of business processes is depicted. High-level business, end-to-end processes, macro flows, and workflows can all be expressed with the same business process element in the ArchiMate language. It does not, however, list the flow of activities in detail. This is typically done during business process modeling, where a business process can be expanded using a business process design language; e.g., BPMN [12].

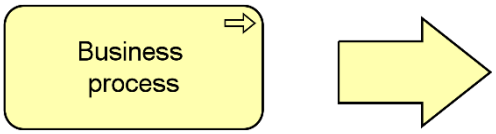


Figure 56: Business Process Notation

8.3.2 Business Function

A business function is a collection of business behavior based on a chosen set of criteria (typically required business resources and/or competences), closely aligned to an organization, but not necessarily explicitly governed by the organization.

Just like a business process, a business function also describes internal behavior performed by a business role. However, while a business process groups behavior based on a sequence or flow of activities that is needed to realize a product or service, a business function typically groups behavior based on required business resources, skills, competences, knowledge, etc.

There is a potential many-to-many relation between business processes and business functions. Complex processes in general involve activities that offer various functions. In this sense a business process forms a string of business functions. In general, a business function delivers added value from a business point of view. Organizational units or applications may coincide with business functions due to their specific grouping of business activities.

A business function may be triggered by, or trigger, any other business behavior element (business event, business process, business function, or business interaction). A business function may access business objects. A business function may realize one or more business services and may be served by business, application, or technology services. A business role may be assigned to a business function. The name of a business function should clearly indicate a well-defined behavior. Examples are customer management, claims administration, member services, recycling, or payment processing.

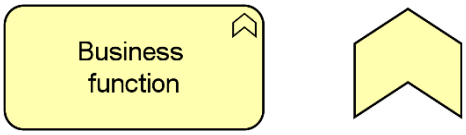


Figure 57: Business Function Notation

8.3.3 Business Interaction

A business interaction is a unit of collective business behavior performed by (a collaboration of) two or more business roles.

A business interaction is similar to a business process/function, but while a process/function may be performed by a single role, an interaction is performed by a collaboration of multiple roles. The roles in the collaboration share the responsibility for performing the interaction.

A business interaction may be triggered by, or trigger, any other business behavior element (business event, business process, business function, or business interaction). A business interaction may access business objects. A business interaction may realize one or more business services and may use (internal) business services or application services. A business collaboration or an application collaboration may be assigned to a business interaction. The name of a business interaction should preferably be a verb in the simple present tense.

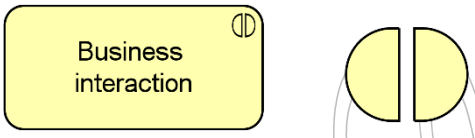


Figure 58: Business Interaction Notation

8.3.4 Business Event

A business event is a business behavior element that denotes an organizational state change. It may originate from and be resolved inside or outside the organization.

Business processes and other business behavior may be triggered or interrupted by a business event. Also, business processes may raise events that trigger other business processes, functions, or interactions. Unlike business processes, functions, and interactions, a business event is instantaneous: it does not have duration. Events may originate from the environment of the organization (e.g., from a customer), but also internal events may occur generated by, for example, other processes within the organization.

A business event may have a time attribute that denotes the moment or moments at which the event happens. For example, this can be used to model time schedules; e.g., to model an event that triggers a recurring business process to execute every first Monday of the month.

A business event may trigger or be triggered (raised) by a business process, business function, or business interaction. A business event may access a business object and may be composed of other business events. The name of a business event should preferably be a verb in the perfect tense; e.g., claim received.

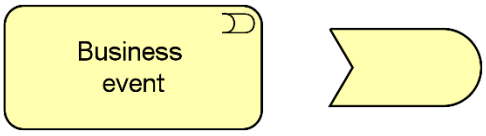


Figure 59: Business Event Notation

8.3.5 Business Service

A business service represents an explicitly defined exposed business behavior.

A business service exposes the functionality of business roles or collaborations to their environment. This functionality is accessed through one or more business interfaces. A business service is realized by one or more business processes, business functions, or business interactions that are performed by the business roles or business collaborations, respectively. It may access business objects.

A business service should provide a unit of behavior that is meaningful from the point of view of the environment. It has a purpose, which states this utility. The environment includes the (behavior of) users from outside as well as inside the organization. Business services can be external, customer-facing services (e.g., a travel insurance service) or internal support services (e.g., a resource management service).

A business service is associated with a value. A business service may serve a business process, business function, or business interaction. A business process, business function, or business interaction may realize a business service. A business interface may be assigned to a business service. A business service may access business objects. The name of a business service should preferably be a verb ending with "ing"; e.g., transaction processing. Also, a name explicitly containing the word "service" may be used.

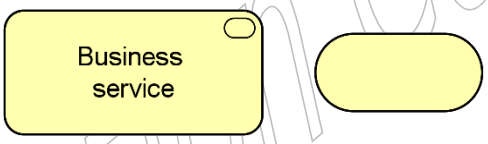
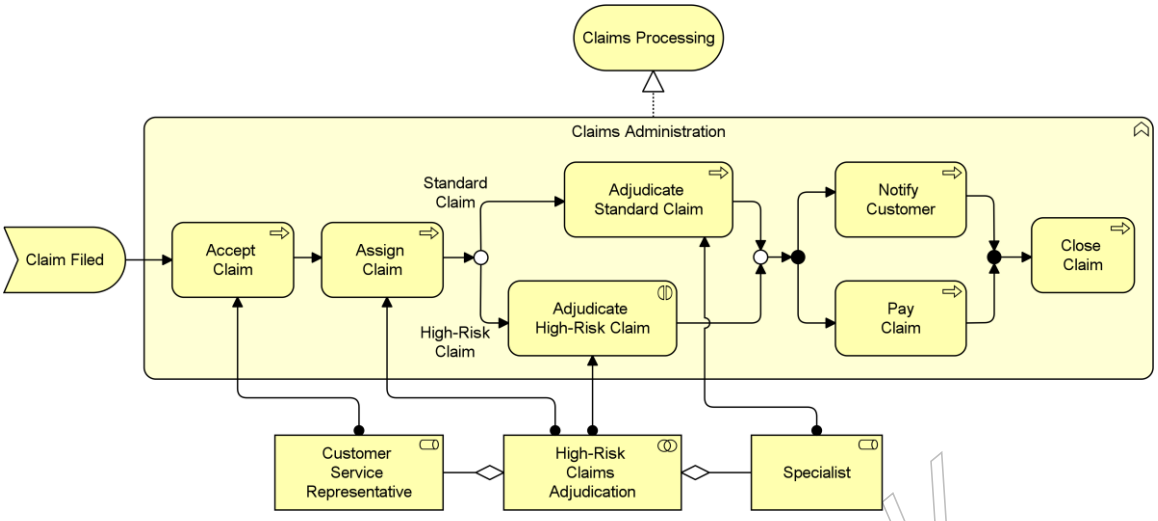


Figure 60: Business Service Notation

8.3.6 Example

Claims Administration is a business function that is composed of a number of business processes and a business interaction. This business function realizes a Claims Processing business service. A business event Claim Filed triggers the first business process, Accept Claim, which in turn triggers a business process Assign Claim. Depending on the type of claim, either the business process Adjudicate Standard Claim or the business interaction Adjudicate High-Risk Claim is performed. Adjudication of high-risk claims is a business interaction because, according to the company policy, two people should always be involved in this activity to minimize the risk of fraud. After adjudication, the business processes Notify Customer and Pay Claim are performed in parallel, and when both have finished, business process Close Claim is triggered.



Example 23: Business Behavior Elements

8.4 Passive Structure Elements

The passive structure aspect of the Business Layer contains the passive structure elements (business objects) that are manipulated by behavior, such as business processes or functions. The passive entities represent the important concepts in which the business thinks about a domain.

In the Business Layer, there are two main types of passive structure elements: business object and representation. Furthermore, a contract, used in the context of a product, is a specialization of a business object.

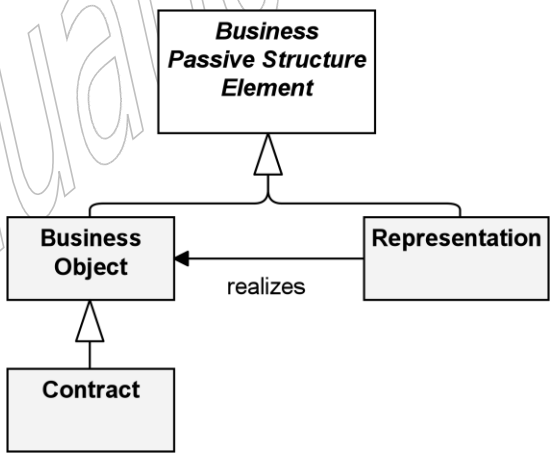


Figure 61: Business Passive Structure Elements

8.4.1 Business Object

A business object represents a concept used within a particular business domain.

As explained in Section 3.6, the ArchiMate language in general focuses on the modeling of types, not instances, since this is the most relevant at the Enterprise Architecture level of description. Hence a business object typically models an object type (*cf.* a UML class) of which

multiple instances may exist in operations. Only occasionally, business objects represent actual instances of information produced and consumed by behavior elements such as business processes. This is in particular the case for singleton types; i.e., types that have only one instance.

A wide variety of types of business objects can be defined. Business objects are passive in the sense that they do not trigger or perform processes. A business object could be used to represent information assets that are relevant from a business point of view and can be realized by data objects.

Business objects may be accessed (e.g., in the case of information objects, they may be created, read, written) by a business process, function, business interaction, business event, or business service. A business object may have association, specialization, aggregation, or composition relationships with other business objects. A business object may be realized by a representation or by a data object (or both). The name of a business object should preferably be a noun.

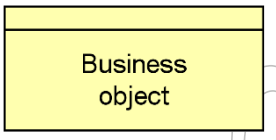


Figure 62: Business Object Notation

8.4.2 Contract

A contract represents a formal or informal specification of an agreement between a provider and a consumer that specifies the rights and obligations associated with a product and establishes functional and non-functional parameters for interaction.

The contract element may be used to model a contract in the legal sense, but also a more informal agreement associated with a product. It may also be or include a Service-Level Agreement (SLA), describing an agreement about the functionality and quality of the services that are part of a product. A contract is a specialization of a business object.

The relationships that apply to a business object also apply to a contract. In addition, a contract may have an aggregation relationship with a product. The name of a contract is preferably a noun.

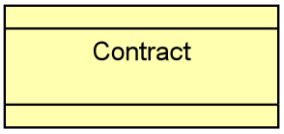


Figure 63: Contract Notation

8.4.3 Representation

A representation represents a perceptible form of the information carried by a business object.

Representations (for example, messages or documents) are the perceptible carriers of information that are related to business objects. If relevant, representations can be classified in various ways; for example, in terms of medium (electronic, paper, audio, etc.) or format (HTML,

ASCII, PDF, RTF, etc.). A single business object can have a number of different representations. Also, a single representation can realize one or more specific business objects.

A meaning can be associated with a representation that carries this meaning. The name of a representation is preferably a noun.

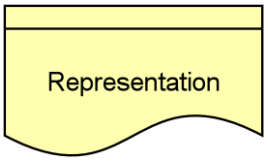
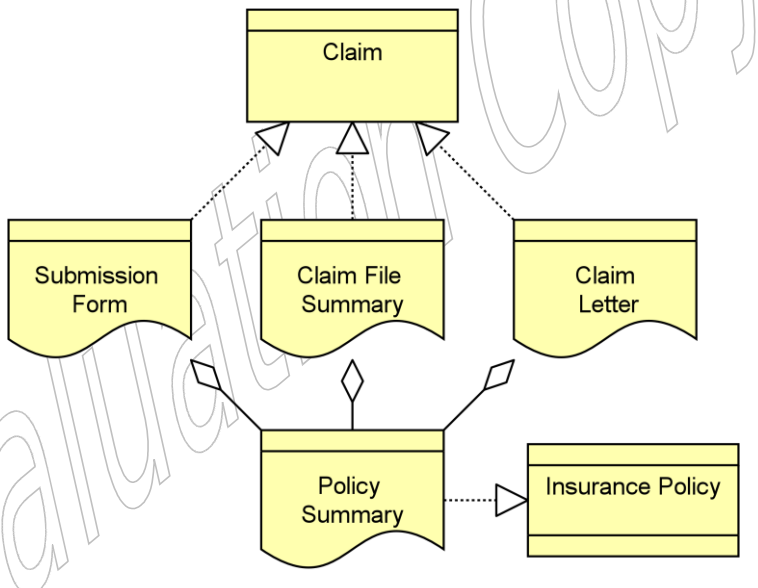


Figure 64: Representation Notation

8.4.4 Example

Business object Claim may be realized by either of the following three physical representations (in different stages of the Claims Administration process): Submission Form, Claim File Summary, or Claim Letter. All of these representations refer to a representation Policy Summary, which realizes a contract Insurance Policy.



Example 24: Business Passive Structure Elements

8.5 Composite Elements

The Business Layer contains one composite element: product. This aggregates or composes services and passive structure elements across the layers of the ArchiMate core language.

Figure 65 shows the applicable part of the metamodel. This crosses layers, as also described in Chapter 12.

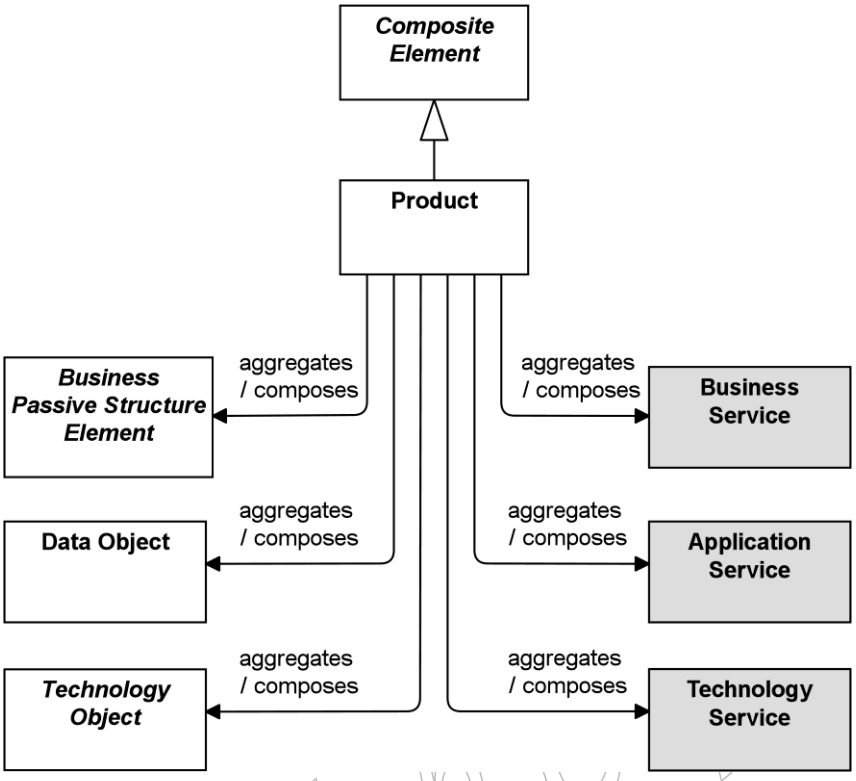


Figure 65: Product

8.5.1 Product

A product represents a coherent collection of services and/or passive structure elements, accompanied by a contract/set of agreements, which is offered as a whole to (internal or external) customers.

This definition covers both intangible, services-based, or information products that are common in information-intensive organizations, and tangible, physical products. A financial or information product consists of a collection of services, and a contract that specifies the characteristics, rights, and requirements associated with the product. "Buying" a product gives the customer the right to use the associated services.

Generally, the product element is used to specify a product *type*. The number of product types in an organization is typically relatively stable compared to, for example, the processes that realize or support the products. "Buying" is usually one of the services associated with a product, which results in a new instance of that product (belonging to a specific customer). Similarly, there may be services to modify or destroy a product.

A product may aggregate or compose business services, application services, and technology services, business objects, data objects, and technology objects, as well as a contract. Hence a product may aggregate or compose elements from other layers than the Business Layer.

A value may be associated with a product. The name of a product is usually the name which is used in the communication with customers, or possibly a more generic noun (e.g., "travel insurance").

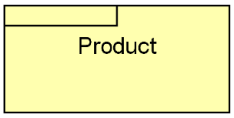
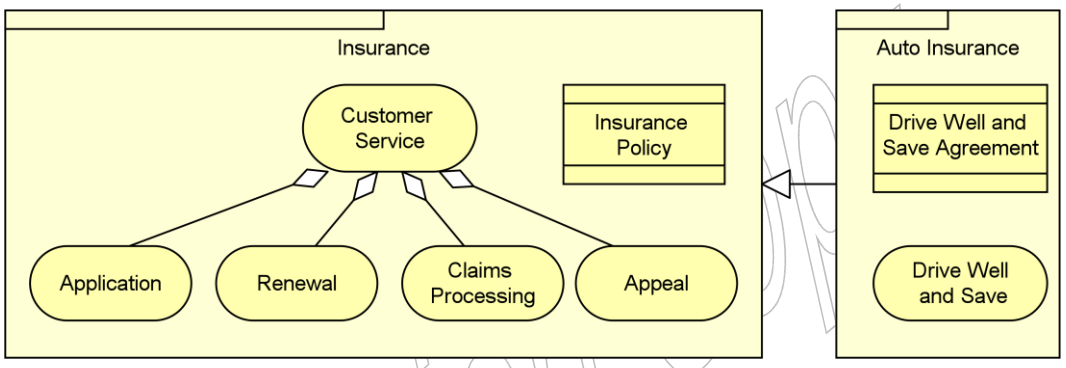


Figure 66: Product Notation

8.5.2 Example

A product Insurance consists of a contract Insurance Policy and a business service Customer Service, which aggregates four other business services: Application, Renewal, Claims Processing, and Appeal. An Auto Insurance product is a specialization of the generic Insurance product, with an additional business service Drive Well and Save, and accompanying contract Drive Well and Save Agreement.



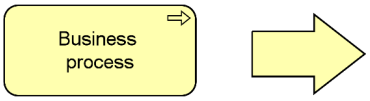
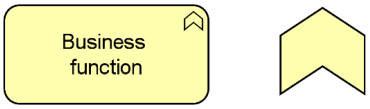
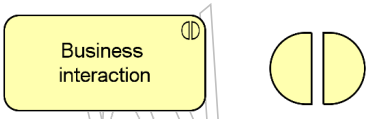
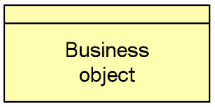
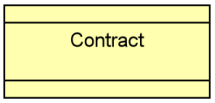
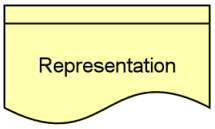
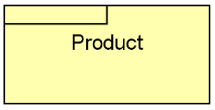
Example 25: Business Composite Element: Product

8.6 Summary of Business Layer Elements

Table 6 gives an overview of the Business Layer elements, with their definitions.

Table 6: Business Layer Elements

Element	Description	Notation
Business actor	A business entity that is capable of performing behavior.	Business actor
Business role	The responsibility for performing specific behavior, to which an actor can be assigned, or the part an actor plays in a particular action or event.	Business role
Business collaboration	An aggregate of two or more business internal active structure elements that work together to perform collective behavior.	Business collaboration
Business interface	A point of access where a business service is made available to the environment.	Business interface

Element	Description	Notation
Business process	A sequence of business behaviors that achieves a specific outcome such as a defined set of products or business services.	
Business function	A collection of business behavior based on a chosen set of criteria (typically required business resources and/or competences), closely aligned to an organization, but not necessarily explicitly governed by the organization.	
Business interaction	A unit of collective business behavior performed by (a collaboration of) two or more business roles.	
Business event	A business behavior element that denotes an organizational state change. It may originate from and be resolved inside or outside the organization.	
Business service	An explicitly defined exposed business behavior.	
Business object	A concept used within a particular business domain.	
Contract	A formal or informal specification of an agreement between a provider and a consumer that specifies the rights and obligations associated with a product and establishes functional and non-functional parameters for interaction.	
Representation	A perceptible form of the information carried by a business object.	
Product	A coherent collection of services and/or passive structure elements, accompanied by a contract/set of agreements, which is offered as a whole to (internal or external) customers.	

9 Application Layer

9.1 Application Layer Metamodel

Figure 67 gives an overview of the Application Layer elements and their relationships. Whenever applicable, inspiration has been drawn from the analogy with the Business Layer.

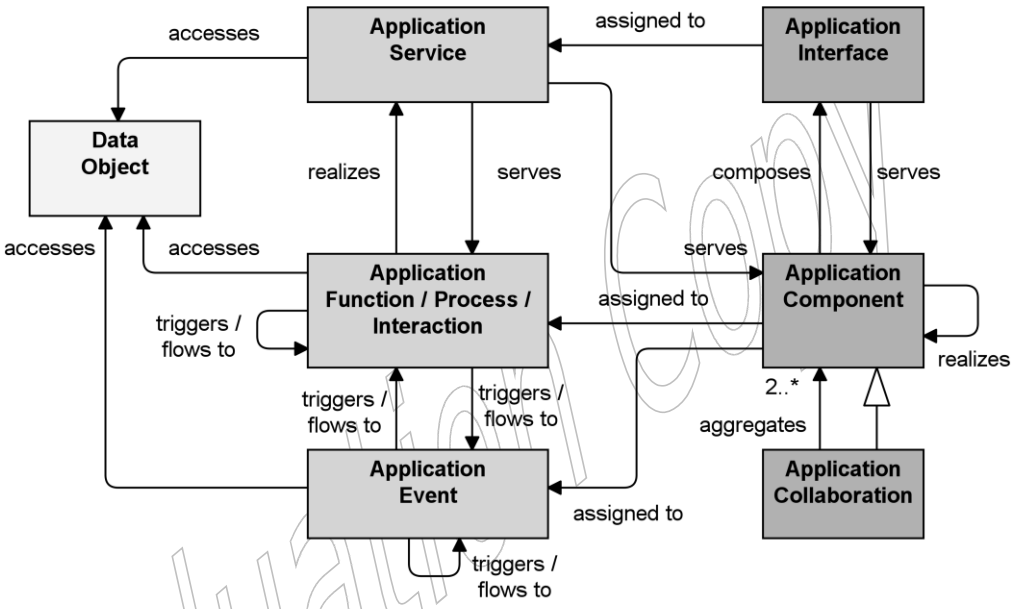


Figure 67: Application Layer Metamodel

Note: This figure does not show all permitted relationships: every element in the language can have composition, aggregation, and specialization relationships with elements of the same type; furthermore, there are indirect relationships that can be derived as explained in Section 5.6.

The Application Layer is typically used to model the information systems architectures of the enterprise, including the application architecture that, as defined by the TOGAF framework [4], describes the structure and interaction of the applications.

9.2 Active Structure Elements

The main active structure element for the Application Layer is the *application component*. This element is used to model any structural entity in the Application Layer: not just (re-usable) software components that can be part of one or more applications, but also complete software applications, sub-applications, or information systems. Although very similar to the UML component, the ArchiMate application component element strictly models the structural aspect of an application; its behavior is modeled by an explicit relationship to the behavior element.

Also in the application architecture, the inter-relationships of components are an essential ingredient. Therefore, we also introduce the element of *application collaboration* here, defined as a collective of application components which perform application interactions. The element is very similar to the collaboration as defined in the UML standard [7], [8].

In the purely structural sense, an *application interface* is the (logical) channel through which the services of a component can be accessed. In a broader sense (as used in, among others, the UML definition), an application interface defines some elementary behavioral characteristics: it defines the set of operations and events that are provided by the component, or those that are required from the environment. Thus, it is used to describe the functionality of a component. The application interface element can be used to model both *application-to-application* interfaces, which offer internal application services, and *application-to business* interfaces (and/or *user interfaces*), which offer external application services.

9.2.1 Application Component

An application component represents an encapsulation of application functionality aligned to implementation structure, which is modular and replaceable. It encapsulates its behavior and data, exposes services, and makes them available through interfaces.

An application component is a self-contained unit. As such, it is independently deployable, re-usable, and replaceable. An application component performs one or more application functions. It encapsulates its contents: its functionality is only accessible through a set of application interfaces. Cooperating application components are connected via application collaborations.

An application component may be assigned to one or more application functions. An application component has one or more application interfaces, which expose its functionality. Application interfaces of other application components may serve an application component. The name of an application component should preferably be a noun.

The application component element is used to model entire applications (i.e., deployed and operational IT systems, as defined by the TOGAF framework [4]) and individual parts of such applications, at all relevant levels of detail.

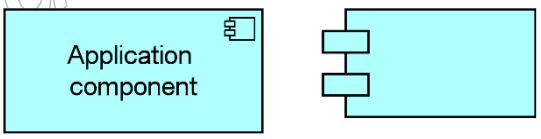


Figure 68: Application Component Notation

9.2.2 Application Collaboration

An application collaboration represents an aggregate of two or more application components that work together to perform collective application behavior.

An application collaboration specifies which components cooperate to perform some task. The collaborative behavior, including, for example, the communication pattern of these components, is modeled by an application interaction. An application collaboration typically models a logical or temporary collaboration of application components, and does not exist as a separate entity in the enterprise.

Application collaboration is a specialization of component, and aggregates two or more (cooperating) application components. An application collaboration is an active structure element that may be assigned to one or more application interactions, business interactions, or other application or business internal behavior elements, which model the associated behavior. An application interface may serve an application collaboration, and an application collaboration may be composed of application interfaces. The name of an application collaboration should preferably be a noun.

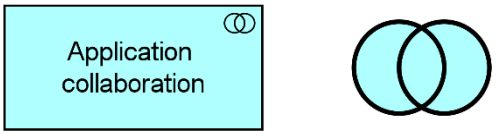


Figure 69: Application Collaboration Notation

9.2.3 Application Interface

An application interface represents a point of access where application services are made available to a user, another application component, or a node.

An application interface specifies how the functionality of a component can be accessed by other elements. An application interface exposes application services to the environment. The same application service may be exposed through different interfaces, and the same interface may expose multiple services.

In a sense, an application interface specifies a kind of contract that a component realizing this interface must fulfill. This may include parameters, protocols used, pre- and post-conditions, and data formats.

An application interface may be part of an application component through composition (not shown in the standard notation), which means that these interfaces are provided by that component, and can serve other application components. An application interface can be assigned to application services, which means that the interface exposes these services to the environment. The name of an application interface should preferably be a noun.

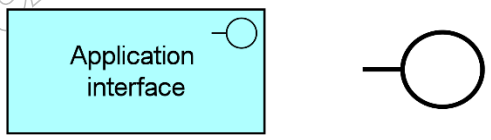
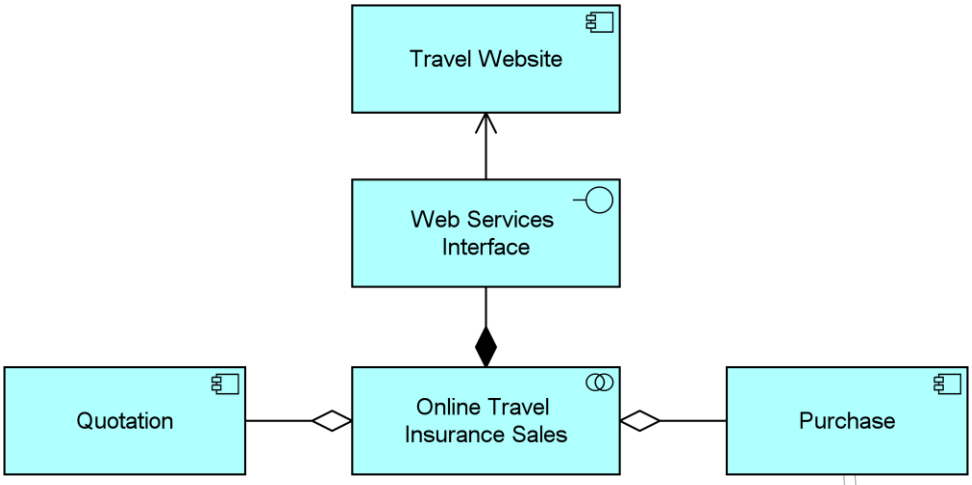


Figure 70: Application Interface Notation

9.2.4 Example

The Online Travel Insurance Sales application collaboration aggregates two application components: Quotation and Purchase. The application collaboration provides an application interface Web Services Interface that serves another application component Travel Website.



Example 26: Application Active Structure Elements

9.3 Behavior Elements

Behavior in the Application Layer is described in a way that is very similar to Business Layer behavior. Also here, a distinction is made between the external behavior of application components in terms of *application services*, and the internal behavior of these components; i.e., *application functions* that realize these services.

An *application service* is an externally visible unit of behavior, provided by one or more components, exposed through well-defined interfaces, and meaningful to the environment. The service element provides a way to explicitly describe the functionality that components share with each other and the functionality that they make available to the environment. The concept fits well within service-oriented application architecture. The functionality that an interactive computer program provides through a user interface is also modeled using an application service, exposed by an application-to-business interface representing the user interface. Internal application services are exposed through an application-to-application interface.

An *application function* describes the internal behavior of a component needed to realize one or more application services. In analogy with the Business Layer, an *application process* models an ordering of application behavior, as a counterpart of a business process. Note that the internal behavior of a component should in most cases not be modeled in too much detail in an architectural description, because for the description of this behavior we may soon be confronted with detailed design issues.

An *application interaction* is the behavior of a collaboration of two or more application components. An application interaction is external behavior from the perspective of each of the participating components, but the behavior is internal to the collaboration as a whole.

9.3.1 Application Function

An application function represents automated behavior that can be performed by an application component.

An application function describes the internal behavior of an application component. If this behavior is exposed externally, this is done through one or more services. An application function abstracts from the way it is implemented. Only the necessary behavior is specified.

An application function may realize one or more application services. Application services of other application functions and technology services may serve an application function. An application function may access data objects. An application component may be assigned to an application function (which means that the application component performs the application function). The name of an application function should preferably be a verb ending with "ing"; e.g., "accounting".

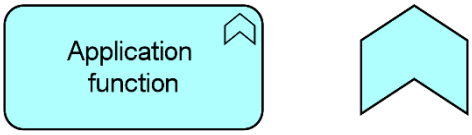


Figure 71: Application Function Notation

9.3.2 Application Interaction

An application interaction represents a unit of collective application behavior performed by (a collaboration of) two or more application components.

An application interaction describes the collective behavior that is performed by the components that participate in an application collaboration. This may, for example, include the communication pattern between these components. An application interaction can also specify the externally visible behavior needed to realize an application service. The details of the interaction between the application components involved in an application interaction can be expressed during the detailed application design using, for example, a UML interaction diagram.

An application collaboration may be assigned to an application interaction. An application interaction may realize an application service. Application services and technology services may serve an application interaction. An application interaction may access data objects. The name of an application interaction should clearly identify a series of application behaviors; e.g., "Client profile creation" or "Update customer records".

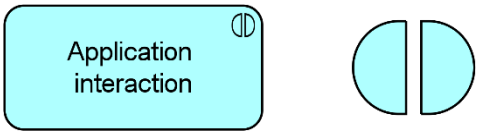


Figure 72: Application Interaction Notation

9.3.3 Application Process

An application process represents a sequence of application behaviors that achieves a specific outcome.

An application process describes the internal behavior performed by an application component that is required to realize a set of services. For a (human or automated) consumer the services are relevant and the required behavior is merely a black box, hence the designation "internal".

An application process may realize application services. Other application services may serve (be used by) an application process. An application process may access data objects. An application component may be assigned to an application process (which means that this component performs the process). The name of an application process should clearly identify a series of application behaviors; e.g., "Claims adjudication process", or "General ledger update job".

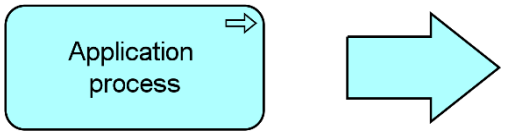


Figure 73: Application Process Notation

9.3.4 Application Event

An application event is an application behavior element that denotes a state change.

Application functions and other application behavior may be triggered or interrupted by an application event. Also, application behavior may raise events that trigger other application behavior. Unlike processes, functions, and interactions, an event is instantaneous; it does not have duration. Events may originate from the environment of the organization (e.g., from an external application), but also internal events may occur generated by, for example, other applications within the organization.

An application event may have a time attribute that denotes the moment or moments at which the event happens. For example, this can be used to model time schedules; e.g., an event that triggers a daily batch process.

An application event may trigger or be triggered (raised) by an application function, process, or interaction. An application event may access a data object and may be composed of other application events. The name of an application event should preferably be a verb in the perfect tense; e.g., "claim received".

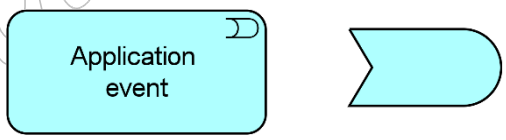


Figure 74: Application Event Notation

9.3.5 Application Service

An application service represents an explicitly defined exposed application behavior.

An application service exposes the functionality of components to their environment. This functionality is accessed through one or more application interfaces. An application service is realized by one or more application functions that are performed by the component. It may require, use, and produce data objects.

An application service should be meaningful from the point of view of the environment; it should provide a unit of behavior that is, in itself, useful to its users. It has a purpose, which

states this utility to the environment. This means, for example, that if this environment includes business processes, application services should have business relevance.

A purpose may be associated with an application service. An application service may serve business processes, business functions, business interactions, or application functions. An application function may realize an application service. An application interface may be assigned to an application service. An application service may access data objects. The name of an application service should preferably be a verb ending with "ing"; e.g., "transaction processing". Also, a name explicitly containing the word "service" may be used.

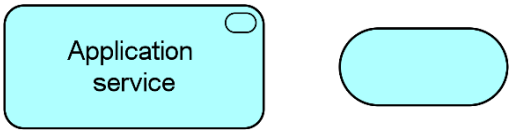
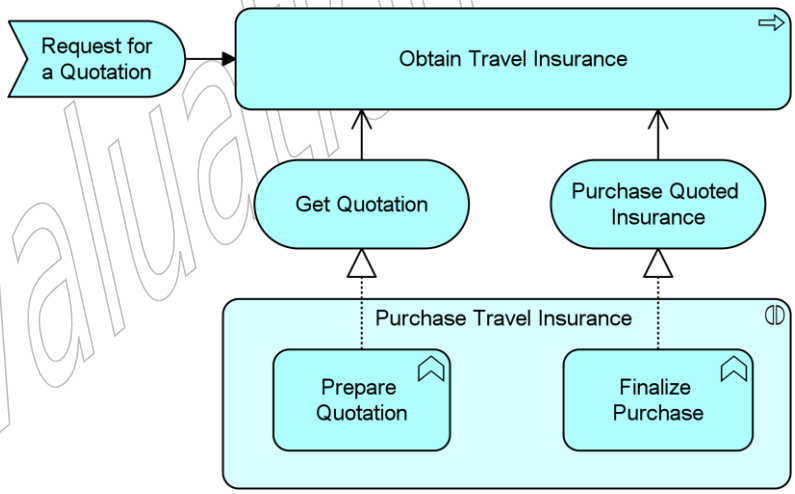


Figure 75: Application Service Notation

9.3.6 Example

The Purchase Travel Insurance application interaction is composed of two application functions: Prepare Quotation, realizing an application service Get Quotation, and Finalize Purchase, realizing an application service Purchase Quoted Insurance. This application interaction models the cooperative behavior of the Quotation and Purchase application components, modeled as the application collaboration Online Travel Insurance Sales in Example 26. An application event Request for a Quotation triggers an application process Obtain Travel Insurance, which is served by the two aforementioned application services.



Example 27: Application Behavior Elements

9.4 Passive Structure Elements

The passive counterpart of the application component in the Application Layer is called a *data object*. This element is used in the same way as data objects (or object types) in well-known data modeling approaches, most notably the "class" concept in UML class diagrams. A data object can be seen as a representation of a business object, as a counterpart of the representation element in the Business Layer. The ArchiMate language does not define a specific layer for

information; however, elements such as business objects and data objects are used to represent the information entities and also the logical data components that realize the business objects.

9.4.1 Data Object

A data object represents data structured for automated processing.

A data object should be a self-contained piece of information with a clear meaning to the business, not just to the application level. Typical examples of data objects are a customer record, a client database, or an insurance claim.

As explained in Section 3.6, the ArchiMate language in general focuses on the modeling of types, not instances, since this is the most relevant at the Enterprise Architecture level of description. Hence a data object typically models an object type (*cf.* a UML class) of which multiple instances may exist in operational applications. An important exception is when a data object is used to model a data collection such as a database, of which only one instance exists.

An application function or process can operate on data objects. A data object may be communicated via interactions and used or produced by application services. A data object can be accessed by an application function, application interaction, or application service. A data object may realize a business object, and may be realized by an artifact. A data object may have association, specialization, aggregation, or composition relationships with other data objects. The name of a data object should preferably be a noun.

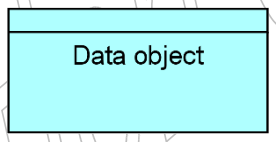
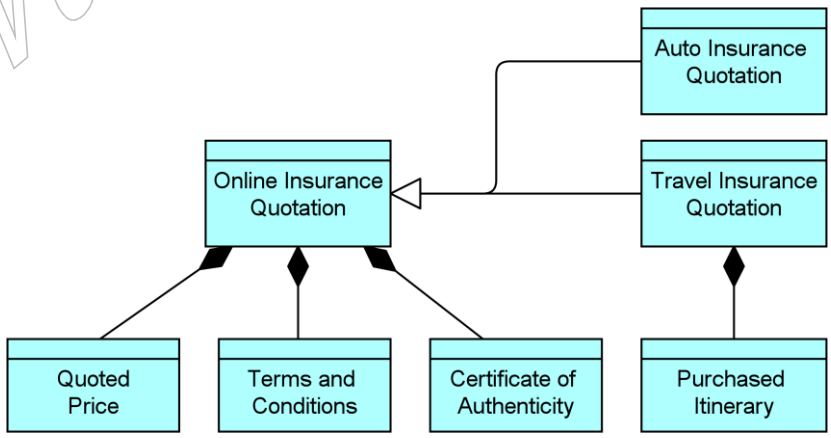


Figure 76: Data Object Notation

9.4.2 Example

An Online Insurance Quotation data object is composed of three other data objects: Quoted Price, Terms and Conditions, and Certificate of Authenticity. Auto Insurance Quotation and Travel Insurance Quotation are two specializations of the Online Insurance Quotation data object. Travel Insurance Quotation contains an additional data object Purchased Itinerary.

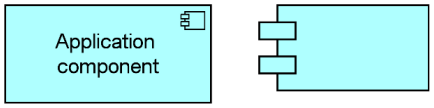
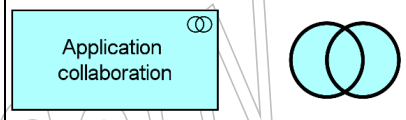
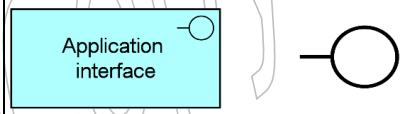
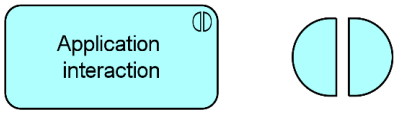
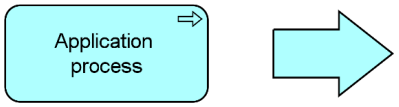
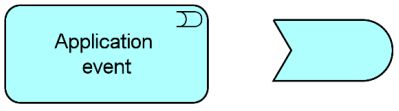
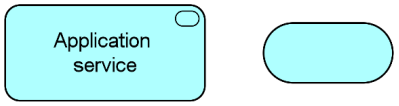
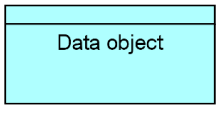


Example 28: Application Passive Structure Elements

9.5 Summary of Application Layer Elements

Table 7 gives an overview of the Application Layer elements, with their definitions.

Table 7: Application Layer Element

Element	Definition	Notation
Application component	An encapsulation of application functionality aligned to implementation structure, which is modular and replaceable. It encapsulates its behavior and data, exposes services, and makes them available through interfaces.	
Application collaboration	An aggregate of two or more application components that work together to perform collective application behavior.	
Application interface	A point of access where application services are made available to a user, another application component, or a node.	
Application function	Automated behavior that can be performed by an application component.	
Application interaction	A unit of collective application behavior performed by (a collaboration of) two or more application components.	
Application process	A sequence of application behaviors that achieves a specific outcome.	
Application event	An application behavior element that denotes a state change.	
Application service	An explicitly defined exposed application behavior.	
Data object	Data structured for automated processing.	

10 Technology Layer

10.1 Technology Layer Metamodel

Figure 77 gives an overview of the Technology Layer elements and their relationships. Whenever applicable, inspiration is drawn from the analogy with the Business and Application Layers.

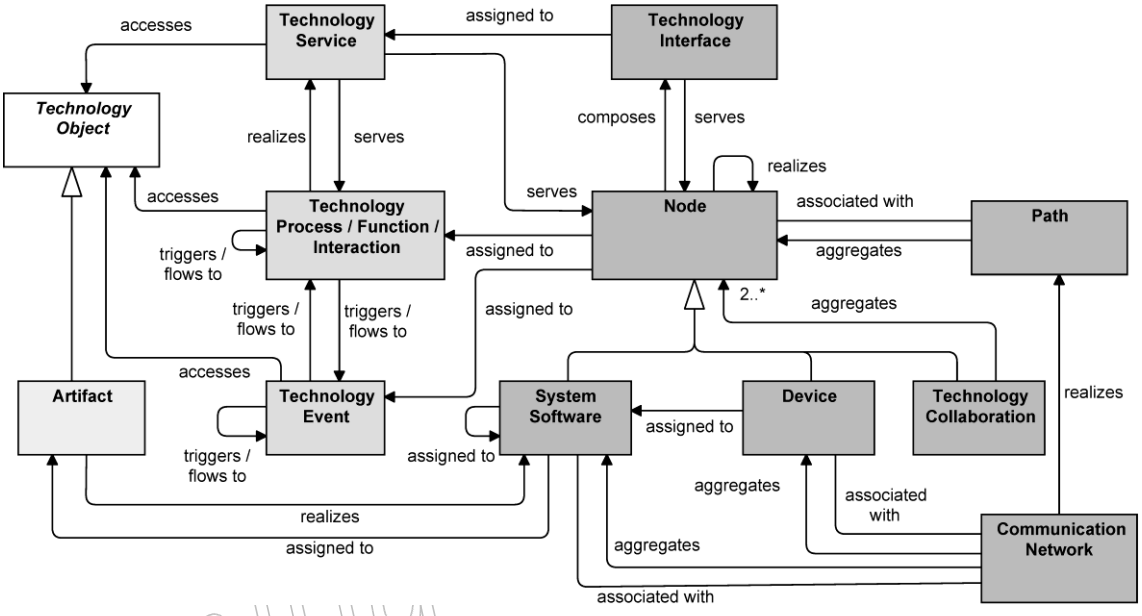


Figure 77: Technology Layer Metamodel

Note: This figure does not show all permitted relationships: every element in the language can have composition, aggregation, and specialization relationships with element of the same type; furthermore, there are indirect relationships that can be derived as explained in Section 5.6.

The Technology Layer is typically used to model the technology architecture of the enterprise, defined by the TOGAF framework [4] as: "*the structure and interaction of the platform services, and logical and physical technology components*".

10.2 Active Structure Elements

The main active structure element for the Technology Layer is the *node*. This element is used to model structural entities in this layer. It strictly models the structural aspect of a system: its behavior is modeled by an explicit relationship to the behavior element.

A *technology interface* is the (logical) place where the technology services offered by a node can be accessed by other nodes or by application components from the Application Layer.

Nodes come in two flavors: *device* and *system software*. A *device* models a physical computational resource, upon which artifacts may be deployed for execution. *System software* is an infrastructural software component running on a device. Typically, a node consists of a number of sub-nodes; for example, a device such as a server and system software to model the operating system.

The inter-relationships of components in the Technology Layer are mainly formed by the communication infrastructure. The *path* models the relation between two or more nodes, through which these nodes can exchange information. The physical realization of a path is modeled with a *network*; i.e., a physical communication medium between two or more devices (or other networks).

10.2.1 Node

A node represents a computational or physical resource that hosts, manipulates, or interacts with other computational or physical resources.

Nodes are active structure elements that perform technology behavior and execute, store, and process technology objects such as artifacts (or materials, as outlined in Chapter 11). For instance, nodes are used to model application platforms, defined by the TOGAF framework [4] as: "*a collection of technology components of hardware and software that provide the services used to support applications*".

Nodes can be interconnected by paths. A node may be assigned to an artifact to model that the artifact is deployed on the node.

The name of a node should preferably be a noun. A node may consist of sub-nodes.

Artifacts deployed on a node may either be drawn inside the node or connected to it with an assignment relationship.

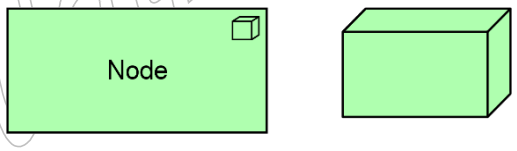


Figure 78: Node Notation

10.2.2 Device

A device is a physical IT resource upon which system software and artifacts may be stored or deployed for execution.

A device is a specialization of a node that represents a physical IT resource with processing capability. It is typically used to model hardware systems such as mainframes, PCs, or routers. Usually, they are part of a node together with system software. Devices may be composite; i.e., consist of sub-devices.

Devices can be interconnected by networks. Devices can be assigned to artifacts and to system software, to model that artifacts and system software are deployed on that device. A node can contain one or more devices.

The name of a device should preferably be a noun referring to the type of hardware; e.g., "IBM System z mainframe".

Different icons may be used to distinguish between different types of devices; e.g., mainframes and PCs.

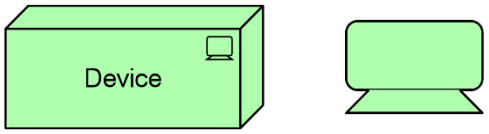


Figure 79: Device Notation

10.2.3 System Software

System software represents software that provides or contributes to an environment for storing, executing, and using software or data deployed within it.

System software is a specialization of a node that is used to model the software environment in which artifacts run. This can be, for example, an operating system, a JEE application server, a database system, or a workflow engine. Also, system software can be used to represent, for example, communication middleware. Usually, system software is combined with a device representing the hardware environment to form a general node.

A device or system software can be assigned to other system software; e.g., to model different layers of software running on top of each other. System software can be assigned to artifacts, to model that these artifacts are deployed on that software. System software can realize other system software. A node can be composed of system software.

The name of system software should preferably be a noun referring to the type of execution environment; e.g., "JEE server". System software may be composed of other system software; e.g., an operating system containing a database.

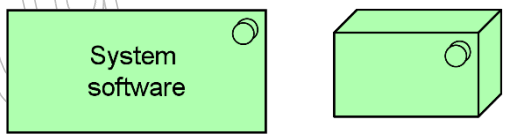


Figure 80: System Software Notation

10.2.4 Technology Collaboration

A technology collaboration represents an aggregate of two or more nodes that work together to perform collective technology behavior.

A technology collaboration specifies which nodes cooperate to perform some task. The collaborative behavior, including, for example, the communication pattern of these nodes, is modeled by a technology interaction. A technology collaboration typically models a logical or temporary collaboration of nodes, and does not exist as a separate entity in the enterprise.

Technology collaboration is a specialization of node, and aggregates two or more (cooperating) nodes. A technology collaboration is an active structure element that may be assigned to one or more technology interactions or other technology internal behavior elements, which model the

associated behavior. A technology interface may serve a technology collaboration, and a technology collaboration may be composed of technology interfaces. The name of a technology collaboration should preferably be a noun.

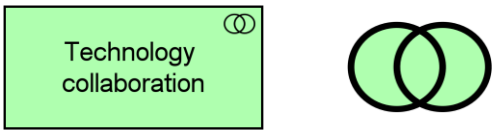


Figure 81: Technology Collaboration Notation

10.2.5 Technology Interface

A technology interface represents a point of access where technology services offered by a node can be accessed.

A technology interface specifies how the technology services of a node can be accessed by other nodes. A technology interface exposes a technology service to the environment. The same service may be exposed through different interfaces.

In a sense, a technology interface specifies a kind of contract that a component realizing this interface must fulfill. This may include, for example, parameters, protocols used, pre- and post-conditions, and data formats.

A technology interface may be part of a node through composition (not shown in the standard notation), which means that these interfaces are provided by that node, and can serve other nodes. A technology interface can be assigned to a technology service, to expose that service to the environment.

The name of a technology interface should preferably be a noun.

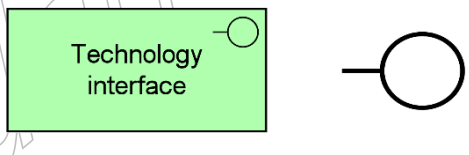


Figure 82: Technology Interface Notations

Note: In previous versions of this standard, this element was called ‘infrastructure interface’. This usage is still permitted but deprecated, and will be removed from a future version of the standard.

10.2.6 Path

A path represents a link between two or more nodes, through which these nodes can exchange data or material.

A path is used to model the logical communication (or distribution) relations between nodes. It is realized by one or more networks, which represent the physical communication (or distribution) links. The properties (e.g., bandwidth, latency) of a path are usually aggregated from these underlying networks.

A path connects two or more nodes. A path is realized by one or more networks. A path can aggregate nodes.

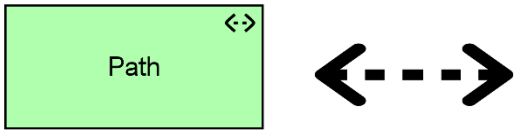


Figure 83: Path Notation, as Connection and as Box

10.2.7 Communication Network

A communication network represents a set of structures and behaviors that connects computer systems or other electronic devices for transmission, routing, and reception of data or data-based communications such as voice and video.

A communication network represents the physical communication infrastructure. It represents "*a set of products, concepts, and services that enable the connection of computer systems or devices for the purpose of transmitting data and other forms (e.g., voice and video) between the systems*", as defined by the TOGAF framework [4].

A communication network connects two or more devices. The most basic communication network is a single link between two devices, but it may comprise multiple links and associated network equipment. A network has properties such as bandwidth and latency. A communication network realizes one or more paths. It embodies the physical realization of the logical path between nodes.

A communication network can consist of sub-networks. It can aggregate devices and system software, for example, to model the routers, switches, and firewalls that are part of the network infrastructure.

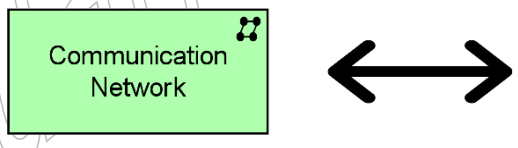
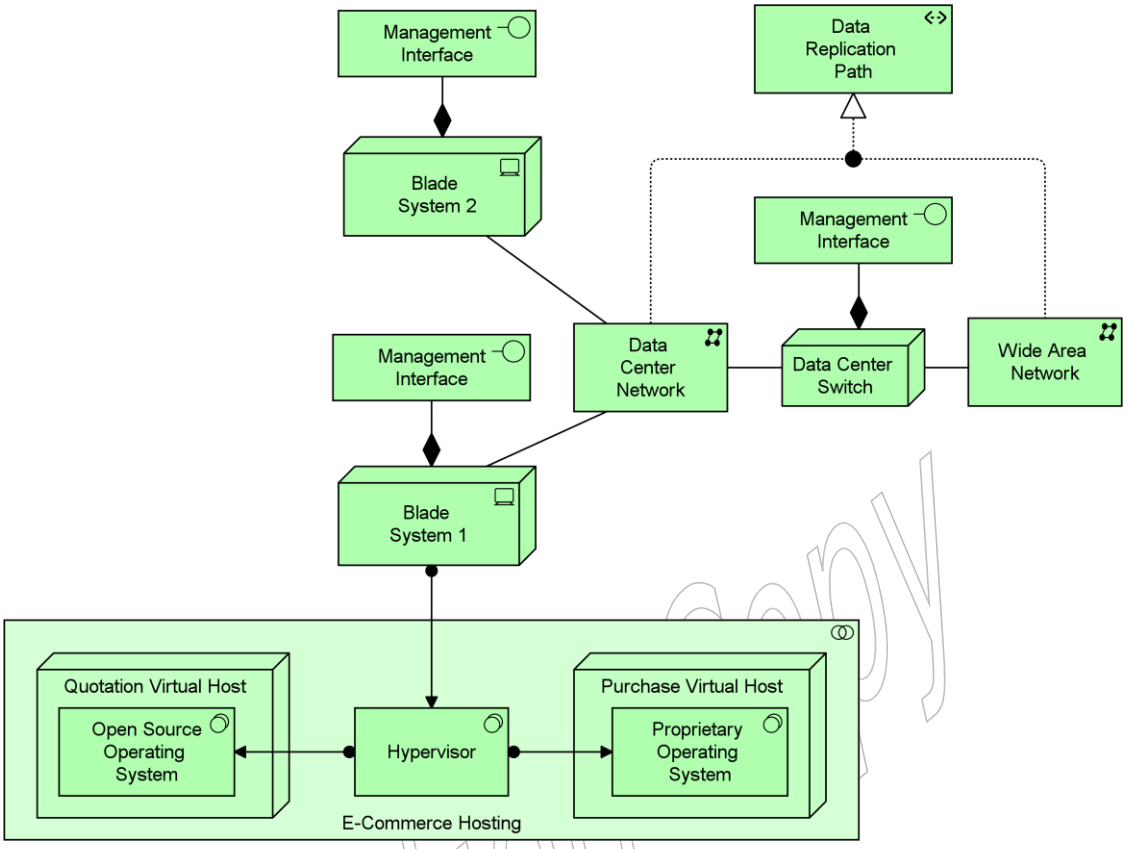


Figure 84: Network Notation, as Connection and as Box

Note: Formerly, this element was called 'network'. This usage is still permitted but deprecated, and will be removed from a future version of the standard.

10.2.8 Example

Two Blade System devices are connected to a communication network Data Center Network. This in turn is connected to another communication network Wide Area Network through a node Data Center Switch. The two communication networks together realize a path Data Replication Path. Both Blade System devices and the Data Center Switch node have a technology interface Management Interface. Device Blade System 1 deploys Hypervisor system software for hardware virtualization. Two system software components are deployed on the Hypervisor: an Open Source Operating System and a Proprietary Operating System, creating two virtual hosts, modeled as nodes Quotation Virtual Host and Purchase Virtual Host.



Example 29: Technology Active Structure Elements

10.3 Behavior Elements

Behavior elements in the Technology Layer are similar to the behavior elements in the other layers. Also here, a distinction is made between the external behavior of nodes in terms of *technology services*, and the internal behavior of these nodes; i.e., *technology functions* that realize these services.

10.3.1 Technology Function

A technology function represents a collection of technology behavior that can be performed by a node.

A technology function describes the internal behavior of a node; for the user of a node that performs a technology function, this function is invisible. If its behavior is exposed externally, this is done through one or more technology services. A technology function abstracts from the way it is implemented. Only the necessary behavior is specified.

A technology function may realize technology services. Technology services of other technology functions may serve technology functions. A technology function may access technology objects. A node may be assigned to a technology function (which means that the node performs the technology function). The name of a technology function should preferably be a verb ending with “ing”.

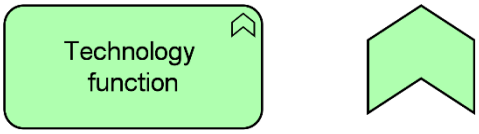


Figure 85: Technology Function Notation

Note: In previous versions of this standard, this element was called 'infrastructure function'. This usage is still permitted but deprecated, and will be removed from a future version of the standard.

10.3.2 Technology Process

A technology process represents a sequence of technology behaviors that achieves a specific outcome.

A technology process describes internal behavior of a node; for the user of that node, this process is invisible. If its behavior is exposed externally, this is done through one or more technology services. A technology process abstracts from the way it is implemented. Only the necessary behavior is specified. It can use technology objects as input and use or transform these to produce other technology objects as output.

A technology process may realize technology services. Other technology services may serve (be used by) a technology process. A technology process may access technology objects. A node may be assigned to a technology process, which means that this node performs the process. The name of a technology process should clearly identify a series of technology behaviors; e.g., "System boot sequence" or "Replicate database".

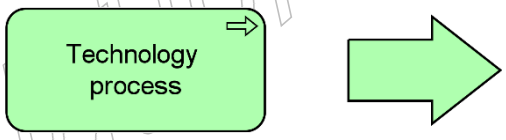


Figure 86: Technology Process Notation

10.3.3 Technology Interaction

A technology interaction represents a unit of collective technology behavior performed by (a collaboration of) two or more nodes.

A technology interaction describes the collective behavior that is performed by the nodes that participate in a technology collaboration. This may, for example, include the communication pattern between these components. A technology interaction can also specify the externally visible behavior needed to realize a technology service. The details of the interaction between the nodes involved in a technology interaction can be expressed during the detailed design using, for example, a UML interaction diagram.

A technology collaboration may be assigned to a technology interaction. A technology interaction may realize a technology service. Technology services may serve a technology interaction. A technology interaction may access artifacts. The name of a technology interaction should clearly identify a series of technology behaviors; e.g., "Client profile creation" or "Update customer records".

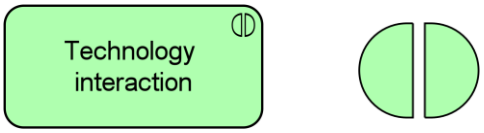


Figure 87: Technology Interaction Notation

10.3.4 Technology Event

A technology event is a technology behavior element that denotes a state change.

Technology functions and other technology behavior may be triggered or interrupted by a technology event. Also, technology functions may raise events that trigger other infrastructure behavior. Unlike processes, functions, and interactions, an event is instantaneous: it does not have duration. Events may originate from the environment of the organization, but also internal events may occur generated by, for example, other devices within the organization.

A technology event may have a time attribute that denotes the moment or moments at which the event happens. For example, this can be used to model time schedules; e.g., to model an event that triggers a recurring infrastructure function such as making a daily backup.

A technology event may trigger or be triggered (raised) by a technology function, process, or interaction. A technology event may access a data object and may be composed of other technology events. The name of a technology event should preferably be a verb in the perfect tense; e.g., "message received".

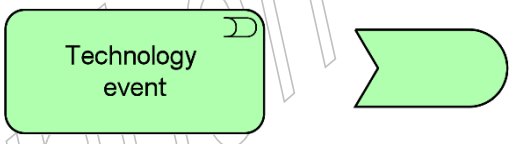


Figure 88: Technology Event Notation

10.3.5 Technology Service

A technology service represents an explicitly defined exposed technology behavior.

A technology service exposes the functionality of a node to its environment. This functionality is accessed through one or more technology interfaces. It may require, use, and produce artifacts.

A technology service should be meaningful from the point of view of the environment; it should provide a unit of behavior that is, in itself, useful to its users, such as application components and nodes.

Typical technology services may, for example, include messaging, storage, naming, and directory services. It may access artifacts; e.g., a file containing a message.

A technology service may serve application components or nodes. A technology service is realized by a technology function or process. A technology service is exposed by a node by assigning technology interfaces to it. A technology service may access artifacts. A technology service may consist of sub-services.

The name of a technology service should preferably be a verb ending with "ing"; e.g., "messaging". Also, a name explicitly containing the word "service" may be used.

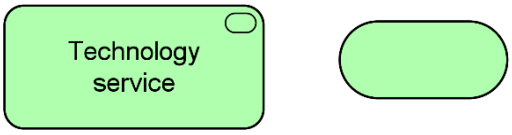
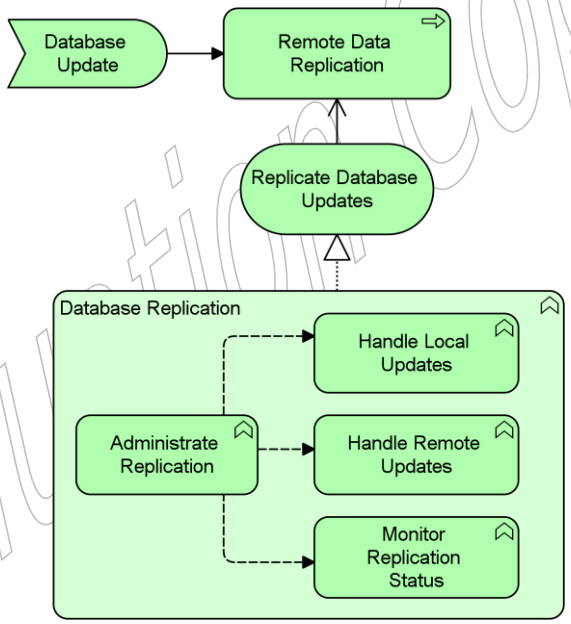


Figure 89: Technology Service Notation

Note: In previous versions of this standard, this element was called 'infrastructure service'. This usage is still permitted but deprecated, and will be removed from a future version of the standard.

10.3.6 Example

A technology event Database Update triggers a technology process Remote Data Replication, which is served by a technology service Replicate Database Updates. This technology service is realized by a technology function Database Replication, which is composed of four other technology functions: Administrate Replication, Handle Local Updates, Handle Remote Updates, and Monitor Replication Status. There are information flows from the Administrate Replication technology function to the other three technology functions.



Example 30: Technology Behavior Elements

10.4 Passive Structure Elements

A *technology object* models the passive structure elements that are used and processed by the infrastructure. An *artifact* is a physical piece of information that is used or produced in a software development process, or by deployment and operation of a system. It is the representation, in the form of, for example, a file, of a data object, or an application component, and can be deployed on a node. The artifact element has been taken from UML [8].

10.4.1 Technology Object

A technology object represents a passive element that is used or produced by technology behavior.

Technology objects represent the "physical" objects manipulated by the infrastructure of an enterprise. Technology objects are abstract elements; i.e., they are not instantiated in models but serve as the generic type of the things manipulated by the Technology Layer. This may include both artifacts (e.g., files) and physical material.

Technology objects may be accessed by technology behavior (functions, processes, interactions, events, and services). A technology object may have association, specialization, aggregation, or composition relationships with other technology objects. A technology object may realize a data object or business object. It may be realized by an artifact or material (from the physical elements). The name of a technology object should preferably be a noun.

10.4.2 Artifact

An artifact represents a piece of data that is used or produced in a software development process, or by deployment and operation of an IT system.

An artifact represents a tangible element in the IT world. Artifact is a specialization of technology object. It is typically used to model (software) products such as source files, executables, scripts, database tables, messages, documents, specifications, and model files. An instance (copy) of an artifact can be deployed on a node. An artifact could be used to represent a physical data component that realizes a data object.

An application component or system software may be realized by one or more artifacts. A data object may be realized by one or more artifacts. A node may be assigned to an artifact to model that the artifact is deployed on the node. Thus, the two typical ways to use the artifact element are as an *execution component* or as a *data file*. In fact, these could be defined as specializations of the artifact element.

The name of an artifact should preferably be the name of the file it represents; e.g., "order.jar". An artifact may consist of sub-artifacts.

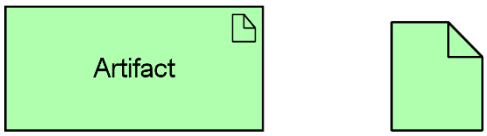
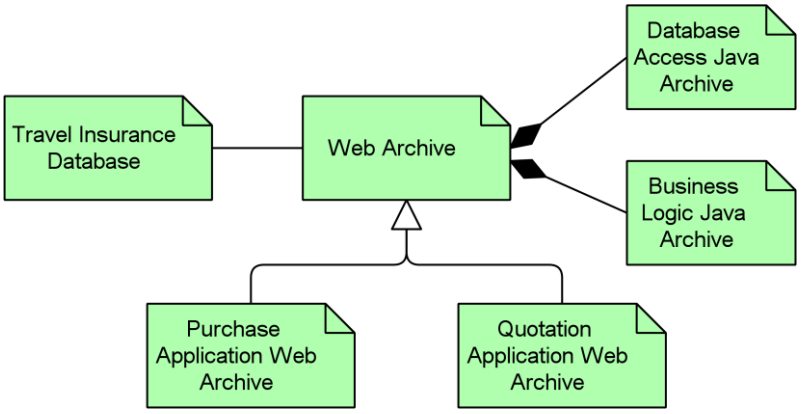


Figure 90: Artifact Notation

10.4.3 Example

A Web Archive artifact (which may realize an application component) is composed of two other artifacts: Database Access Java Archive and Business Logic Java Archive. Two specializations of the Web Archive artifact are a Purchase Application Web Archive and a Quotation Application Web Archive. A Travel Insurance Database artifact (which may realize a data object) is associated with the Web Archive artifact.

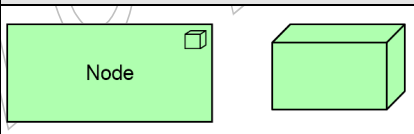
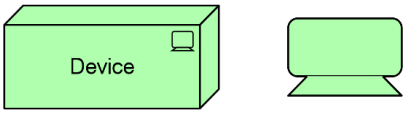
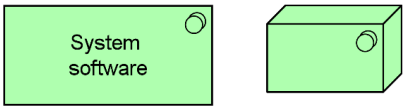
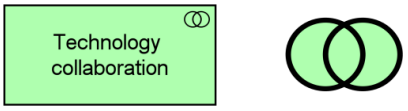
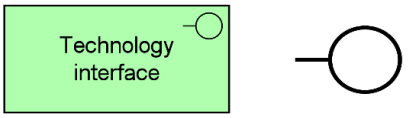
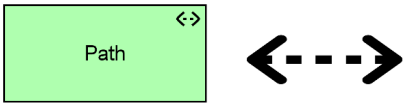


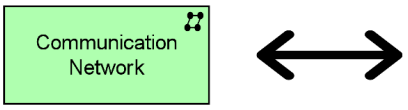
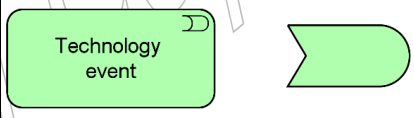
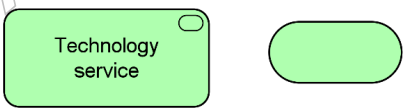
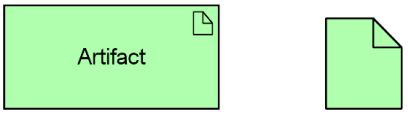
Example 31: Technology Passive Structure Element: Artifact

10.5 Summary of Technology Layer Elements

Table 8 gives an overview of the Technology Layer elements, with their definitions.

Table 8: Technology Layer Elements

Element	Definition	Notation
Node	A computational or physical resource that hosts, manipulates, or interacts with other computational or physical resources.	
Device	A physical IT resource upon which system software and artifacts may be stored or deployed for execution.	
System software	Software that provides or contributes to an environment for storing, executing, and using software or data deployed within it.	
Technology collaboration	An aggregate of two or more nodes that work together to perform collective technology behavior.	
Technology interface	A point of access where technology services offered by a node can be accessed.	
Path	A link between two or more nodes, through which these nodes can exchange data or material.	

Element	Definition	Notation
Communication network	A set of structures and behaviors that connects computer systems or other electronic devices for transmission, routing, and reception of data or data-based communications such as voice and video.	
Technology function	A collection of technology behavior that can be performed by a node.	
Technology process	A sequence of technology behaviors that achieves a specific outcome.	
Technology interaction	A unit of collective technology behavior performed by (a collaboration of) two or more nodes.	
Technology event	A technology behavior element that denotes a state change.	
Technology service	An explicitly defined exposed technology behavior.	
Technology object	A passive element that is used or produced by technology behavior.	Abstract element
Artifact	A piece of data that is used or produced in a software development process, or by deployment and operation of a system.	

11 Physical Elements

11.1 Physical Elements Metamodel

Figure 91 gives an overview of the physical elements and their relationships. These are based on the Technology Layer.

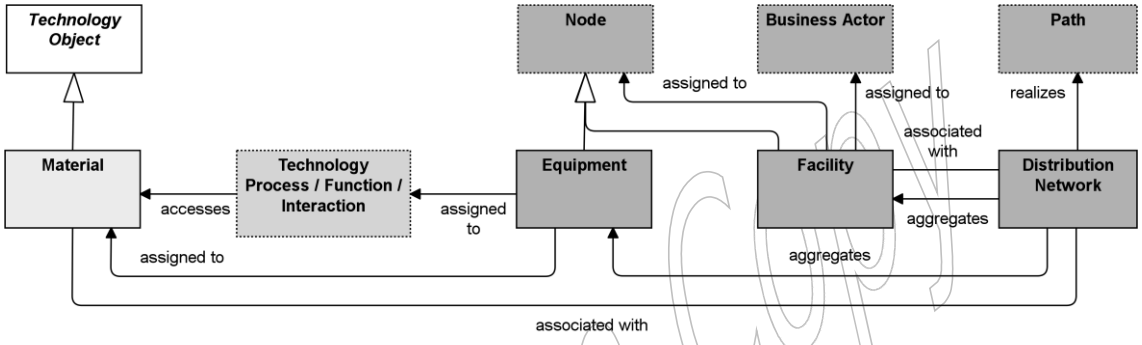


Figure 91: Physical Elements Metamodel

Note: This figure does not show all permitted relationships: every element in the language can have composition, aggregation, and specialization relationships with elements of the same type; furthermore, there are indirect relationships that can be derived as explained in Section 5.6.

11.2 Active Structure Elements

The *equipment* element is the main active structure element within the physical elements. This element is used to model structural entities in this layer. It is used to model any physical machinery, tools, instruments, or implements. It strictly models the structural aspect of a system; its behavior is modeled by an explicit relationship to the behavior elements.

The inter-relationships of physical elements are mainly formed by the logistics infrastructure. The path element from the Technology Layer models the relation between two or more nodes, through which these nodes can exchange information or material. The physical realization of a path is modeled with a *distribution network*; i.e., a physical connection between two or more pieces of equipment (or other physical networks). This can be used to model, for example, rail or road networks, the water supply, power grid, or gas network.

11.2.1 Equipment

Equipment represents one or more physical machines, tools, or instruments that can create, use, store, move, or transform materials.

Equipment comprises all active processing elements that carry out physical processes in which materials (which are a special kind of technology object) are used or transformed. Equipment is

a specialization of the node element from the Technology Layer. Therefore, it is possible to model nodes that are formed by a combination of IT infrastructure (devices, system software) and physical infrastructure (equipment); e.g., an MRI scanner at a hospital, a production plant with its control systems, etc.

Material can be accessed (e.g., created, used, stored, moved, or transformed) by equipment. Equipment can serve other equipment, and also other active structure elements such as business roles and actors, and facilities can be assigned to equipment. A piece of equipment can be composed of other pieces of equipment. Equipment can be assigned to (i.e., installed and used in or on) a facility and can be aggregated in a location.

The name of a piece of equipment should preferably be a noun.

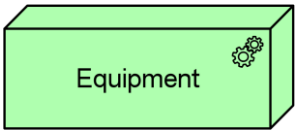


Figure 92: Equipment Notation

A useful specialization of equipment is vehicle, for describing, for example, trucks, cars, trains, ships, and airplanes.

11.2.2 Facility

A facility represents a physical structure or environment.

A facility is a specialization of a node. It represents a physical resource that has the capability of facilitating (e.g., housing or locating) the use of equipment. It is typically used to model factories, buildings, or outdoor constructions that have an important role in production or distribution processes. Examples of facilities include a factory, a laboratory, a warehouse, a shopping mall, a cave, or a spaceship. Facilities may be composite; i.e., consist of sub-facilities.

Facilities can be interconnected by distribution networks. Material can be accessed (e.g., created, used, stored, moved, or transformed) by equipment. A facility can serve other facilities, and also other active structure elements such as business roles and actors, and locations can be assigned to facilities. A facility can be composed of other facilities and can be aggregated in a location.

The name of a facility should preferably be a noun referring to the type of facility; e.g., "Rotterdam harbor oil refinery".

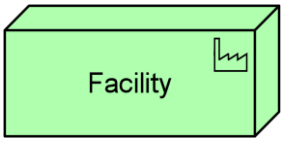


Figure 93: Facility Notation

11.2.3 Distribution Network

A distribution network represents a physical network used to transport materials or energy.

A distribution network represents the physical distribution or transportation infrastructure. It embodies the physical realization of the logical paths between nodes.

A distribution network connects two or more nodes. A distribution network may realize one or more paths. A distribution network can consist of sub-networks and can aggregate facilities and equipment, for example, to model railway stations and trains that are part of a rail network.

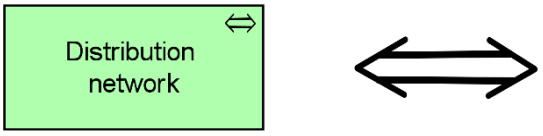


Figure 94: Distribution Network Notation

11.3 Behavior Elements

No separate physical behavior elements are defined. Rather, the behavior elements from the Technology Layer (technology function, process, interaction, service, and event) are used to model the behavior of all nodes, including physical equipment. Since equipment will very often be computer-controlled or in other ways have a close relationship to IT (also think of sensors, Internet of Things), their behavior can be described in an integral way using the existing technology behavior concepts.

11.4 Passive Structure Elements

11.4.1 Material

Material represents tangible physical matter or physical elements.

Material represents tangible physical matter, with attributes such as size and weight. It is typically used to model raw materials and physical products, and also energy sources such as fuel. Material can be accessed by physical processes.

The name of material should be a noun. Pieces of material may be composed of other pieces of material.

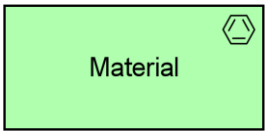
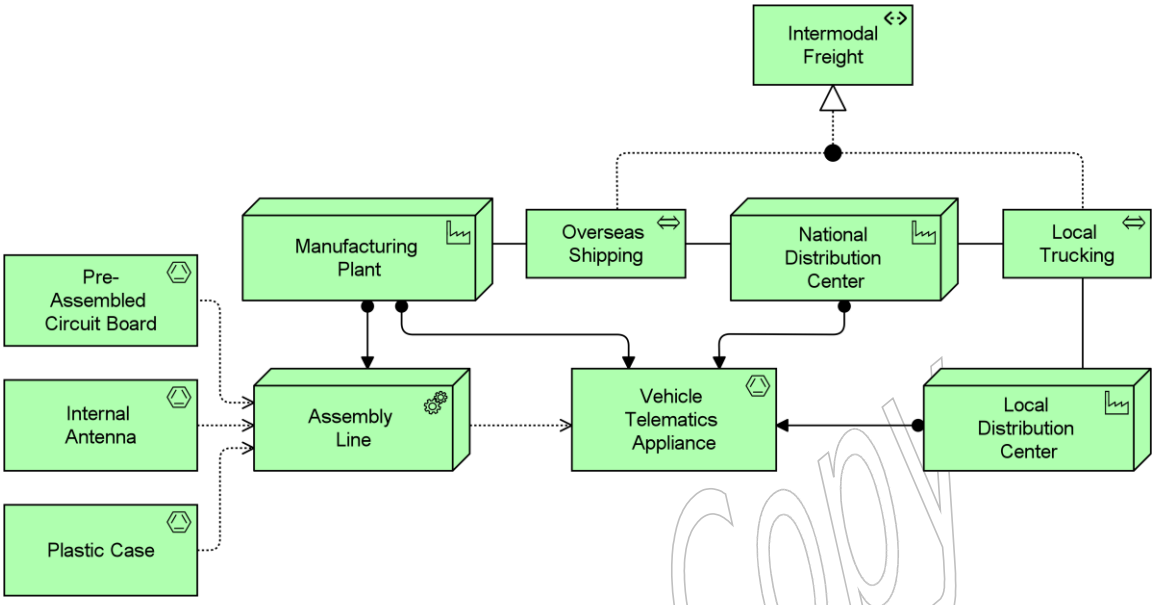


Figure 95: Material Notation

11.5 Example

An Assembly Line, modeled as equipment, and installed at a facility Manufacturing Plant, makes use of materials Pre-Assembled Circuit Board, Internal Antenna, and Plastic Case to produce material Vehicle Telematics Appliance. The appliance, initially located at the Manufacturing Plant facility, is subsequently transported to the facilities National Distribution Center and Local Distribution Center, making use of the distribution networks Overseas

Shipping and Local Trucking. These distribution networks together realize the path Intermodal Freight.

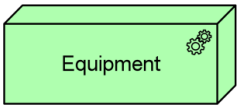
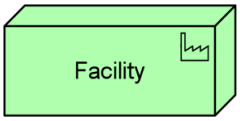
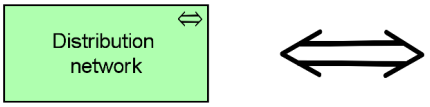
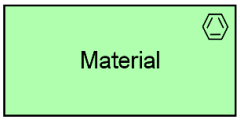


Example 32: Physical Elements

11.6 Summary of Physical Elements

Table 9 gives an overview of the physical elements, with their definitions.

Table 9: Physical Elements

Element	Definition	Notation
Equipment	One or more physical machines, tools, or instruments that can create, use, store, move, or transform materials.	
Facility	a physical structure or environment.	
Distribution network	A physical network used to transport materials or energy.	
Material	Tangible physical matter or physical elements.	

12 Cross-Layer Dependencies

The previous chapters have presented the concepts to model the Business, Application, and Technology Layers of an enterprise. However, a central issue in Enterprise Architecture is business-IT alignment: how can these layers be matched? This chapter describes the relationships that the ArchiMate language offers to model the link between business, applications, and technology.

12.1 Alignment of Business Layer and Lower Layers

Figure 96 shows the relationships between the Business Layer, the Application Layer, and the Technology Layer elements. There are two main types of relationships between these layers:

1. *Serving* relationships; for example, between application service and the different types of business behavior elements, and between application interface and business role; *vice versa*, serving relationships between business service and application behavior elements, and between business interface and application component. These relationships represent the behavioral and structural aspects of the support of the business by applications.

2. *Realization* relationships; for example, from an application process or function to a business process or function, or from a data object or a technology object to a business object, to indicate that the data object is a digital representation of the corresponding business object, or the technology object is a physical representation of the business object.

In addition, there may be an aggregation relationship between a product and an application or technology service, and a data or technology object, to indicate that these services or objects can be offered directly to a customer as part of the product.

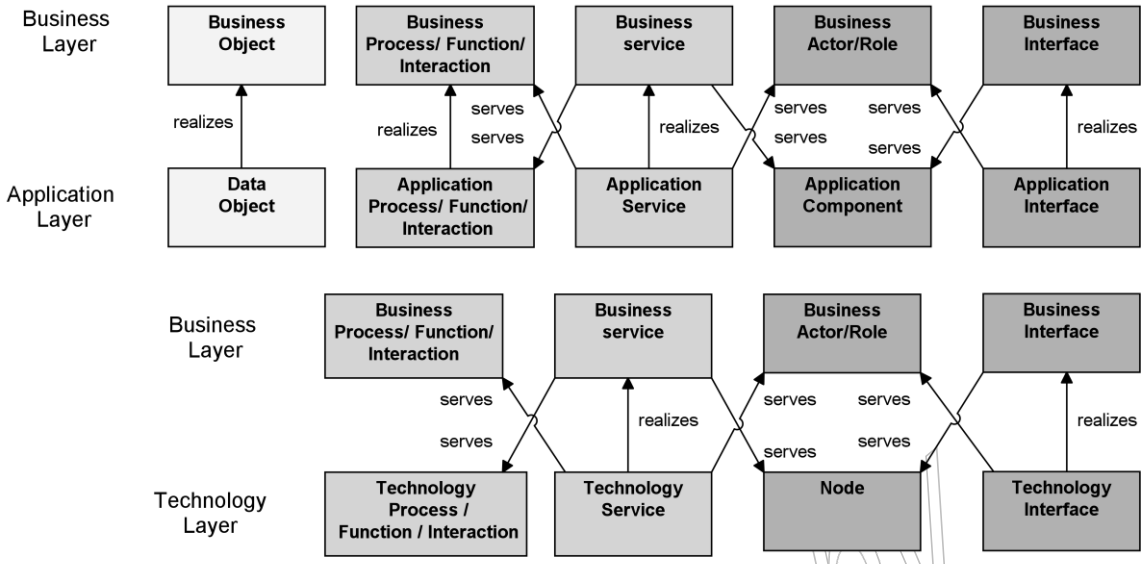


Figure 96: Relationships between Business Layer and Application and Technology Layer Elements

Note: This figure does not show all permitted relationships: there are indirect relationships that can be derived as explained in Section 5.6.

12.2 Alignment of Application and Technology Layers

Figure 97 shows the relationships between Application Layer and Technology Layer elements. There are two types of relationships between these layers:

1. *Serving* relationships, between technology service and the different types of application behavior elements, and between technology interface and application component; *vice versa*, serving relationships between application service and technology behavior, and application interface and node. These relationships represent the behavioral and structural aspects of the use of technology infrastructure by applications and *vice versa*.

2. *Realization* relationships from technology process or function to application process or function, from technology object to data object, to indicate that the data object is realized by, for example, a physical data file, from technology object to application component, to indicate that a physical data file is an executable that realizes an application or part of an application. (Note: In this case, an artifact represents a "physical" component that is deployed on a node; this is modeled with an assignment relationship. A (logical) application component is realized by an artifact and, indirectly, by the node on which the artifact is deployed.)

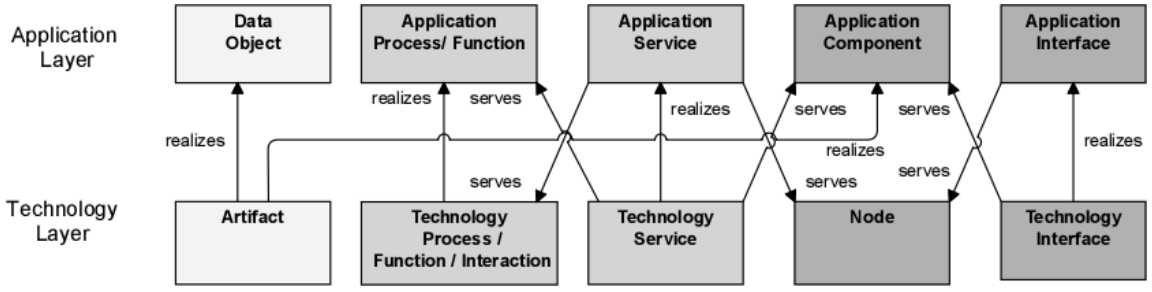


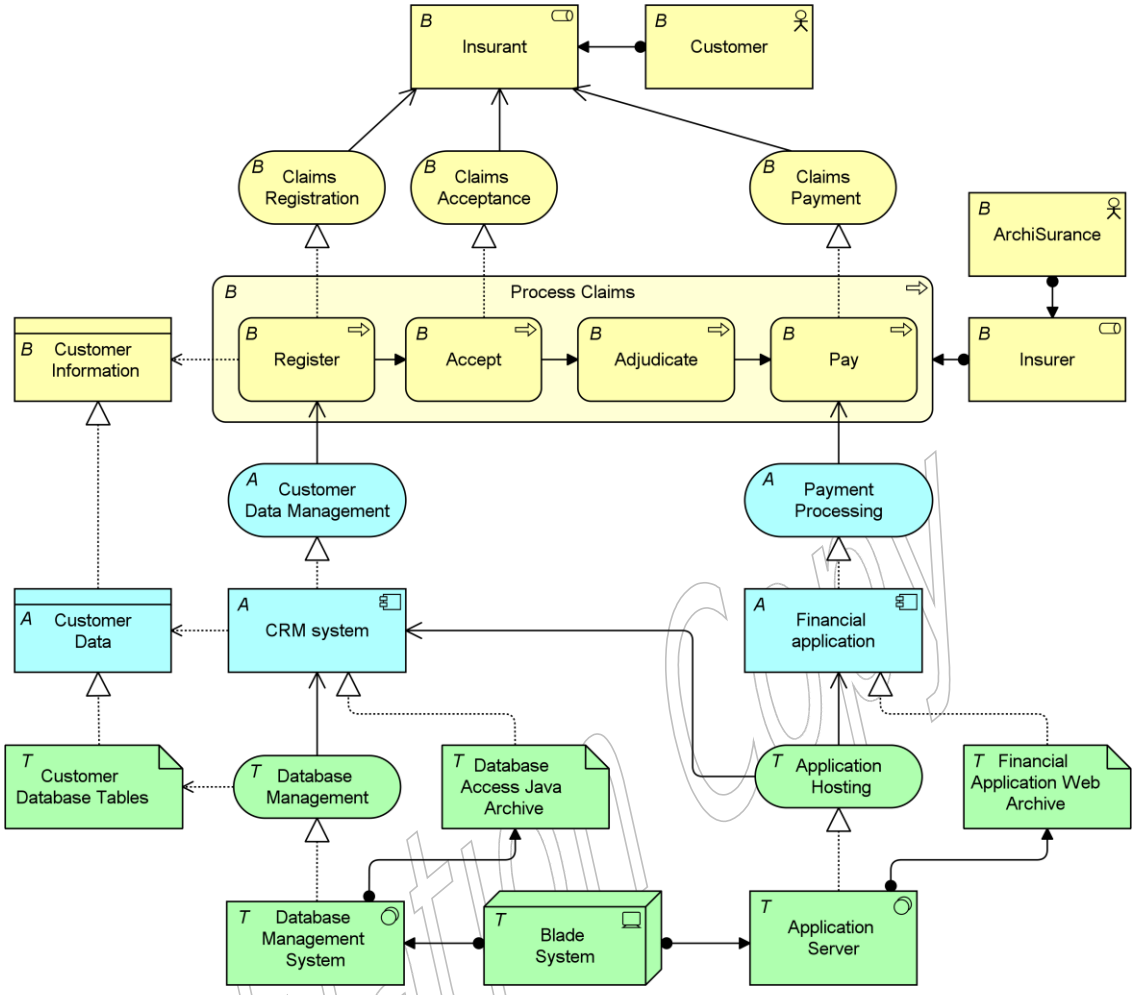
Figure 97: Relationships between Application Layer and Technology Layer Elements

Note: This figure does not show all permitted relationships: there are indirect relationships that can be derived as explained in Section 5.6.

Due to the derived relationships that are explained in Section 5.6, it is also possible to draw relationships directly between the Business and Technology Layers. For example, if a business object is realized by a data object, which in turn is realized by a technology object, this technology object indirectly realizes the business object.

12.3 Example

The example below shows how the cross-layer relationships integrate the different layers, and how you can depict this in one view. It also shows how the optional notation with letters in the upper-left corner is used to distinguish between layers.



Example 33: Cross-Layer Relationships

13 Implementation and Migration Elements

13.1 Implementation and Migration Elements Metamodel

Figure 98 gives an overview of the implementation and migration elements and their relationships.

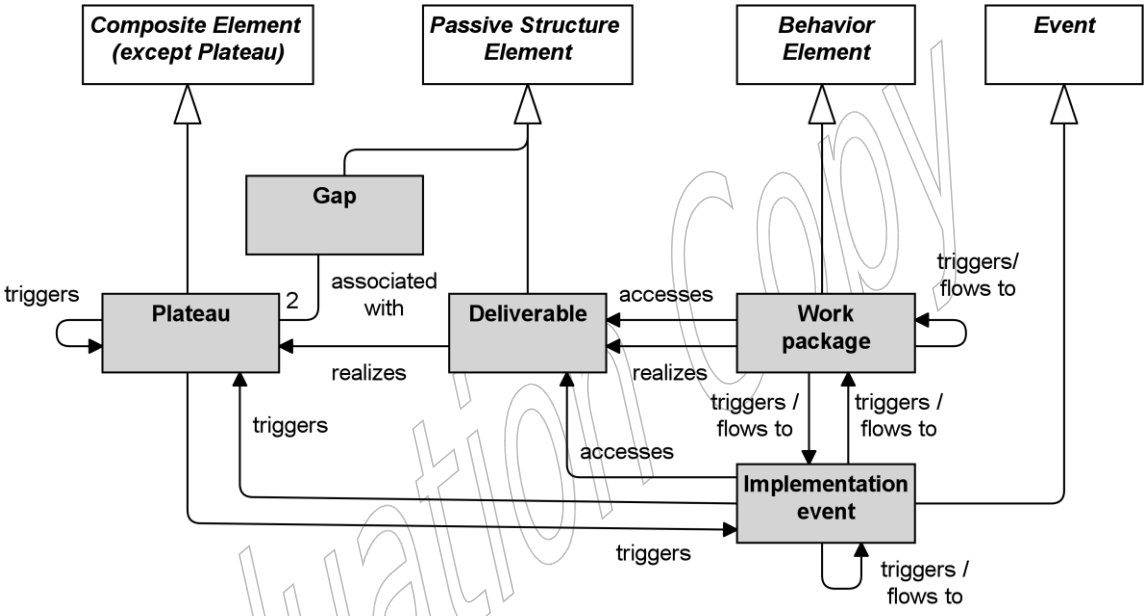


Figure 98: Implementation and Migration Metamodel

Note: This figure does not show all permitted relationships: every element in the language can have composition, aggregation, and specialization relationships with elements of the same type; furthermore, there are indirect relationships that can be derived as explained in Section 5.6.

13.2 Implementation and Migration Elements

13.2.1 Work Package

A work package represents a series of actions identified and designed to achieve specific results within specified time and resource constraints.

The central behavioral element is a *work package*. A work package is a behavior element that has a clearly defined start and end date, and realizes a well-defined set of goals or deliverables. The work package element can be used to model sub-projects or tasks within a project, complete projects, programs, or project portfolios.

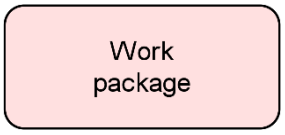


Figure 99: Work Package Notation

Conceptually, a work package is similar to a business process, in that it consists of a set of causally-related tasks, aimed at producing a well-defined result. However, a work package is a unique, "one-off" process. Still, a work package can be described in a way very similar to the description of a process.

13.2.2 Deliverable

A deliverable represents a precisely-defined outcome of a work package.

Work packages produce *deliverables*. These may be results of any kind; e.g., reports, papers, services, software, physical products, etc., or intangible results such as organizational change. A deliverable may also be the implementation of (a part of) an architecture.

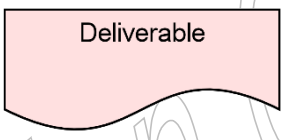


Figure 100: Deliverable Notation

Often, deliverables are contractually specified and in turn formally reviewed, agreed, and signed off by the stakeholders as is, for example, prescribed by the TOGAF framework [4].

13.2.3 Implementation Event

An implementation event is a behavior element that denotes a state change related to implementation or migration.

Work packages may be triggered or interrupted by an implementation event. Also, work packages may raise events that trigger other behavior. Unlike a work package, an event is instantaneous: it does not have duration.

An implementation event may have a time attribute that denotes the moment or moments at which the event happens. For example, this can be used to model project schedules and milestones; e.g., an event that triggers a work package, an event that denotes its completion (with a triggering relationship from the work package to the event), or an event that denotes a lifecycle change of a deliverable (via an access relationship to that deliverable).

Implementation events access deliverables to fulfill project objectives. For example, in a project to deliver a completely new application along with the technology needed to host it, an implementation event "Release to production" could access the deliverables "Final build", "staging environment", and "Production environment".

An implementation event may trigger or be triggered (raised) by a work package or a plateau. An implementation event may access a deliverable and may be composed of other implementation events.

An implementation event may be associated with any core element; e.g., to indicate a lifecycle state change. The name of an implementation event should preferably be a verb in the perfect tense; e.g., "project initiation phase completed".

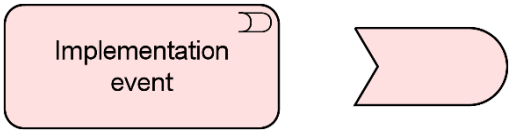


Figure 101: Implementation Event Notation

13.2.4 Plateau

A plateau represents a relatively stable state of the architecture that exists during a limited period of time.

An important premise in the TOGAF framework is that the various architectures are described for different stages in time. In each of the Phases B, C, and D of the ADM, a Baseline Architecture and Target Architecture are created, describing the current situation and the desired future situation. In Phase E (Opportunities and Solutions), so-called Transition Architectures are defined, showing the enterprise at incremental states reflecting periods of transition between the Baseline and Target Architectures. Transition Architectures are used to allow for individual work packages and projects to be grouped into managed portfolios and programs, illustrating the business value at each stage.

In order to support this, the *plateau* element is defined.

Figure 102: Plateau Notation

13.2.5 Gap

A gap represents a statement of difference between two plateaus.

The *gap* element is associated with two plateaus (e.g., Baseline and Target Architectures, or two subsequent Transition Architectures), and represents the differences between these plateaus.

In the TOGAF framework [4], a gap is an important outcome of a gap analysis in Phases B, C, and D of the ADM process, and forms an important input for the subsequent implementation and migration planning.

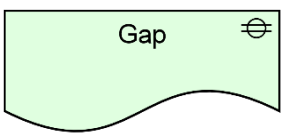
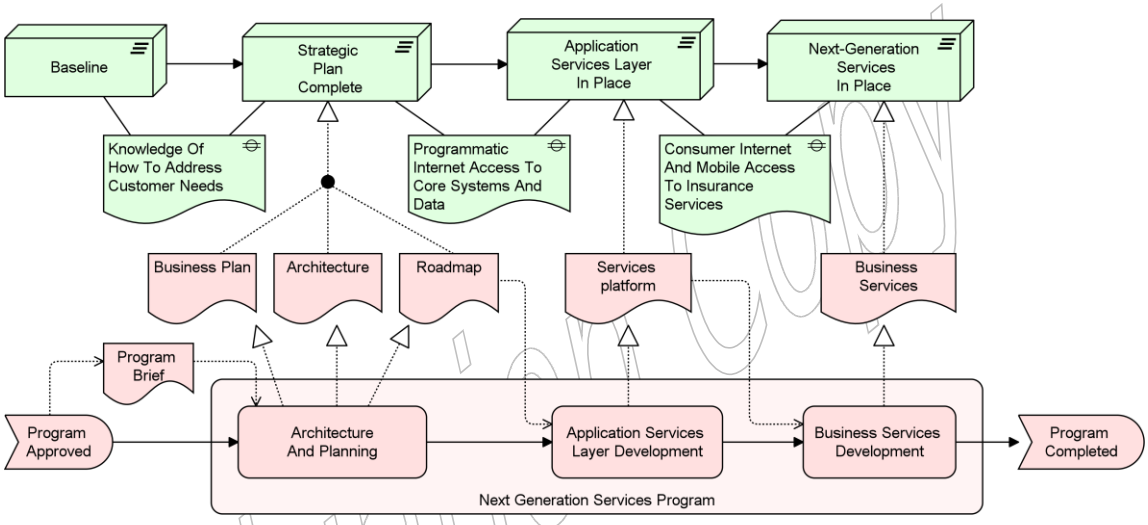


Figure 103: Gap Notation

13.2.6 Example

The Next Generation Services Program work package is composed of three other work packages. An implementation event Program Approved triggers the first work package, Architecture And Planning, which triggers the work package Application Services Layer Development, which triggers the work package Business Services Development, which triggers the implementation event Program Completed. The Program Approved implementation event also provides a deliverable Program Brief, as input for the first work package. Work package Architecture And Planning realizes three deliverables: Business Plan, Architecture, and Roadmap (which is accessed by the Application Services Layer Development work package), which collectively realize the plateau Strategic Plan Complete. This plateau follows the initial plateau Baseline, filling the gap Knowledge Of How To Address Customer Needs. Similarly, the other work packages realize other deliverables that realize the subsequent plateaus.



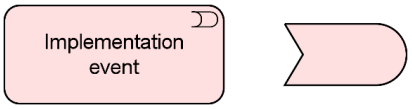
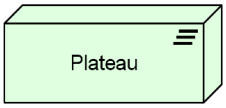
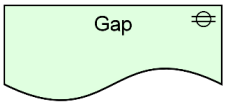
Example 34: Implementation and Migration Elements

13.2.7 Summary of Implementation and Migration Elements

Table 10 gives an overview of the implementation and migration elements, with their definitions.

Table 10: Implementation and Migration Elements

Element	Definition	Notation
Work package	A series of actions identified and designed to achieve specific results within specified time and resource constraints.	Work package
Deliverable	A precisely-defined outcome of a work package.	Deliverable

Element	Definition	Notation
Implementation event	A behavior element that denotes a state change related to implementation or migration.	
Plateau	A relatively stable state of the architecture that exists during a limited period of time.	
Gap	A statement of difference between two plateaus.	

13.3 Relationships

The implementation and migration elements use the standard ArchiMate relationships.

13.4 Cross-Aspect Dependencies

Figure 104 shows how the implementation and migration elements can be related to the ArchiMate core elements.

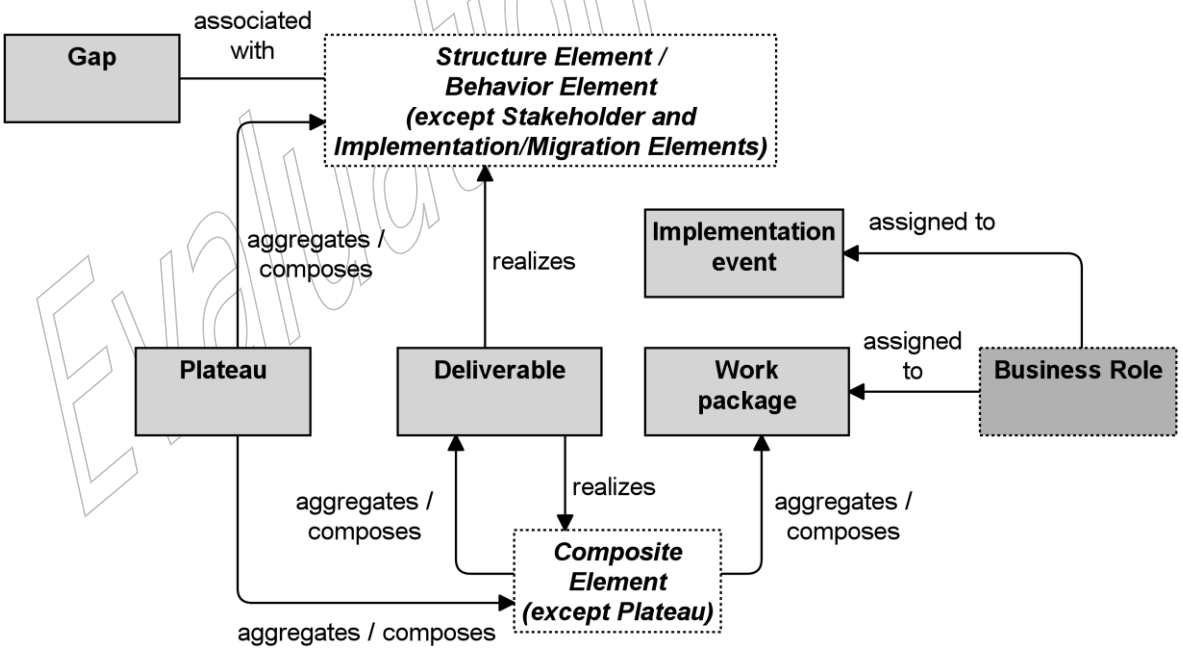


Figure 104: Relationships of Implementation and Migration Elements with Core Elements

A business role may be assigned to a work package.

A plateau is linked to an architecture that is valid for a certain time span. To indicate which parts of the architecture belong to a certain plateau, a plateau may aggregate or compose any of the concepts of the ArchiMate core language.

A gap is associated with the core concepts that are unique to one of the plateaus linked by the gap; i.e., the core concepts that make up the difference between these plateaus.

A deliverable may realize, among others, the implementation of an architecture or a part of an architecture. Therefore, any of the concepts of the ArchiMate core language may be linked to a deliverable by means of a realization relationship.

Like most of the core language concepts, a composite element may be aggregate a work package or deliverable.

Weaker relationships may also be defined. For example, the association relationship may be used to show that parts of the architecture are affected in some way by certain work packages.

Strictly speaking, the relationships between the implementation and migration elements and the motivation elements are indirect relationships; e.g., a deliverable realizes a requirement or goal through the realization of an ArchiMate core element (e.g., an application component, business process, or service). However, it is still useful to make these relationships explicit, to show directly that a deliverable is needed to realize certain requirements and goals.

Also, goals, outcomes, capabilities, and requirements can be associated with a certain plateau; e.g., certain requirements may only be applicable to the Target Architecture, while others may apply to a certain Transition Architecture. Similarly, plateaus can be used for capability-based planning. This can be modeled by means of the aggregation or composition relationships.

Figure 105 summarizes the relationships between implementation and migration elements and motivation elements. Goals, outcomes, capabilities, and requirements can be aggregated or composed in plateaus. Requirements and capabilities can be realized by deliverables. Since outcomes and goals are realized by capabilities and requirements, they can of course be realized indirectly by deliverables as well.

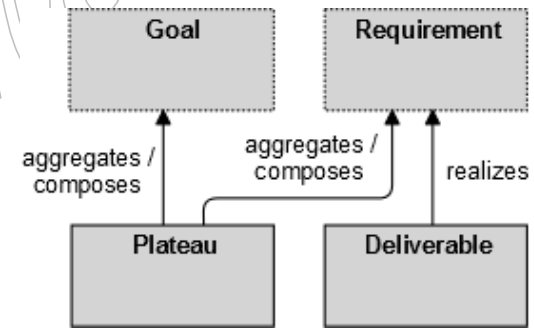


Figure 105: Relationships of Implementation and Migration Elements with Motivation Elements

14 Stakeholders, Viewpoints, and Views

14.1 Introduction

Establishing and maintaining a coherent Enterprise Architecture is clearly a complex task, because it involves many different people with differing backgrounds using various notations. In order to get a handle on this complexity, researchers have initially focused on the definition of architectural frameworks for classifying and positioning the various architectural descriptions with respect to each other (e.g., the Zachman framework [5], [8]).

Architecture frameworks provide general guidance to deliver Architecture Descriptions along with a process. The ArchiMate language, as a modeling notation, provides a detailed insight into the structure and coherence of different architectures, so its use complements and supports architecture frameworks.

The ArchiMate language provides a flexible approach in which architects and other stakeholders can use their own views on the Enterprise Architecture. In this approach, views are specified by viewpoints. Viewpoints define abstractions on the set of models representing the Enterprise Architecture, each aimed at a particular type of stakeholder and addressing a particular set of concerns. Viewpoints can be used to view certain aspects in isolation, and to relate two or more aspects.

In the domain of Enterprise Architecture, the TOGAF framework describes a taxonomy of views for different categories of stakeholders. In addition to this description of views, the TOGAF framework also provides guidelines for the development and use of viewpoints and views in Enterprise Architecture models.

The viewpoints and views proposed by any of the above-mentioned frameworks should not be considered in isolation: views are inter-related and, often, it is exactly a combination of views together with their underlying inter-dependency relationships that is the best way to describe and communicate a piece of architecture. It should, however, be noted that viewpoints and views have a limiting character. They are eventually a restriction of the whole system (and architecture) to a partial number of aspects – a view is just a partial incomplete depiction of the system.

14.2 Stakeholders and Concerns

This chapter introduces a method for using the ArchiMate language to systematically address stakeholder concerns, the viewpoint mechanism. This viewpoint mechanism conforms to the ISO/IEC 42010 standard [14], which provides a model for Architecture Description. Stakeholders, concerns, viewpoints, and views are important elements in this model, as depicted in Figure 106.

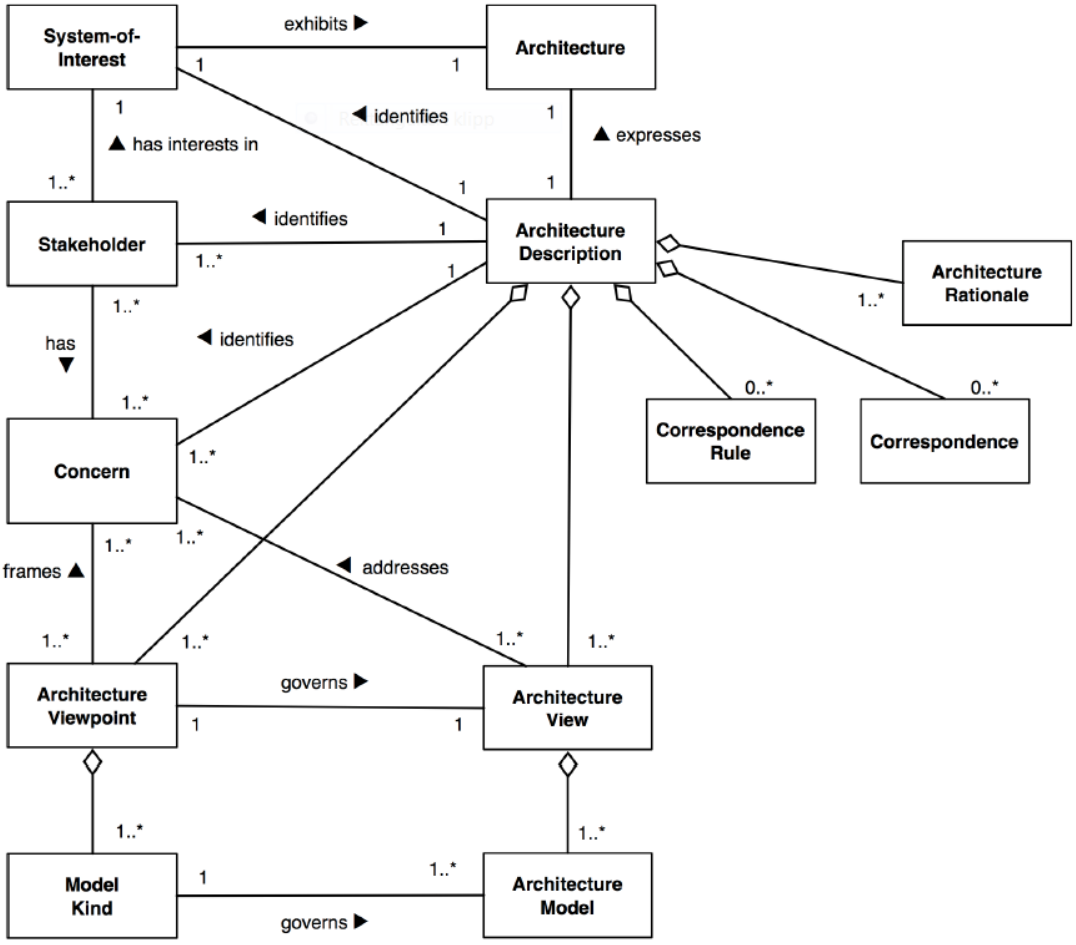


Figure 106: Conceptual Model of an Architecture Description (from [14])

The ArchiMate language with the viewpoint mechanism described in Section 14.4 assists and guides the architect in definition and classification of governing viewpoints. The architect will use this mechanism in his work to construct and design views for stakeholder communication.

14.3 Views and Viewpoints

Views are an ideal mechanism to purposefully convey information about architecture areas. In general, a view is defined as a part of an Architecture Description that addresses a set of related concerns and is tailored for specific stakeholders. A view is specified by means of a viewpoint, which prescribes the concepts, models, analysis techniques, and visualizations that are provided by the view. Simply put, a view is what you see and a viewpoint is where you are looking from.

An Architecture Description includes one or more architecture views. An architecture view (or simply view) addresses one or more of the concerns held by a stakeholder of the system.

An architecture view expresses the architecture of the system of interest in accordance with an architecture viewpoint (or simply viewpoint). There are two aspects to a viewpoint: the concerns it frames for the stakeholders and the conventions it establishes on views.

An architecture viewpoint frames one or more concerns. A concern can be framed by more than one viewpoint.

A view is governed by its viewpoint: the viewpoint establishes the conventions for constructing, interpreting, and analyzing the view to address concerns framed by that viewpoint. Viewpoint conventions can include languages, notations, model kinds, design rules and/or modeling methods, analysis techniques, and other operations on views.

Viewpoints are a means to focus on particular aspects and layers of the architecture. These aspects and layers are determined by the concerns of a stakeholder with whom communication takes place. What should and should not be visible from a specific viewpoint is therefore entirely dependent on the argumentation with respect to a stakeholder's concerns.

Viewpoints are designed for the purpose of communicating certain aspects and layers of an architecture. The communication enabled by a viewpoint can be strictly informative, but in general is bi-directional. The architect informs stakeholders, and stakeholders give their feedback (critique or consent) on the presented aspects and layers. What is and what is not shown in a view depends on the scope of the viewpoint and on what is relevant to the concerns of the stakeholder. Ideally, these are the same; i.e., the viewpoint is designed with specific concerns of a stakeholder in mind. Relevance to a stakeholder's concern, therefore, is the selection criterion that is used to determine which elements and relationships are to appear in a view.

14.4 Viewpoint Mechanism

An architect is confronted with many different types of stakeholders and concerns. To help him in selecting the right viewpoints for the task at hand, we introduce a framework for the definition and classification of viewpoints, the viewpoint mechanism. The framework is based on two dimensions: purpose and content. Figure 107 shows how the viewpoint mechanism is used to create views addressing stakeholder concerns.

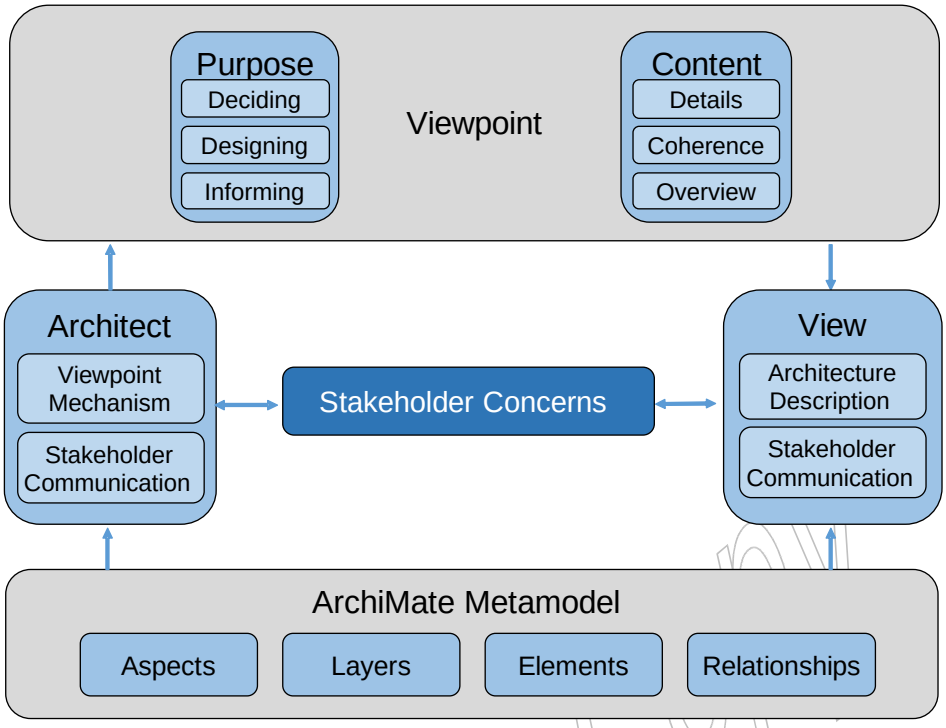


Figure 107: Framing Stakeholder Concerns using the Viewpoint Mechanism

The architect communicates with the stakeholder to understand and document their concerns. The viewpoint mechanism is used to identify purpose and content and to help define and classify the viewpoint. The viewpoint governs the construction and design of the view. The view is a description of the architecture addressing stakeholder concerns and is governed by the viewpoint.

Creating an ArchiMate viewpoint consists of two steps:

1. Selecting a subset of relevant concepts (elements and relationships) from the ArchiMate metamodel, based on the information that is needed to address the stakeholder's concerns.

2. Defining a representation to depict these concepts in a way that is understood by the stakeholders. This can be a diagram that uses standard or customized ArchiMate notation, a catalog of elements, a matrix showing the relationships between two groups of elements, or an entirely different visualization.

Applying this viewpoint to an architecture model means that those parts of the architecture are selected that match the chosen set of concepts (step 1) and are depicted in the manner prescribed by step 2.

14.4.1 Defining and Classifying Viewpoints

To help defining and classifying viewpoints based on a repeatable structure, the ArchiMate language assists the architect in selecting purpose and content relevant for the stakeholder's concerns.

The purpose dimension is supported by the following three categories:

- *Designing*: Design viewpoints support architects and designers in the design process from initial sketch to detailed design. Typically, design viewpoints consist of diagrams, like those used in, for example, UML.
- *Deciding*: Decision support viewpoints assist managers in the process of decision-making by offering insight into cross-domain architecture relationships, typically through projections and intersections of underlying models, but also by means of analytical techniques. Typical examples are cross-reference tables, landscape maps, lists, and reports.
- *Informing*: Informing viewpoints help to inform any stakeholder about the Enterprise Architecture, in order to achieve understanding, obtain commitment, and convince adversaries. Typical examples are illustrations, animations, cartoons, flyers, etc.

The content dimension uses the ArchiMate Core Framework to select relevant aspects and layers. This is supported by the following three categories:

- *Details*: Views on the detailed level typically consider one layer and one aspect from the ArchiMate Core Framework. Typical stakeholders are a software engineer responsible for design and implementation of a software component or a process owner responsible for effective and efficient process execution.
- *Coherence*: At the coherence abstraction level, multiple layers or multiple aspects are spanned. Extending the view to more than one layer or aspect enables the stakeholder to focus on architecture relationships like process-uses-system (multiple layer) or application-uses-object (multiple aspect). Typical stakeholders are operational managers responsible for a collection of IT services or business processes.
- *Overview*: The overview abstraction level addresses both multiple layers and multiple aspects. Typically, such overviews are addressed to Enterprise Architects and decision-makers, such as CEOs and CIOs.

14.4.2 Creating the View

With a governing viewpoint, the architect can create and design a view. The view contains elements and relationships (concepts) from the ArchiMate metamodel. The architect can design and create an appropriate representation for these elements and relationships, suitable for the stakeholder(s) and concern(s) being framed. The architect may use the profile mechanism described in Section 15.1 to create representations based on attributes of elements and relationships; for example, to create color-coded heat maps. The view does not have to be visual or graphical in nature.

14.5 Example Viewpoints

See Appendix C for a set of example viewpoints.

15 Language Customization Mechanisms

Every specific purpose and usage of an architecture modeling language brings about its own specific demands on the language. Yet, it should be possible to use a language for only a limited, though non-specific, modeling purpose. Therefore, the ArchiMate language, specified in the ArchiMate metamodel and described in Chapter 4 to Chapter 10, contains only the basic elements and relationships that serve general Enterprise Architecture modeling purposes. However, the language should also be able to facilitate, through customization[3] mechanisms, specialized, or domain-specific purposes, such as:

- Support for specific types of model analysis
- Support the communication of architectures
- Capture the specifics of a certain application domain (e.g., the financial sector)

The argument behind this statement is to provide a means to allow customization of the language that is tailored towards such specific domains or applications, without burdening the language with a lot of additional concepts and notations which most people would barely use. The remainder of this chapter is devoted to the customization mechanisms that are part of the ArchiMate language, and to a series of illustrative examples of such customizations.

15.1 Adding Attributes to ArchiMate Elements and Relationships

As stated earlier in this standard, the ArchiMate language contains only the elements and relationships that are necessary for general architecture modeling. However, users might want to be able to, for example, perform model-based performance or cost calculations, or to attach supplementary information (textual, numerical, etc.) to the model elements and relationships. A simple way to enrich ArchiMate elements and relationships in a generic way is to add supplementary information by means of a "profiling" specialization mechanism (see also [9]).

A *profile* is a data structure which can be defined separately from the ArchiMate language, but can be dynamically coupled with elements or relationships; i.e., the user of the language is free to decide whether and when the assignment of a profile to a model element is necessary. Profiles are specified as sets of typed attributes. Each of these attributes may have a default value that can be changed by the user.

Two types of profiles can be distinguished:

- *Pre-defined profiles*: These are profiles that have a predefined attribute structure and can be implemented beforehand in any tool supporting the ArchiMate language. Examples of such profiles are sets of attributes for ArchiMate elements and relationships that have to be specified in order to execute common types of analysis.

[3] Note that this chapter was called Language Extension Mechanisms in previous versions of this standard. Since these customization mechanisms do not actually *extend* the language, it was decided to rename this chapter and these mechanisms.

- *User-defined profiles:* Through a profile definition language, the user is able to define new profiles, thus extending the definition of ArchiMate elements or relationships with supplementary attribute sets.

Example

Table 11 below shows possible profiles with input attributes needed for certain types of cost and performance analysis of architecture models [13]. Each "serving" relationship may have a weight (indicating the average number of uses); each (business, application, or technology) "service" may have fixed and variable costs and an (average) service time; and each structure element (e.g., business role, business actor, application component, device) may have fixed and variable costs and a capacity.

Table 11: Profile Example

"Serving" Profile		**"Service" Profile**		**"Structure Element" Profile**	
Attribute	**Type**	**Attribute**	**Type**	**Attribute**	**Type**
Weight	Real	Fixed cost	Currency	Fixed cost	Currency
		Variable cost	Currency	Variable cost	Currency
		Service time	Time	Capacity	Integer

15.2 Specialization of Elements and Relationships

Specialization is a simple and powerful way to define new elements or relationships based on the existing ones. Specialized elements inherit the properties of their generalized elements (including the relationships that are allowed for the element), but some of the relationships that apply to the specialized element need not be allowed for the generalized element. Also, new graphical notation could be introduced for a specialized concept, but preferably with a resemblance to the notation of the generalized concept; e.g., by adding an icon or other graphical marker, or changing the existing icon. A specialized element or relationship strongly resembles a stereotype as it is used in UML. The stereotype notation with angled brackets may also be used to denote a specialized concept. Finally, for a specialized concept, certain attributes may be predefined, as described in the previous section.

Specialization of relationships is also allowed. Similar to specialization of elements, a specialized relationship inherits all properties of its "parent" relationship, with possible additional restrictions. For example, two specializations of the assignment relationship may be used to model responsibility *versus* accountability. Another example is a specialization of the flow relationship to model material flow in a supply chain.

Specialization of elements and relationships provides extra flexibility, as it allows organizations or individual users to customize the language to their own preferences and needs, while the underlying precise definition of the concepts is preserved. This also implies that analysis and visualization techniques developed for the ArchiMate language still apply when the specialized elements or relationships are used.

Specialization of concepts is done by using the profile mechanism described in Section 15.1. The name of the profile is the name of the specialization, and it may have other attributes if relevant to the specialization. The specialized concept is modeled by assigning such a profile to the generalized concept.

The profile may also define a specific notation to denote the specialization. The default is the guillemet notation of UML for stereotypes ("«specialization name»"). Other options include specific icons, colors, fonts, or symbols. Note that multiple specialization profiles may be assigned to the same generalized concept; in the default notation, these are shown as a comma-separated list ("«specialization 1, specialization 2»").

15.2.1 Examples of Specializations of Business Layer Elements (Informative)

The table below shows examples of specializations of Business Layer concepts.

Parent Concept	Specialized Concept	Description
Business Actor	Individual	A natural person capable of performing behavior in the context of an enterprise.
	Organizational Unit	Any named subdivision of an organization (e.g., a department).
	Organization	An entity such as an institution, corporation, or association that has a collective goal and is linked to an external environment.
	Threat Agent	Anything (e.g., an object, substance, individual, or group) that is capable of acting against an asset in a manner that can result in harm. This can be intentional; i.e., an attacker, but also unintentional; e.g., a well-intentioned, but inept, computer operator who trashes a daily batch job by typing the wrong command.
Business Service	Business Decision	A conclusion that a business arrives at through business logic and which the business is interested in managing.
Business Collaboration	Social Network	A social structure made up of social actors (individuals or organizations) and the connections between these actors.
Business Process	Business Activity	Atomic internal behavior element (at the considered abstraction level) that will not be decomposed any further.
Business Event	Threat Event (Risk & Security Overlay)	Event with the potential to adversely impact an asset. An a*ttack* is a specific type of threat event that is the result of an intentional malicious activity of an attacker, which is a specific type of threat agent.
	Loss Event (Risk & Security Overlay)	Any circumstance that causes a loss or damage to an asset.

15.2.2 Examples of Specializations of Application Layer Elements (Informative)

The table below shows examples of specializations of Application Layer elements.

Parent Concept	Specialized Concept	Description
Application Component	Logical Application Component	An encapsulation of application functionality that is independent of a particular implementation.
	Physical Application Component	An application, application module, application service, or other deployable component of functionality.
Application Interface	Application-to-Application Interface	Interface that is used to communicate between application components.
	Graphical User Interface	On-screen interface (GUI) with which a human user can interact with an application component.

15.2.3 Examples of Specializations of Technology Layer Elements (Informative)

The table below shows examples of specializations of Technology Layer elements.

Parent Concept	Specialized Concept	Description
Node	Logical Technology Component	An encapsulation of technology infrastructure that is independent of a particular product. A class of technology product.
	Physical Technology Component	A specific technology infrastructure product or technology infrastructure product instance.
Device	Mobile Device	A portable device such as a smartphone or tablet.
	Embedded Device	A computing device that is part of a piece of equipment.
Network	WiFi Network	Wireless Local Area Network (WLAN).
	Wide Area Network	Long-range data communication network.
Technology Service	Processing Service	Service used for processing data by a node.
	Storage Service	Service used for storing data on a node, typically offered by a database or file system.
	Communication Service	Service used for transporting information (e.g., voice, data) between nodes.

15.2.4 Examples of Specializations of Physical Elements (Informative)

The table below shows examples of specializations of physical elements.

Parent Concept	Specialized Concept	Description
Equipment	Vehicle	A movable piece of equipment used for transportation purposes.
	Train	A vehicle intended for use on a rail network.
Facility	Factory	A large-scale physical resource used for receipt, temporary storage, and redistribution of goods.
Material	Ore	Rock containing minerals, raw material in mining, and related industries.
	Building Material	Material used in building and construction such as concrete, bricks and mortar, beams and girders, etc.
	Fuel	Material used as an energy source in, for example, production or transportation.
Distribution Network	Rail Network	Network for rail transport, on which trains are used.
	Energy Grid	Network for distribution of energy, such as an electrical power grid or a gas distribution network.

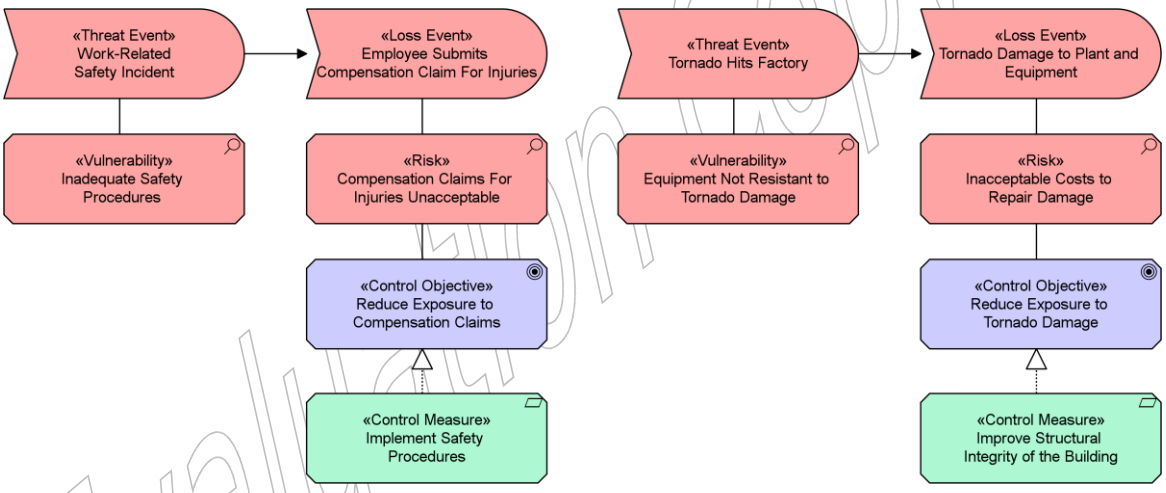
15.2.5 Examples of Specializations of Motivation Elements (Informative)

The table below shows examples of specializations of motivation elements.

Parent Concept	Specialized Concept	Description
Driver	Metric	The extent, quantity, amount, or degree of something, as determined by measurement or calculation.
Assessment	Vulnerability (Risk & Security Overlay)	The probability that an asset will be unable to resist the actions of a threat agent.
	Risk (Risk & Security Overlay)	The probable frequency and probable magnitude of future loss.
Goal	Business Objective	A time-bound milestone for an organization used to demonstrate progress towards a goal.
	Control Objective (Risk & Security Overlay)	Aim or purpose of specified control measures which address the risks that these control measures are intended to mitigate.

Parent Concept	Specialized Concept	Description
Principle	Business Policy	A directive that is not directly enforceable, whose purpose is to govern or guide the enterprise.
Requirement	Control Measure (Risk & Security Overlay)	An action, device, procedure, or technique that reduces a threat, a vulnerability, or an attack by eliminating or preventing it, by minimizing the harm it can cause, or by discovering and reporting it so that corrective action can be taken.
	Business Rule	An enforceable directive intended to govern, guide, or influence business behavior.

The model below illustrates the use of specializations of Business Layer and Motivation elements to model the results of a risk analysis, and the control objectives and required control measures to mitigate the identified risks. This example uses the UML stereotype notation with angled brackets to denote specialized elements.



Example 35: Specializations of Business Layer and Motivation Elements

15.2.6 Examples of Specializations of Strategy Elements (Informative)

The table below shows examples of specializations of strategy elements.

Parent Concept	Specialized Concept	Description
Capability	Capability Increment	A specialization of a capability realized by a specific plateau or a state in the architecture that represents a stage in the evolution of that capability.
Course of Action	Strategy	A high-level, broad-scope approach to achieve a long-term goal.
	Tactic	A narrow-scope approach to achieve a short-term goal, used to detail a strategy.

15.2.7 Examples of Specializations of Implementation and Migration Elements (Informative)

The table below shows examples of specializations of implementation and migration elements.

Parent Concept	Specialized Concept	Description
Work Package	Program	A coordinated set of projects that deliver business benefits to the organization.
	Project	A time- and resource-bound activity that delivers specific business benefits to an organization.

15.2.8 Examples of Specializations of Composite Elements (Informative)

The table below shows examples of specializations of compound elements. In addition to the specialization of single model elements, grouping can also be used to define specific *compound* elements.

Parent Concept	Specialized Concept	Description
Grouping	Risk Domain (Risk & Security Overlay)	A domain consisting of entities that share one or more characteristics relevant to risk management or security. A risk domain is also a context or set of conditions that affects a risk exposure level.
Grouping of Application Component, Application Function, and Data Object	Data Store	A repository for persistently storing and managing collections of data.

15.2.9 Examples of Specializations of Relationships (Informative)

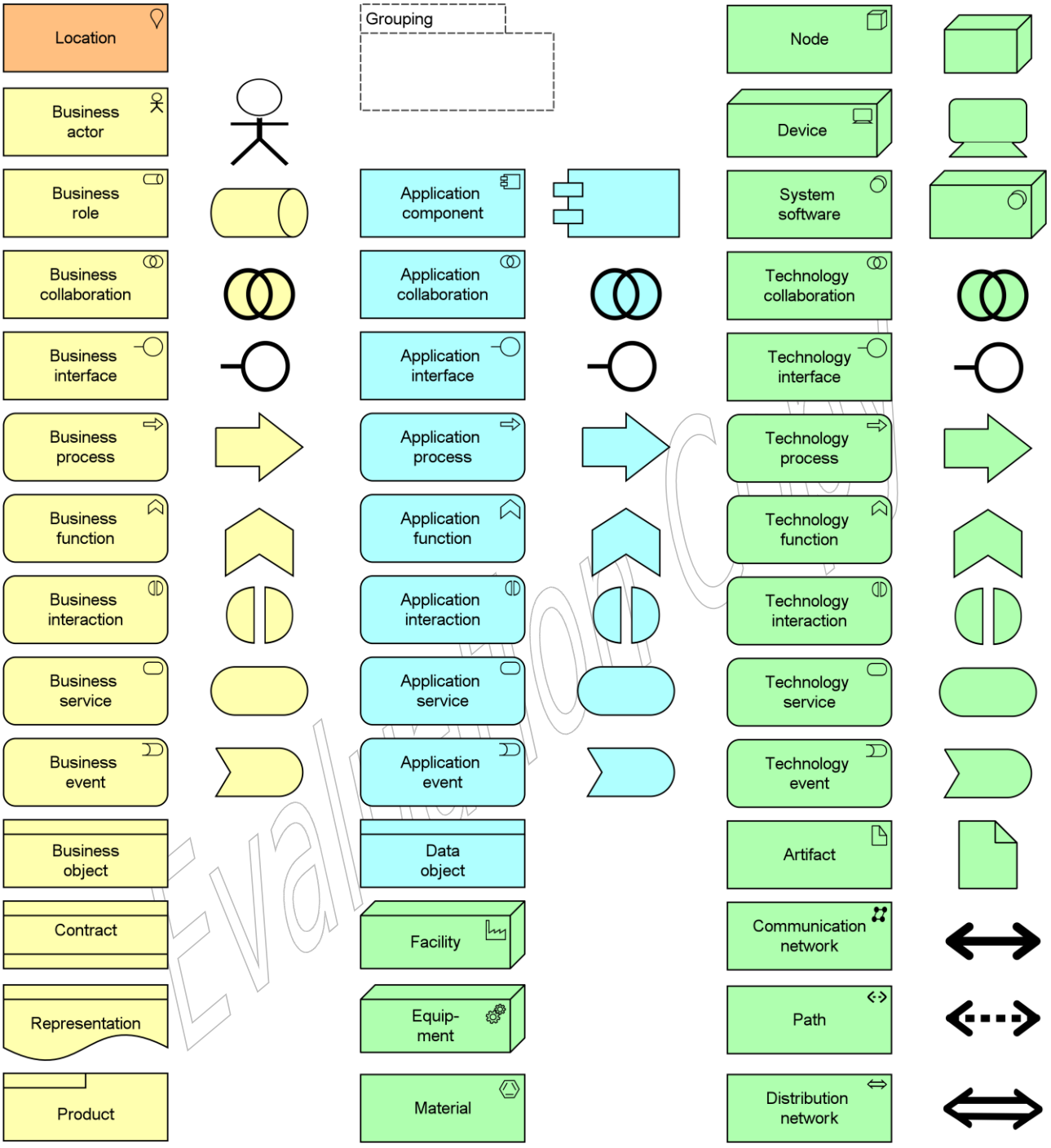
The table below shows examples of specializations of relationships.

Parent Concept	Specialized Concept	Description
Flow	Money Flow	A flow of money between behavior elements.
Assignment	Responsibilities assignment	Assignment from a business actor to a business role.
	Behavior assignment	Assignment from an active structure to a behavior element.
Or-junction	Or-join	A junction with two or more incoming triggering and one outgoing triggering relationship, representing that at least one of the incoming relationships must be triggered to trigger the outgoing one.

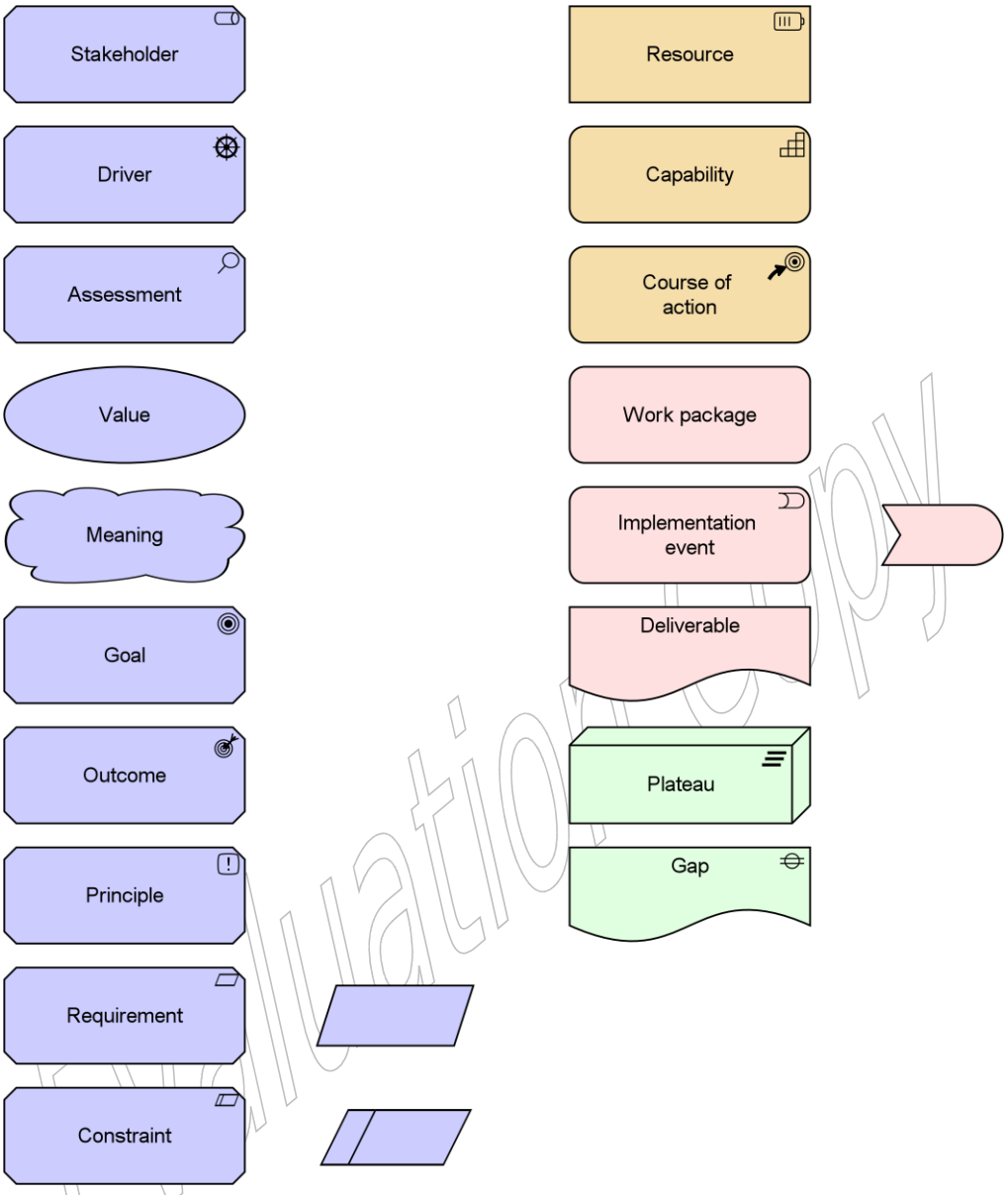
A Summary of Language Notation

This appendix describes the default iconography of the ArchiMate language. Modelers can choose to use a different iconography on any diagram they desire, if it will help communicate better with the stakeholder for which the viewpoint was designed. It is, however, recommended to use the default iconography so that teams using the ArchiMate language have a collective understanding of the view being developed. Conforming tools shall at least support these notations.

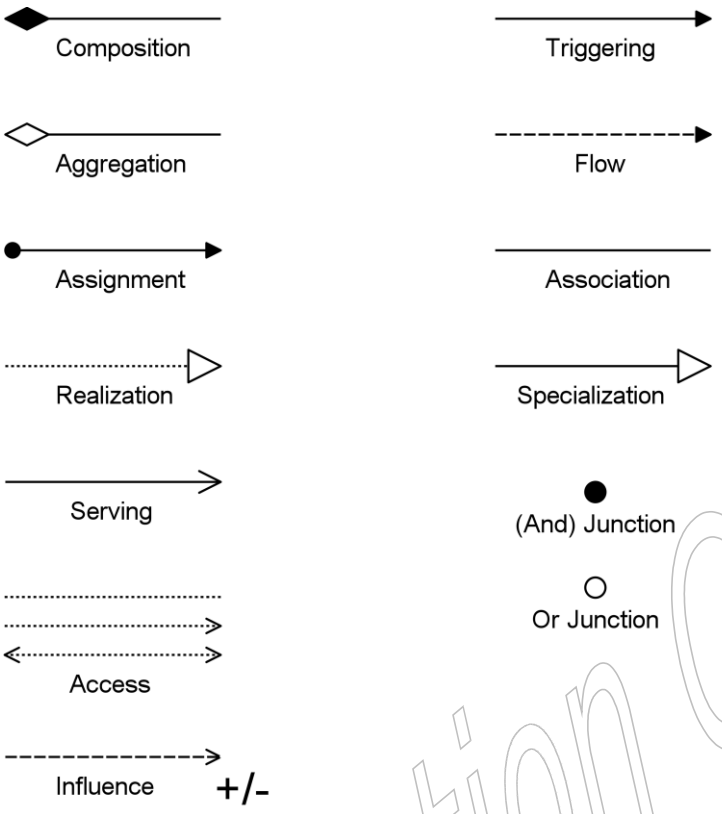
A.1 Core Elements



A.2 Motivation, Strategy, Implementation and Migration Elements



A.3 Relationships



B Relationship Tables

This appendix details the normative requirements for relationships between elements of the ArchiMate modeling language.

The letters in the tables have the following meaning:

(a)ccess	(c)omposition	(f)low	a(g)gregation	ass(i)gnment	
i(n)fluence	ass(o)ciation	(r)ealization	(s)pecialization	(t)riggering	ser(v)ing

↓ From / → To	Assessment	Constraint	Driver	Goal	Meaning	Outcome	Principle	Requirement	Stakeholder	Value	Capability	Course of Action	Resource	Business Actor	Business Collaboration	Contract	Business Event	Business Function	Business Interaction	Business Interface	Business Object	Business Process	Product	Representation	Business Role	Business Service
Assessment	cgnos	no	no	no	no	no	no	no	o	no	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Constraint	no	cgnos	no	no	no	no	no	no	o	no	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Driver	no	no	cgnos	no	no	no	no	no	o	no	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Goal	no	no	no	cgnos	no	no	no	nor	o	no	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Meaning	no	no	no	no	cgnos	no	no	no	o	no	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Outcome	no	no	no	nor	no	cgnos	no	nor	o	no	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Principle	no	no	no	nor	no	nor	cgnos	no	o	no	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Requirement	no	cgnos	no	nor	no	nor	nor	cgnos	o	no	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Stakeholder	o	o	o	o	o	o	o	o	cgnos	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Value	no	no	no	no	no	no	no	no	o	cgnos	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Capability	no	nor	no	nor	no	nor	nor	nor	o	no	cfgostv	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Course of Action	no	nor	no	nor	no	nor	nor	nor	o	no	fot	cfgostv	fortv	fot	o	o	o	o	o	o	o	o	o	o	o	o
Resource	no	nor	no	nor	no	nor	nor	nor	o	no	fiotv	fortv	cfgost	o	o	o	o	o	o	o	o	o	o	o	o	o
Business Actor	no	nor	no	nor	no	nor	nor	nor	ino	no	or	or	or	cfgostv	fotv	ao	fiotv	fiortv	fiortv	cfgiotv	ao	fiortv	fot	ao	fiotv	fiortv
Business Collaboration	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	or	fgotv	cfgostv	ao	fiotv	fiortv	fiortv	cfgiotv	ao	fiortv	fot	ao	fgiotv	fiortv
Contract	no	nor	no	nor	no	nor	nor	nor	o	no	or	or	or	o	o	cgos	o	o	o	o	cgos	o	o	o	o	o
Business Event	no	nor	no	nor	no	nor	nor	nor	o	no	o	o	o	fot	fot	ao	cfgost	fot	fot	fot	ao	fot	fot	ao	fot	fot
Business Function	no	nor	no	nor	no	nor	nor	nor	o	no	o	o	o	fotv	fotv	ao	fotv	cfgortv	fotv	fotv	ao	cfgorstv	fot	ao	fotv	fortv
Business Interaction	no	nor	no	nor	no	nor	nor	nor	o	no	o	o	o	fotv	fotv	ao	fotv	cfgortv	cfgortv	fotv	ao	cfgortv	fot	ao	fotv	fortv
Business Interface	no	nor	no	nor	no	nor	nor	nor	o	no	o	o	o	fotv	fotv	ao	fotv	fotv	cfgostv	cfgiotv	ao	fotv	fot	ao	fotv	fiotv
Business Object	no	nor	no	nor	no	nor	nor	nor	o	no	or	or	or	o	o	cgos	o	o	o	o	cgos	o	o	o	o	o
Business Process	no	nor	no	nor	no	nor	nor	nor	o	no	or	or	o	fotv	fotv	ao	fotv	cfgortv	cfgortv	fotv	ao	cfgorstv	fot	ao	fotv	fortv
Product	no	nor	no	nor	no	nor	nor	nor	o	no	or	or	or	fotv	fotv	acgor	fotv	fotv	fotv	fotv	acgor	fotv	cfgost	acgo	fotv	cfgortv
Representation	no	nor	no	nor	no	nor	nor	nor	o	no	or	or	or	o	o	or	o	o	o	o	or	o	o	cgos	o	o
Business Role	no	nor	no	nor	no	nor	nor	nor	o	no	or	or	or	o	o	ao	fiotv	fiortv	fiortv	cfgotv	ao	fiortv	fot	ao	cfgostv	fiortv
Business Service	no	nor	no	nor	no	nor	nor	nor	o	no	or	or	o	fotv	fotv	ao	fotv	fotv	fotv	fotv	ao	fotv	fot	ao	fotv	cfgostv

↓ From / → To	Application Collaboration	Application Component	Data Object	Application Event	Application Function	Application Interaction	Application Interface	Application Process	Application Service	Artifact	Technology Collaboration	Communication Network	Device	Technology Event	Technology Function	Technology Interaction	Technology Interface	Node	Path	Technology Process	Technology Service	System Software	Distribution Network	Equipment	Facility	Material	Deliverable	Implementation Event	Workpackage	Gap	Plateau	Location
Assessment	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Constraint	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Driver	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Goal	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Meaning	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Outcome	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Principle	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Requirement	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Stakeholder	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Value	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Capability	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Course of Action	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Resource	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Business Actor	fotv	fotv	ao	fotv	fortv	fortv	fotv	fortv	fortv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	aov	o	o	o	o	o	fot
Business Collaboration	fotv	fotv	ao	fotv	fortv	fortv	fotv	fortv	fortv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	aov	o	o	o	o	o	fot
Contract	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Business Event	fot	fot	o	o	fot	fot	fot	fot	fot	fot	fot	fot	fot	o	fot	fot	fot	fot	fot	fot	fot	fot	fot	fot	fot	o	o	o	o	o	o	fot
Business Function	fotv	fotv	ao	fotv	fortv	fortv	fotv	fortv	fortv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	aov	o	o	o	o	o	fot
Business Interaction	fotv	fotv	ao	fotv	fortv	fortv	fotv	fortv	fortv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	aov	o	o	o	o	o	fot
Business Interface	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	aov	o	o	o	o	o	fot
Business Object	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Business Process	fotv	fotv	ao	fotv	fortv	fortv	fotv	fortv	fortv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	aov	o	o	o	o	o	fot
Product	fotv	fotv	acgor	fotv	fotv	fotv	fotv	fotv	cfgortv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	cfgotv	fotv	fot	fotv	acgov	o	o	o	o	o	fot
Representation	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Business Role	fotv	fotv	ao	fotv	fortv	fortv	fotv	fortv	fortv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	aov	o	o	o	o	o	fot
Business Service	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	aov	o	o	o	o	o	fot

↓ From / → To	Assessment	Constraint	Driver	Goal	Meaning	Outcome	Principle	Requirement	Stakeholder	Value	Capability	Course of Action	Resource	Business Actor	Business Collaboration	Contract	Business Event	Business Function	Business Interaction	Business Interface	Business Object	Business Process	Product	Representation	Business Role	Business Service
Application Collaboration	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	or	fotv	fotv	ao	fotv	fortv	fortv	fortv	ao	fortv	fot	ao	fotv	fortv
Application Component	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	or	fotv	fotv	ao	fotv	fortv	fortv	fortv	ao	fortv	fot	ao	fotv	fortv
Data Object	no	nor	no	nor	no	nor	nor	nor	o	no	or	or	or	o	fot	ao	o	o	o	o	or	o	o	o	o	o
Application Event	no	nor	no	nor	no	nor	nor	nor	o	no	o	o	o	fot	fot	ao	o	fot	fot	fot	ao	fot	fot	o	fot	fot
Application Function	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	o	fotv	fot	ao	fotv	fortv	fortv	fotv	ao	fortv	fot	ao	fotv	fortv
Application Interaction	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	o	fotv	fotv	ao	fotv	fortv	fortv	fotv	ao	fortv	fot	ao	fotv	fortv
Application Interface	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	or	fotv	fotv	ao	fotv	fortv	fotv	fotv	ao	fotv	fot	ao	fotv	fortv
Application Process	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	o	fotv	fotv	ao	fotv	fortv	fortv	fotv	ao	fortv	fot	ao	fotv	fortv
Application Service	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	o	fotv	fotv	ao	fotv	fortv	fortv	fotv	ao	fotv	fot	ao	fotv	fortv
Artifact	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	or	fortv	fotv	ao	fortv	fortv	fortv	fortv	ao	fortv	fot	ao	fortv	fortv
Technology Collaboration	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	or	fiortv	fotv	ao	fiortv	fiortv	fiortv	fiortv	ao	fiortv	fot	ao	fiortv	fiortv
Communication Network	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	or	fiortv	fotv	ao	fiortv	fiortv	fiortv	fiortv	ao	fiortv	fot	ao	fiortv	fiortv
Device	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	or	fiortv	fotv	ao	fiortv	fiortv	fiortv	fiortv	ao	fiortv	fot	ao	fiortv	fiortv
Technology Event	no	nor	no	nor	no	nor	nor	nor	o	no	o	o	o	fot	fot	ao	o	fot	fot	fot	ao	fot	fot	o	fot	fot
Technology Function	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	o	fotv	fotv	ao	fotv	fortv	fortv	fotv	ao	fotv	fot	ao	fotv	fortv
Technology Interaction	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	o	fotv	fotv	ao	fotv	fortv	fortv	fotv	ao	fotv	fot	ao	fotv	fortv
Technology Interface	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	or	fotv	fotv	ao	fiortv	fiortv	fiortv	fortv	ao	fotv	fot	ao	fotv	fortv
Node	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	or	fiortv	fotv	ao	fiortv	fiortv	fiortv	fiortv	ao	fiortv	fot	ao	fiortv	fiortv
Path	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	or	fiortv	fotv	ao	fiortv	fiortv	fiortv	fiortv	ao	fiortv	fot	ao	fiortv	fiortv
Technology Process	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	o	fotv	fotv	ao	fotv	fotv	fotv	fotv	ao	fotv	fot	ao	fotv	fortv
Technology Service	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	o	fotv	fotv	ao	fotv	fotv	fotv	fotv	ao	fotv	fot	ao	fotv	fortv
System Software	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	o	fiortv	fotv	ao	fiortv	fiortv	fiortv	fiortv	ao	fiortv	fot	ao	fiortv	fiortv
Distribution Network	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	or	fiortv	fotv	ao	fiortv	fiortv	fiortv	fiortv	ao	fiortv	fot	ao	fiortv	fiortv
Equipment	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	or	fiortv	fotv	ao	fiortv	fiortv	fiortv	fiortv	ao	fiortv	fot	ao	fiortv	fiortv
Facility	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	or	fiortv	fotv	ao	fiortv	fiortv	fiortv	fiortv	ao	fiortv	fot	ao	fiortv	fiortv
Material	no	nor	no	nor	no	nor	nor	nor	no	no	or	or	or	o	fotv	ao	o	o	o	o	o	o	o	o	o	o
Deliverable	or	or	or	or	or	or	or	or	or	or	or	or	or	fiortv	fotv	ao	fiortv	fiortv	fiortv	fiortv	ao	fiortv	fot	ao	fiortv	fiortv
Implementation Event	o	o	o	o	o	o	o	o	o	o	o	o	o	fiortv	fotv	ao	fiortv	fiortv	fiortv	fiortv	ao	fiortv	fot	ao	fiortv	fiortv
Workpackage	or	or	or	or	or	or	or	or	or	or	or	or	or	fiortv	fotv	ao	fiortv	fiortv	fiortv	fiortv	ao	fiortv	fot	ao	fiortv	fiortv
Gap	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Plateau	o	o	o	cgor	o	or	or	cgor	o	o	o	o	cgor	cfgiortv	cfgotv	acgor	cfgiortv	cfgiortv	cfgiortv	cfgiortv	acgor	cfgiortv	cgor	acgo	cfgiortv	cfgiortv
Location	cgno	cgnor	cgno	cgnor	cgno	cgnor	cgnor	cgnor	nor	cgno	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor

↓ From / → To	Application Collaboration	Application Component	Data Object	Application Event	Application Function	Application Interaction	Application Interface	Application Process	Application Service	Artifact	Technology Collaboration	Communication Network	Device	Technology Event	Technology Function	Technology Interaction	Technology Interface	Node	Path	Technology Process	Technology Service	System Software	Distribution Network	Equipment	Facility	Material	Deliverable	Implementation Event	Workpackage	Gap	Plateau	Location
Application Collaboration	cfgorstv	cfgorstv	ao	fiortv	fiortv	fiortv	cfgortv	fiortv	fiortv	fotv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aiorv	o	o	o	o	o	fot
Application Component	cfgorstv	cfgorstv	ao	fiortv	fiortv	fiortv	cfgortv	fiortv	fiortv	fotv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aiorv	o	o	o	o	o	fot
Data Object	o	o	cgos	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Application Event	fot	fot	ao	cfgost	fot	fot	fot	fot	fot	fot	fot	fot	fot	o	fot	fot	fot	fot	fot	fot	fot	fot	fot	fot	fot	aov	o	o	o	o	o	fot
Application Function	fotv	fotv	ao	fotv	cfgorstv	cfgorstv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aov	o	o	o	o	o	fot
Application Interaction	fotv	fotv	ao	fotv	fotv	cfgorstv	fotv	cfgortv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aov	o	o	o	o	o	fot
Application Interface	fotv	fotv	ao	fotv	fotv	fotv	cfgostv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aov	o	o	o	o	o	fot
Application Process	fotv	fotv	ao	fotv	fotv	fotv	fotv	cfgorstv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aov	o	o	o	o	o	fot
Application Service	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	cfgostv	fotv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aov	o	o	o	o	o	fot
Artifact	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	cfgorstv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aov	o	o	o	o	o	fot
Technology Collaboration	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	fotv	cfgijorstv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aiorv	o	o	o	o	o	fot
Communication Network	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	cfgost	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aiorv	o	o	o	o	o	fot
Device	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fot	cfgijorstv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aiorv	o	o	o	o	o	fot
Technology Event	fot	fot	ao	o	fot	fot	fot	fot	fot	fot	fot	fot	fot	cfgost	fot	fot	fot	fot	fot	fot	fot	fot	fot	fot	fot	ao	o	o	o	o	o	fot
Technology Function	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	cfgostv	cfgotv	cfgotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aov	o	o	o	o	o	fot
Technology Interaction	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	cfgotv	cfgostv	cfgotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aov	o	o	o	o	o	fot
Technology Interface	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fotv	cfgostv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aov	o	o	o	o	o	fot
Node	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aiorv	o	o	o	o	o	fot
Path	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aiorv	o	o	o	o	o	fot
Technology Process	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aiorv	o	o	o	o	o	fot
Technology Service	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aiorv	o	o	o	o	o	fot
System Software	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aiorv	o	o	o	o	o	fot
Distribution Network	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aiorv	o	o	o	o	o	fot
Equipment	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aiorv	o	o	o	o	o	fot
Facility	fotv	fotv	ao	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fotv	fotv	fotv	fot	fotv	fotv	fotv	fot	fotv	fotv	aiorv	o	o	o	o	o	fot
Material	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Deliverable	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	cgos	o	o	o	o	o	or
Implementation Event	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	ao	o	o	o	o	o
Workpackage	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	or	aor	fot	o	o	o	or
Gap	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	cfgost	o	o	o
Plateau	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	cgor	o	ot	o	cgos	o	cgor
Location	cfgortv	cfgortv	acgor	cfgiortv	cfgiortv	cfgiortv	cfgortv	cfgiortv	cfgiortv	cfgiortv	cfgiortv	cfgiortv	cfgiortv	cfgiortv	cfgiortv	cfgiortv	cfgiortv	cfgiortv	cfgort	cfgiortv	cfgiortv	cfgiortv	cfgot	cfgiortv	cfgiortv	acgiorv	o	o	o	o	cgost	cfgost

B.1 Grouping, Plateau, and Relationships Between Relationships

For grouping, plateau, and relationships, the following conditions hold:

- Grouping and plateau elements may have an aggregation or composition relationship to any concept (element or relationships).
- A grouping element may have any relationship with any element (provided that the element is a possible target element for the relationship).
- Any element may have any relationship with a grouping element (provided that the element is a possible source element for the relationship).
- A grouping element may have any relationship with another grouping element.
- Any relationship may have an association relationship with any element.

From ↓ / To →	**Grouping**	**Plateau**	**Element**	**Relationship**
Grouping	*any*	cfgorst	*any**	cgo
Plateau	cfgiorst			cgo
Element	*any***			o
Relationship	o	o	o	

* Provided that element is a possible target element of the relationship.

** Provided that element is a possible source element of the relationship.

(c)omposition ass(i)gnment ass(o)ciation (t)riggering
(f)low a(g)gregation (r)ealization (s)pecialization

C Example Viewpoints (Informative)

C.1 Basic Viewpoints in ArchiMate

A viewpoint in the ArchiMate language is a selection of a relevant subset of the ArchiMate elements and their relationships. This is the representation of that part of an architecture that is expressed in different diagrams.

The most basic type of viewpoint is a simple selection of a relevant subset of the ArchiMate concepts and the representation of that part of an architecture that is expressed in this selection, geared towards the stakeholders that will use the resulting views.

The following are examples of stakeholders and concerns as a basis for the specification of viewpoints:

- *End user*: For example, what are the consequences for his work and workplace?
- *Architect*: What is the consequence for the maintainability of a system, with respect to corrective, preventive, and adaptive maintenance?
- *Upper-level management*: How can we ensure our policies are followed in the development and operation of processes and systems? What is the impact of decisions (on personnel, finance, ICT, etc.)?
- *Operational manager*: Responsible for exploitation or maintenance: For example, what new technologies are there to prepare for? Is there a need to adapt maintenance processes? What is the impact of changes to existing applications? How secure are my systems?
- *Project manager*: Responsible for the development of new applications: What are the relevant domains and their relationships? What is the dependence of business processes on the applications to be built? What is their expected performance?
- *Developer*: What are the modifications with respect to the current situation that need to be done?

In each basic viewpoint, concepts from the three layers of Business, Application, and Technology may be used. However, not every combination of these would give meaningful results. In some cases, for example, separate viewpoints for the different layers are advisable. Based on common architectural practice and on experiences with the use of ArchiMate models in practical cases, useful combinations in the form of a set of basic viewpoints have been defined. These are listed in Table 12. The table also shows the perspective for the viewpoint. Some viewpoints have a scope that is limited to a single layer or aspect, when others link multiple layers and/or aspects. The different viewpoints are grouped into categories that indicate which direction and which elements the viewpoint is looking at:

1. *Composition*: Viewpoints that defines internal compositions and aggregations of elements.
2. *Support*: Viewpoints where you are looking at elements that are supported by other elements. Typically from one layer and upwards to an above layer.

3. *Cooperation*: Towards peer elements which cooperate with each other. Typically across aspects.

4. *Realization*: Viewpoints where you are looking at elements that realize other elements. Typically from one layer and downwards to a below layer.

Table 12: Basic Viewpoints

Category: Composition		
Name	**Perspective**	**Scope**
Organization	Structure of the enterprise in terms of roles, departments, etc.	Single layer/ Single aspect
Application Platform	Shows structure of a typical application platform and how it relates to supporting technology.	Multiple layer/ Multiple aspect
Information Structure	Shows the structure of the information used in the enterprise.	Multiple layer/ Single aspect
Technology	Infrastructure and platforms underlying the enterprise's information systems in terms of networks, devices, and system software.	Single layer/ Multiple aspect
Layered	Provides overview of architecture(s).	Multiple layer/ Multiple aspect
Physical	Physical environment and how this relates to IT infrastructure.	Multiple layer/ Multiple aspect
Category: Support		
Name	**Perspective**	**Scope**
Product	Shows the contents of products.	Multiple layer/ Multiple aspect
Application Usage	Relates applications to their use in, for example, business processes.	Multiple layer/ Multiple aspect
Technology Usage	Shows how technology is used by applications.	Multiple layer/ Multiple aspect
Category: Cooperation		
Name	**Perspective**	**Scope**
Business Process Cooperation	Shows the relationships between various business processes.	Multiple layer/ Multiple aspect

Application Cooperation	Shows application components and their mutual relationships.	Multiple layer/ Multiple aspect
Category: Realization		
Name	**Perspective**	**Scope**
Service Realization	Shows how services are realized by the requisite behavior.	Multiple layer/ Multiple aspect
Implementation and Deployment	Shows how applications are mapped onto the underlying technology.	Multiple layer/ Multiple aspect

In the following sections, the ArchiMate viewpoints are described in more detail. For each viewpoint the comprised elements are listed, guidelines for the viewpoint's use, and the stakeholders and concerns addressed by the viewpoint are indicated. In addition to the specified elements, the Grouping element, Junction, and Or Junction can be used in every viewpoint. For more details on the goal and use of viewpoints, refer to Chapter 14 of [1]. The diagrams illustrating the permitted element and relationships for each viewpoint do not show all permitted relationships: every element in a given viewpoint can have composition, aggregation, and specialization relationships with elements of the same type; furthermore, there are indirect relationships that can be derived as explained in Section 5.6.

These basic viewpoints are starting points for modeling efforts. They can accelerate architectural efforts, support organizational standards, facilitate peer review, and aid new modelers. However, these basic viewpoints should not constrain modeling activities. Organizations and individual modelers should address stakeholder concerns by selecting from the basic viewpoints, modifying them, or defining new ones. The viewpoints listed here are therefore intended as examples, not as a normative or exhaustive list.

Moreover, these examples use the standard ArchiMate language notation. As outlined before, a viewpoint's representation should be geared towards the intended stakeholder(s). This means that these basic viewpoints are mainly useful for architects and their peers. Other stakeholders may require a different representation, even if they are interested in the same content.

C.1.1 Organization Viewpoint

The organization viewpoint focuses on the (internal) organization of a company, department, network of companies, or of another organizational entity. It is possible to present models in this viewpoint as nested block diagrams, but also in a more traditional way, such as organizational charts. The Organization viewpoint is very useful in identifying competencies, authority, and responsibilities in an organization.

Table 13: Organization Viewpoint Description

Organization Viewpoint	
Stakeholders	Enterprise, process and domain architects, managers, employees, shareholders
Concerns	Identification of competencies, authority, and responsibilities
Purpose	Designing, deciding, informing
Scope	Single layer/Single aspect

Elements

- Business actor
- Business role
- Business collaboration
- Location
- Business interface

C.1.2 Business Process Cooperation Viewpoint

The business process cooperation viewpoint is used to show the relationships of one or more business processes with each other and/or with their environment. It can both be used to create a high-level design of business processes within their context and to provide an operational manager responsible for one or more such processes with insight into their dependencies. Important aspects of business process cooperation are:

- Causal relationships between the main business processes of the enterprise
- Mapping of business processes onto business functions
- Realization of services by business processes
- Use of shared data

Each of these can be regarded as a "sub-viewpoint" of the business process cooperation viewpoint.

Table 14: Business Process Cooperation Viewpoint Description

Business Process Cooperation Viewpoint	
Stakeholders	Process and domain architects, operational managers
Concerns	Dependencies between business processes, consistency and completeness, responsibilities
Purpose	Designing, deciding
Scope	Multiple layer/Multiple aspect

Elements

- Business actor
- Business role
- Business collaboration
- Location
- Business interface
- Business process/function/interaction
- Business event
- Business service
- Business object
- Representation
- Application component/collaboration
- Application interface
- Application process/function/interaction
- Application event
- Application service
- Data object

C.1.3 Product Viewpoint

The product viewpoint depicts the value that these products offer to the customers or other external parties involved and shows the composition of one or more products in terms of the constituting (business, application, or technology) services, and the associated contract(s) or other agreements. It may also be used to show the interfaces (channels) through which this product is offered, and the events associated with the product. A product viewpoint is typically used in product development to design a product by composing existing services or by

identifying which new services have to be created for this product, given the value a customer expects from it. It may then serve as input for business process architects and others that need to design the processes and ICT realizing these products.

Table 15: Product Viewpoint Description

Product Viewpoint	
Stakeholders	Product developers, product managers, process and domain architects
Concerns	Product development, value offered by the products of the enterprise
Purpose	Designing, deciding
Scope	Multiple layer/Multiple aspect

Elements

- Business actor
- Business role
- Business collaboration
- Business interface
- Business process/function/interaction
- Business event
- Business service
- Business object
- Product
- Contract
- Application component/collaboration
- Application interface
- Application process/function/interaction
- Application event
- Application service
- Data object
- Technology service
- Artifact
- Material
- Value

C.1.4 Application Cooperation Viewpoint

The application cooperation viewpoint describes the relationships between applications components in terms of the information flows between them, or in terms of the services they offer and use. This viewpoint is typically used to create an overview of the application landscape of an organization. This viewpoint is also used to express the (internal) cooperation or orchestration of services that together support the execution of a business process.

Table 16: Application Cooperation Viewpoint Description

Application Cooperation Viewpoint	
Stakeholders	Enterprise, process, application, and domain architects
Concerns	Relationships and dependencies between applications, orchestration/choreography of services, consistency and completeness, reduction of complexity
Purpose	Designing
Scope	Multiple layer/Multiple aspect

Elements

- Location
- Application component/collaboration
- Application interface
- Application process/function/interaction
- Application event
- Application service
- Data object

C.1.5 Application Usage Viewpoint

The application usage viewpoint describes how applications are used to support one or more business processes, and how they are used by other applications. It can be used in designing an application by identifying the services needed by business processes and other applications, or in designing business processes by describing the services that are available. Furthermore, since it identifies the dependencies of business processes upon applications, it may be useful to operational managers responsible for these processes.

Table 17: Application Usage Viewpoint Description

Application Usage Viewpoint	
Stakeholders	Enterprise, process, and application architects, operational managers
Concerns	Consistency and completeness, reduction of complexity
Purpose	Designing, deciding
Scope	Multiple layer/Multiple aspect

Elements

- Business actor
- Business role
- Business collaboration
- Business process/function/interaction
- Business event
- Business object
- Application component/collaboration
- Application interface
- Application process/function/interaction
- Application event
- Application service
- Data object

C.1.6 Implementation and Deployment Viewpoint

The implementation and deployment viewpoint shows how one or more applications are realized on the infrastructure. This comprises the mapping of applications and components onto artifacts, and the mapping of the information used by these applications and components onto the underlying storage infrastructure.

Table 18: Implementation and Deployment Viewpoint Description

Implementation and Deployment Platform Viewpoint	
Stakeholders	Application and domain architects
Concerns	Structure of application platforms and how they relate to supporting technology
Purpose	Designing, deciding
Scope	Multiple layer/Multiple aspect

Elements

- Application component/collaboration
- Application interface
- Application process/function/interaction
- Application event
- Application service
- Data object
- System software
- Technology interface
- Path
- Technology process/function/interaction
- Technology service
- Artifact

C.1.7 Technology Viewpoint

The technology viewpoint contains the software and hardware technology elements supporting the Application Layer, such as physical devices, networks, or system software (e.g., operating systems, databases, and middleware).

Table 19: Technology Viewpoint Description

Technology Viewpoint	
Stakeholders	Infrastructure architects, operational managers
Concerns	Stability, security, dependencies, costs of the infrastructure
Purpose	Designing
Scope	Single layer/Multiple aspect

Elements

- Location
- Node
- Technology collaboration
- Device
- System software
- Technology interface
- Communication network
- Path
- Technology process/function/interaction
- Technology service
- Technology event
- Artifact

C.1.8 Technology Usage Viewpoint

The technology usage viewpoint shows how applications are supported by the software and hardware technology: the technology services are delivered by the devices; system software and networks are provided to the applications. This viewpoint plays an important role in the analysis of performance and scalability, since it relates the physical infrastructure to the logical world of applications. It is very useful in determining the performance and quality requirements on the infrastructure based on the demands of the various applications that use it.

Table 20: Technology Usage Viewpoint Description

Technology Usage Viewpoint	
Stakeholders	Application, infrastructure architects, operational managers
Concerns	Dependencies, performance, scalability
Purpose	Designing
Scope	Multiple layer/Multiple aspect

Elements

- Application component/collaboration
- Application process/function/interaction
- Application event
- Data object
- Node
- Device
- Technology collaboration
- System software
- Technology interface
- Communication network
- Path
- Technology process/function/interaction
- Technology service
- Technology event
- Artifact

C.1.9 Information Structure Viewpoint

The information structure viewpoint is comparable to the traditional information models created in the development of almost any information system. It shows the structure of the information used in the enterprise or in a specific business process or application, in terms of data types or (object-oriented) class structures. Furthermore, it may show how the information at the business level is represented at the application level in the form of the data structures used there, and how these are then mapped onto the underlying technology infrastructure; e.g., by means of a database schema.

Table 21: Information Structure Viewpoint Description

Information Structure Viewpoint	
Stakeholders	Domain and information architects
Concerns	Structure and dependencies of the used data and information, consistency and completeness
Purpose	Designing
Scope	Multiple layer/Single aspect

Elements

- Business object
- Representation
- Data object
- Artifact
- Meaning

C.1.10 Service Realization Viewpoint

The service realization viewpoint is used to show how one or more business services are realized by the underlying processes (and sometimes by application components). Thus, it forms the bridge between the business products viewpoint and the business process view. It provides a "view from the outside" on one or more business processes.

Table 22: Service Realization Viewpoint Description

Service Realization Viewpoint	
Stakeholders	Process and domain architects, product and operational managers
Concerns	Added-value of business processes, consistency and completeness, responsibilities
Purpose	Designing, deciding
Scope	Multiple layer/Multiple aspect

Elements

- Business actor
- Business role
- Business collaboration

- Business interface
- Business process/function/interaction
- Business event
- Business service
- Business object
- Representation
- Application component/collaboration
- Application interface
- Application process/function/interaction
- Application event
- Application service
- Data object

C.1.11 Physical Viewpoint

The physical viewpoint contains equipment (one or more physical machines, tools, or instruments) that can create, use, store, move, or transform materials, how the equipment is connected via the distribution network, and what other active elements are assigned to the equipment.

Table 23: Physical Viewpoint Description

Physical Viewpoint	
Stakeholders	Infrastructure architects, operational managers
Concerns	Relationships and dependencies of the physical environment and how this relates to IT infrastructure
Purpose	Designing
Scope	Multiple layer/Multiple aspect

Elements

- Location
- Node
- Device
- Equipment
- Facility

- Path
- Communication network
- Distribution network
- Material

C.1.12 Layered Viewpoint

The layered viewpoint pictures several layers and aspects of an Enterprise Architecture in one diagram. There are two categories of layers, namely *dedicated layers* and *service layers*. The layers are the result of the use of the "grouping" relationship for a natural partitioning of the entire set of objects and relationships that belong to a model. The technology, application, process, and actor/role layers belong to the first category. The structural principle behind a fully layered viewpoint is that each dedicated layer exposes, by means of the "realization" relationship, a layer of services, which are further on "serving" the next dedicated layer. Thus, we can easily separate the internal structure and organization of a dedicated layer from its externally observable behavior expressed as the service layer that the dedicated layer realizes. The order, number, or nature of these layers are not fixed, but in general a (more or less) complete and natural layering of an ArchiMate model should contain the succession of layers depicted in the example given below. However, this example is by no means intended to be prescriptive. The main goal of the layered viewpoint is to provide an overview in one diagram. Furthermore, this viewpoint can be used as support for impact of change analysis and performance analysis or for extending the service portfolio.

Table 24: Layered Viewpoint Description

Layered Viewpoint	
Stakeholders	Enterprise, process, application, infrastructure, and domain architects
Concerns	Consistency, reduction of complexity, impact of change, flexibility
Purpose	Designing, deciding, informing
Scope	Multiple layer/Multiple aspect

Elements

All core elements and all relationships are permitted in this viewpoint.

C.2 Motivation Viewpoints

A number of standard viewpoints for modeling motivational aspects have been defined. Each of these viewpoints presents a different perspective on modeling the motivation that underlies some Enterprise Architecture and allows a modeler to focus on certain aspects. Therefore, each viewpoint considers only a selection of the elements and relationships that have been described in the preceding sections.

The following viewpoints are distinguished:

- The *stakeholder viewpoint* focuses on modeling the stakeholders, drivers, the assessments of these drivers, and the initial goals to address these drivers and assessments.
- The *goal realization viewpoint* focuses on refining the initial, high-level goals into more concrete (sub-)goals using the aggregation relationship, and finally into requirements and constraints using the realization relationship.
- The *goal contribution viewpoint* focuses on modeling and analyzing the influence relationships between goals (and requirements).
- The *principles viewpoint* focuses on modeling the relevant principles and the goals that motivate these principles.
- The *requirements realization viewpoint* focuses on modeling the realization of requirements and constraints by means of core elements, such as actors, services, processes, application components, etc.
- The *motivation viewpoint* covers the entire motivational aspect and allows use of all motivational elements.
- The *capability map viewpoint* provides an overview of the capabilities of the enterprise.

All viewpoints are separately described below. For each viewpoint, its elements and relationships, the guidelines for its use, and its goal and target group are indicated. Furthermore, each viewpoint description contains example models. For more details on the goal and use of viewpoints, refer to Chapter 14 of [1].

C.2.1 Stakeholder Viewpoint

The stakeholder viewpoint allows the analyst to model the stakeholders, the internal and external drivers for change, and the assessments (in terms of strengths, weaknesses, opportunities, and threats) of these drivers. Also, the links to the initial (high-level) goals that address these concerns and assessments may be described. These goals form the basis for the requirements engineering process, including goal refinement, contribution and conflict analysis, and the derivation of requirements that realize the goals.

Table 25: Stakeholder Viewpoint Description

Stakeholder Viewpoint	
Stakeholders	Stakeholders, business managers, enterprise and ICT architects, business analysts, requirements managers
Concerns	Architecture mission and strategy, motivation
Purpose	Designing, deciding, informing
Scope	Motivation

Elements

- Stakeholder

- Driver
- Assessment
- Goal
- Outcome

C.2.2 Goal Realization Viewpoint

The goal realization viewpoint allows a designer to model the refinement of (high-level) goals into more tangible goals, and the refinement of tangible goals into requirements or constraints that describe the properties that are needed to realize the goals. The refinement of goals into sub-goals is modeled using the aggregation relationship. The refinement of goals into requirements is modeled using the realization relationship.

In addition, the principles may be modeled that guide the refinement of goals into requirements.

Table 26: Goal Realization Viewpoint Description

Goal Realization Viewpoint	
Stakeholders	Stakeholders, business managers, enterprise and ICT architects, business analysts, requirements managers
Concerns	Architecture mission, strategy and tactics, motivation
Purpose	Designing, deciding
Scope	Motivation

Elements

- Goal
- Principle
- Requirement
- Constraint
- Outcome

C.2.3 Requirements Realization Viewpoint

The requirements realization viewpoint allows the designer to model the realization of requirements by the core elements, such as business actors, business services, business processes, application services, application components, etc. Typically, the requirements result from the goal refinement viewpoint.

In addition, this viewpoint can be used to refine requirements into more detailed requirements. The aggregation relationship is used for this purpose.

Table 27: Requirements Realization Viewpoint Description

Requirements Realization Viewpoint	
Stakeholders	Enterprise and ICT architects, business analysts, requirements managers
Concerns	Architecture strategy and tactics, motivation
Purpose	Designing, deciding, informing
Scope	Motivation

Elements

- Goal
- Requirement/constraint
- Outcome
- Value
- Meaning
- Core element

C.2.4 Motivation Viewpoint

The motivation viewpoint allows the designer or analyst to model the motivation aspect, without focusing on certain elements within this aspect. For example, this viewpoint can be used to present a complete or partial overview of the motivation aspect by relating stakeholders, their primary goals, the principles that are applied, and the main requirements on services, processes, applications, and objects.

Table 28: Motivation Viewpoint Description

Motivation Viewpoint	
Stakeholders	Enterprise and ICT architects, business analysts, requirements managers
Concerns	Architecture strategy and tactics, motivation
Purpose	Designing, deciding, informing
Scope	Motivation

Elements

- Stakeholder
- Driver
- Assessment

- Goal
- Principle
- Requirement
- Constraint
- Outcome
- Value
- Meaning

C.3 Strategy Viewpoints

To describe strategic aspects of the enterprise, the viewpoints below have been defined. Each of these viewpoints presents a different perspective on modeling the high-level strategic direction and make-up of the enterprise and allows a modeler to focus on certain aspects. Therefore, each viewpoint considers only a selection of the elements and relationships that have been described in the preceding sections.

The following viewpoints are distinguished:

- The *capability map viewpoint* provides an overview of the capabilities of the enterprise.
- The *outcome realization viewpoint* describes how high-level, business-oriented results are produced by the capabilities and resources of the enterprise.
- The *resource map viewpoint* provides a structured overview of the resources of the enterprise.

All viewpoints are separately described below. For each viewpoint, its elements and relationships, the guidelines for its use, and its goal and target group are indicated. For more details on the goal and use of viewpoints, refer to Chapter 14 of [1].

C.3.1 Strategy Viewpoint

The strategy viewpoint allows the business architect to model a high-level, strategic overview of the strategies (courses of action) of the enterprise, the capabilities and resources supporting those, and the envisaged outcomes.

Table 29: Strategy Viewpoint Description

Strategy Viewpoint	
Stakeholders	CxOs, business managers, enterprise and business architects
Concerns	Strategy development
Purpose	Designing, deciding
Scope	Strategy

Elements

- Course of action
- Capability
- Resource
- Outcome

C.3.2 Capability Map Viewpoint

The capability map viewpoint allows the business architect to create a structured overview of the capabilities of the enterprise. A capability map typically shows two or three levels of capabilities across the entire enterprise. It can, for example, be used as a heat map to identify areas of investment. In some cases, a capability map may also show specific outcomes delivered by these capabilities.

Table 30: Capability Map Viewpoint Description

Capability Map Viewpoint	
Stakeholders	Business managers, enterprise and business architects
Concerns	Architecture strategy and tactics, motivation
Purpose	Designing, deciding
Scope	Strategy

Elements

- Outcome
- Capability
- Resource

C.3.3 Outcome Realization Viewpoint

The outcome realization viewpoint is used to show how the highest-level, business-oriented results are produced by the capabilities and underlying core elements.

Table 31: Outcome Realization Viewpoint Description

Outcome Realization Viewpoint	
Stakeholders	Business managers, enterprise and business architects
Concerns	Business-oriented results
Purpose	Designing, deciding
Scope	Strategy

Elements

- Capability
- Resource
- Outcome
- Value
- Meaning
- Core element

C.3.4 Resource Map Viewpoint

The resource map viewpoint allows the business architect to create a structured overview of the resources of the enterprise. A resource map typically shows two or three levels of resources across the entire enterprise. It can, for example, be used as a heat map to identify areas of investment. In some cases, a resource map may also show relationships between resources and the capabilities they are assigned to.

Table 32: Resource Map Viewpoint Description

Resource Map Viewpoint	
Stakeholders	Business managers, enterprise and business architects
Concerns	Architecture strategy and tactics, motivation
Purpose	Designing, deciding
Scope	Strategy

Elements

- Resource
- Capability
- Work package

C.4 Implementation and Migration Viewpoints

The following standard viewpoints for modeling implementation and migration aspects are distinguished:

- The *project viewpoint* is primarily used to model the management of architecture change.
- The *migration viewpoint* is used to model the transition from an existing architecture to a target architecture.
- The *implementation and migration viewpoint* is used to model the relationships between the programs and projects and the parts of the architecture that they implement.

All viewpoints are described separately below. For each viewpoint the comprised elements and relationships, the guidelines for the viewpoint use, and the goal and target group and of the viewpoint are indicated. Furthermore, each viewpoint description contains example models. For more details on the goal and use of viewpoints, refer to Chapter 14 of [1].

C.4.1 Project Viewpoint

A project viewpoint is primarily used to model the management of architecture change. The "architecture" of the migration process from an old situation (current state Enterprise Architecture) to a new desired situation (target state Enterprise Architecture) has significant consequences on the medium and long-term growth strategy and the subsequent decision-making process. Some of the issues that should be taken into account by the models designed in this viewpoint are:

- Developing a fully-fledged organization-wide Enterprise Architecture is a task that may require several years.
- All systems and services must remain operational regardless of the presumed modifications and changes of the Enterprise Architecture during the change process.
- The change process may have to deal with immature technology standards (e.g., messaging, security, data, etc.).
- The change has serious consequences for the personnel, culture, way of working, and organization.

Furthermore, there are several other governance aspects that might constrain the transformation process, such as internal and external cooperation, project portfolio management, project management (deliverables, goals, etc.), plateau planning, financial and legal aspects, etc.

Table 33: Project Viewpoint Description

Project Viewpoint	
Stakeholders	(operational) managers, enterprise and ICT architects, employees, shareholders
Concerns	Architecture vision and policies, motivation
Purpose	Deciding, informing
Scope	Implementation and Migration

Elements

- Goal
- Work package
- Implementation event
- Deliverable
- Business actor
- Business role

C.4.2 Migration Viewpoint

The migration viewpoint entails models and concepts that can be used for specifying the transition from an existing architecture to a desired architecture. Since the plateau and gap elements have been quite extensively presented in Section 13.2, here the migration viewpoint is only briefly described and positioned by means of the table below.

Table 34: Migration Viewpoint Description

Migration Viewpoint	
Stakeholders	Enterprise architects, process architects, application architects, infrastructure architects and domain architects, employees, shareholders
Concerns	History of models
Purpose	Designing, deciding, informing
Scope	Implementation and Migration

Elements

- Plateau
- Gap

C.4.3 Implementation and Migration Viewpoint

The implementation and migration viewpoint is used to relate programs and projects to the parts of the architecture that they implement. This view allows modeling of the scope of programs, projects, project activities in terms of the plateaus that are realized or the individual architecture elements that are affected. In addition, the way the elements are affected may be indicated by annotating the relationships.

Furthermore, this viewpoint can be used in combination with the programs and projects viewpoint to support portfolio management:

- The programs and projects viewpoint is suited to relate business goals to programs and projects. For example, this makes it possible to analyze at a high level whether all business goals are covered sufficiently by the current portfolio(s).
- The implementation and migration viewpoint is suited to relate business goals (and requirements) via programs and projects to (parts of) the architecture. For example, this makes it possible to analyze potential overlap between project activities or to analyze the consistency between project dependencies and dependencies among plateaus or architecture elements.

Table 35: Implementation and Migration Viewpoint Description

Implementation and Migration Viewpoint	
Stakeholders	(operational) managers, enterprise and ICT architects, employees, shareholders
Concerns	Architecture vision and policies, motivation
Purpose	Deciding, informing
Scope	Multiple layer/Multiple aspect

Elements

- Goal
- Requirement
- Constraint
- Work package
- Implementation event
- Deliverable
- Plateau
- Gap
- Business actor
- Business role

- Location
- Core element

D Relationship to Other Standards (Informative)

This appendix describes the relationship of the ArchiMate language to other standards, including the TOGAF framework, UML, BPMN, and BMM.

D.1 The TOGAF Framework

The ArchiMate language, as described in this standard, complements the TOGAF framework [4] in that it provides a vendor-independent set of concepts, including a graphical representation, that helps to create a consistent, integrated model "below the waterline", which can be depicted in the form of TOGAF views.

The structure of the ArchiMate core language closely corresponds with the three main architectures as addressed in the TOGAF ADM. The strategy, motivation, implementation, and migration elements approximately map onto the remainder of the ADM (although these elements may also be used in Phases B, C, and D). This is illustrated in Figure 108. This correspondence indicates a fairly easy mapping between TOGAF views and the ArchiMate viewpoints. A more detailed description of this correspondence is given in [6].

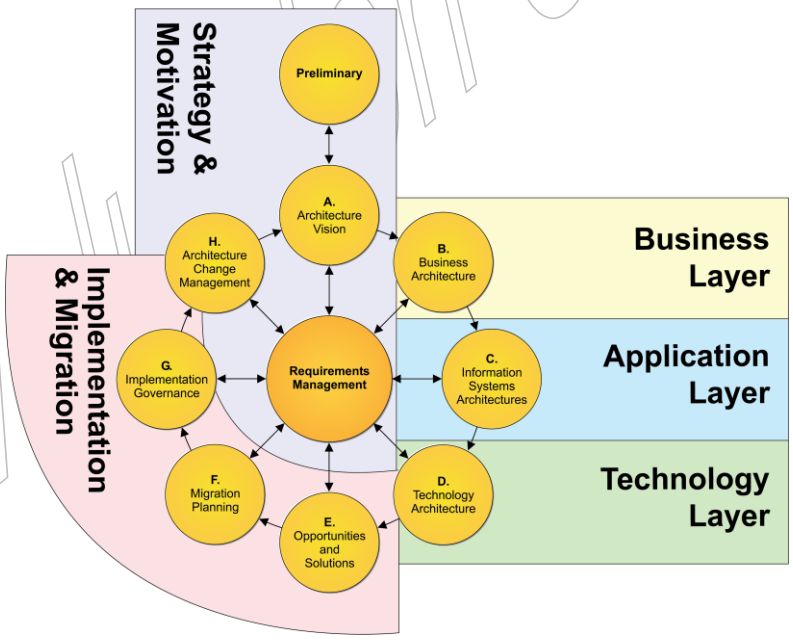


Figure 108: Correspondence between the ArchiMate Language and the TOGAF ADM

Although some of the viewpoints that are defined in the TOGAF standard cannot easily be mapped onto ArchiMate viewpoints, the ArchiMate language and its analysis techniques support the concepts addressed in these viewpoints. While there is no one-to-one mapping between them, there is still a fair amount of correspondence between the ArchiMate viewpoints and the TOGAF viewpoints. Although corresponding viewpoints from the two standards do not necessarily have identical coverage, many viewpoints from both address largely the same issues.

Moreover, the viewpoint mechanism described in Section 14.4 lends itself well to define TOGAF viewpoints using ArchiMate concepts.

Is it important to reiterate that the ArchiMate standard is a modeling language and not a framework, and therefore the viewpoint definitions are more detailed and they specify the stakeholders, concerns, level of detail, or abstraction level, and also the entity types involved in the viewpoints. In the TOGAF standard this is presented in a more general way, so sometimes there cannot be a one-to-one mapping between the entities and some interpretation or transformation will be required.

In conclusion, the TOGAF and ArchiMate standards can easily be used in conjunction:

- The two standards complement each other with respect to the definition of an architecture development process and the definition of an Enterprise Architecture modeling language.
- The two standards overlap in their use of viewpoints, and the concept of an underlying common repository of architectural artifacts and models; i.e., they have a firm common foundation.
- The combined use of the two standards can support a better communication with stakeholders.

See [6] for a detailed explanation of how the ArchiMate and TOGAF standards can be used together.

D.2 The BPMN Standard

Both the ArchiMate language and BPMN [12] can be used for modeling business processes. Their aims are different, however. ArchiMate notation is typically used for high-level processes and their relations to the enterprise context, but is not intended for detailed workflow modeling, whereas BPMN supports detailed sub-process and task modeling down to the level of executable specifications, but lacks the broader enterprise context, for example, to model the application services that support a process or the goals and requirements it has to fulfill.

Both languages share the concepts of (business) process and event. In the ArchiMate notation there is a single business process element that may be decomposed in other processes that are related using flow and triggering relationships, possibly using junctions to represent more complex connections. BPMN has a more fine-grained set of elements, with various types of events, tasks, and gateways. Its metamodel also distinguishes explicitly between process and sub-process (although it lacks a graphical representation of a business process itself). The BPMN concept of participant (or pool) and the ArchiMate concepts of business role or business actor (or application component for automated processes) also correspond.

In a typical scenario, both languages can be used in conjunction. Mapping from ArchiMate notation down to BPMN is fairly straightforward. The other way around loses the detailed elements of BPMN. Moreover, there are structural differences between the languages that preclude a direct concept-to-concept mapping and may merit a pattern-based approach. A detailed description of such a mapping is beyond the scope of this standard.

D.3 The UML Standard

The ArchiMate language has derived a number of concepts from UML [8]. For other concepts, straightforward correspondences can be defined.

In the Business Layer, the ArchiMate business process concept can be mapped onto UML activity diagrams, where more detailed specifications of such processes can be given (although BPMN would be the preferred language for detailed process and workflow modeling). The ArchiMate business actor and role concepts can both be mapped onto UML actors, although the latter can also be used for modeling automated actors. Business collaborations have been inspired by collaborations as defined in the UML standard [8], although the UML collaborations apply to components in the Application Layer.

In the Application Layer, the application component element corresponds to the UML component. This facilitates the direct linkage between higher-level Enterprise Architecture models described in ArchiMate notation and lower-level solution architecture and implementation models in UML in one continuous development chain. In less direct manner, the ArchiMate application function concept can be mapped onto UML activity diagrams, and an application service to a use-case diagram. Application collaborations also correspond to UML collaborations.

Many of the elements of the ArchiMate Technology Layer correspond directly to UML. The node, artifact, device, system software, and path elements have a direct counterpart in UML (where system software is called execution environment).

In addition to these elements, many relationships in the ArchiMate language have close ties to UML as well. The ArchiMate association, composition, aggregation, specialization, and realization relationships have a direct counterpart in UML.

There are also some notable differences between the two languages. The ArchiMate serving relationship (formerly used by) is different from UML dependency. Although their notations are similar, their directions are different. UML dependency is often used to model, for example, function calls in software programs, but in ArchiMate notation, the direction of the serving relationship denotes the direction of service delivery, independent of whether this service is called by the user or offered pro-actively by the provider. At the architectural level at which the ArchiMate language is aimed, the run-time operational details of such call graphs is less important than the more stable and generic notion of service provision.

This also points to another important difference: UML does not have a separate service concept, since in its object-oriented paradigm the behavior expressed by a service is encapsulated within the interface offering that behavior (i.e., its operations). The ArchiMate language differentiates between interfaces and the services they provide to allow, for example, specifying that the same service is offered through multiple interfaces. Hence, an ArchiMate application interface does not equate directly with a UML interface.

Finally, UML has a predefined, fixed set of diagram types, whereas the ArchiMate viewpoint mechanism allows for the construction of custom, stakeholder-oriented views on an architecture.

See [16] for a more detailed explanation about how the UML language and the ArchiMate standard can be used together.

D.4 The BMM Standard

The ArchiMate strategy and motivation elements have been inspired partly by the Business Motivation Model (BMM) [15]. BMM distinguishes between means, ends, and influencers and assessments. It provides fairly detailed concepts for these categories. The ArchiMate course of action element corresponds directly with the course of action element in BMM, whereas its directive concepts can be modeled with the ArchiMate principle, requirement, and constraint elements.

BMM concepts for modeling ends are typically mapped onto the ArchiMate goal element. Its influencers correspond to the ArchiMate element of driver, whereas its assessments map directly onto the ArchiMate assessment element.

Although a mapping between many of the ArchiMate motivation and implementation elements and BMM concepts is possible, BMM provides a more detailed, fine-grained description of business motivation. In that sense, it is comparable to the other languages described in this appendix. Where the ArchiMate language aims to cover a broad scope and interlink various domains, these more specialized languages zoom in on the details of their specific domains.

E Changes from ArchiMate 2.1 to ArchiMate 3.0 (Informative)

The main changes between Version 2.1 and Version 3.0 of the ArchiMate Specification are listed below. Note that this is not an exhaustive list; various smaller improvements have been made throughout the text of the document.

- Changed various definitions to increase alignment with the TOGAF framework.
- Added an upper-level generic metamodel to explain the full structure of the language.
- Restructured the set of relationships into structural, dynamic, dependency, and other relationships.
- Allowed relationships to other relationships in some cases; e.g., to associate objects with flows or aggregate relationships within plateaus.
- Improved the derivation of relationships, relaxed the constraints on relationships between layers in the ArchiMate core language, and improved the grouping and junction concepts.
- Renamed the 'used by' relationship to 'serving', in line with the other active names of relationships.
- Changed the notation of the influence relationship for consistency with the other dependency relationships (access and serving).
- Introduced a directional notation for the assignment relationship by replacing the black circle at the 'to' end by an arrow.
- Added an optional notation to denote the layer of an element. A letter 'M', 'S', 'B', 'A', 'T', 'P', or 'I' in the top-left corner of an element can be used to denote a Motivation, Strategy, Business, Application, Technology, Physical, or Implementation & Migration element, respectively.
- Changed the notation of the representation and contract elements, to distinguish these from deliverable and business object, respectively.
- Added events (with a time attribute) at all layers in the ArchiMate core language as well as to the Implementation and Migration elements.
- Renamed the Motivation Extension to Motivation elements and introduced a new outcome element.
- Moved the value and meaning concepts from the Business Layer of the ArchiMate core language to the Motivation elements.
- Introduced new Strategy elements for modeling the enterprise at a strategic level, notably capability, resource, and course of action.
- Moved the location element to the generic metamodel.

- Abolished the ‘required interface’ notation.
- Renamed the elements in the Technology Layer from infrastructure x to technology x.
- Added application process, technology process , technology interaction, and technology collaboration, to increase the regularity of the layers.
- Extended the Technology Layer with elements for modeling the physical world: facility, equipment, material, and distribution network.
- Renamed the ‘communication path’ element to ‘path’ and extended its meaning, to integrate with the physical elements.
- Improved the description of viewpoints and the viewpoints mechanism, removed the introductory viewpoint, and moved the basic viewpoints listed in the standard to an informative appendix to indicate they are intended as examples, not as a normative or exhaustive list.
- Replaced the examples throughout the document.
- Described the relationships of the ArchiMate standard with several other standards.
- Created new tables of relationships based on the changes in the metamodel and derivation properties.

ArchiMate 2.1 models are still mostly valid in ArchiMate 3.0. Two transformations may be applied to ensure conformance to the new version of the standard:

- Rename “used by” relationships to “serving”.
- If a relationship between two elements in a model is no longer permitted (according to Appendix B), replace it by an association. If it concerns an assignment of an application component to a business process or function, this may be replaced by a realization relationship from the application component to the business process or function. If it concerns an assignment of a location to another element, this may be replaced by an aggregation. In some cases, the modeler may want to replace the location by a facility.

Acronyms

ADM	Architecture Development Method (TOGAF framework)
ASCII	American Standard Code for Information Interchange
B2B	Business-to-Business
BMM	Business Motivation Model
BPMN	Business Process Model and Notation
CRM	Customer Relationship Management
DBMS	Database Management System
ERD	Entity Relationship Diagram
GUI	Graphical User Interface
HTML	HyperText Markup Language
JEE	Java, Enterprise Edition (was J2EE)
LAN	Local Area Network
PDF	Portable Document Format
RTF	Rich Text Format
SWOT	Strengths, Weaknesses, Opportunities, and Threats
UML	Unified Modeling Language
WAN	Wide Area Network
WLAN	Wireless Local Area Network

Index