ArchiMate® 4 Specification

The Open Group Standard

Table of Contents

ArchiMate 4 Specification

The Open Group Standard

The Open Group Standard
ArchiMate® 4 Specification
ISBN: 1-957866-75-8
Document Number: C260

Published by The Open Group, April 2026.

Preface

The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through technology standards and open source initiatives by fostering a culture of collaboration, inclusivity, and mutual respect among our diverse group of 900+ memberships. Our membership includes customers, systems and solutions suppliers, tool vendors, integrators, academics, and consultants across multiple industries.

The mission of The Open Group is to drive the creation of Boundaryless Information Flow™ achieved by:

- Working with customers to capture, understand, and address current and emerging requirements, establish policies, and share best practices
- Working with suppliers, consortia, and standards bodies to develop consensus and facilitate interoperability, to evolve and integrate specifications and open source technologies
- Offering a comprehensive set of services to enhance the operational efficiency of consortia
- Developing and operating the industry's premier certification service and encouraging procurement of certified products

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 are available at www.opengroup.org/library.

This Document

This document is the ArchiMate® 4 Specification, a standard of The Open Group. It has been developed and approved by The Open Group.

Intended Audience

The intended audience of this document is threefold:

- 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 document is as follows:

- Chapter 1, *Introduction*, provides the introduction to this document, including the objectives, a brief overview, conformance requirements, and terminology
- Chapter 2, *Definitions*, defines the general terms used in this document
- Chapter 3, *Language Structure*, describes the structure of the ArchiMate modeling language, including the top-level structure, aspects, domains, the ArchiMate core language, and the ArchiMate full language
- Chapter 4, *Common Domain*, describes the structure of the common, generic ArchiMate elements
- Chapter 5, *Relationships and Junctions*, describes the relationships in the language
- Chapter 6, *Motivation Domain*, describes the concepts for expressing the motivation for an architecture, together with examples
- Chapter 7, *Strategy Domain*, provides elements for modeling the enterprise at a strategic level, together with examples
- Chapter 8, *Business Domain*, covers the definition and usage of the Business Domain elements, together with examples
- Chapter 9, *Application Domain*, covers the definition and usage of the Application Domain elements, together with examples
- Chapter 10, *Technology Domain*, covers the definition and usage of the Technology Domain elements, together with examples

- Chapter 11, *Relationships Between Core Domains*, covers the relationships between different domains of the language
- Chapter 12, *Implementation and Migration Domain*, describes the language elements for expressing the implementation and migration aspects of an architecture (e.g., projects, deliverables, and plateaus)
- Chapter 13, *Stakeholders, Architecture Views, and Viewpoints*, describes the ArchiMate viewpoint mechanism
- Chapter 14, *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, *Relationships (Normative)*, is a normative appendix detailing the required relationships between elements of the language and the rules to derive these
- Appendix C, *Example Viewpoints*, 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, Specifications, and Guidance Documents*, describes the relationships of the ArchiMate language to other standards and specifications, including the TOGAF® framework, the BIZBOK® Guide, BPMN™, UML®, and BMM™
- Appendix E, *Changes from Version 2.1 to This Document*, is an informative appendix outlining the changes in the standard between Version 2.1 and this document

Trademarks

ArchiMate, FACE, FACE logo, Future Airborne Capability Environment, Making Standards Work, Open Footprint, Open O logo, Open O and Check certification logo, Open Subsurface Data Universe, OSDU, Sensor Open Systems Architecture, SOSA, SOSA logo, The Open Group, TOGAF, UNIX, UNIXWARE, and X logo are registered trademarks and Boundaryless Information Flow, Build with Integrity Buy with Confidence, Commercial Aviation Reference Architecture, Dependability Through Assuredness, Digital Practitioner Body of Knowledge, DPBoK, Eagle AI Assistant and logo, EMMM, FHIM Profile Builder, FHIM logo, FPB, IT4IT, IT4IT logo, O-AA, O-DA, O-DEF, O-HERA, O-PAS, O-TTPS, O-VBA, Open Agile Architecture, Open FAIR, Open Process Automation, Open Trusted Technology Provider, and Sensor Integration Simplified are trademarks of The Open Group.

A Guide to the Business Architecture Body of Knowledge and BIZBOK are registered trademarks of the Business Architecture Guild.

IBM is a registered trademark of International Business Machines Corp.

Java is a registered trademark of Oracle.

UML and Unified Modeling Language are registered trademarks and BMM, BPMN, Business Motivation Model, and Business Process Model and Notation are trademarks of the Object Management Group.

Zachman is a registered trademark of Zachman International, Inc.

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 Open Group ArchiMate Forum for developing this standard.

The Open Group gratefully acknowledges the contribution of the following people in the development of this document:

- Stefan Alsop
- Miguel Ansaras
- Steven Bradley
- Antonio Carlos Plais do Couto
- Andreas Ditze
- Roger Griessen
- Maamoun Hasuneh
- Mark Heerkens
- Henk Jonkers
- Andrew Josey
- Marinus F de Kok
- Marc Lankhorst
- Leoš Mates
- Milan Rubeš
- Jean-Baptiste Sarrodie
- Ed Walters

The Open Group gratefully acknowledges the following reviewers who participated in the Company Review of this document:

- Antonio Carlos Plais do Couto
- Andreas Ditze
- Roger Griessen
- Mamoun Hasuneh
- Andrew Josey
- Marinus F de Kok
- Marc Lankhorst
- Milan Rubeš

- Jean-Baptiste Sarrodie
- Ed Walters

The first version of this 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 document. These references are informative.

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

[Agile Manifesto]	Agile Manifesto, by Kent Beck et al., 2021; refer to: agilemanifesto.org
[BIZBOK Guide]	A Guide to the Business Architecture Body of Knowledge® (BIZBOK® Guide), Version 14.0, published by the Business Architecture Guild. 2025; refer to: www.businessarchitectureguild.org
[BMM]	Business Motivation Model™ (BMM™), Version 1.3, published by Object Management Group, April 2015; refer to: www.omg.org/spec/BMM
[BPMN]	Business Process Model and Notation™ (BPMN™), Version 2.0.2, published by Object Management Group, April 2015; refer to: www.omg.org/spec/BPMN
[C19C]	ArchiMate® Model Exchange File Format for the ArchiMate Modeling Language, Version 3.1, The Open Group Standard (C19C), published by The Open Group, November 2019; refer to: www.opengroup.org/library/c19c
[C220]	TOGAF® Standard, 10th Edition, a standard of The Open Group (C220), published by The Open Group, April 2022 and updated May 2025 to include Technical Corrigendum 1; refer to: www.opengroup.org/library/c220
[Eertink et al., 1999]	A Business Process Design Language, by H. Eertink, W. Janssen, P. Oude Luttighuis, W. Teeuw, and C. Vissers, published in the Proceedings of the First World Congress on Formal Methods, Toulouse, France, September 1999
[Engelsman et al., 2011]	Extending Enterprise Architecture Modeling with Business Goals and Requirements, by W. Engelsman, D.A.C. Quartel, H. Jonkers, and M.J. van Sinderen, published in Enterprise Information Systems, 5(1):9-36, February 2011
[G175]	TOGAF® Series Guide: The TOGAF® Technical Reference Model (TRM) (G175), published by The Open Group, September 2017; refer to: www.opengroup.org/library/g175
[G178]	TOGAF® Series Guide: Value Streams (G178), published by The Open Group, April 2022 and updated May 2025 to include Technical Corrigendum 1; refer to: www.opengroup.org/library/g178
[G211]	TOGAF® Series Guide: Business Capabilities, Version 2 (G211), published by The Open Group, April 2022; refer to: www.opengroup.org/library/g211
[G21E]	How to Use the ArchiMate® Modeling Language to Support the TOGAF® Standard, The Open Group Guide (G21E), published by The Open Group, April 2022; refer to: www.opengroup.org/library/g21e

[ISO/IEC/IEEE 42010]	ISO/IEC/IEEE 42010:2022: Software, Systems and Enterprise — Architecture Description; refer to: www.iso.org/standard/74393.html
[Jonkers & Iacob, 2009]	Performance and Cost Analysis of Service-Oriented Enterprise Architectures, by H. Jonkers and M.E. Iacob, published in Global Implications of Modern Enterprise Information Systems: Technologies and Applications, edited by A. Gunasekaran, IGI Global, 2009
[Lankhorst et al., 2010]	The Anatomy of the ArchiMate® Language, by M. Lankhorst, E. Proper, H. Jonkers, published in the International Journal of Information Systems Modeling and Design (IJISMD), 1(1):1-32, January-March 2010
[Lankhorst et al., 2017]	Enterprise Architecture at Work: Modeling, Communication, and Analysis, Fourth Edition, by Marc Lankhorst et al., published by Springer, 2017
[Sowa & Jackman, 1992]	Extending and Formalizing the Framework for Information Systems Architecture, by J.F. Sowa and J.A. Zachman, published in IBM Systems Journal, Volume 31, No. 3, pp.590-616, 1992
[UML, 2011]	Unified Modeling Language®: Infrastructure, Version 2.4.1, published by Object Management Group, August 2011; refer to: www.omg.org/spec/UML/2.4.1/Infrastructure/PDF
[W134]	Using the ArchiMate® Language with UML®, The Open Group White Paper (W134), published by The Open Group, September 2013; refer to: www.opengroup.org/library/w134
[Whittle & Myrick, 2004]	Enterprise Business Architecture: The Formal Link Between Strategy and Results, by Ralph Whittle and Conrad B. Myrick, published by CRC Press, August 2004
[van Buuren et al., 2004]	Composition of Relations in Enterprise Architecture, by R. van Buuren, H. Jonkers, M.E. Iacob, and P. Strating, published in the Proceedings of the Second International Conference on Graph Transformation, pp.39-53, edited by H. Ehrig et al., 2004

Chapter 1. Introduction

1.1. Objective

This document is the specification of the ArchiMate® Enterprise Architecture modeling language, a visual language with a default iconography for describing, analyzing, and communicating the many concerns of Enterprise Architectures as they change over time. The standard provides a set of concepts with their corresponding iconography for the representation of Architecture Descriptions. The ArchiMate ecosystem also supports an exchange format in XML, which allows model and diagram exchange between tools [C19C].

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. It provides a structuring mechanism for architecture domains and aspects. It distinguishes between the model concepts 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 Domains of Enterprise Architectures, and uses realization relationships to relate concrete elements to more abstract elements across these domains.

Note that this specification adheres to the TOGAF definition of “enterprise” [C220]: “The highest level (typically) of description of an organization and typically covers all missions and functions. An enterprise will often span multiple organizations”. So, an enterprise in this sense does not need to be contained within the boundaries of a single organization and can cover an entire ecosystem or federation of organizations.

1.3. Conformance

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

A conforming implementation:

1. Shall support the language structure, relationships, domains, cross-domain dependencies, and other elements as specified in Chapter 3, Chapter 4, Chapter 5, Chapter 6, Chapter 7, Chapter 8, Chapter 9, Chapter 10, Chapter 11, and Chapter 12
2. Shall support the standard iconography as specified in Chapter 4, Chapter 5, Chapter 6, Chapter 7, Chapter 8, Chapter 9, Chapter 10, Chapter 11, and Chapter 12, and summarized in Appendix A
3. Shall support the viewpoint mechanism as specified in Chapter 13
4. Shall support the language customization mechanisms as specified in Chapter 14 in an implementation-defined manner
5. Shall support the relationships between elements as specified in Appendix B
6. May support the example viewpoints described in Appendix C and Chapter 3, Chapter 4, Chapter 5, Chapter 6, Chapter 7, Chapter 8, Chapter 9, Chapter 10, Chapter 11, and Chapter 12

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

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.

Chapter 2. Definitions

For the purposes of this standard, the following terms and definitions apply. The TOGAF® Standard [C220] should be referenced for Enterprise Architecture-related terms not defined in this chapter. Merriam-Webster's Collegiate Dictionary should be referenced for all other terms not defined in this chapter.

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

2.1. ArchiMate Core Language

The central part of the ArchiMate language that defines the core elements to model Enterprise Architectures. It includes elements from four domains: Common, Business, Application, and Technology.

NOTE The ArchiMate core language is defined in detail in Section 3.2.1.

2.2. ArchiMate Full Language

The full content of the ArchiMate language that defines all elements that can be used to model Enterprise Architectures. It includes elements from the following domains: Motivation, Strategy, Common, Business, Application, Technology, and Implementation and Migration.

NOTE The ArchiMate full language is defined in detail in Section 3.2.1.

2.3. Architecture Domain

A grouping construct used to structure architectural descriptions into broad areas such as Business, Application, Technology, Motivation, Strategy, and Implementation & Migration.

2.4. Architecture View

A representation of a system from the perspective of a related set of concerns.

NOTE In some sections of this standard, the term "view" is used as a synonym for "architecture view".

2.5. Architecture Viewpoint

A specification of the conventions for a particular kind of architecture view.

NOTE In some sections of this standard, the term "viewpoint" is used as a synonym for "architecture viewpoint".

2.6. Aspect

Classification of elements based on domain-independent characteristics related to the concerns of different stakeholders. Used for positioning elements in the ArchiMate metamodel.

NOTE Aspects are described in Section 3.2.2.

2.7. Attribute

A property associated with an ArchiMate language concept.

2.8. Composite Element

Unlike elements from the active structure, behavior, and passive structure aspects, composite elements are collections of concepts that can be from multiple aspects of the language.

2.9. Concept

Either an element, a relationship, or a junction. See also Section 2.14 and Section 2.15.

NOTE The top-level language structure is defined in detail in Section 3.3.

2.10. Conformance

Fulfillment of specified requirements. See Section 1.3.

2.11. Conforming Implementation

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

2.12. Core Element

An element in one of the core domains (Common Domain, Business Domain, Application Domain, or Technology Domain) of the ArchiMate language.

2.13. Domain

A subset of the ArchiMate language for modeling specific characteristics of an enterprise and its architecture.

NOTE The TOGAF Standard uses *architecture domain* in much the same way, and distinguishes Business, Data, Application, and Technology as domains. The ArchiMate language covers this scope with the *Core Domains*, but extends the notion of domain to the entirety of the language.

2.14. Element

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

2.15. 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 3.3.

For a general definition of model, see the TOGAF Standard [C220].

2.16. 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.

Chapter 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 12.

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 general set of architecture concepts that reflects a view of systems as a mere set of inter-related entities.

To guide the development evolution of the standard, The Open Group ArchiMate Forum has established a set of guiding principles:

- Principle 1: Simplicity over Comprehensiveness

  Statement: The most important design consideration is that the language has been explicitly designed to be as small as possible, but still usable for most Enterprise Architecture modeling tasks.

  Rationale: Many other languages try to accommodate the 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.

  Implications:

  - Architecture models will focus on essential elements rather than exhaustive detail
  - Some specialized modeling needs may require supplemental approaches
  - Training and adoption will be streamlined due to reduced complexity
  - Stakeholder communication will improve through clarity and focus
- Principle 2: Needs of the Many over Wants of the Few

  Statement: Any change to the language shall cover a clearly identified and important use case of a substantial part of its users.

  Rationale: In line with the previous principle, keeping things simple for the “80%” is more important than catering for the wants of the “20%”.

  Implications:

  - Change requests require validation against broad organizational needs
  - Specialized requirements affecting only a small percentage of users may be deprioritized
  - Governance processes must evaluate impact across the majority of stakeholders
  - Custom extensions may be needed for specialized use cases

- Principle 3: Collaboration over Coverage

  Statement: If a need is already covered sufficiently by another modeling language, it is preferred to create a mapping to that language rather than incorporate those features into the architecture.

  Rationale: Integration with specialized notations is more effective than expanding the architecture framework to incorporate all modeling needs. For example, a detailed software design model is better written in Unified Modeling Language® (UML®), and the ArchiMate application component concept can be used to map between both models.

  Implications:

  - Architects must maintain knowledge of complementary modeling approaches
  - Integration points between different modeling languages must be established
  - Tool environments may need to support multiple notations
  - Training may cover integration approaches rather than just a single framework

- Principle 4: People over Tools

  Statement: Understandability is key.

  Rationale: The language was designed for communication among human users rather than for technical usage. There are many technical languages already and, per Principle 3, the architecture should work together with those rather than try to cover their scope and level of detail too.

  Implications:

  - Models should use clear visualization and consistent terminology
  - Appropriate levels of abstraction must be maintained for different audiences
  - Validation should include stakeholder comprehension assessment
  - Technical precision may sometimes be secondary to communicative clarity

- Principle 5: Communicating Architecture over Other Use Cases

  Statement: The architecture language was designed first and foremost to communicate (Enterprise) Architectures.

  Rationale: Maintaining effectiveness for core use cases and practitioners ensures the integrity and quality of architectural outputs while supporting broader stakeholder needs.

  Implications:

  - User experience design for architecture tools should optimize for architect workflows first
  - Other stakeholders may need simplified views or interfaces
  - Feedback processes should weight architect input appropriately

These guiding principles were inspired by the Agile Manifesto, which also says: *"While there is value in the items on the right, we value the items on the left more"* [Agile Manifesto].

In addition to these principles, the interested reader is referred to *The Anatomy of the ArchiMate® Language* [Lankhorst et al., 2010], *Enterprise Architecture at Work: Modeling, Communication, and Analysis* [Lankhorst et al., 2017], and *Extending Enterprise Architecture Modeling with Business Goals and Requirements* [Engelsman et al., 2011], which provide detailed descriptions of the language construction and design considerations.

3.2. The ArchiMate Language

The concepts of the ArchiMate language are organized in domains and aspects.

3.2.1. Domains of the ArchiMate Language

The ArchiMate core language defines a set of domains with specific active, behavior, and passive structure elements. Four domains are defined within the ArchiMate core language as follows:

- The Common Domain (see Chapter 4) contains generic elements (in particular behavior elements) that can be used in combination with elements from different domains
- The Business Domain (see Chapter 8) depicts the organization, how it relates to its internal and external customers, and what it consumes and produces
- The Application Domain (see Chapter 9) depicts the applications, how they work together to support the business, and which data they process
- The Technology Domain (see Chapter 10) comprises both information technology and operational (physical) technology

Figure 3-1 depicts the contents of the core language.

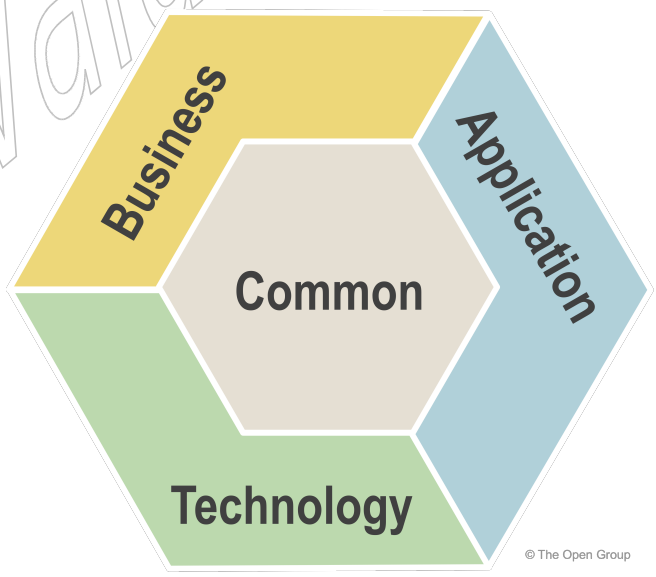


Figure 3-1: ArchiMate Core Language

Next to these core domains, the language also comprises:

- The Motivation Domain (see Chapter 6), containing elements to model stakeholders, goals, and other related contexts and motivations that drive the architecture
- The Strategy Domain (see Chapter 7), used to model strategic direction and choices, based on the motivation of the enterprise
- The Implementation and Migration Domain (see Chapter 12), with elements for modeling the evolution and implementation of architectures

The ArchiMate language does not define a specific domain for information or data. Instead, information modeling is supported across the different ArchiMate domains as elements from the passive structure aspect. This implies that business objects, data objects, and artifacts are used to represent information entities at different levels of abstraction.

In alignment with service orientation, the most important relationship between domains is the "serving" relationship, which shows how the elements in one domain are served by the services of other domains. (Note that services need not only serve elements in another domain, but also can serve elements in the same domain.)

A second type of link is formed by realization relationships: more concrete elements from one domain may realize more abstract elements from another domain; e.g., a "data object" (Application Domain) may realize a "business object" (Business Domain); or an "artifact" (Technology Domain) may realize either a "data object" or an "application component" (Application Domain).

The structure of the language allows for the modeling of the enterprise from different viewpoints, where a stakeholder can have concerns that cover multiple aspects and domains.

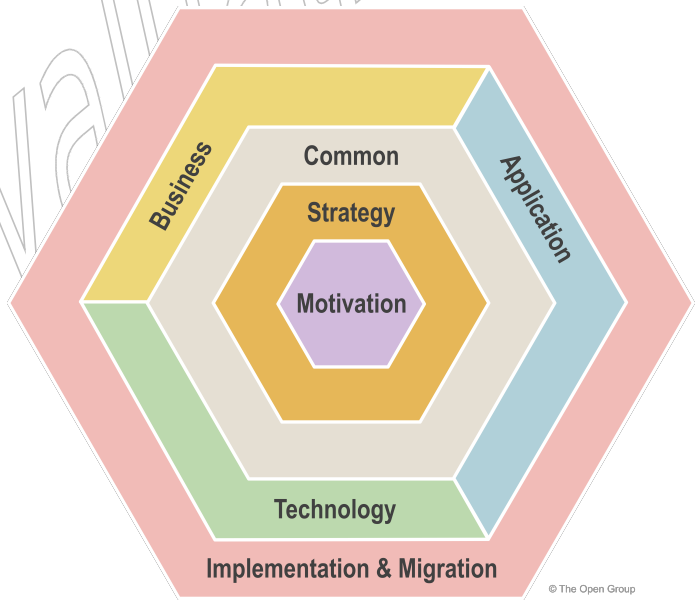


Figure 3-2: ArchiMate Full Language

Figure 3-2 shows a symbolic representation of the full language. Intentionally, “Motivation” is put in the center, since it describes the goals that the architecture is aiming for. Working outward from the center, these goals are translated into high-level and more concrete architectural designs, and a plan for the implementation of those designs.

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

3.2.2. Aspects of the ArchiMate Language

The three core aspects are:

- The *Active Structure Aspect*, which represents the structural elements (such as the business actors, application components, devices, and equipment 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 this behavior
- The *Passive Structure Aspect*, which represents the objects on which behavior is performed; these are usually information in the Business Domain and data in the Application Domain, but they may also be used to represent physical materials in the Technology Domain

These aspects were inspired by natural language, where a sentence has a subject (active structure), a verb (behavior), and an object (passive structure). By using the same constructs that people are used to in their own languages, the ArchiMate language is easier to learn and read.

A composite element, as shown in Figure 3-3, is an element that does not necessarily fit in a single aspect but may combine multiple aspects.

3.3. Top-Level Language Structure

Figure 3-3 outlines the top-level hierarchical structure of the language:

- A model is a collection of *concepts* — a concept is either an *element*, a *relationship*, or a *junction*
- An element is either a behavior element, a structure element, a motivation element, or a composite element
- A relationship is either a structural relationship, a dependency relationship, a dynamic relationship, or an other relationship

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. *Concrete* concepts are subtypes of these abstract concepts, which are defined later in this document. Those concrete concepts can be used in models and depicted in views (see also Section 13.3). This document does not define how these concepts are organized or stored in any tool that implements the standard; that is left to the implementers of such tools.

See Chapter 4 for an explanation of the notation used in Figure 3-3.

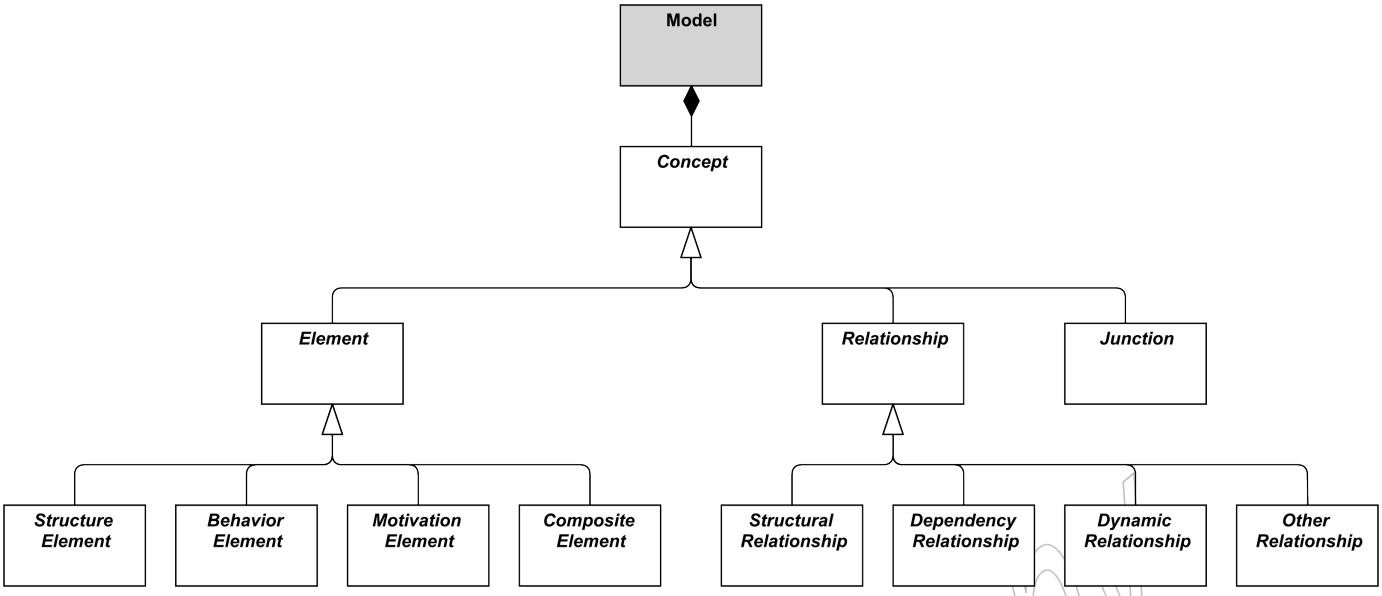


Figure 3-3: Top-Level Hierarchy of ArchiMate Concepts

3.4. Structure and Behavior Elements

The main hierarchy of behavior and structure elements of the ArchiMate language is presented in the metamodel fragment of Figure 3-4. It defines these elements in a generic, domain-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 domains) are depicted.

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

Structure elements can be subdivided into *active structure* elements and *passive structure* elements. Active structure elements can be further divided into *external* active structure elements (also called *interfaces*) and *internal* active structure elements.

Importantly, the designation of elements as "internal" or "external" as used here does not mean internal or external to the enterprise. The scope of this designation is only the specific element that performs some process or function (internal behavior), offers a service (external behavior), or provides an interface (external active structure). Hence, that scope can range from, for example, federations of organizations to individuals and from large systems to small components thereof.

Behavior elements can be subdivided into *internal behavior* elements, *services* (which equal external behavior elements), and *events*.

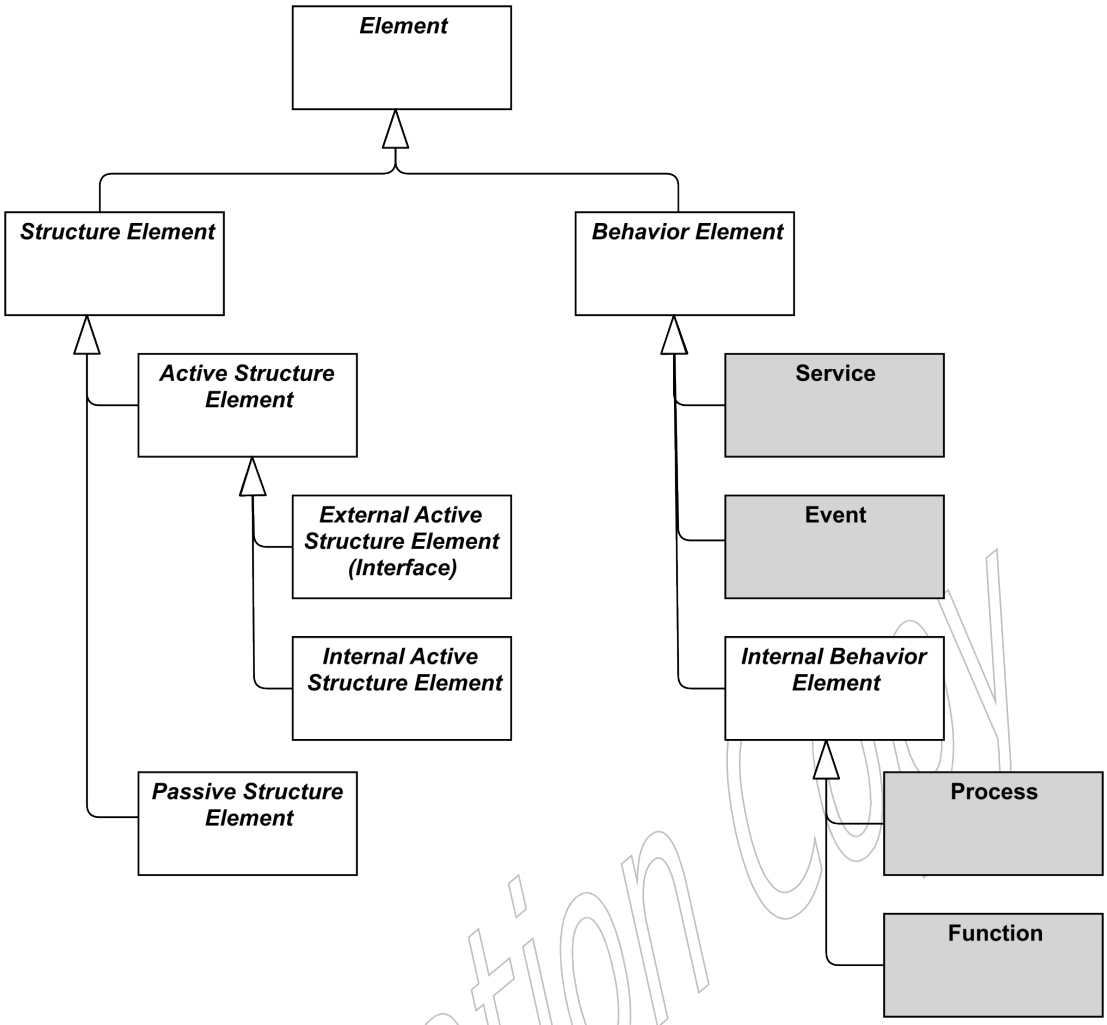


Figure 3-4: Hierarchy of Behavior and Structure Elements

3.4.1. Active Structure Elements

Active structure elements are the subjects that can perform behavior. These can be divided into internal active structure elements (the business actors, application components, nodes, etc.) that perform this behavior, and external active structure elements; i.e., the interfaces that expose this behavior to human consumers or non-human systems. 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 an internal active structure element makes one or more services available to its environment.

3.4.2. Passive Structure Elements

Passive structure elements can be accessed by behavior elements.

> A passive structure element represents an element on which behavior is performed.

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.

3.4.3. Behavior Elements

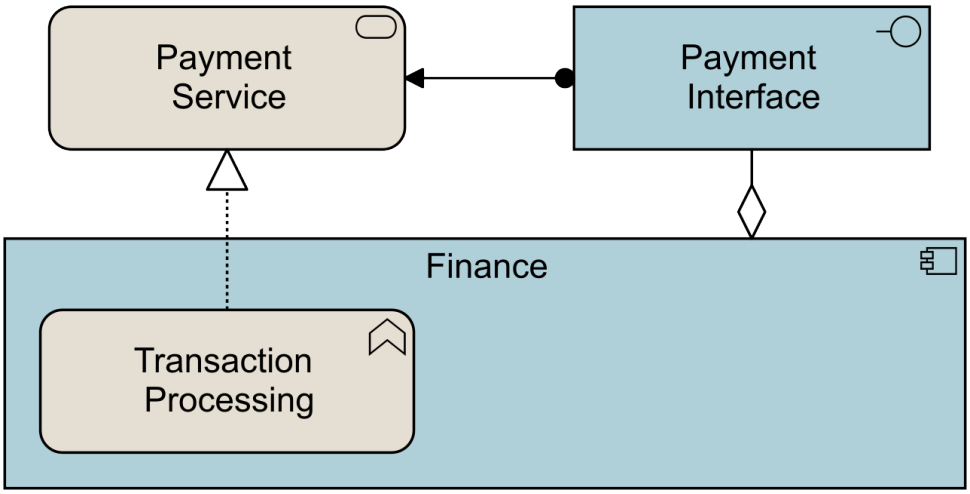
Behavior elements represent the dynamic aspects of an enterprise. Similar to active structure elements, behavior elements can be divided into *internal* behavior elements and services that are exposed to the environment.

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

> An external behavior element, called a service, represents an explicitly defined behavior, exposed to the environment.

3.4.4. Example

Example 3-1 illustrates both active structure and behavior elements. The internal active structure element “Finance” application component performs the internal behavior element “Transaction Processing” function. It aggregates the external active structure element “Payment Interface”, which provides access to the service “Payment Service”.



Example 3-1: Active Structure and Behavior Elements

3.5. 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) view (abstracting from the contents of the box) and an 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 constituents (people, applications, and infrastructure) must do it. In modeling new systems, it is often useful to start with the behaviors that the system must perform. 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, along with the realization relationships between them.

The distinction between logical and physical elements has also been carried over to the description of applications. The TOGAF Enterprise Metamodel [C220] includes a set of entities that describe business, data, application, technology components and services to describe architecture concepts. Logical components are implementation or product-independent encapsulations of data or functionality, whereas physical components are tangible software components, devices, etc. This distinction is captured in the TOGAF Standard in the form of Architecture Building Blocks (ABBs) and Solution Building Blocks (SBBs). 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 that are typically modeled using the grouping concept in the ArchiMate language.

The ArchiMate language has three ways of modeling such abstractions. First, as described in The Open Group Guide *How to Use the ArchiMate® Modeling Language to Support the TOGAF® Standard* [G21E], behavior elements such as functions can be used to model logical components, since they represent implementation-independent encapsulations of functionality. These logical components can be implemented as physical components, which can 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 Domain upwards. The Technology Domain 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 metamodel-level specializations of the application component element, as described in Chapter 14 (see also the examples in Section 14.2.3). 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 14.2.4).

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 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.

3.6. 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 Business Process Model and Notation™ (BPMN™), which have only one standardized notation. The viewpoint mechanism explained in Chapter 13 provides the means for defining such stakeholder-oriented visualizations.

Although the notation of the ArchiMate concepts can 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 toward an audience used to existing technical modeling techniques such as Entity Relationship Diagrams (ERDs), 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 elements consists of a box with an icon in the upper-right corner (the boxed notation) or the icon by itself (the icon notation). Both notations may be mixed in a view, if there are reasons to do so. It is recommended to use this standard iconography whenever possible, so that anyone familiar with the ArchiMate language can read the diagrams produced in it.

3.7. Use of Nesting

Nesting elements inside other elements can be used as an alternative graphical notation to express some relationships. This may lead to ambiguity though, and it is advisable to add an explanation to views where the meaning of nesting is not immediately clear. This is explained in more detail in Chapter 5 and in the definition of each of these relationships.

3.8. Use of Colors and Notational Cues

In the metamodel pictures within this document, shades of gray are used to distinguish elements belonging to the different aspects of the ArchiMate language, as follows:

- White for abstract (i.e., non-instantiable) concepts
- Gray for concrete concepts

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 document, colors are used to distinguish between the domains of the ArchiMate language, as follows:

- Purple for the Motivation Domain
- Orange for the Strategy Domain
- Warm gray for the Common Domain
- Yellow for the Business Domain
- Blue for the Application Domain
- Green for the Technology Domain
- Pink for the Implementation and Migration Domain

They can also be used for visual emphasis; see Chapter 6 in [Lankhorst et al., 2017]. In addition to the colors, other notational cues can be used to distinguish between the domains of the language. A letter M, S, C, B, A, T, or I in the top-left corner of an element can be used to denote a Motivation, Strategy, Common, Business, Application, Technology, or Implementation and Migration element, respectively. An example of this notation is depicted in Example 11-1.

The standard notation 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
- Rounded corners are used to denote behavior elements
- Diagonal corners are used to denote motivation elements

Chapter 4. Common Domain

The common, generic concepts, which are used across different domains, are shown in Figure 4-1. These are explained in more detail in the next sections.

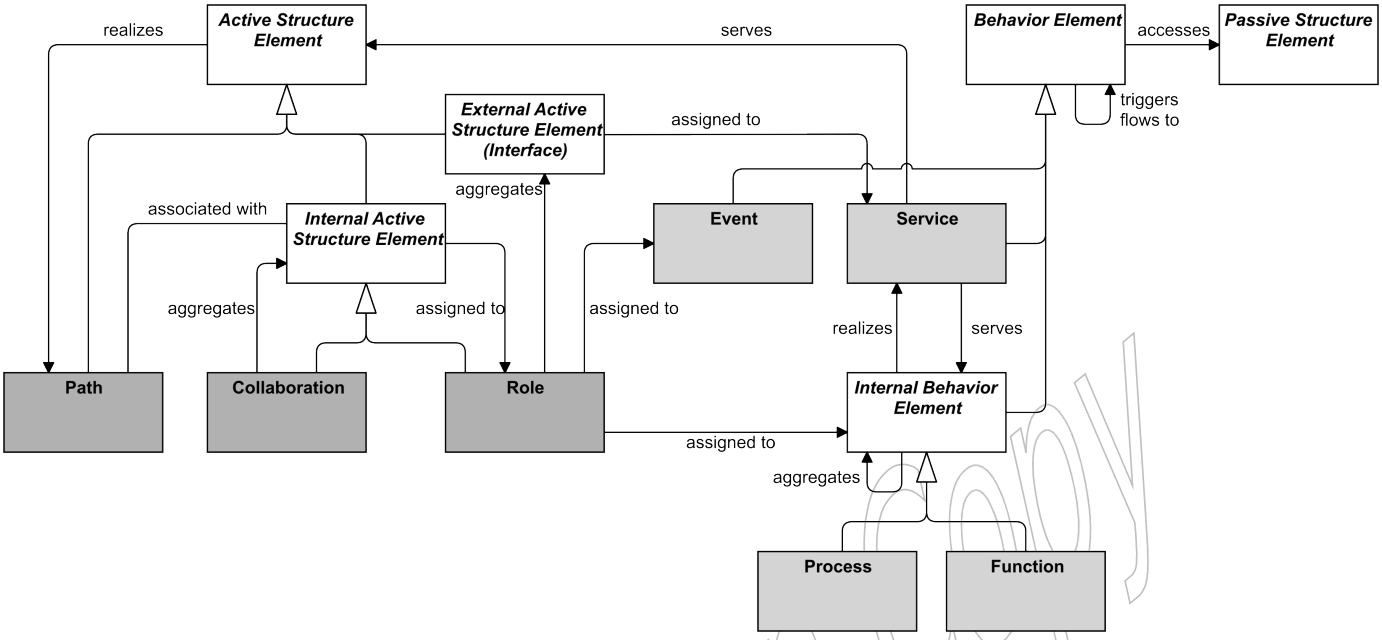


Figure 4-1: Common Domain Metamodel

NOTE The relationships shown in this and other metamodel figures are not to be confused with ArchiMate relationships. They are metamodel relationships expressing the structure of the language rather than a model *in* the language. The label of a relationship signifies the role of the source element in the relationship; e.g., a service serves an internal behavior element.

NOTE This and all other metamodel figures do not show all permitted relationships; every element in the language can have aggregation and specialization relationships to elements of the same type. Furthermore, there are indirect relationships that can be derived, as explained in Section 5.8. The full specification of permitted relationships can be found in Appendix B.

4.1. Active Structure Elements

4.1.1. Role

A role represents the position or purpose that a business actor, application component, node, device, system software, equipment, facility, or a collaboration has in performing specific behavior.

Roles with certain responsibilities or skills are assigned to processes or functions. An internal active structure element that is assigned to a role is responsible for ensuring 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 role with behavior, a role is also useful in a (structural) organizational sense; for instance, in the division of labor within an organization.

A role may aggregate interfaces that it provides to its environment, and interfaces may serve a role. The name of a role should preferably be a noun. For example, a business actor can be assigned to a “Requester” or “Approver” role, or an application component can be assigned to a “Policy Enforcement Point” or “Policy Decision Point” role.

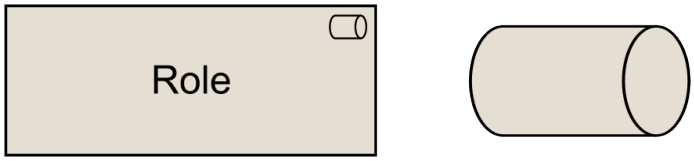


Figure 4-2: Role Notation

4.1.2. Collaboration

A collaboration represents a collection of business actors, application components, nodes, devices, system software, equipment, facilities, roles, and/or other collaborations that work together to perform collective behavior.

In some cases, behavior is the collective effort of more than one internal active structure element. In fact, a collaboration of two or more internal active structure elements results in collective behavior that may be more than simply the sum of the behavior of the separate elements. Collaborations represent this collective effort. A collaboration is a (possibly temporary) collection of internal active structure elements (including other collaborations) which perform collaborative behavior. Unlike, say, a department, a collaboration need not have an official (permanent) status within the organization.

A collaboration may be assigned to one or more processes or functions. A collaboration may aggregate interfaces that it provides to its environment, and interfaces may serve a collaboration. The name of a collaboration should preferably be a noun.

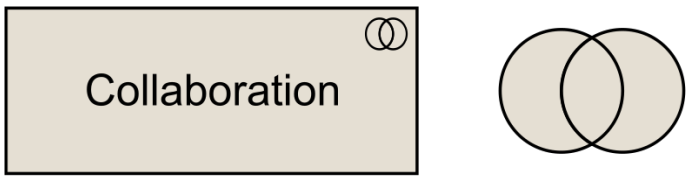


Figure 4-3: Collaboration Notation

4.1.3. Path

A path represents a logical link between business actors, application components, nodes, devices, system software, equipment, facilities, roles, and/or collaborations, through which these elements can exchange information, data, energy, or material.

A path is used to model the logical communication or distribution relations between internal active structure elements. It may be realized by one or more communication networks or distribution networks (which represent the physical communication or distribution links), and by any other active structure elements (to model the people and technology on which the path relies).

A path can be associated with the internal active structure elements it connects. Typical examples of paths are a Virtual Private Network (VPN), realized by WiFi, Local Area Network (LAN), and Wide Area Network (WAN) communication networks; intermodal freight, realized by a combination of shipping, trucking, and rail distribution networks; or a social network that lets users communicate with each other.

Figure 4-4: Path Notation

4.2. 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 service. A service represents a coherent piece of functionality that offers added value to the environment, independent of the way this functionality is realized internally.

When modeling the internal behavior, it is often useful to distinguish a *process view* and a *function view* on behavior; two elements associated with these views, *process* and *function*, are defined. Both elements can be used to aggregate more detailed processes/functions, but based on different aggregation criteria. A *process* represents a workflow consisting of smaller processes/functions executed in a certain order, 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, or some system in the case of automated processes. The goal of such a process is to “satisfy or delight the customer” [Whittle & Myrick, 2004]. A *function* aggregates behavior based on required skills, resources, (application) support, etc. Typically, the processes of an organization or system are defined based on the *products* and *services* that this organization or system offers, while the functions are often the basis for the assignment of resources to tasks.

It is permitted to use aggregation relationships between processes and functions; e.g., a process can aggregate other processes or functions, as can a function.

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

4.2.1. Service

A service represents an explicitly defined behavior that a business actor, application component, node, device, system software, equipment, facility, role, or collaboration provides to its environment.

Thus, a service exposes specific functionality of the providing active structure element (for example, an organization unit or IT system) from the perspective of others that use that service; the environment consists of everything outside this providing organization unit or 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. Services are accessible through interfaces.

A service may serve other behavior or active structure elements. A process or function may realize a service. An interface may be assigned to a service. A service may access passive elements. The name of a service should clearly describe the functionality being offered (for example, using a verb ending with “-ing”); e.g., “claims processing”, "transaction processing”, or “messaging”. Also, a name explicitly containing the word “service” may be used; e.g., “data replication service”.

Figure 4-5: Service Notation

4.2.2. Process

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

A process describes the internal behavior performed by a role that is required to produce a set of services.

A complex 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 processes and functions. Informally speaking, processes describe some kind of “flow” of activities, whereas functions group activities according to required skills, knowledge, resources, etc.

A process may be triggered by, or trigger, any other behavior element (i.e., event, process, or function). A process may access passive elements. A process may realize one or more services and may be served by other services. A role may be assigned to a process to model the responsibility for performing this process.

The name of a process should clearly indicate a predefined sequence of actions using a verb or verb-noun combination and may include the word “process”. Examples are “adjudicate claim”, “employee on-boarding”, “approval process”, “general ledger update job”, or “replicate database”.

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

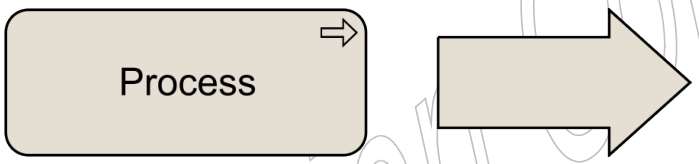


Figure 4-6: Process Notation

4.2.3. Function

A function represents a collection of behavior based on a chosen set of criteria.

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

There is a potential many-to-many relation between processes and functions. Complex processes in general involve activities that require various functions. In this sense, a process can form a string of functions. Organizational units or applications may coincide with functions due to their specific grouping of activities. In the Business Domain, functions are often closely aligned to an organization, but not necessarily explicitly governed by the organization.

Functions can realize capabilities. Where functions are often aligned with organization structure and describe the current, day-to-day activities of the enterprise, capabilities (see Section 7.3.1) represent its potential behavior and are independent from the implementation in the organization structure.

A function may be triggered by, or trigger, any other behavior element (event, process, or function). A function may access passive elements. A function may realize one or more services and may be served by other services. A role may be assigned to a function. The name of a function should clearly indicate a well-defined behavior, but unlike processes, it is typically not in the form of a verb-noun phrase. Examples are customer management, claims administration, recycling, or payment processing.

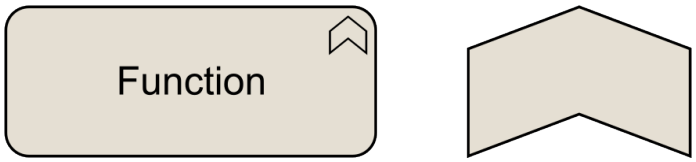


Figure 4-7: Function Notation

4.2.4. Event

An event represents something that happens or occurs inside or outside the enterprise, such as a state change.

Processes and other behavior may be triggered or interrupted by an event. Processes may raise events that trigger other processes or functions. Unlike processes or functions, an event is instantaneous: it does not have a duration. Events may originate from the environment of the organization (e.g., from a customer) or some external physical or IT system, as well as internal events generated within the organization or system world.

An event may trigger or be triggered (raised) by a process or function. An event may access passive elements and may aggregate other events. The name of an event should preferably be a verb in the perfect tense; e.g., “Claim Received”.

This event concept is similar to a BPMN event, and to the initial state and final state elements in UML activity diagrams. However, the ArchiMate event is more generally applicable in the sense that it can also be used to model other types of events, in addition to triggers of processes.

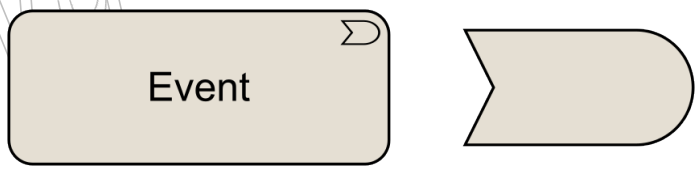
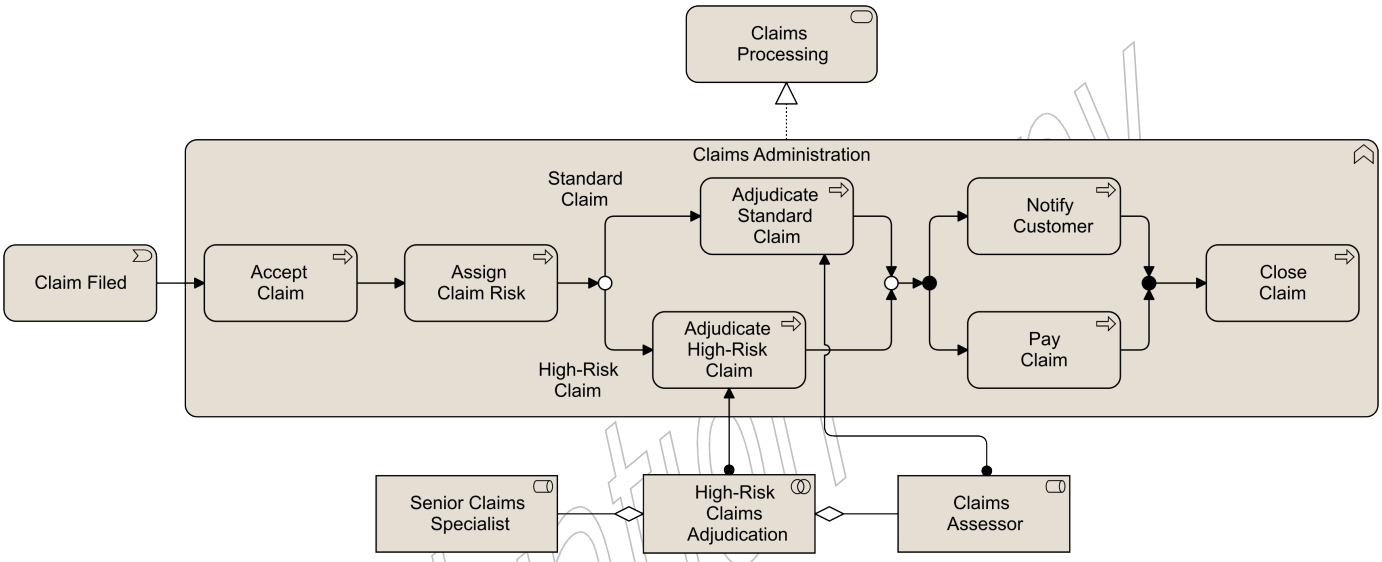


Figure 4-8: Event Notation

Example

"Claims Administration" is a function that aggregates a number of processes. This function realizes a "Claims Processing" service. An event "Claim Filed" triggers the first process "Accept Claim Risk", which in turn triggers a process "Assign Claim". Depending on the type of claim, either the process "Adjudicate Standard Claim" or the process "Adjudicate High-Risk Claim" is performed. According to the company policy, two roles should always be involved in the latter process, to minimize the risk of fraud. This is modeled with the collaboration "High-Risk Claims Adjudication". After adjudication, the processes "Notify Customer" and "Pay Claim" are performed in parallel, and when both have finished, process "Close Claim" is triggered.



Example 4-1: Common Domain Elements

4.3. Composite Elements

Unlike elements from the active structure, behavior, and passive structure aspects, composite elements are collections of concepts that can be from multiple aspects of the language. Grouping and location are common composite elements (see Figure 4-9). Other composite elements include product (see Section 8.4.1) in the Business Domain and plateau (see Section 12.2.3) in the Implementation and Migration Domain. Composite elements can themselves aggregate other composite elements.

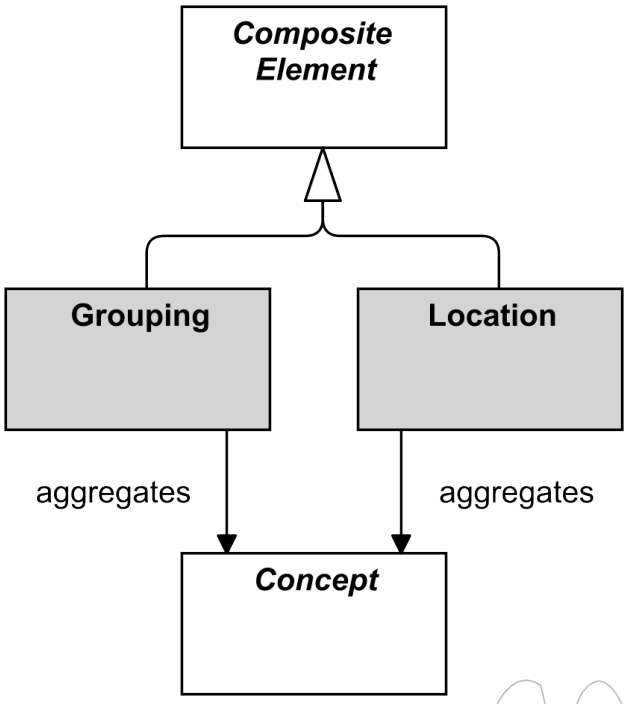


Figure 4-9: Composite Elements

4.3.1. Grouping

A grouping represents a collection of concepts that belong together based on some common characteristic.

The grouping element is used to aggregate an arbitrary group of concepts, which can be elements and/or relationships of the same or of different types. An aggregation relationship is used to link the grouping element to the grouped concepts. Grouping elements can also have other relationships to and from them, as shown in Appendix B.

Figure 4-10: Grouping Notation

Concepts may be aggregated by multiple (overlapping) groups.

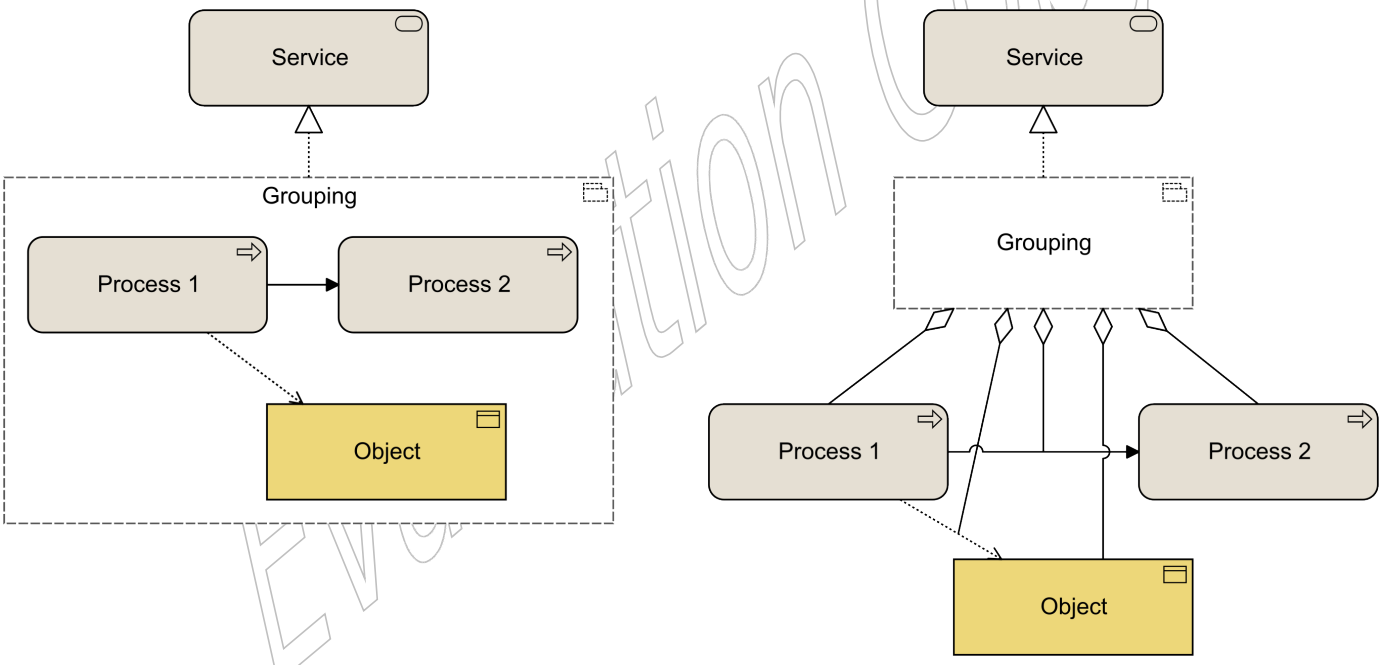
One useful way of employing grouping is for modeling ABBs and SBBs, as described in the TOGAF Standard [C220].

Another useful application of grouping is for modeling domains. For example, the TOGAF Standard [C220] Glossary of Supplementary Definition 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"*.

NOTE The use of grouping is not to be confused with creating views on the architecture (see Section 13.3). Although, like a view, it comprises concepts that belong together for some reason, it does not provide a separate visualization of these concepts. Moreover, groupings are used *within* architecture views to provide additional structure to an architecture model and its visualization.

Example

In Example 4-2, 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 4-2: Grouping

NOTE The semantics of grouping imply that a relationship from or to a group should be interpreted as a collective relationship with the group's contents. In the example, the implied meaning is that the contents of the group together, or parts thereof, realize the service. However, this is not always easily expressed in simple derivable relationships.

4.3.2. Location

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

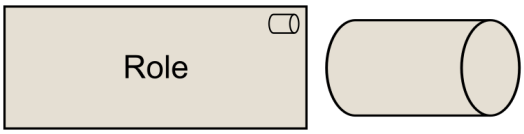
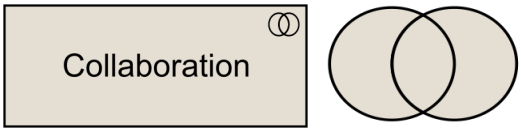
The location element is used to model the place where (active and passive) structure elements such as business actors, application components, and devices are located. This covers both physical (e.g., geographical) locations and logical places such as the regions of cloud providers. This is modeled by 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 [Sowa & Jackman, 1992].

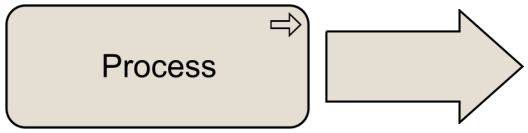
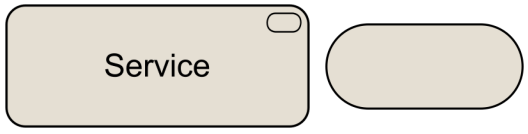
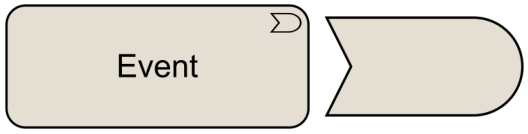
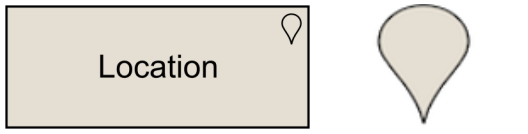
Figure 4-11: Location Notation

4.4. Summary of Common Domain Elements

Table 4-1 gives an overview of the common core elements, their definitions, and their default graphical notation.

Table 4-1: Common Core Elements

Element	Definition	Notation
Active Structure		
Role	The position or purpose that a business actor, application component, node, device, system software, equipment, facility, or a collaboration has in performing specific behavior.	
Collaboration	A collection of business actors, application components, nodes, devices, system software, equipment, facilities, roles, and/or other collaborations that work together to perform collective behavior.	

Element	Definition	Notation
Path	A logical link between business actors, application components, nodes, devices, system software, equipment, facilities, roles, and/or collaborations, through which these elements can exchange data, energy, or material.	
Behavior		
Process	A sequence of behaviors that achieves a specific result.	
Function	A collection of behavior based on a chosen set of criteria.	
Service	An explicitly defined behavior that a business actor, application component, node, device, system software, equipment, facility, role, or collaboration provides to its environment.	
Event	Something that happens or occurs inside or outside the enterprise, such as a state change.	
Composite		
Grouping	A collection of concepts that belong together based on some common characteristic.	
Location	A place or position where structure elements can be located or behavior can be performed.	

Chapter 5. Relationships and Junctions

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 5-1):

- *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

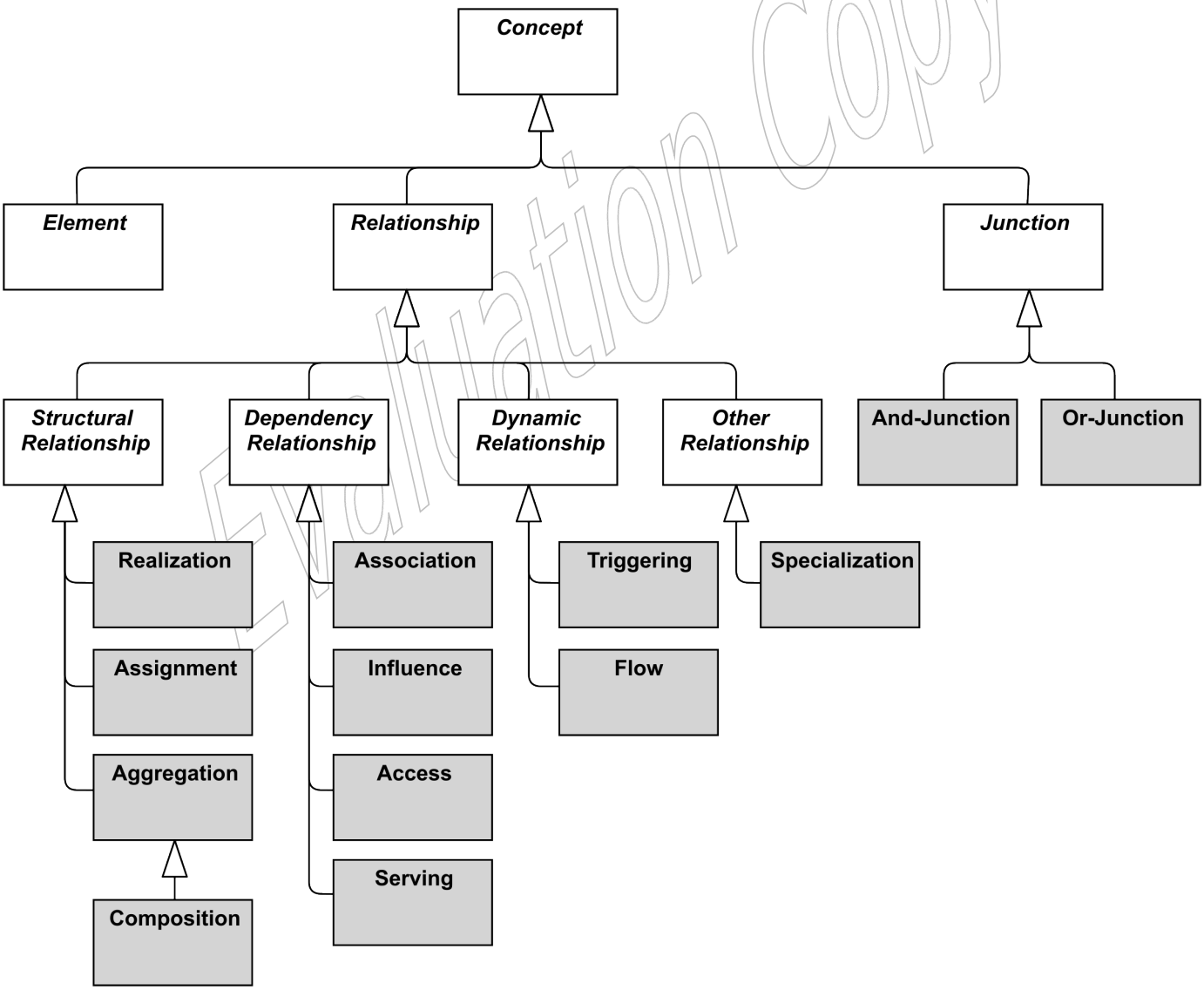


Figure 5-1: Overview of Relationships

Each relationship has exactly one “from” and one “to” concept (element, relationship, or junction) as endpoints. The following restrictions apply:

- No relationships are allowed between two relationships
- All relationships connected with the same junction must be of the same type
- A chain of relationships of the same type that connects two elements, and is in turn connected via junctions, is valid only if a direct relationship of that same type between those two elements is valid
- A relationship connecting an element with a second relationship can only be an aggregation or association; aggregation is valid only from a composite element to that second relationship

It is good practice to explicitly name or label any relationship that would otherwise be ambiguous or misunderstood.

For the sake of readability, the metamodel figures throughout this document do not show all possible relationships in the language. Section 5.8 describes a set of derivation rules to derive indirect relationships between elements in a model. Aggregation 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.

As an alternative to the graphical notations proposed in this section, structural relationships may also be expressed by nesting the united concept within the uniting element. Note, however, that this can lead to ambiguous views (although unambiguous in the model), in case multiple structural relationships are allowed between these elements.

5.1.1. Aggregation Relationship

An aggregation relationship represents that an element combines one or more other concepts.

The aggregation relationship has been inspired by the aggregation relationship in UML class diagrams.

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

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

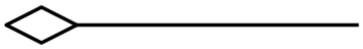
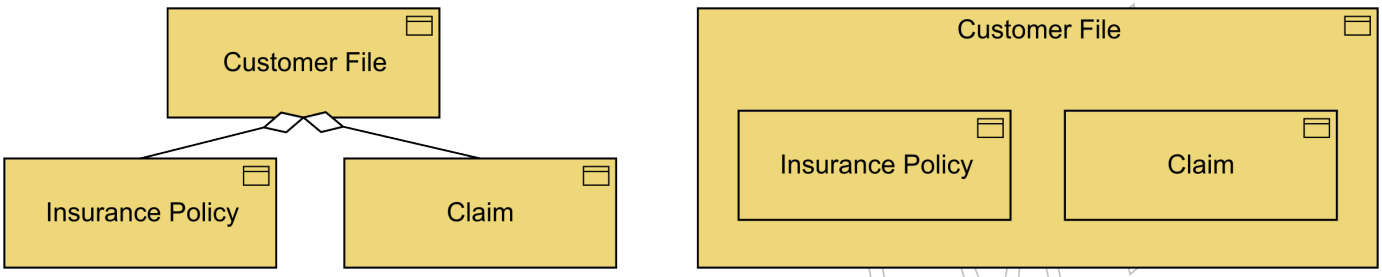


Figure 5-2: Aggregation Notation

The interpretation of an aggregation relationship is that the *whole or part* of the source element aggregates the *whole of* the target concept. See also Section 5.1.5.

Example

Example 5-1 shows two ways to express that the “Customer File” aggregates an “Insurance Policy” and “Claim”.



Example 5-1: Aggregation

5.1.2. Composition Relationship

A composition relationship represents that an element combines one or more other concepts, where the existence of those concepts depends on that of the element.

The composition relationship is a specialization of the aggregation relationship that expresses an existence dependency: If a composite is deleted, its parts are deleted as well. When you model real-world elements (for example, an organization structure of departments and teams expressed as business actors), this dependency applies to these elements themselves. When you model exemplars or categories — as is common in Enterprise Architecture — this dependency may be interpreted as applying to their real-world instances. For example, a specific kind of server can be modeled as a node composed of a device and system software; this implies an existence dependency between individual servers of that kind and the individual devices and system software instances of which they consist.

A composition relationship is allowed in exactly the same cases where an aggregation relationship would be allowed. It follows the same semantics as aggregation relationships. See also Section 5.1.5.

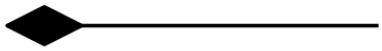
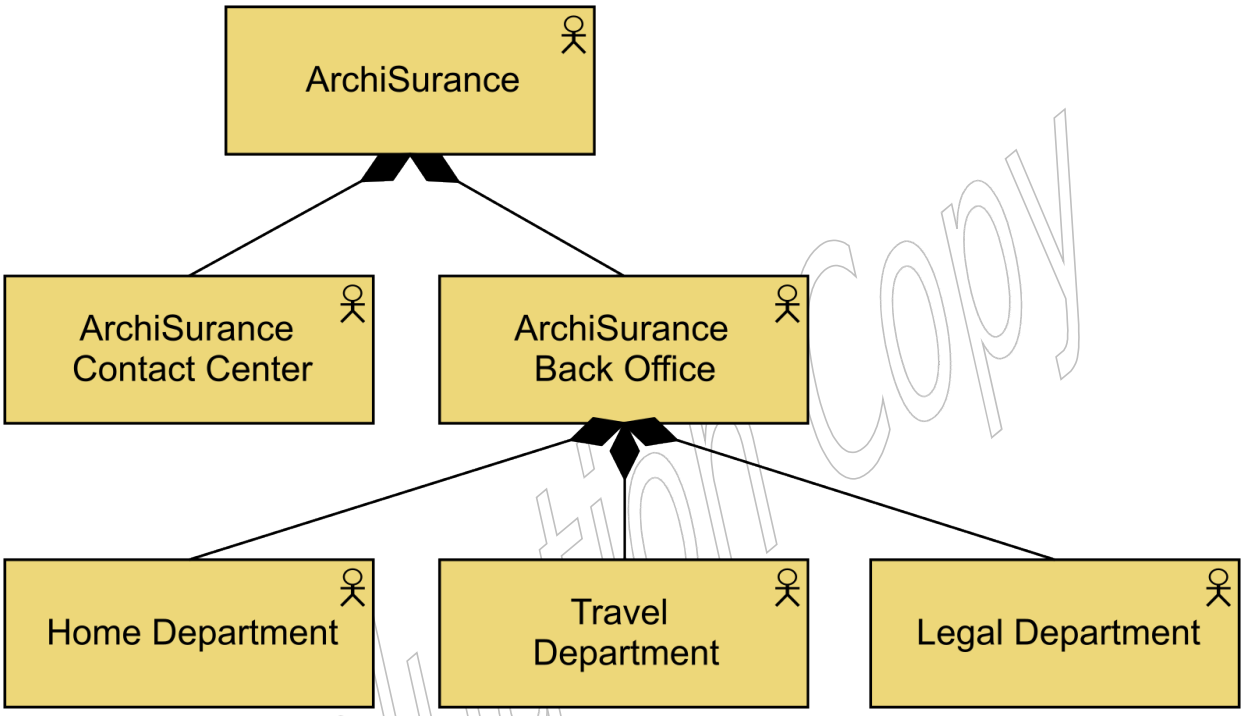


Figure 5-3: Composition Notation

Example

Example 5-2 shows that the company “ArchiSurance” consists of “ArchiSurance Contact Center” and “ArchiSurance Back Office”, which in turn consists of three departments.



Example 5-2: Composition

5.1.3. Assignment Relationship

An assignment relationship represents the allocation of responsibility, performance of behavior, storage, or execution.

The assignment relationship links active structure elements with units of behavior that are performed by them, internal active structure elements (e.g., business actors) with roles fulfilled by them, and active structure elements with passive structure elements from the same domain. It can, for example, relate an internal active structure element with an internal behavior element, an interface with a service, or a node, device, and system software with an artifact. The full set of permitted relationships is listed in Appendix B.

Figure 5-4: Assignment Notation

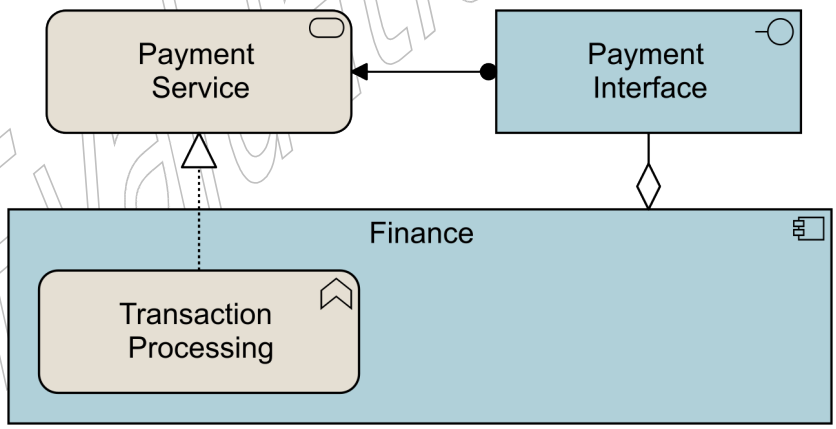
In the ArchiMate language described in Section 3.2, it always points from active structure to behavior, from behavior to passive structure, and from active to passive structure.

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 role inside the business actor performing that role, a function inside an application component executing that function, or an artifact inside a node that stores it.

The interpretation of an assignment relationship is that the *whole or part* of the source element is assigned the *whole of* the target element (see also Section 5.1). 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 5.5) 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

Example 5-3 includes the two ways to express the assignment relationship. The “Finance” application component is assigned to the “Transaction Processing” function, and the “Payment Interface” is assigned to the “Payment Service”.



Example 5-3: Assignment

5.1.4. Realization Relationship

A realization relationship represents that an element plays a critical role in the creation, achievement, sustenance, or operation of a more abstract element.

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

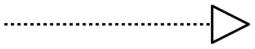


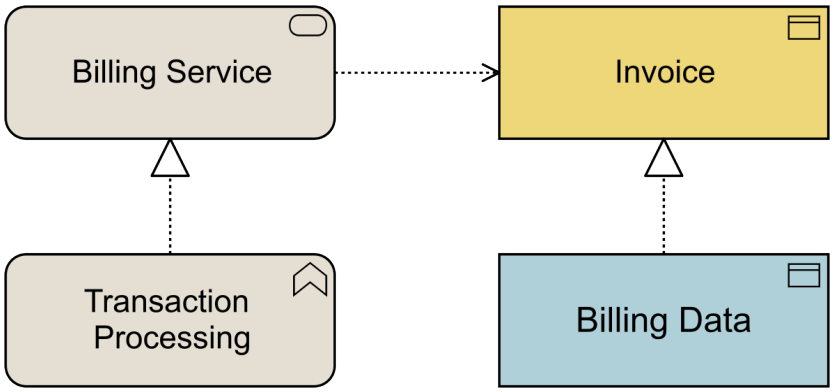
Figure 5-5: Realization Notation

The interpretation of a realization relationship is that the *whole or part* of the source element realizes the *whole of* the target element (see also Section 5.1). For example, this means that if 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 an *And*-junction (see Section 5.5.1) can be used. For weaker types of effects on the realization of a motivation element, the influence relationship (see Section 5.2.3) should be used.

For realization, nesting as a notation is allowed in both directions of the relationship. For instance, a service can be nested in an application component to express that it is realized by that application component, and a process nested in a capability may express that this process realizes the capability. Since this can lead to ambiguity, it is advisable to use it only in one direction in the same view and add an explanation to any view in which such ambiguity may arise.

Example

Example 5-4 illustrates two ways to use the realization relationship. A "Transaction Processing" function realizes a "Billing Service"; the "Invoice" business object is realized by the data object "Billing Data".



Example 5-4: Realization

5.1.5. Semantics of Structural Relationships

Structural relationships describe that the element on the source side contains, groups, performs, or realizes the concept on the target side of the relationship. Structural relationships can be transitively applied to (possibly unmodeled) parts of the source element.

Below are some examples of how these semantics work:

- Aggregation relationships from parts also apply to the whole

  For example, if a part of A aggregates B, A itself is also considered to aggregate B. Conversely, if A aggregates B, that can be interpreted as some part of A aggregating B.

- An assignment relationship from an active structure element to a behavior element may also apply to parts of this active structure element

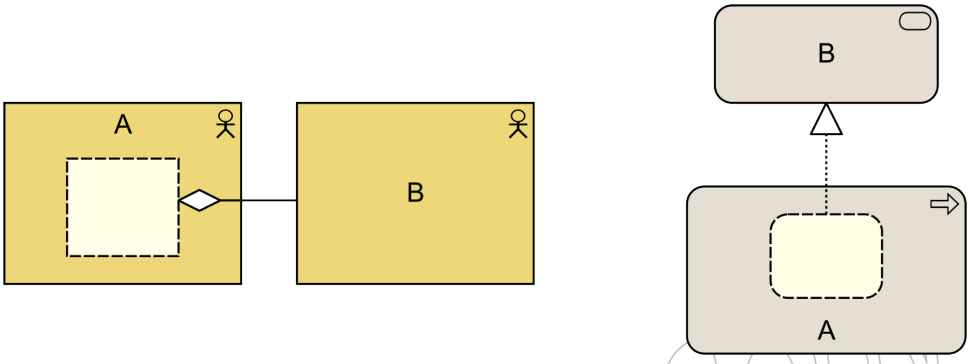
  For example, if Role A is assigned to Process B, some part of A may perform B. Conversely, if a part of A is assigned to B, A itself is also considered to be assigned to B.

- A realization relationship from an internal behavior element to a service may also apply to parts of that internal behavior element

  For example, if a Service B is realized by a Process A, B may be realized by some part of A. Conversely, if a part of A realizes B, A itself is also considered to realize B.

Example

In the left-hand side of Example 5-5, the entire Business Actor B (possibly a department) is aggregated in Business Actor A (possibly a division), via some unmodeled element inside A. In the example on the right, Process A completely realizes Service B, via some unmodeled element inside A.



Example 5-5: Semantics of Structural Relationships

5.2. Dependency Relationships

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

- A *serving* relationship represents a *control* dependency, denoted by a solid line
- An *access* relationship represents a *data* dependency, denoted by a dotted line
- An *influence* relationship represents an *impact* dependency, denoted by a dashed line
- An *association* relationship represents a dependency not covered by any of the other relationships

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. As you can see in Example 11-1, most of the arrows that represent such realizing, supporting, influencing, or serving dependencies point "upwards" toward the client/user/business. Another reason for this direction, particularly 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 (see 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

A serving relationship represents that an element provides its functionality to another element.

A 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 to both the behavior aspect and the active structure aspect.

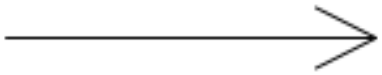
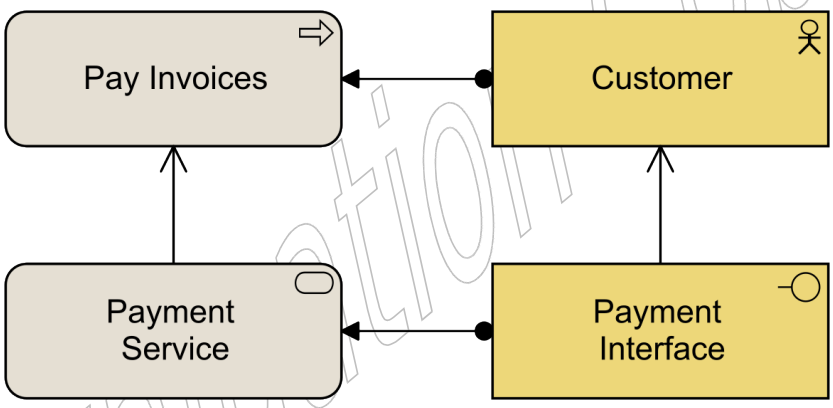


Figure 5-6: Serving Notation

Example

Example 5-6 illustrates the serving relationship. The “Payment Interface” serves the “Customer”, while the “Payment Service” serves the “Pay Invoices” process of that customer.



Example 5-6: Serving

5.2.2. Access Relationship

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

An access relationship indicates that a process, function, service, or event “does something” with a passive structure element; e.g., create, read, modify or delete a business or data object or an artifact; or produce, consume, move, transform, or destroy material. The relationship can also be used to indicate that the passive structure element . 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 arrowhead, if present, indicates the creation, change, or usage of passive structure elements. 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” or “usage” access, and in both directions to denote “read-write” or “change” access. Care must be taken when using access with derived relationships because the arrow on the relationship has no bearing to its directionality.

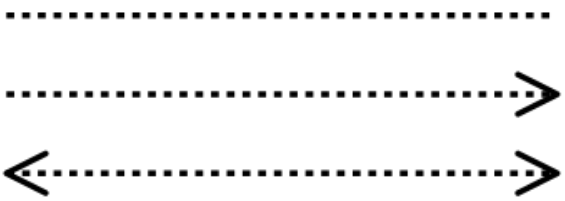
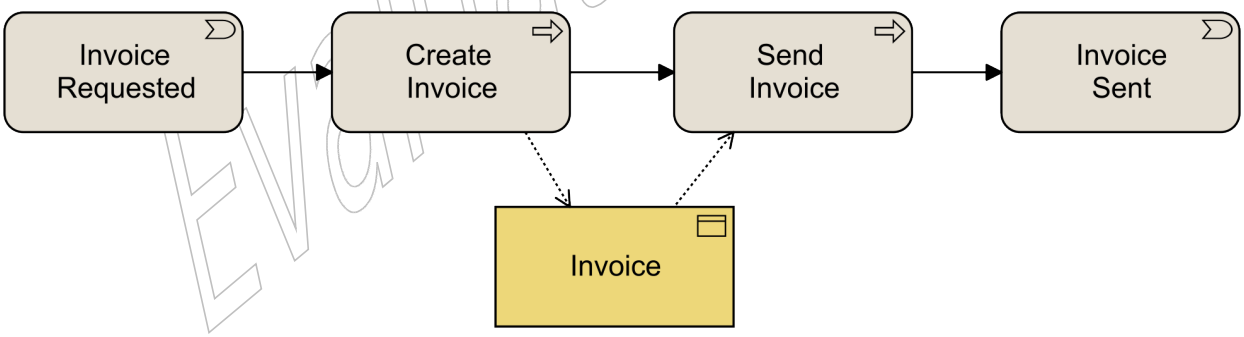


Figure 5-7: Access Notation

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.

Example

Example 5-7 illustrates the access relationship. The “Create Invoice” process writes/creates the “Invoice” business object; the “Send Invoice” process reads that object.



Example 5-7: Access

5.2.3. Influence Relationship

An influence relationship represents that an element affects the implementation or achievement of some motivation element.

An influence relationship is used to describe some architectural elements that influence the 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 to make 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 depending on the degree to which the other element is satisfied itself. For example, the degree to which a 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. 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. The influence relationship should be used to represent relationships that are not critical to the target element’s existence or realization. For example, a business actor representing a construction crew may realize the goal of constructing a building, and a requirement to add additional (skilled) workers to an already adequate crew may influence the goal of constructing the building. However, the business actor also realizes an additional goal of opening the building by a particular date. 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
- 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.

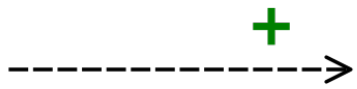
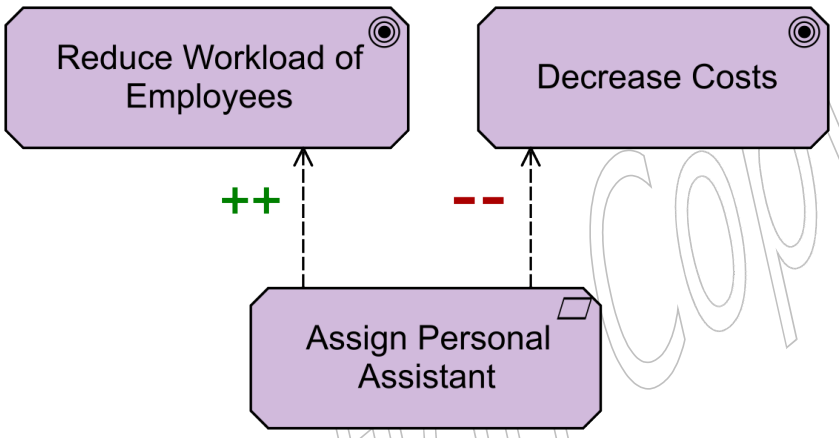


Figure 5-8: Influence Notation

Example

Example 5-8 illustrates the use of the influence relationship to model the different effects of the same requirement, “Assign Personal Assistant”. This has a strongly positive influence on “Reduce Workload Of Employees”, but a strongly negative influence on “Decrease Costs”.



Example 5-8: Influence

5.2.4. Association Relationship

An association relationship represents 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. An association is undirected or directed. See also Section 5.2.5.

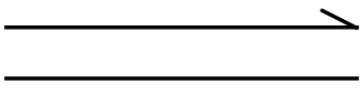
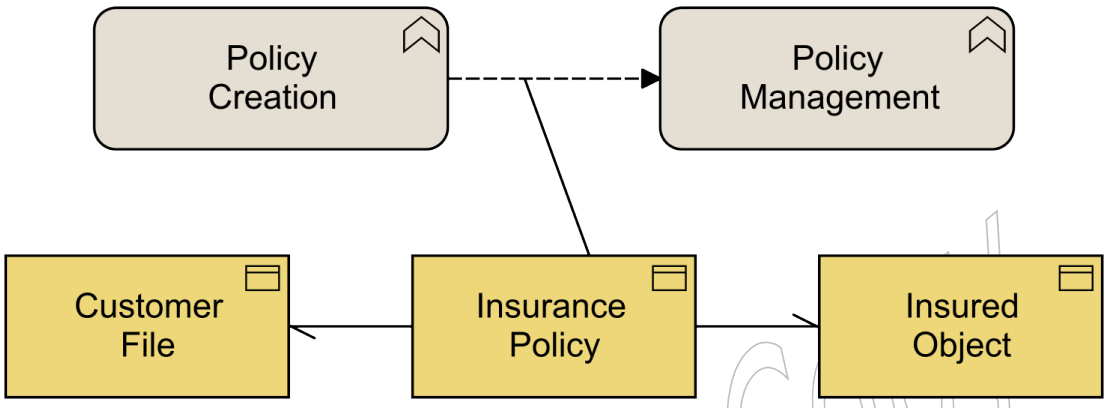


Figure 5-9: Association Notation

Example

Example 5-9 illustrates two directed association relationships between a business object "Insurance Policy" and two business objects to which this business object refers. It also shows an association between a flow relationship and this "Insurance Policy", to indicate that the business object is transferred from "Policy Creation" to "Policy Management".



Example 5-9: Association

5.2.5. Semantics of Dependency Relationships

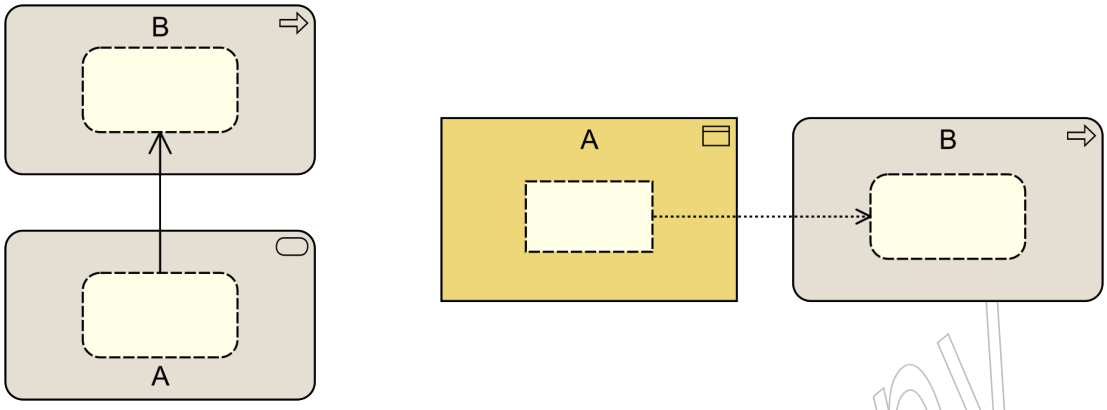
Dependency relationships describe that a part of the target element has a dependency on a part of the source element. Although there is a dependency between the two elements, it does not necessarily mean this applies to all of the parts of the element as defined by any structural relationships.

This semantic allows you to model dependencies at a high level (with details removed) without implying specific dependencies at a more detailed level. This means, for example, that:

- In serving relationships, some part of an internal behavior element is served by some part of a service; for example, if a service A serves a process B, some unmodeled sub-service of A may serve an unmodeled sub-process of B
- In access relationships, some part of a behavior element accesses some part of a passive structure element; for example, if a function A accesses a data object B, some unmodeled sub-function of A may access an unmodeled part of B
- In influence relationships, some part of a core element influences some part of a motivational element; for example, if an application component A influences a requirement B, some unmodeled part of A may influence some unmodeled part of B
- In association relationships, some part of an element is related to some part of another element; if it is directed, it can only be used in derivations in that direction (see Section 5.8)

Example

In the left-hand side of Example 5-10, a part of Process B is served by a part of Service A. In the right-hand example, a part of Process B accesses (reads) a part of Business Object A.



Example 5-10: Semantics of Dependency Relationships

5.3. Dynamic Relationships

While structural and dependency relationships are used to model the architecture of a system “at rest”, dynamic relationships describe the architecture of a system “in motion”. Two types of dynamic relationships are distinguished: *triggering* and *flow*.

5.3.1. Triggering Relationship

The triggering relationship represents 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 interpretation of a triggering relationship is that some part of the source element should be completed before the target element can start (see also Section 5.3.3). Note that this does not necessarily mean that one behavior element actively starts another; in a hypothetical daily process, the “Have Lunch” activity follows (is triggered by) the “Have Breakfast” activity, even though there is no active signal between the two.

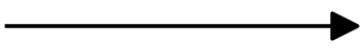
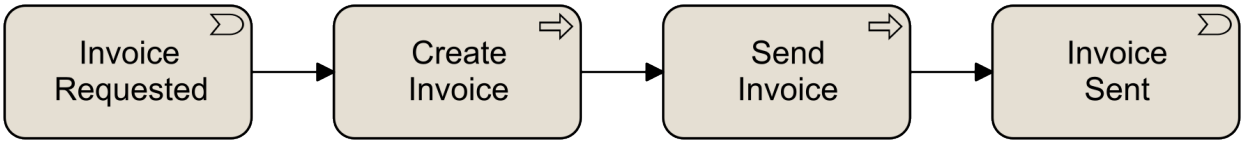


Figure 5-10: Triggering Notation

Example

Example 5-11 illustrates that triggering relationships are used to model causal dependencies between processes and/or events.



Example 5-11: 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.

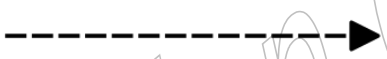
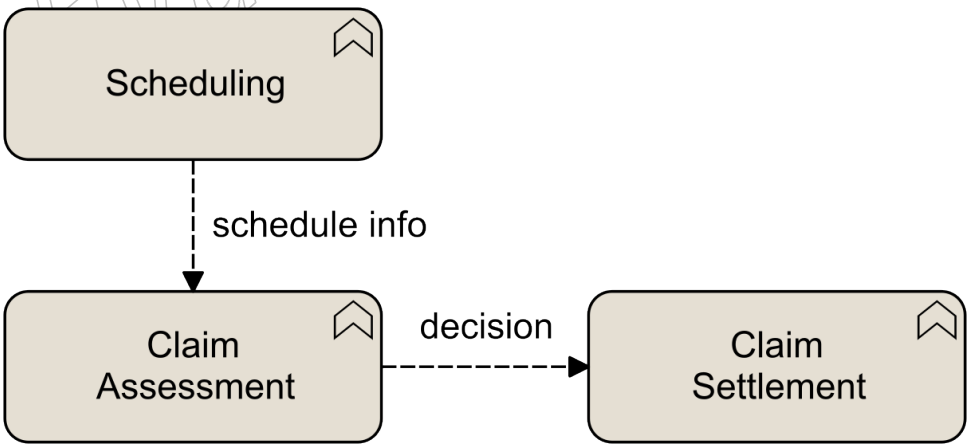


Figure 5-11: Flow Notation

Example

Example 5-12 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 5-12: Flow

5.3.3. Semantics of Dynamic Relationships

The semantics of triggering and flow relationships differ. The triggering relationship follows the same semantics as structural relationships (see Section 5.1.5). A triggering relationship from A to B indicates that everything in B is preceded by a part of A. When A and B are processes, it means that all steps in process B are performed after a part of A has occurred; but some steps in A can occur after some or all steps in B have occurred.

A stronger interpretation of triggering (everything in B is preceded by everything in A) could be imposed on the ArchiMate model by a modeling group wishing to do so.

The flow relationships follow the same semantics as dependency relationships (see Section 5.2.5). A flow relationship from A to B indicates that the whole or some part of A transfers something (e.g., information) to the whole or some part of B.

5.4. Other Relationships

5.4.1. Specialization Relationship

> The specialization relationship represents 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.

A specialization relationship is always allowed between two elements of the same type.

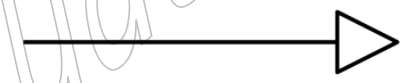
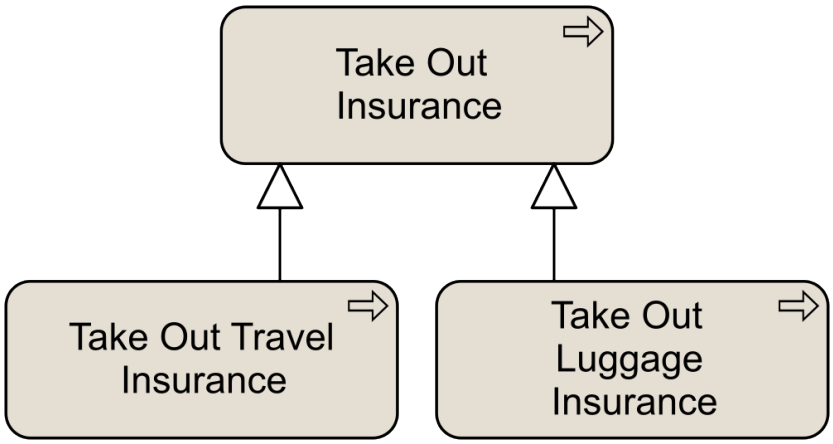


Figure 5-12: Specialization Notation

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

Example

Example 5-13 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 5-13: Specialization

5.4.2. Semantics of Other Relationships

The semantics of the specialization relationship are that the whole of the generic element is specialized by the specialized element.

5.5. Junctions

5.5.1. Junction

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

> A junction connects two or more relationships of the same type.

A chain of junctions and relationships of a specific type is only allowed between two concepts if a direct relationship of that type between these concepts is also permitted. Simply put, you cannot use junctions to create relationships between concepts that would otherwise not be allowed.

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 contiguous junctions).

A junction is used to explicitly express that all elements together must participate in the relationship (and-junction) or that at least one of the elements participates in the relationship (or-junction). The or-junction can be used to express both inclusive and exclusive or conditions, which could be indicated by a modeler by naming the junction to reflect its type.

The default interpretation of incoming relationships without a junction is (inclusive) *or*, and for outgoing relationships it is *and*. Even though it is not strictly necessary, using junctions in such a case can still be useful for reasons of clarity or to stress the importance of a combination of relationships.

In addition to the above, a junction (which connects some relationships), may also be aggregated in a plateau, grouping, or location element. If that junction connects other aggregation relationships, it should be interpreted without the aggregation to this containing element. The aggregation merely states that the junction is part of a plateau, grouping, or location. Without that relationship, the junction must fulfill the same conditions as above: it connects relationships of the same type, with at least one incoming and one outgoing relationship.

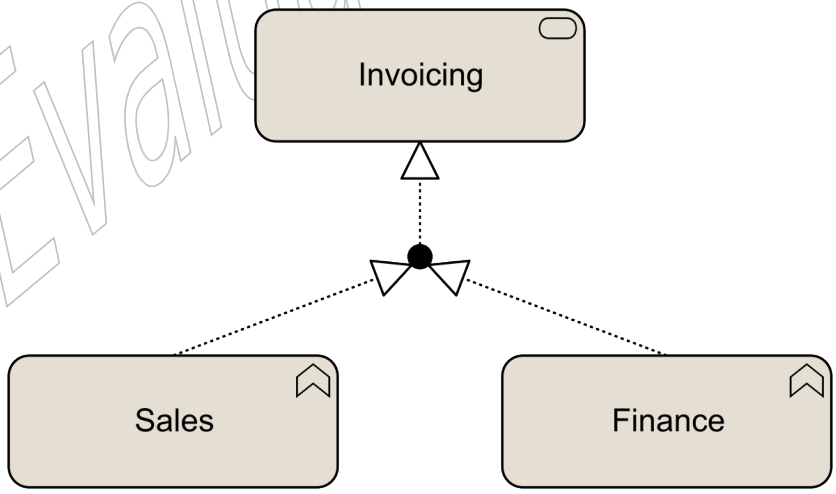
It is allowed to omit arrowheads of relationships leading into a junction.

Figure 5-13: Junction Notation

Junctions may be used on triggering relationships. This is a technique used by other modeling languages. For example, the BPMN notation uses gateways as junctions, and the UML notation uses forks and joins in their modeling 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 have 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.

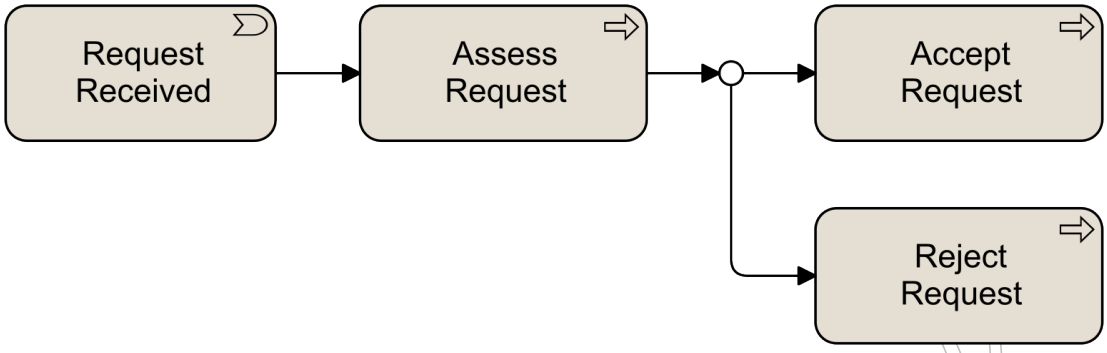
Examples

In Example 5-14, the and-junction in the model is used to denote that the “Sales” and “Finance” functions together realize the “Invoicing” service.



Example 5-14: And-Junction

In Example 5-15, 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 5-15: Or-Junction

5.6. Multiplicity

Even though the ArchiMate language intentionally does not support a difference between types and instances (this is explained in more detail in Section 3.5), we can express the constraints put on the instances of elements.

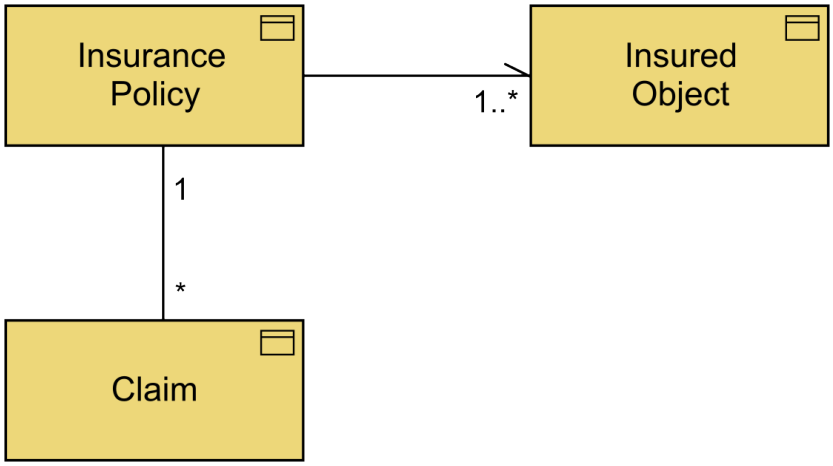
All relationships can have on each end (except when connected to a junction) an attribute specifying how many instances are allowed at that end. The notation for this attribute can be:

- A single number *n* being a positive integer, meaning that exactly *n* instances are required on that end
- * (equivalent to 0..*), meaning that this end is unbounded
- *n..m* where *n* and *m* are non-negative integers and *m* is greater than *n*, meaning that any number of instances from *n* to *m* are allowed on that end

Using this notation, it is possible to indicate if the participation of an element in the relationship is optional (multiplicity on its end set to * or 0..m) or mandatory (multiplicity on its end set to n or n..m, with n being a positive integer).

Example

In Example 5-16, an “Insurance Policy” refers to at least one “Insured Object”. A “Claim” is related to exactly one “Insurance Policy” and zero or more Claims may be related to that Insurance Policy.



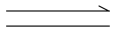
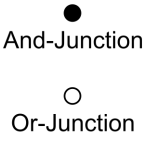
Example 5-16: Multiplicity

5.7. Summary of Relationships and Junctions

Table 5-1 gives an overview of the ArchiMate relationships and junctions with their definitions.

Table 5-1: Relationships and Junctions

Relationships	Definition	Notation	Role Names
Structural Relationships			
Aggregation	Represents that an element combines one or more other concepts.		→ aggregates ← aggregated in
Composition	Represents that an element combines one or more other concepts, where the existence of those concepts depends on that of the element.		→ composed of ← composed in
Assignment	Represents the allocation of responsibility, performance of behavior, storage, or execution.		→ assigned to ← has assigned
Realization	Represents that an element plays a critical role in the creation, achievement, sustenance, or operation of a more abstract element.		→ realizes ← realized by
Dependency Relationships			
Serving	Represents that an element provides its functionality to another element.		→ serves ← served by
Access	Represents the ability of behavior and active structure elements to observe or act upon passive structure elements.		→ accesses ← accessed by
Influence	Represents that an element affects the implementation or achievement of some motivation element.	+	→ influences ← influenced by

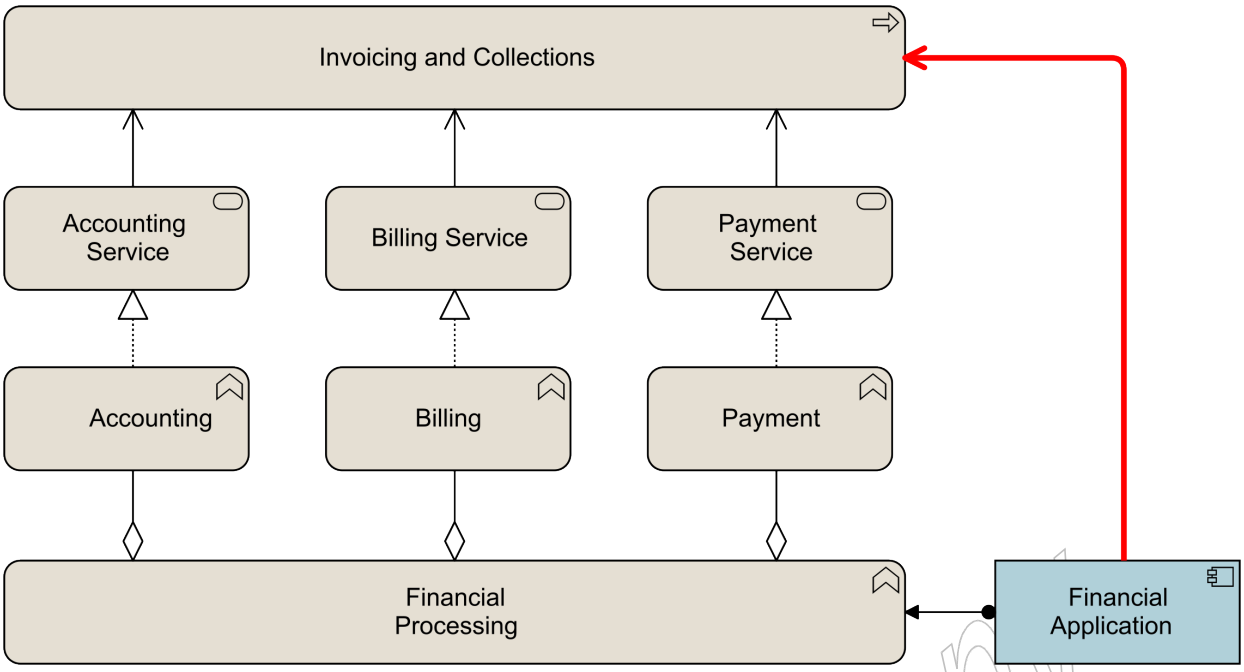
Relationships	Definition	Notation	Role Names
Association	Represents an unspecified relationship, or one that is not represented by another ArchiMate relationship.		associated with → associated to ← associated from
Dynamic Relationships			
Triggering	Represents a temporal or causal relationship between elements.		→ triggers ← triggered by
Flow	Represents transfer from one element to another.		→ flows to ← flows from
Other Relationships			
Specialization	Represents that an element is a particular kind of another element.		→ specializes ← specialized by
Junctions			
Junction	Used to connect relationships of the same type.		

5.8. Derivation of Relationships

In the ArchiMate language, you can derive indirect relationships between elements in a model, based on the modeled relationships. This allows for the abstraction of irrelevant intermediary elements to present a specific model or view of the architecture that supports impact analysis. The precise rules for making such derivations are specified in Appendix B.

Example

In Example 5-17, assume that the goal is to abstract from the functions, sub-functions, and services in the model. In this case, an indirect serving relationship (thick red arrow on the right) can be derived from “Financial Application” to the “Invoicing and Collections” process (from the chain assignment — aggregation — realization — serving).



Example 5-17: Derivation from a Chain of Relationships

Derivation of relationships is intended as a way to create summaries of detailed models. It is a way to remove (to abstract from) details in a model while still making valid "statements". Hence, derivation is always meant to go from more detail to less detail. This mechanism is one of the unique properties of the ArchiMate language compared to other modeling languages.

The language allows the modeler to directly create relationships that are necessarily valid derived relationships without the constituents of the derivation being available in the model. These relationships (for example, a realization relationship between an application component and an application service) assume that the required constituents (for example, a function) needed for the derived relationship exist; however, these missing elements need not be modeled explicitly, and the derived relationships can be used as if they have not been derived. Thus, the modeler has full freedom in choosing the required level of detail.

Because the essence of derivation is to make simplifications or summaries, it cannot be used to infer more detail. Therefore, a realization relationship from an application component to an application service can be modeled, but from it, no conclusions can be drawn about the exact source of this derivation (e.g., which functions realize which services).

This is information that should be added by a modeler during the design process: a higher-level, more abstract model can be refined by elaborating the derived relationships (in the previous example by adding a function that realizes the application service and to which the application component is assigned).

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

Chapter 6. Motivation Domain

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

6.1. 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 [Sowa & Jackman, 1992].

Several *motivation elements* are included in the language: stakeholder, value, meaning, driver, assessment, goal, outcome, principle, and requirement. The motivation elements address the way the Enterprise Architecture is aligned to its context, as described by these intentions.

A motivation element represents the context of or reason behind the architecture of an enterprise.

6.2. Motivation Elements Metamodel

Figure 6-1 gives an overview of the motivation elements and their relationships.

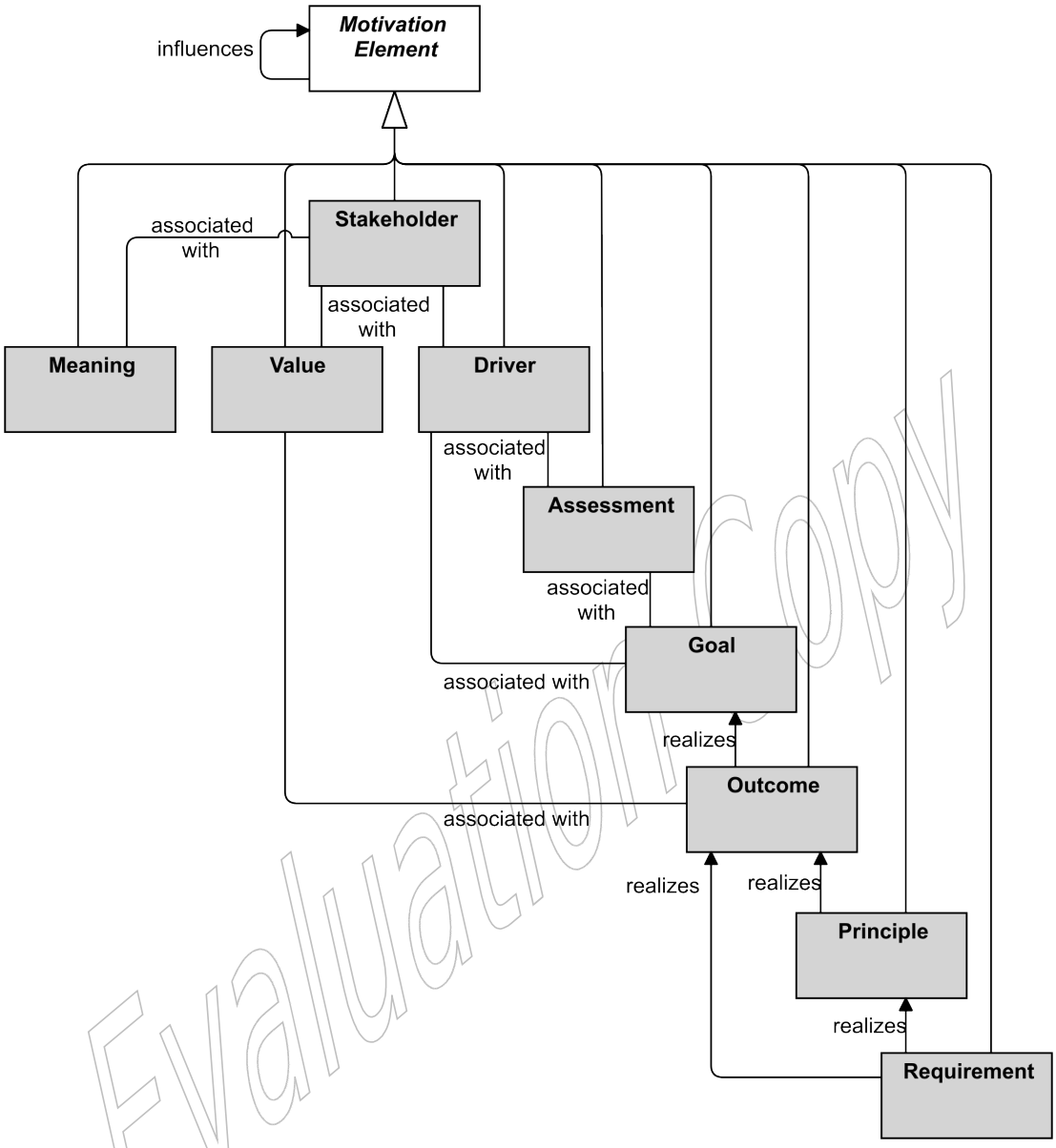


Figure 6-1: Motivation Elements Metamodel

6.3. Stakeholder, Driver, and Assessment

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. A *stakeholder* can be an individual or a group of people, such as a project team, enterprise, or society. Examples of 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 Strengths, Weaknesses, Opportunities, and Threats (SWOT) analysis, in order to respond in the best way.

6.3.1. Stakeholder

A stakeholder represents the perspective from which a business actor perceives the effects of the architecture.

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 Chief Executive Officer (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.

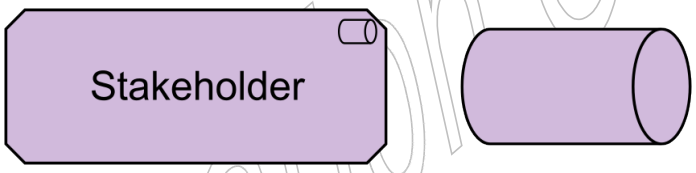


Figure 6-2: Stakeholder Notation

6.3.2. 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 that are associated with a stakeholder are often called “concerns” of that stakeholder. Stakeholder concerns are defined in the TOGAF Standard [C220] as “*an interest in a system relevant to one or more of its stakeholders. Concerns may pertain to any aspect of the system’s functioning, development, or operation, including considerations such as performance, reliability, security, distribution, and evolvability and may determine the acceptability of the system.*” Drivers of change may be external to the enterprise (e.g., economic changes or changing legislation) and need not have a stakeholder associated with them. The name of a driver should preferably be a noun.

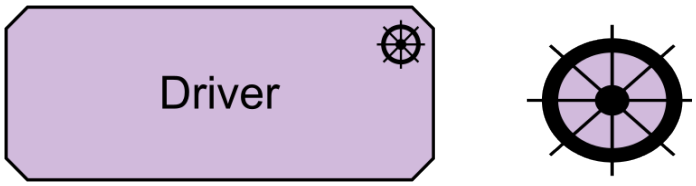


Figure 6-3: Driver Notation

6.3.3. 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.

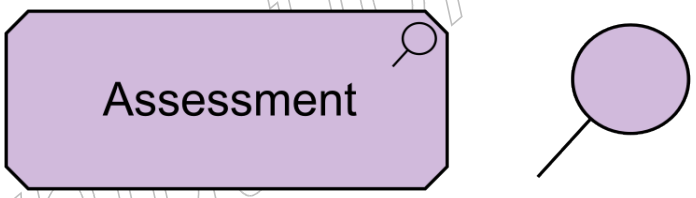
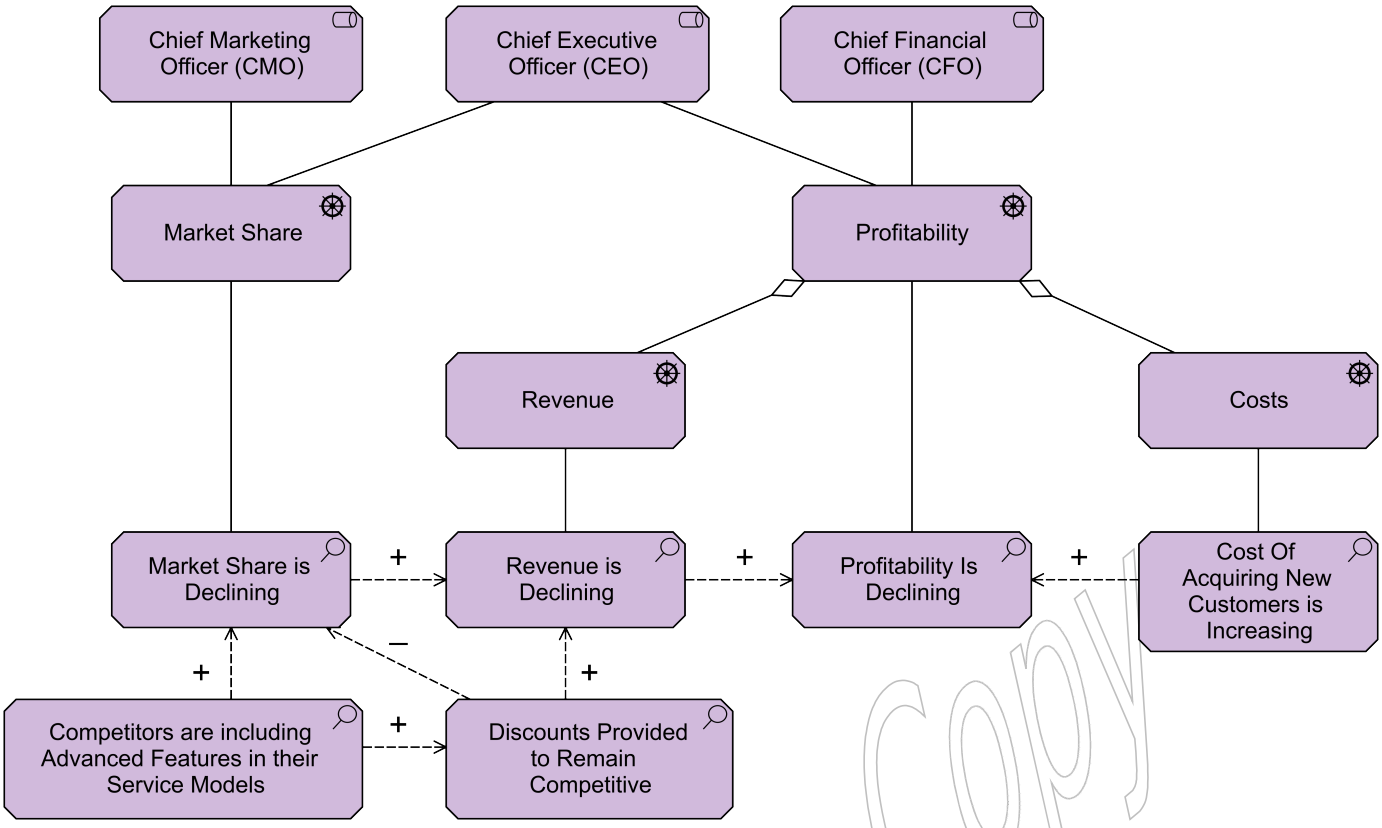


Figure 6-4: Assessment Notation

6.3.4. 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" consists 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 6-1: Stakeholder, Driver, and Assessment

6.4. Goal, Outcome, Principle, and Requirement

The motivation of an organization or individual to achieve certain results is represented by goals, principles, and requirements. *Goals* represent that a stakeholder wants to realize a certain outcome; e.g., “Increase customer satisfaction by 10%”. The end results achieved in the pursuit of goals are outcomes, which may or may not indicate the realization of such goals. When applying a capability-based planning approach, capabilities may realize outcomes that realize the intended goals. 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 Customer Relationship Management (CRM) system” conforms to the aforementioned principle by applying it to the current organization’s architecture in the context of CRM and the management of customer data.

6.4.1. 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 any objective a stakeholder may desire, such as a state of affairs, or a produced value. Examples of goals include: 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.

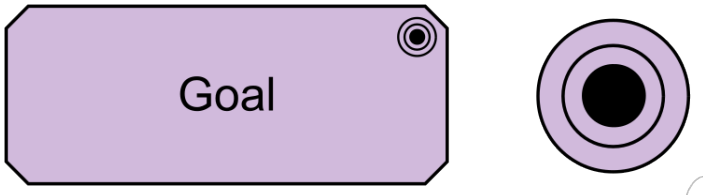


Figure 6-5: Goal Notation

6.4.2. Outcome

An outcome represents an end result, effect, or consequence of a certain state of affairs.

Outcomes are high-level, business-oriented results produced by the 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. The distinction between goals and outcomes is important. Simply put, a goal is what you want, and an outcome is what you get. Outcomes are the end results, and goals or requirements are often related to outcomes that should be realized. Capabilities can be designed to achieve such outcomes.

However, not all outcomes relate to the successful achievement of goals. When modeling a future state, an outcome models some result or effect that is expected to have been achieved or occur at that future point in time. Unlike goals, outcomes can also be used to model potentially unwanted effects; for example, in order to design appropriate mitigating measures.

Outcome names should unambiguously identify end results that have been achieved or are expected to be achieved at a definite point in the future. Examples include “First-place customer satisfaction ranking achieved” and “Key supplier partnerships in place”. Outcome names can also be more specific; e.g., “10% year-over-year quarterly profits increase in 2018”.

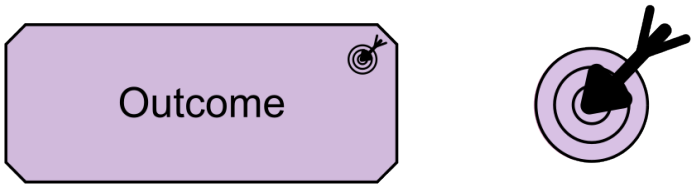


Figure 6-6: Outcome Notation

6.4.3. Principle

A principle represents a statement of intent defining a general property that applies to any system in a certain context in the architecture.

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 structure element, behavior element, or passive structure element of some organization, such as a business actor, application component, process, service, business object, or data object.

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, whereas a requirement defines a property that applies to a specific system as described by an architecture. 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.

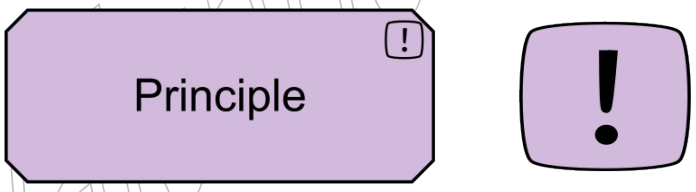


Figure 6-7: Principle Notation

6.4.4. Requirement

A requirement represents a statement of need defining a property that applies to a specific system as described 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.

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, two alternative requirements may be identified to realize the goal “Improve portfolio management”:

- By assigning a personal assistant to each customer
- 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.

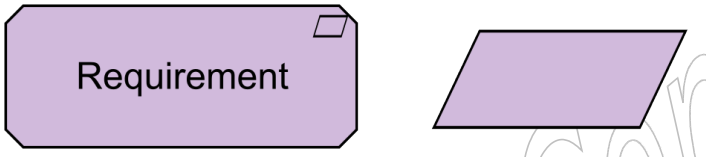
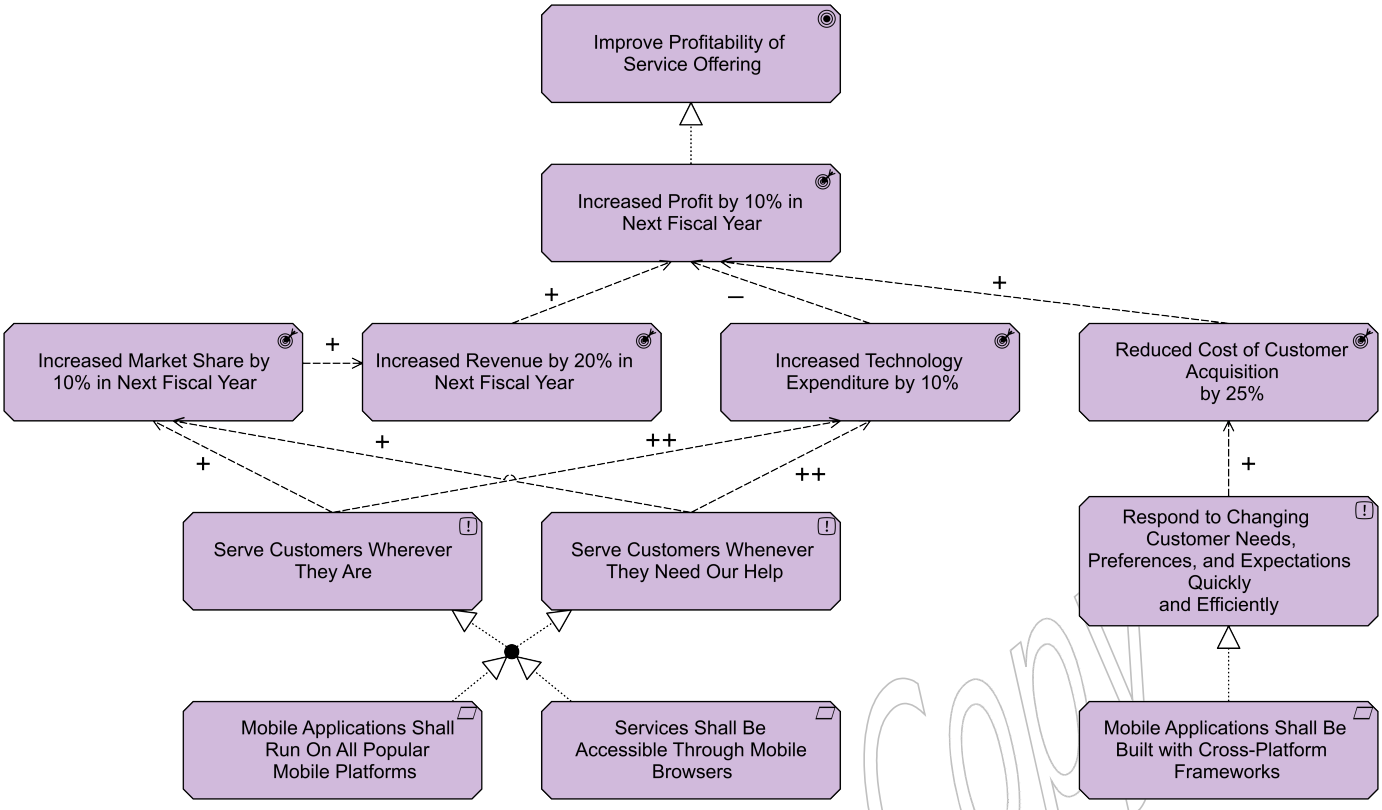


Figure 6-8: Requirement Notation

6.4.5. Example

The goal “Improve Profitability of Service Offering” is realized by the outcome “Increased Profit by 10% in Next Fiscal Year”. This outcome is influenced positively by the outcomes “Increased Revenue by 20% in Next Fiscal Year” and “Reduced Cost of Customer Acquisition by 25%”. The outcome “Increased Revenue by 20% in Next Fiscal Year” is influenced positively by an outcome “Increased Market Share by 10% in Next Fiscal Year”. There is also a negative outcome: “Increased Technology Expenditure by 10%”. 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 outcome “Reduced Cost Of Customer Acquisition by 25%” is realized by a principle “Respond To Changing Customer Needs, Preferences, and Expectations Quickly and Efficiently”, which in turn is realized by a requirement “Mobile Applications Shall Be Built With Cross-Platform Frameworks”.



Example 6-2: Goal, Outcome, Principle, and Requirement

6.5. 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.

6.5.1. Meaning

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

A meaning represents the interpretation of a concept of the architecture. In particular, this is used to describe the meaning of passive structure elements (for example, a document, a 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 concept. 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.

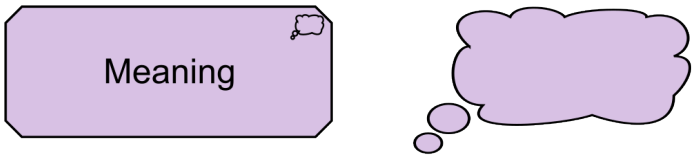


Figure 6-9: Meaning Notation

6.5.2. Value

Value represents the relative worth, utility, or importance of a concept.

Value represents the usefulness, advantage, benefit, desirability, or gain for a customer, stakeholder, or end user by, for example, selling a product, using a service, or completing some activity. 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 the *external* appreciation of goods, services, information, knowledge, or money, normally as part of some sort of customer-provider relationship. Value may also be negative. For instance, the introduction of an Artificial Intelligence (AI) chatbot on a company website may have a positive value (efficiency) for that company but a negative value (frustration) for customers.

A value can be associated with any concept. 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.

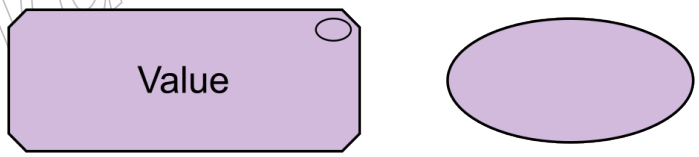
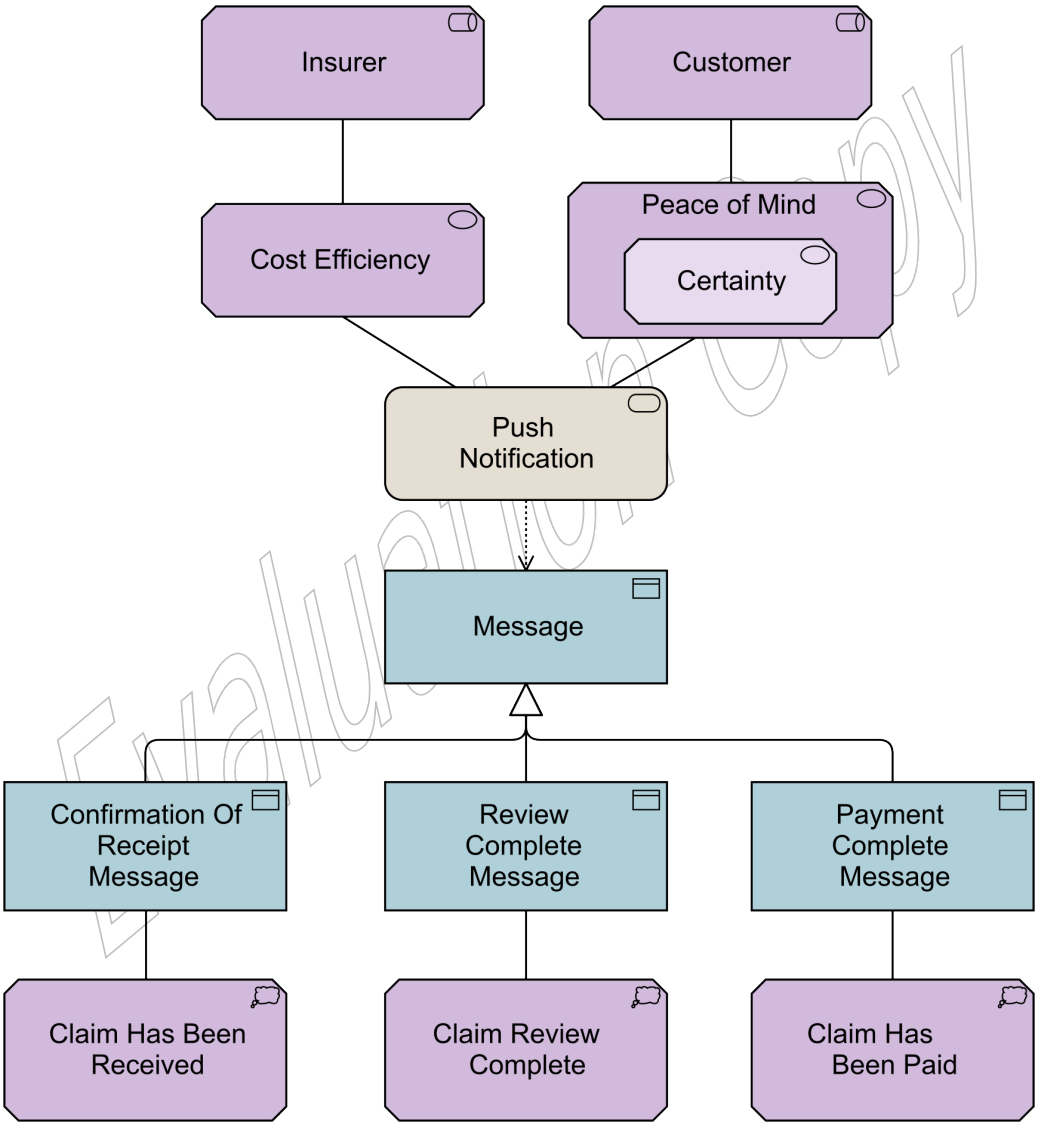


Figure 6-10: Value Notation

6.5.3. Example

Sending a “Push Notification” has a value of “Cost Efficiency” for the stakeholder “Insurer”, and a value 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 “The Insurer has acknowledged and registered a claim”, a “Review Complete Message” has the meaning “Claim review has been completed, so the Customer can expect a decision soon”, and a “Payment Complete Message” has the meaning “Payment has been authorized and funds have been transferred to the Customer nominated account”.

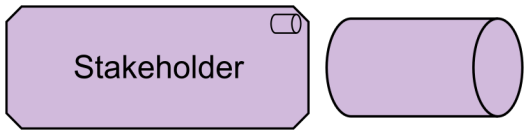
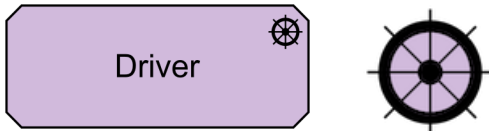
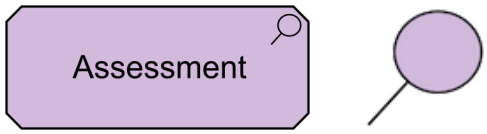
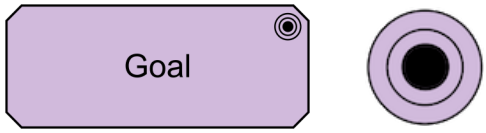
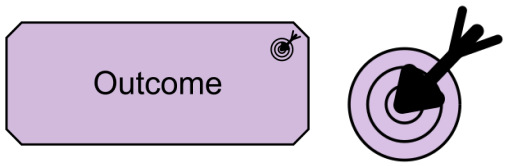
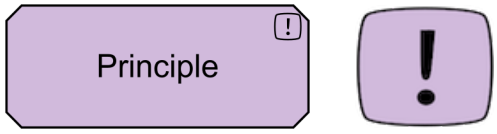
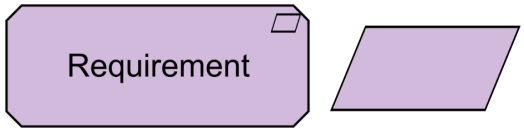
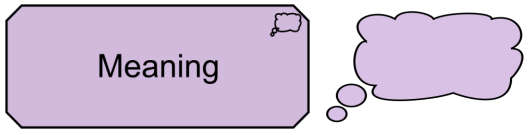
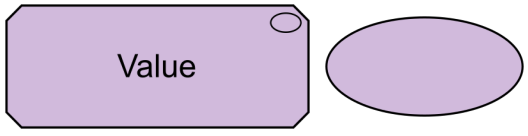


Example 6-3: Meaning and Value

6.6. Summary of Motivation Elements

Table 6-1 gives an overview of the motivation elements, with their definitions.

Table 6-1: Motivation Elements

Element	Definition	Notation
Stakeholder	The perspective from which a business actor perceives the effects 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.	
Outcome	An end result, effect, or consequence of a certain state of affairs.	
Principle	A statement of intent defining a general property that applies to any system in a certain context in the architecture.	
Requirement	A statement of need defining a property that applies to a specific system as described by the architecture.	
Meaning	The knowledge or expertise present in, or the interpretation given to, a concept in a particular context.	
Value	The relative worth, utility, or importance of a concept.	

6.7. Relationships with Other Domains

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 6-11, a “Requirement” can be related to a “Structure Element/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. Other relationships can be derived from this as well, using the derivation rules of Appendix B.

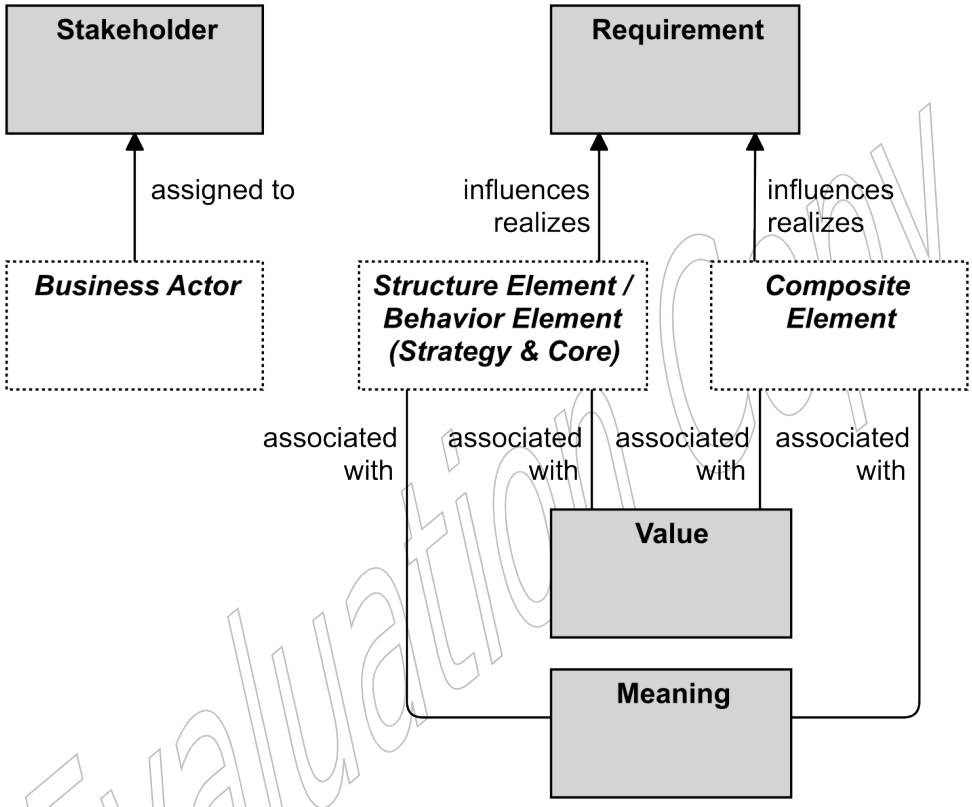


Figure 6-11: Relationships Between Motivation Elements and Core Elements

Additionally, a “Business Actor” may be assigned to a “Stakeholder” to express that someone with an operational position within or outside the enterprise is also a stakeholder of that enterprise.

Chapter 7. Strategy Domain

The strategy elements are typically used to model the strategic direction and choices of an enterprise, as far as the impact on its architecture is concerned. They can be used to express how the enterprise wants to create value for its stakeholders, the capabilities it needs, and the resources needed to support these capabilities, as well as how it plans to configure and use these capabilities and resources to achieve its aims (see Chapter 6). Strategy elements are used to model the strategic direction and choices of the enterprise, whereas Common Domain elements and Business Domain elements (see Chapter 4 and Chapter 8) are used to model the operational organization of an enterprise.

Intentionally, the set of elements in the Strategy Domain is kept as small and simple as possible. The strategy elements are more abstract than the business elements. The Strategy Domain abstracts from the difference between active and passive structure, and from the internal/external distinction in the Common, Business, Application, and Technology Domains. Unlike these domains, it does not have service or interface elements.

7.1. Strategy Elements Metamodel

Figure 7-1 gives an overview of the strategy elements and their relationships.

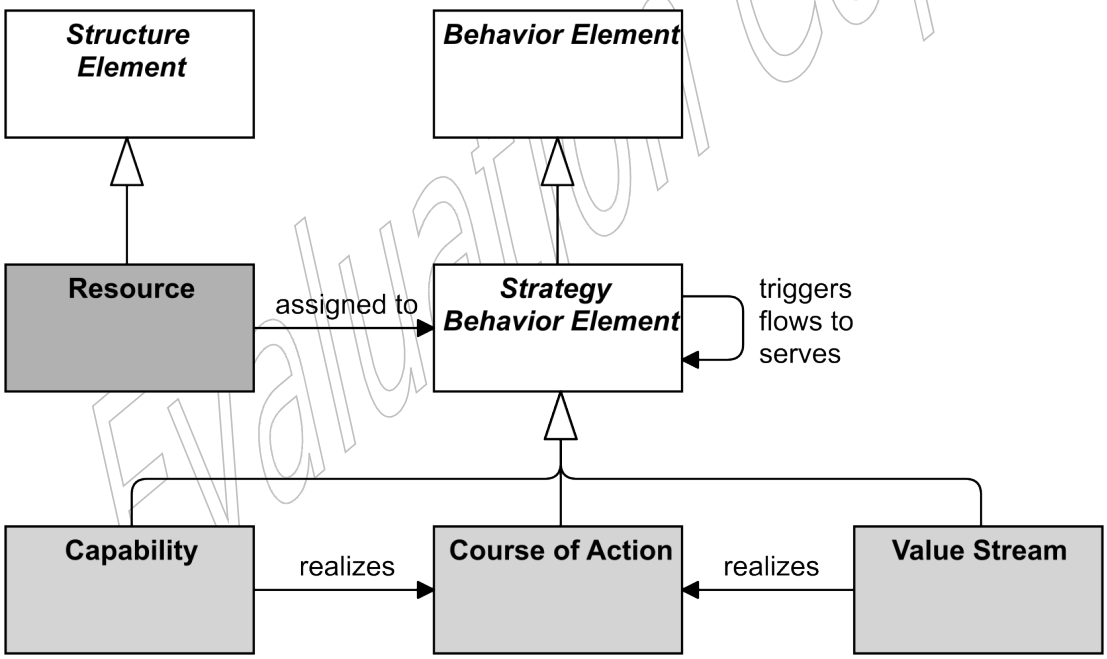


Figure 7-1: Strategy Elements Metamodel

7.2. Structure Elements

7.2.1. Resource

A resource represents a useful or valuable possession or means that can be drawn upon by the enterprise.

Resources are a central concept in the field of strategic management, economics, computer science, portfolio management, and more. They are often recognized along with capabilities, to be sources of competitive advantage for organizations. Resources are analyzed in terms of strengths and weaknesses, and 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 in different ways, including tangible assets, intangible assets, and human assets. Examples of tangible assets include financial assets (cash, securities, borrowing capacity, etc.) and physical assets (plant, equipment, land, mineral reserves, etc.). Examples of intangible assets include technology assets (patents, copyrights, trade secrets, etc.), reputation assets (brand, relationships, etc.), and culture assets. Examples of human assets include skills/know-how, capacity for communication and collaboration, and motivation.

Resources are realized by active and passive structure elements and are therefore classified as structures that are neither active nor passive. The name of a resource should preferably be a noun.

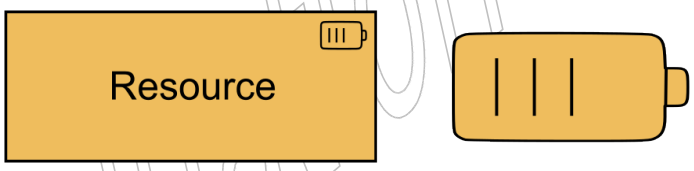


Figure 7-2: Resource Notation

7.3. Behavior Elements

7.3.1. Capability

A capability represents an ability that an organizational entity, 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 at a relatively high abstraction level.

Capabilities help to reduce this gap by focusing on business outcomes. On 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 classified as behavior in the ArchiMate language because they describe abilities: what the enterprise can (is able to) *do* (now or in the future). Capabilities are expressed in general, high-level terms that are typically realized by a combination of organization, people, processes, information, and technology. For example, business planning, customer management, or asset management [G211]. This capability concept corresponds to the business capability in the TOGAF Standard [G21E].

Capabilities are typically aimed at achieving some outcome. Capabilities are themselves realized by core behavior 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 12.2.3) can be used to model the evolution of the capabilities.

The name of a capability should emphasize “what we do” rather than “how we do it”. Typically, it should be expressed as a compound noun or gerund (-ing form of verb); e.g., “Risk Management”, “Market Development”, “Product Engineering”, etc.

Figure 7-3: Capability Notation

7.3.2. Value Stream

> A value stream represents a sequence of activities that create an overall result for a customer, stakeholder, or end user.

A value stream describes how an enterprise organizes its activities to create value. As described in the *TOGAF® Series Guide: Value Streams* [G178], a key principle of value streams is that value is always defined from the perspective of the stakeholder — the customer, end user, or recipient of the product, service, or deliverable produced by the work. The value obtained is in the eye of the beholder; it depends more on the stakeholder’s perception of the worth of the product, service, outcome, or deliverable than on its intrinsic value; i.e., the cost to produce. This is modeled in the ArchiMate language by using the value element and, in turn, is associated on the one hand with the result being produced and, on the other hand, may be associated with the stakeholder.

Value streams may be defined at different levels of the organization; e.g., at the enterprise level, business unit level, or department level. Value streams can be an aggregation of value-adding activities. These are also modeled with the value stream element and known as value (stream) stages, each of which creates and adds incremental value from one stage to the next. These stages are typically related using flow relationships to model the flow of value between them. Resources can be assigned to value streams and capabilities can serve (i.e., enable) a value stream.

Importantly, value streams and processes may seem alike, but they are defined at different abstraction levels and serve separate purposes. A process describes the (time-ordered) sequence of behaviors required to create some result for an individual case, and it may describe alternative paths and decision points (modeled with junctions). In contrast, a value stream focuses on the overall value-creating behavior from the perspective of the importance, worth, or usefulness of what is produced, and is not a description of time-ordered tasks for individual cases. Value streams (and capabilities) reflect an organization's business model and value proposition, whereas processes (and functions) reflect its operating model. Similar to a physical system having potential energy at rest and expending that energy in motion, capabilities and functions describe the potential behavior of the "enterprise at rest", whereas value streams and processes represent the actions of the "enterprise in motion".

Value streams are typically realized by processes and possibly other core behavior elements. The stages in a value stream provide a framework for organizing and defining processes, but different parts of the organization may have their own implementations of processes that realize the same value stream stage.

It is recommended that the name of a value stream be expressed using a verb-noun construct in the active tense; e.g., "Acquire Insurance Product".

Figure 7-4: Value Stream Notation

7.3.3. Course of Action

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

A course of action represents what an enterprise has decided to do to achieve its objectives. 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 short term and narrower in scope.

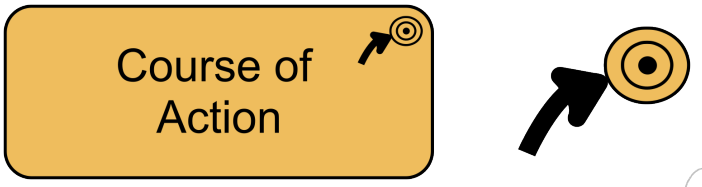
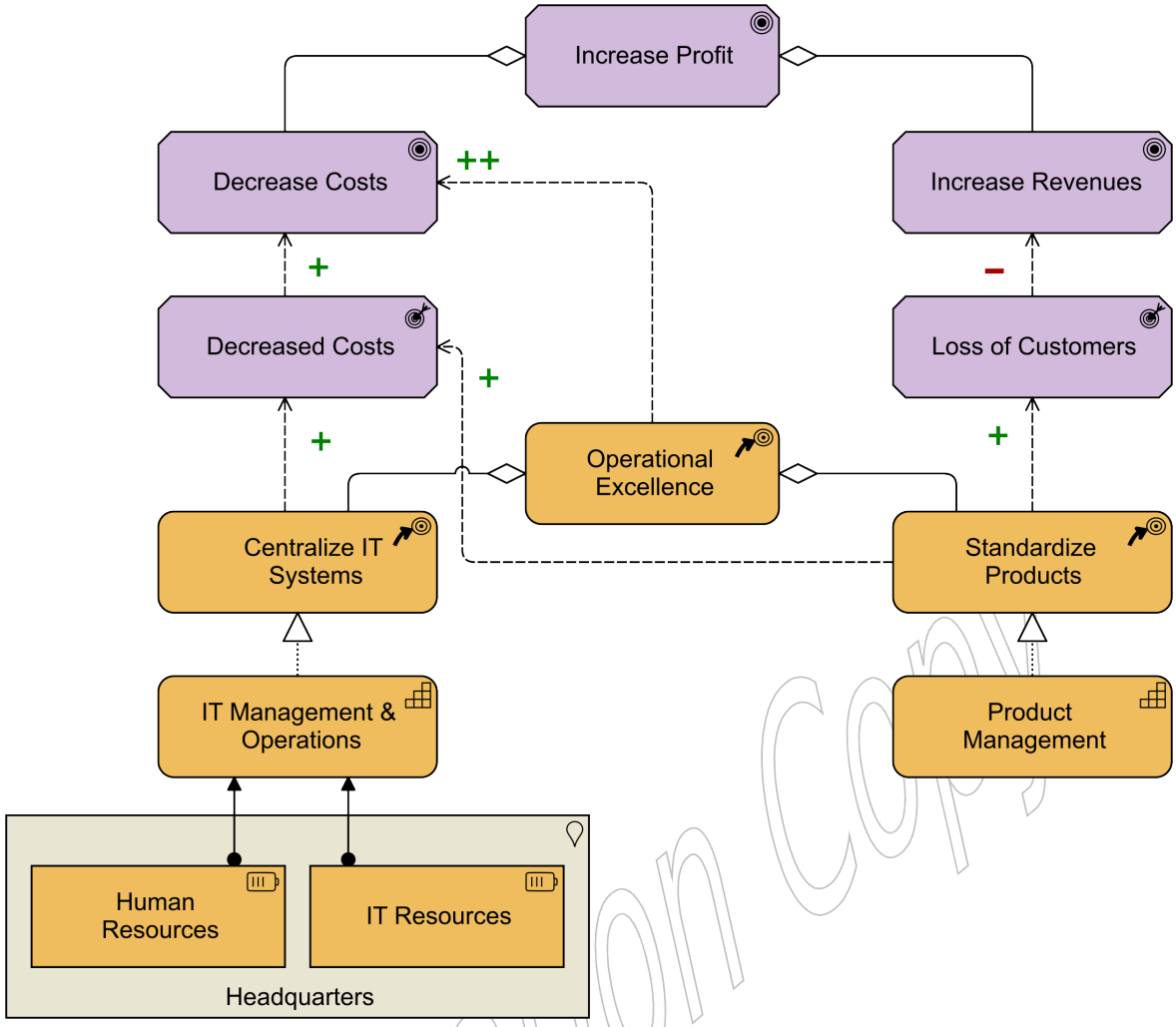


Figure 7-5: Course of Action Notation

7.4. Examples

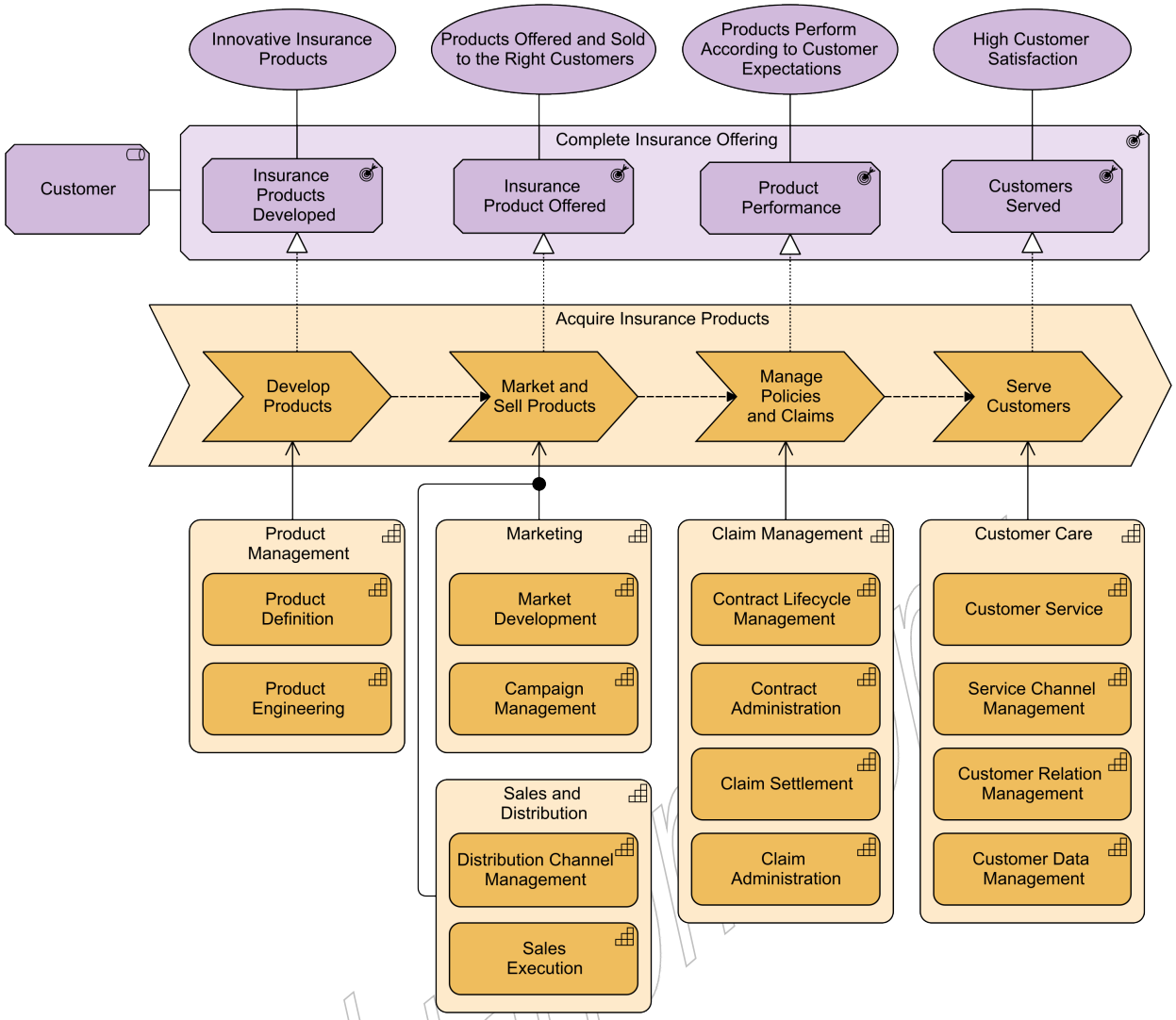
"Increase Profit" is a goal that consists of other goals: "Decrease Costs" and "Increase Revenues". The former is related to the "Operational Excellence" strategy of the company, modeled as a course of action. This is decomposed into two other courses of action: "Centralize IT Systems" and "Standardize 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 7-1: Capability, Resource, and Course of Action

Example 7-2 shows a model of a high-level value stream for an insurance company, where each stage in the value stream is served by a number of capabilities. Between these stages, we see the value flows, and each stage realizes a specific value item (modeled as an outcome), which has a specific value. Together, these outcomes comprise the overall outcome that this value stream realizes for a stakeholder, the customer.

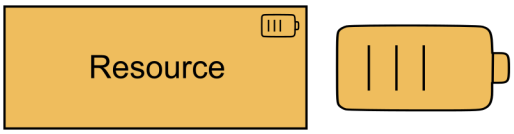
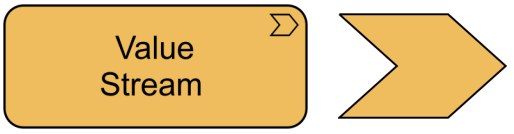
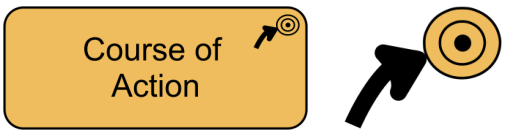


Example 7-2: Value Stream with Capability Cross-Mapping

7.5. Summary of Strategy Elements

Table 7-1 gives an overview of the strategy elements, with their definitions.

Table 7-1: Strategy Elements

Element	Description	Notation
Resource	A useful or valuable possession or means that can be drawn upon by the enterprise.	
Capability	An ability that an organizational entity, person, or system, possesses.	
Value Stream	A sequence of activities that create an overall result for a customer, stakeholder, or end user.	
Course of Action	An approach or plan for configuring some capabilities and resources of the enterprise, to achieve a goal.	

7.6. Relationships with Other Domains

Figure 7-6 shows how the strategy elements are related to core elements and motivation elements. Internal and external behavior elements may realize strategy behavior elements (value streams and capabilities), while an active or passive structure element may realize a resource. Capabilities, value streams, courses of action, and resources may realize or influence requirements (and indirectly, as described in Section 5.8, also principles or goals), and a course of action may also realize or influence an outcome (and, indirectly, also a goal). Other relationships can be derived from this as well, using the derivation of Appendix B.

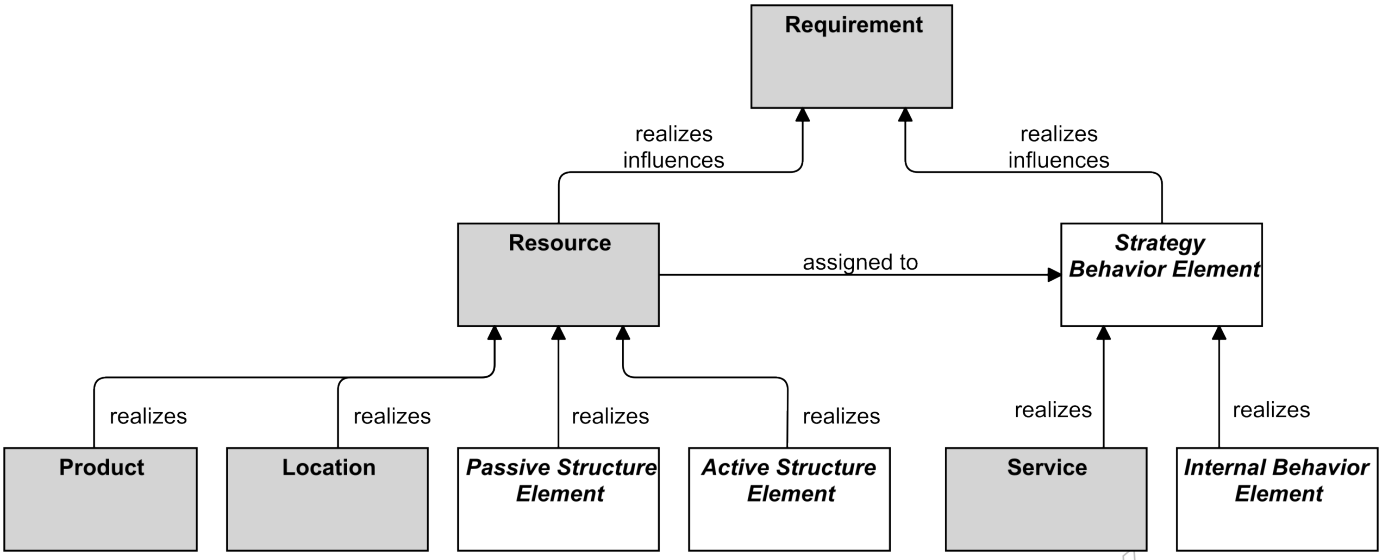


Figure 7-6: Relationships Between Strategy Elements and Motivation and Core Elements

Chapter 8. Business Domain

Business Domain elements are used to model the operational organization of an enterprise in a technology-independent manner, whereas strategy elements (see Chapter 7) are used to model the strategic direction and choices of the enterprise.

8.1. Business Structure Metamodel

Figure 8-1 gives an overview of the active and passive structure elements in the Business Domain and their relationships. “Internal Active Structure Element”, “External Active Structure Element”, and “Passive Structure Element” are abstract elements; only their specializations (as defined in the following sections) are instantiated in models.

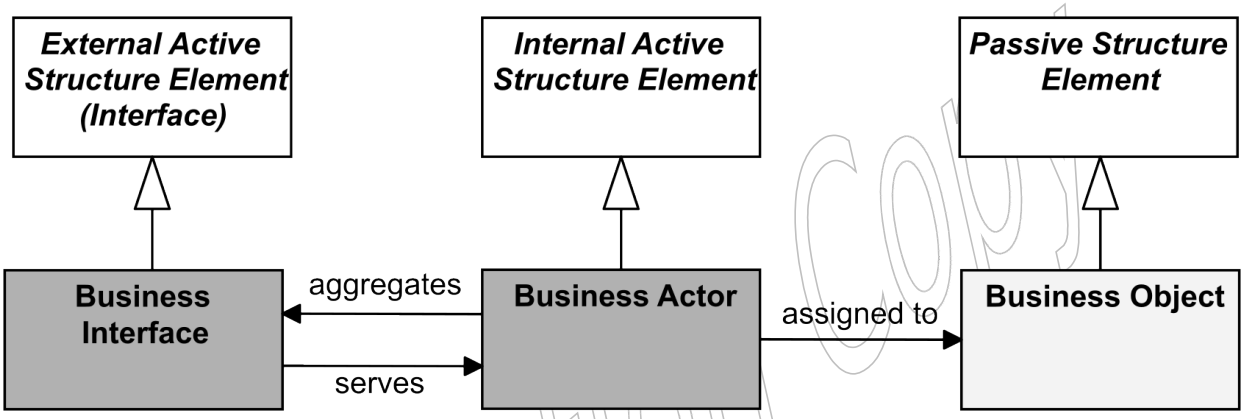


Figure 8-1: Business Structure Metamodel

As explained in Chapter 4, active structure elements can be assigned to common behavior elements. This way, it can be modeled, for example, that a business actor performs a process or a business interface provides a service. These behavior elements, in turn, can access passive elements, so it can be modeled that this process reads or writes a business object.

8.2. Active Structure Elements

The active structure aspect of the Business Domain 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 that perform behavior such as processes or functions. Business actors may be individual persons (e.g., customers or employees), but also groups of people (organization units) that have a permanent (or at least long-term) status within the organizations, such as a department or a business unit.

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.

8.2.1. Business Actor

> A business actor represents 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 Domain. 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 Standard [C220]. Examples of business actors are humans, departments, and business units.

A business actor may be assigned to one or more roles. It can then perform the behavior to which these roles are assigned. Business actors can also be assigned to business objects, to express that someone is responsible for some specific information or another kind of business object.

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”.

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

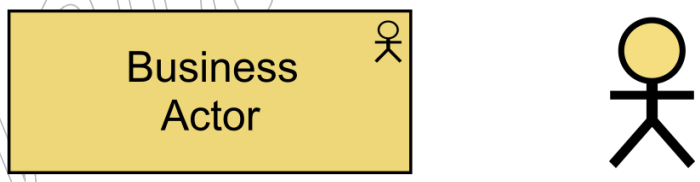


Figure 8-2: Business Actor Notation

8.2.2. Business Interface

A business interface represents a point of access where a business actor makes services available to its environment.

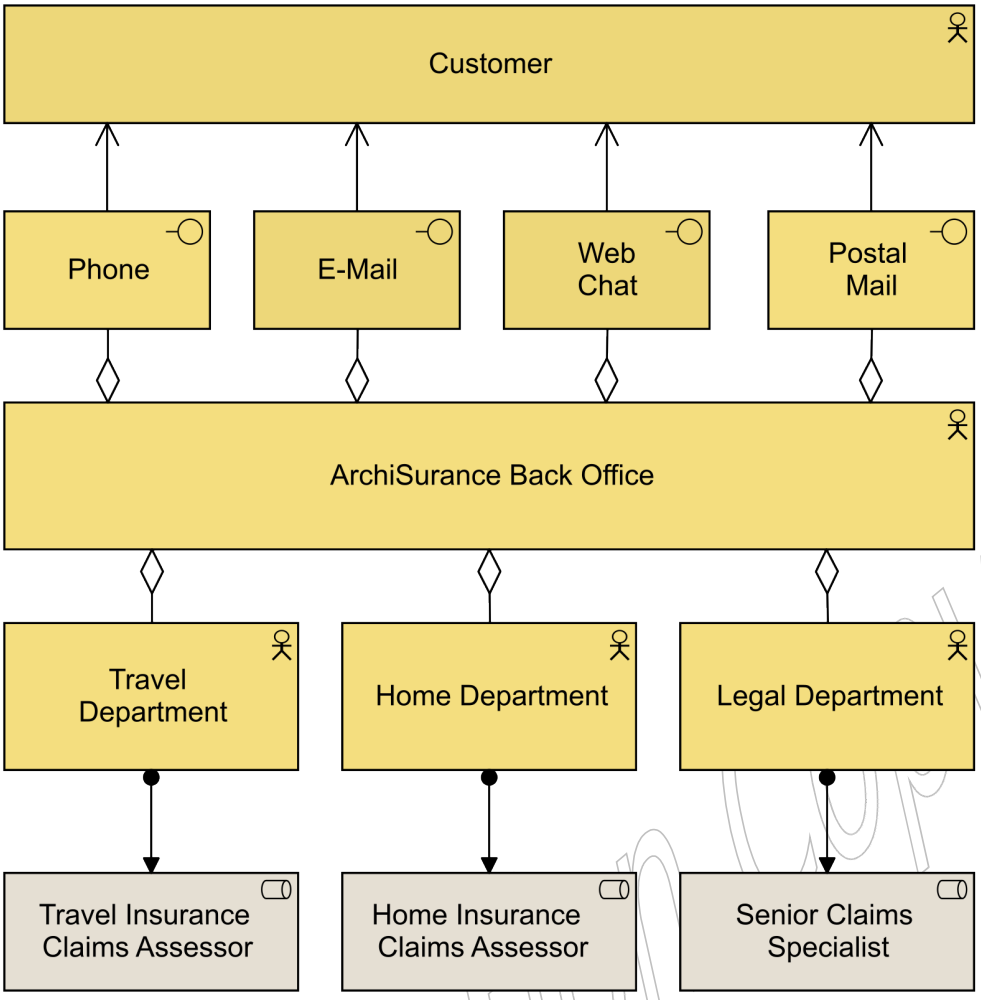
A business interface exposes the functionality of a service that is provided by a business actor. It is often referred to as a channel (telephone, Internet, local office, etc.). The same service may be exposed through different interfaces.

A business interface may be part of a business actor through an aggregation relationship, and a business interface may serve a role or business actor. A business interface may be assigned to one or more services, which means that these services are exposed by the interface. The name of a business interface should preferably be a noun.

Figure 8-3: Business Interface Notation

8.2.3. Example

The “ArchiSurance Back Office”, modeled as a business actor, consists of the “Travel Department”, “Home Department”, and “Legal Department”. Respectively, these are assigned to the roles “Travel Insurance Claims Assessor”, “Home Insurance Claims Assessor”, and “Senior Claims Specialist”. Moreover, the “ArchiSurance Back Office” has four different business interfaces that serve the “Customer” business actor.



Example 8-1: Business Active Structure Elements

8.3. Passive Structure Elements

The passive structure aspect of the Business Domain contains one passive structure element, *business object*, which can be manipulated by behavior such as processes or functions. This represents the important concepts in which the business thinks about a domain.

8.3.1. Business Object

A business object represents a conceptual passive element that has relevance from a business perspective.

The term “conceptual” in this context refers to the conceptual abstraction level used in, for example, data modeling. See also Section 3.5. As explained in that section, 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 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, or written) by a process, function, event, or service. A business object may have association, specialization, or aggregation relationships with other business objects. A business object may be realized by a data object or material. The name of a business object should preferably be a noun.

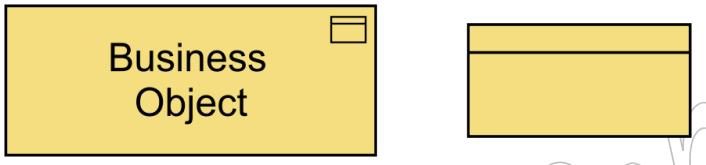
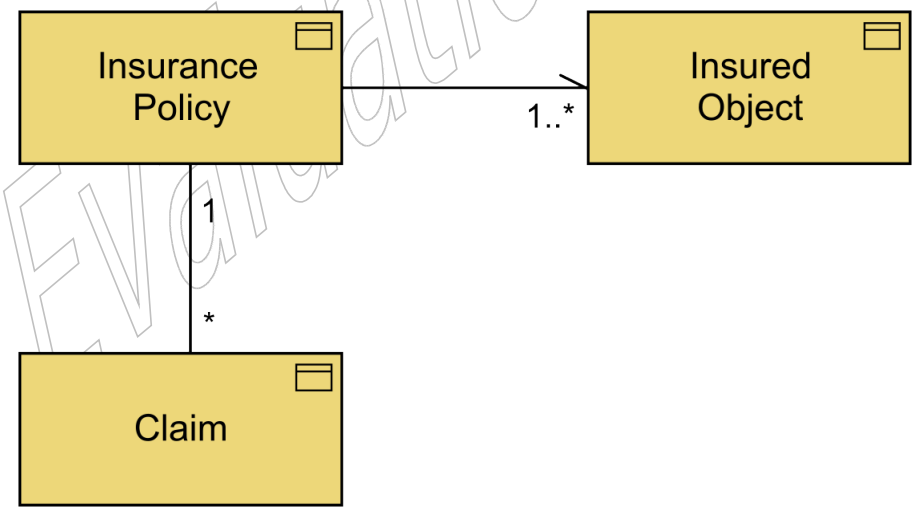


Figure 8-4: Business Object Notation

8.3.2. Example

The example shows that an “Insurance Policy” refers to one or more “Insured Objects”. An “Insurance Policy” can have zero or more “Claims” filed against it, and each “Claim” must refer to exactly one “Insurance Policy”.



Example 8-2: Business Passive Structure Elements

8.4. Composite Elements

The Business Domain contains one composite element: *product*. This aggregates services and passive structure elements across the domains of the ArchiMate core language.

Figure 8-5 shows the applicable part of the metamodel. This crosses domains, as also described in Chapter 11.

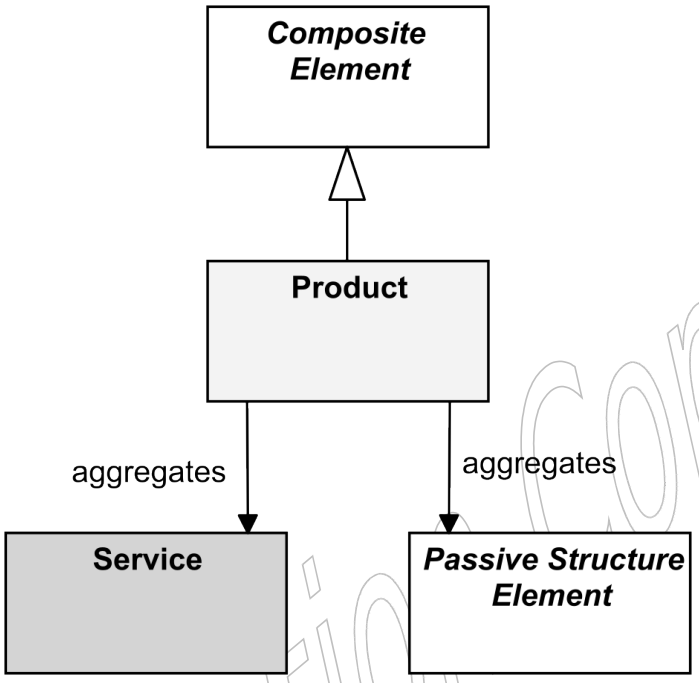


Figure 8-5: Product Metamodel

8.4.1. Product

> A product represents a coherent collection of services, business objects, data objects, artifacts, and/or material, 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, as well as tangible, physical products. A financial or information product typically consists of a collection of services along with a business object representing a contract that specifies the characteristics, rights, and obligations 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 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 services, business objects, data objects, artifacts, and material. Hence a product may aggregate elements from domains other than the Business Domain.

A value may be associated with a product. The name of a product is usually the name used in the communication with customers, or possibly a more generic noun (e.g., “Travel Insurance”).

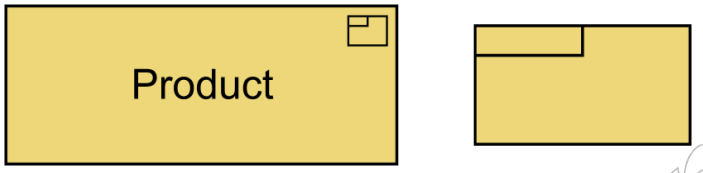
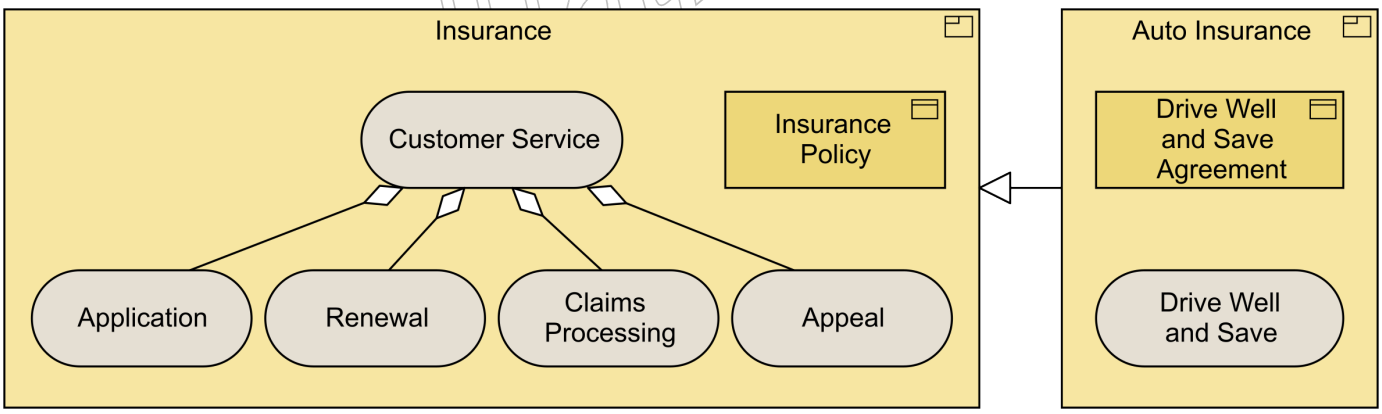


Figure 8-6: Product Notation

8.4.2. Example

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

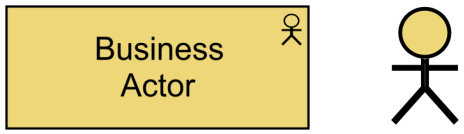
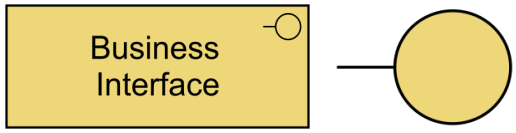
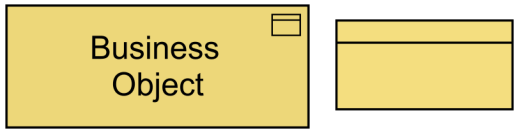
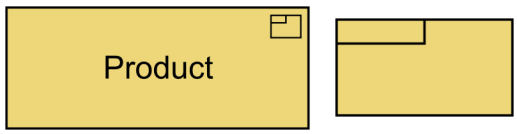


Example 8-3: Business Composite Element: Product

8.5. Summary of Business Domain Elements

Table 8-1 gives an overview of the Business Domain elements, with their definitions.

Table 8-1: Business Domain Elements

Element	Description	Notation
Business Actor	A business entity that is capable of performing behavior.	
Business Interface	A point of access where a business actor makes services available to its environment.	
Business Object	A passive element that has relevance from a business perspective.	
Product	A coherent collection of services, business objects, data objects, artifacts and/or material, which is offered as a whole to (internal or external) customers.	

Chapter 9. Application Domain

The Application Domain elements are typically used to model the Application Architecture that describes the structure, behavior, and interaction of the applications of the enterprise.

9.1. Application Structure Metamodel

Figure 9-1 gives an overview of the active and passive structure elements in the Application Domain and their relationships.

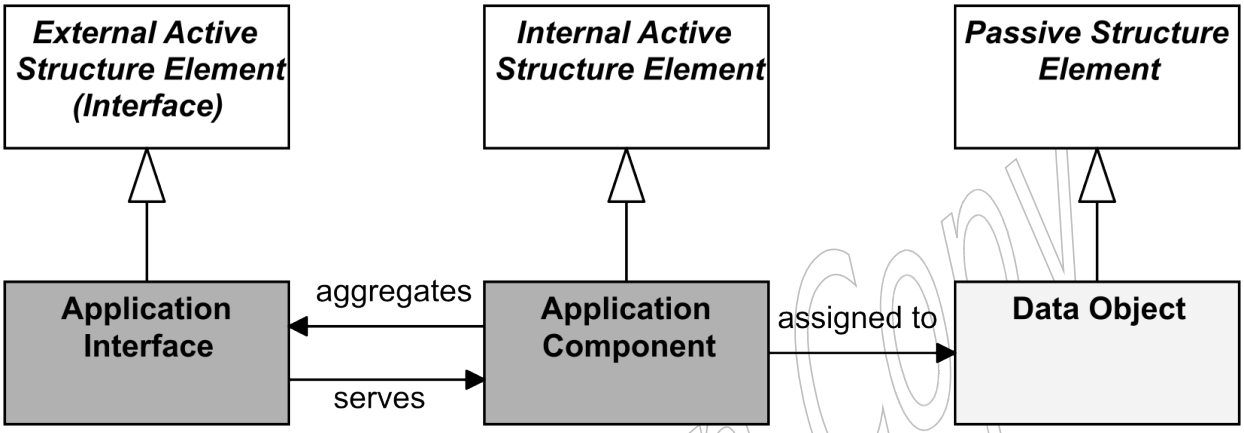


Figure 9-1: Application Structure Metamodel

9.2. Active Structure Elements

The main active structure element for the Application Domain is the *application component*. This element is used to model any structural entity in the Application Domain: 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 explicit relationships to internal behavior elements. Application components can access *data objects*, which are passive structure elements.

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 how the functionality of the component (i.e., its services) is offered to the environment of that component. The application interface element can be used to model *application-to-business* interfaces (e.g., *user interfaces*), *application-to-application* interfaces, and *application-to-technology* interfaces.

9.2.1. Application Component

An application component represents an encapsulation of application functionality aligned to implementation structure, which is modular and replaceable.

An application component is a self-contained unit. As such, it is independently deployable, re-usable, and replaceable. It encapsulates its behavior and data, exposes services, and makes them available through interfaces. An application component may be assigned (through a role) to one or more internal application behavior or application event. Application components can also be assigned to data objects, to express that the application is the repository, golden source, or single source of truth of that data.

Cooperating application components can be aggregated in collaborations. 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 Standard [C220]) and individual parts of such applications, at all relevant levels of detail. This is explained in Section 3.5.

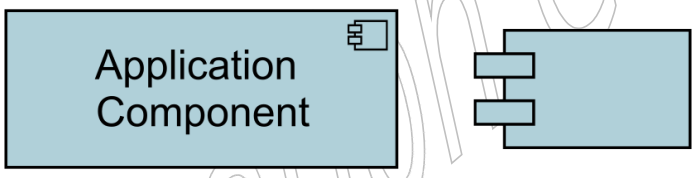


Figure 9-2: Application Component Notation

9.2.2. Application Interface

An application interface represents a point of access where an application component makes its services available to its environment.

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

In a sense, an application interface specifies a contract that a component making this interface available 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 aggregation, which means that this interface is provided by that component and can serve other application components. An application interface can be assigned to services, which means that the interface exposes these services to the environment. The name of an application interface should preferably be a noun.

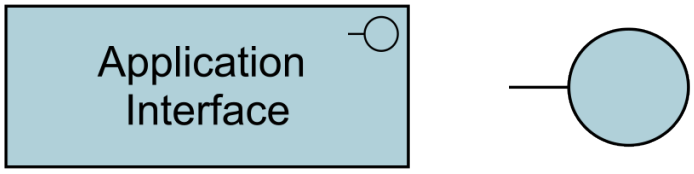
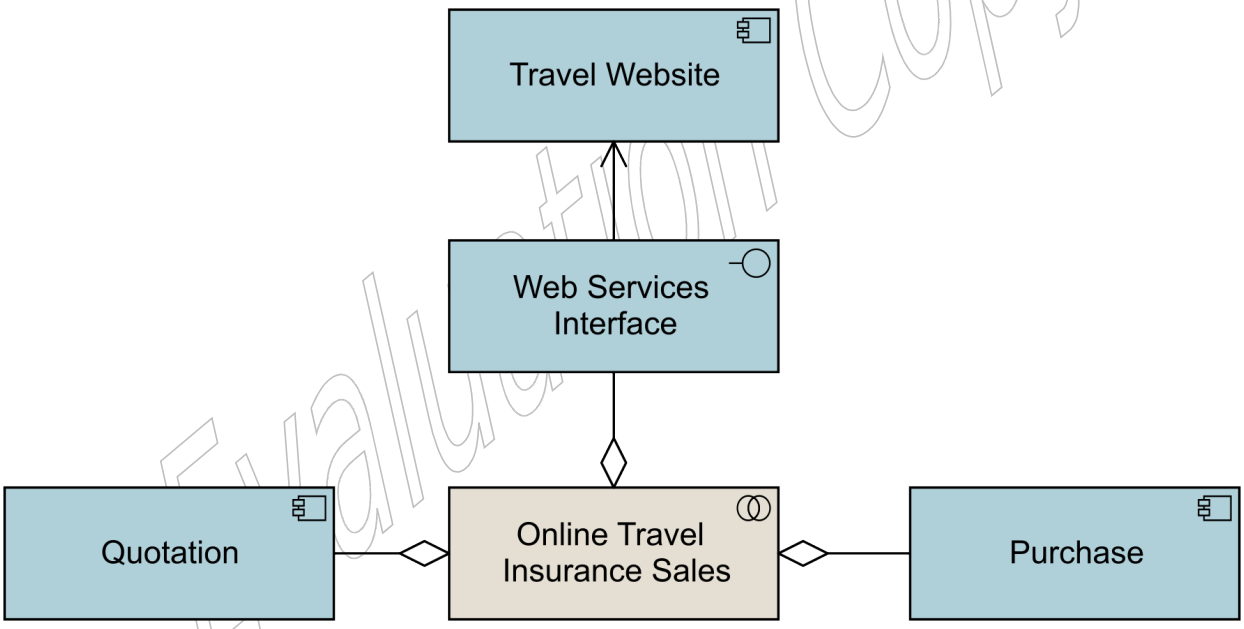


Figure 9-3: Application Interface Notation

9.2.3. Example

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



Example 9-1: Application Active Structure Elements

9.3. Passive Structure Elements

The passive counterpart of the application component in the Application Domain 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 for use by IT applications. The ArchiMate language does not define a specific domain for information; however, elements such as business objects and data objects are used to represent information entities and logical data components.

9.3.1. 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 data set, or an insurance claim.

As explained in Section 3.5, 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 a logical object type of which one or many instances may exist in operational applications.

A function, process, or service can access data objects. A data object may be transferred via flow relations and used or produced by services. A data object may realize a business object and may be realized by an artifact. A data object may have association, specialization, or aggregation relationships with other data objects. The name of a data object should preferably be a noun.

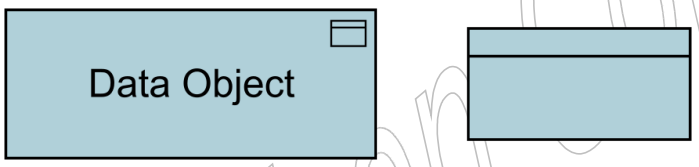
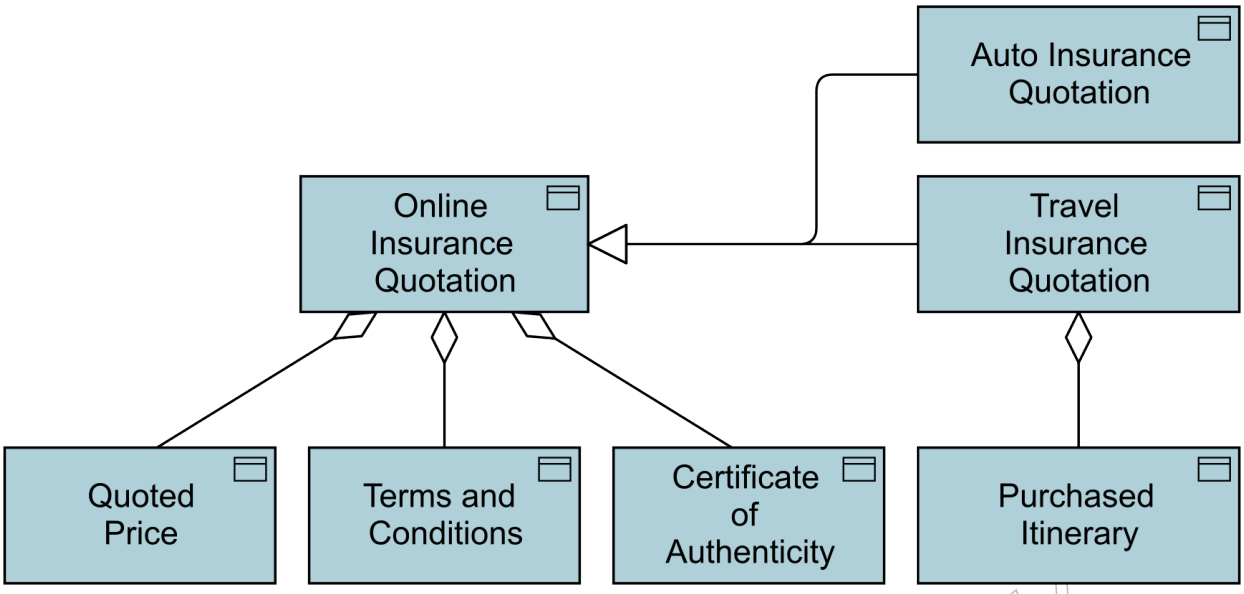


Figure 9-4: Data Object Notation

9.3.2. Example

An “Online Insurance Quotation” data object aggregates 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”.

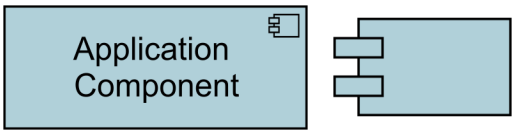
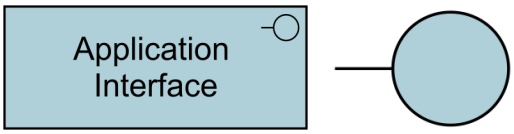
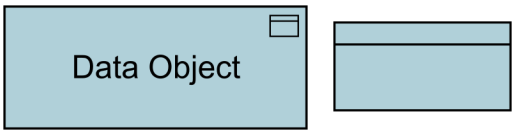


Example 9-2: Application Passive Structure Elements

9.4. Summary of Application Domain Elements

Table 9-1 gives an overview of the Application Domain elements, with their definitions.

Table 9-1: Application Domain Elements

Element	Definition	Notation
Application Component	An encapsulation of application functionality aligned to implementation structure, which is modular and replaceable.	
Application Interface	A point of access where an application component makes services available to its environment.	
Data Object	Data structured for automated processing.	

Chapter 10. Technology Domain

The Technology Domain elements are typically used to model the Technology Architecture of the enterprise, describing the structure and behavior of the technology infrastructure of the enterprise. This comprises both information technology and operational/physical technology, and combinations thereof.

10.1. Technology Metamodel

Figure 10-1 gives an overview of Active Structure Elements in the Technology Domain.

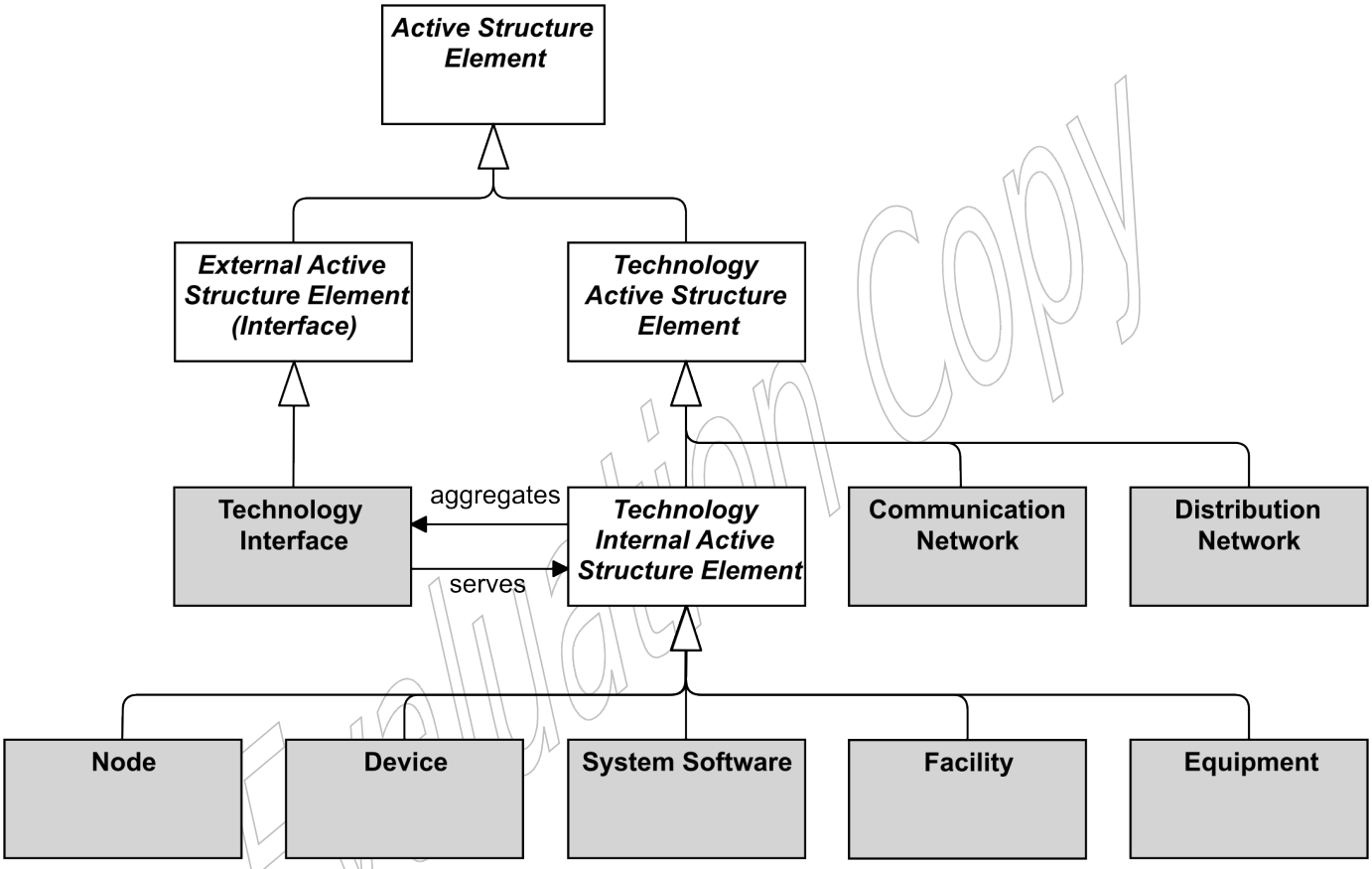


Figure 10-1: Technology Active Structure Elements

Figure 10-2 shows the relationships of the Technology Active Structure Elements.

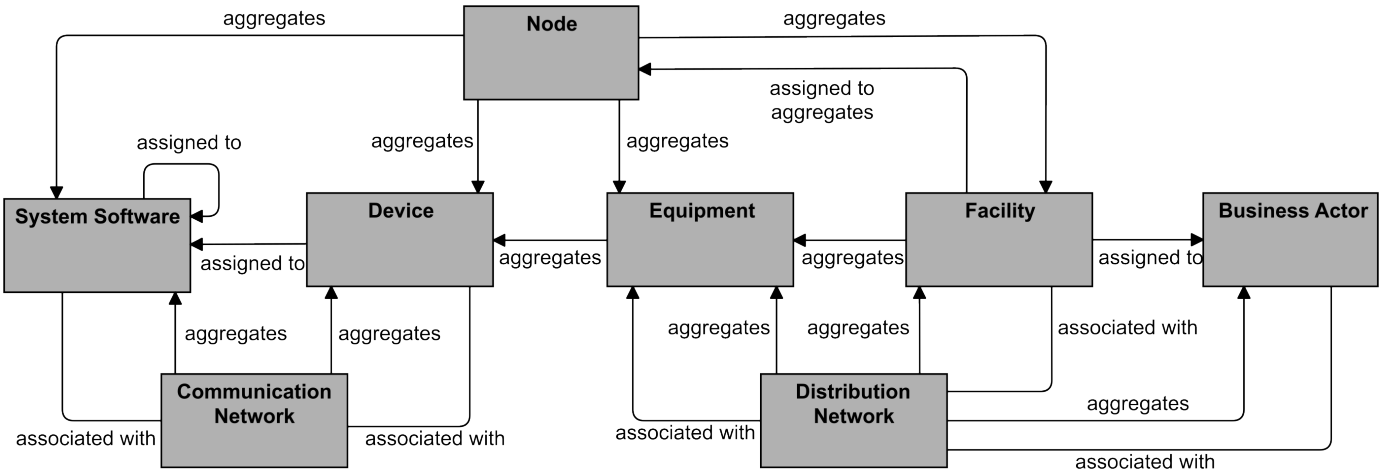


Figure 10-2: Technology Active Structure Relationships

NOTE These figures do not show all permitted relationships; every element in the language can have aggregation and specialization relationships with elements of the same type. Furthermore, there are indirect relationships that can be derived, as explained in Section 5.8.

10.2. Active Structure Elements

The main active structure element in the Technology Domain is the *node*. This element is used to model structural entities in this domain. It strictly models the structural aspect of a system: its behavior is modeled by an explicit relationship to a behavior element. A *technology interface* is the (logical or physical) place where the services offered by a technology internal active structure element can be accessed by other elements.

Nodes can consist of information and technology elements modeled as *devices* and *system software*, and also of operational technology elements modeled as *facilities* and *equipment*. A *device* models a computational resource, upon which artifacts may be deployed for execution. *System software* is an infrastructural software component running on a device. Often, a node consists of a number of sub-elements; for example, a device such as a server and system software to model the operating system.

The *equipment* element is the main active structure element for describing operational technology. 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 behavior elements. The *facility* element is used to model the environment in which this equipment is used; for example, office buildings, factories, laboratories, or data centers.

The inter-relationships of components in the Technology Domain are mainly formed by the communication infrastructure. The common element *path* models the relation between two or more internal active structure elements, through which these can exchange information. The physical realization of a path is modeled with a *communication network* or *distribution network*; i.e., a physical communication medium between two or more devices (or other networks), or a medium for the distribution or transportation of physical materials or energy, such as rail or road networks, the water supply, power grid, or gas network.

10.2.1. Node

A node represents a generic IT or physical structure that hosts, manipulates, or interacts with other such structures.

Nodes are active structure elements that perform behavior and execute, store, and process artifacts or materials. For instance, nodes are used to model application platforms, defined by the TOGAF Standard [C220] as: "*a collection of technology components of hardware and software that provide the services used to support applications*".

Nodes can be interconnected by paths, communication networks or distribution networks, and 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.

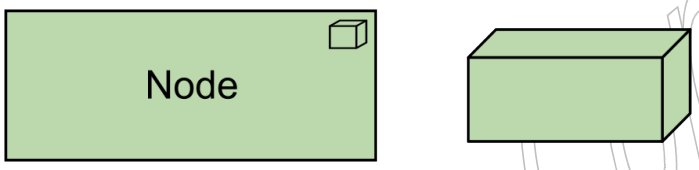


Figure 10-3: Node Notation

10.2.2. Technology Interface

A technology interface represents a point of access where a node, device, system software, facility, or equipment makes services available to its environment.

A technology interface specifies how the services of a technology internal active structure element can be accessed by other elements. A technology interface exposes a 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 node realizing this interface must fulfill. This may include, parameters, protocols used, pre- and post-conditions, and data formats.

A technology interface may be part of a technology internal active structure through aggregation, which means that these interfaces are provided by that element and can serve other elements. A technology interface can be assigned to a service, to expose that service to the environment.

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

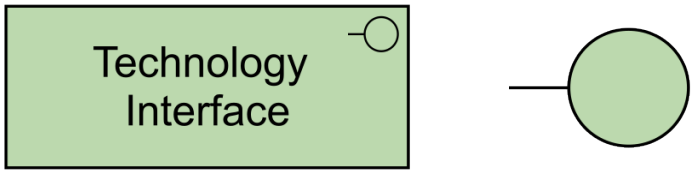


Figure 10-4: Technology Interface Notation

10.2.3. Device

A device represents IT hardware (possibly virtualized) used for processing, data storage, and similar purposes.

A device represents an IT resource with processing capability. It is typically used to model hardware systems such as mainframes, PCs, smartphones, or routers. It can also be used to model virtualized hardware; e.g., in an Infrastructure as a Service (IaaS) environment. Usually, devices are part of a node together with system software. Devices may be composite; i.e., aggregate sub-devices.

Devices can be interconnected by communication networks as well as being assigned to artifacts and to system software, to model deployment of these on those devices. A node can contain one or more devices. Devices can be realized by artifacts; e.g., to model virtualized hardware systems.

The name of a device should preferably be a noun phrase referring to the type of hardware (e.g., "IBM® Z Mainframe") or its usage (e.g., "Application Server").

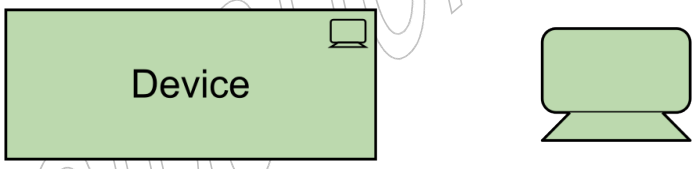


Figure 10-5: Device Notation

10.2.4. 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 used to model the software environment in which artifacts run. This can be an operating system, a Java®, Enterprise Edition (JEE) application server, a database system, or a workflow engine. System software can also be used to represent 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 and can realize other system software. A node can aggregate 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 aggregate other system software; e.g., an operating system containing a database.

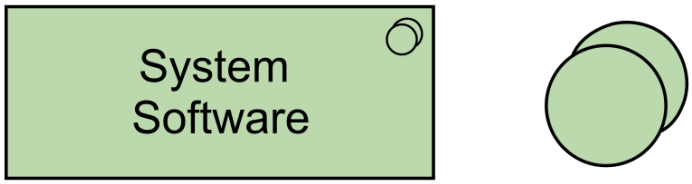


Figure 10-6: System Software Notation

10.2.5. Equipment

> Equipment represents a physical machine, tool, instrument, or livestock.

Equipment comprises all active processing elements that carry out physical processes in which materials are used or transformed. 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, moved, transformed, or otherwise manipulated) by equipment. Equipment may be assigned to material in order to model where the material is stored. Material may realize equipment. Equipment can serve other equipment as well as other active structure elements such as roles and business actors, and facilities can be assigned to equipment. A piece of equipment can aggregate other pieces of equipment or devices. Facilities can be assigned to equipment (i.e., equipment is installed and used in or on a facility). Equipment can be aggregated in a location.

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

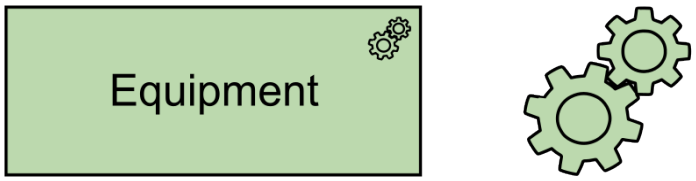


Figure 10-7: Equipment Notation

10.2.6. Facility

A facility represents a physical structure or environment.

A facility 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, and also IT-related facilities. Examples of facilities include a factory, laboratory, warehouse, shopping mall, cave, data center, or spaceship.

Facilities can be interconnected by distribution networks. A facility may be composed of technology interfaces in order to serve other internal active structure elements such as business roles and actors. A facility can aggregate other facilities or nodes 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 oil refinery”.

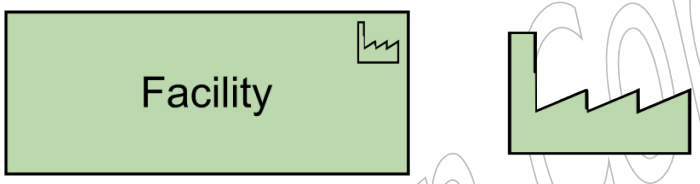


Figure 10-8: Facility Notation

10.2.7. Communication Network

A communication network represents a set of IT structures for transmission, routing, and reception of data.

A communication network represents the physical communication infrastructure. It *“provides the basic services to interconnect systems and provide the basic mechanisms for opaque transfer of data. It contains the hardware and software elements which make up the networking and physical communications links used by a system, and of course all the other systems connected to the network”*, as described by the *TOGAF Series Guide: The TOGAF Technical Reference Model (TRM)* [G175].

A communication network can be associated with the devices or system softwares it connects. The most basic communication network is a single link between two devices, but it may comprise multiple links and required network equipment or software. 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 to model the routers, switches, and firewalls that are part of the network infrastructure.

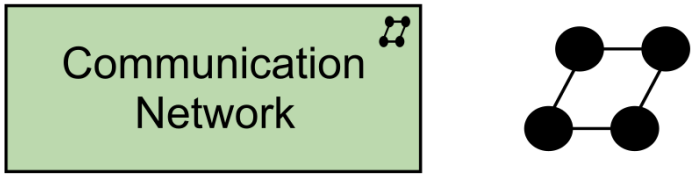


Figure 10-9: Communication Network Notation

10.2.8. 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 facilities.

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

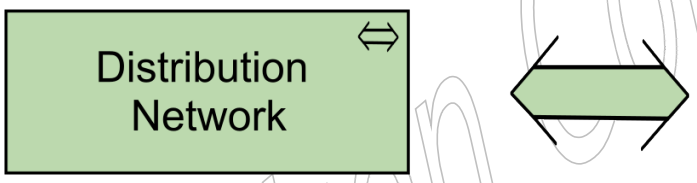


Figure 10-10: Distribution Network Notation

10.3. Passive Structure Elements

Technology passive structure elements may be accessed by behavior (functions, processes, events, and services). A technology passive structure elements may have association, specialization, or aggregation relationships with other technology passive structure elements. It may be an artifact (see Section 10.3.1) or material (see Section 10.3.2). Specific technology internal active structure elements may be assigned to technology passive structure elements. For example, a device may be assigned to an artifact, modeling the deployment of the artifact on that device.

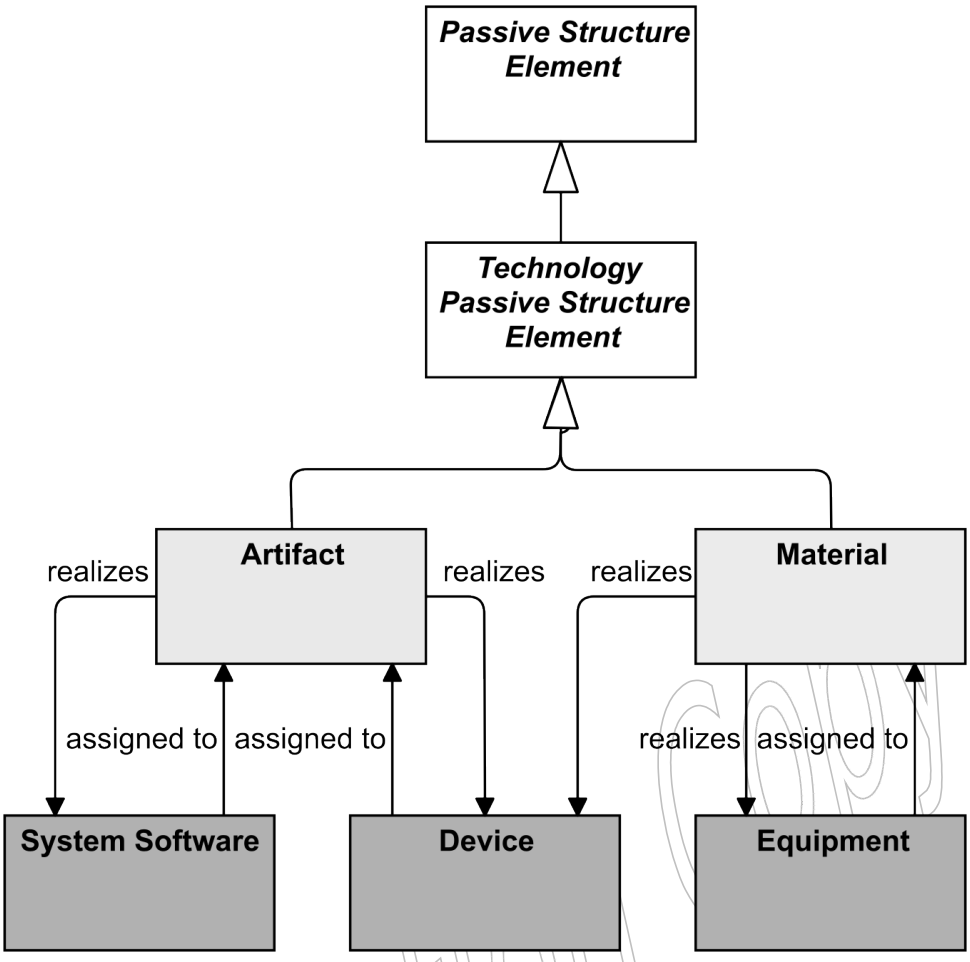


Figure 10-11: Technology Passive Structure Elements

10.3.1. 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 “physical” element in the IT world. Artifact is a specialization of technology passive structure elements. It is typically used to model (software) products such as source files, executables, scripts, database tables, messages, documents, specifications, and model files. It can be accessed (created, deleted, read, written) by behavior elements. An instance (copy) of an artifact can be deployed on a device or system software; this is modeled with an assignment relationship. 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 device or system software element may be assigned to an artifact to model that the artifact is deployed on this element. 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 kind of artifact or the name of the file it represents;

e.g., “virtual image”, “monitoring script”, or “order.jar”. An artifact may consist of sub-artifacts.

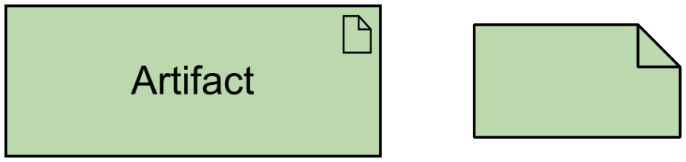


Figure 10-12: Artifact Notation

10.3.2. Material

> Material represents tangible physical matter or energy.

Material is typically used to model raw materials and physical products, and also energy sources such as fuel and electricity. Material can be accessed by behavior elements to model how it is created, used, moved, transformed, or otherwise manipulated. Equipment may be assigned to material, to model where the material is stored.

This allows a distinction to be modeled between, for instance, “the sand is in the dump truck” and “the sand is poured into the concrete mixer”. Material may realize equipment.

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

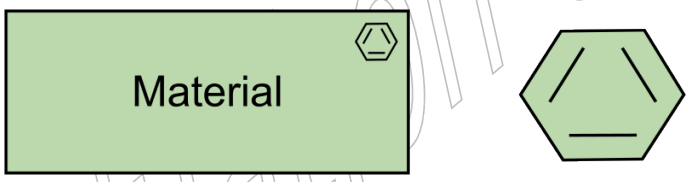
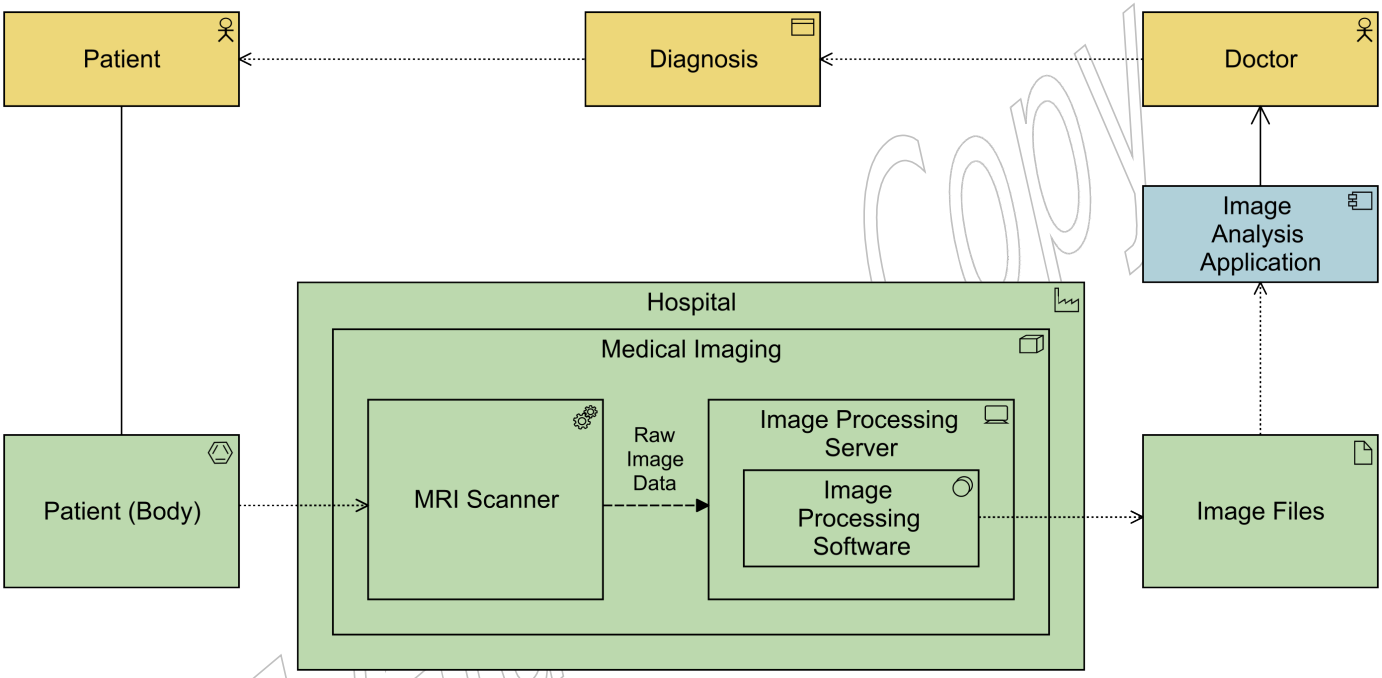


Figure 10-13: Material Notation

10.4. Example

The example shows a “Patient” and “Doctor”, where the ”Patient (Body)” (modeled as material) is scanned by an “MRI Scanner” that is part of ”Medical Imaging” at the ”Hospital. The “Raw Image Data” coming out of this scanner is transferred to the “Image Processing Server”, on which the “Image Processing Software” runs. The resulting “Image Files” are read by the “Image Analysis Application” that the “Doctor” uses to provide a “Diagnosis” to the “Patient”.

A more elaborate view could also show, for instance, how a Communication Network is used to transfer image data, or how a Distribution Network transports patients around the hospital. For the sake of clarity, these details have been left out here.

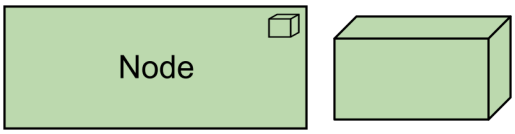
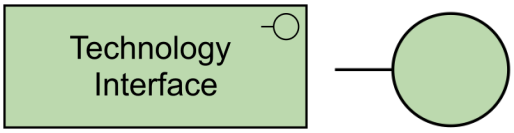
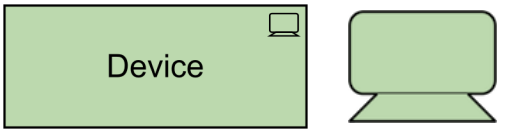
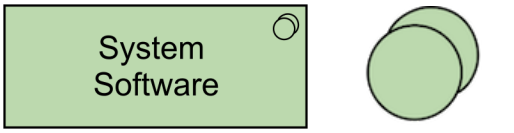
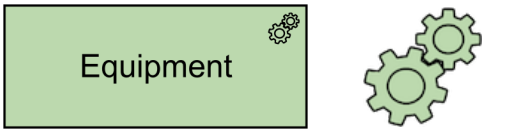
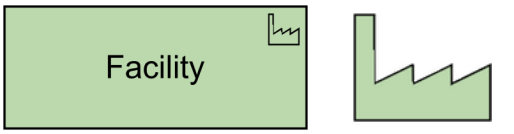
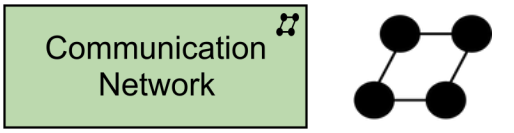
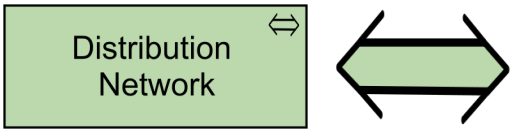
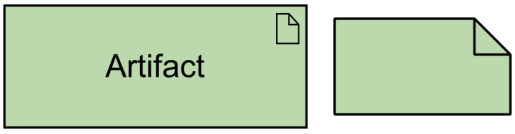
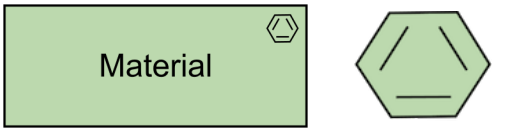


Example 10-1: Technology Elements

10.5. Summary of Technology Domain Elements

Table 10-1 gives an overview of the Technology Domain elements, with their definitions.

Table 10-1: Technology Domain Elements

Element	Definition	Notation
Node	A generic IT or physical structure that hosts, manipulates, or interacts with other such structures.	
Technology Interface	A point of access where a node, device, system software, facility, or equipment makes services available to its environment.	
Device	IT hardware (possibly virtualized) used for processing, data storage, and similar purposes.	
System Software	Software that provides or contributes to an environment for storing, executing, and using software or data deployed within it.	
Equipment	A physical machine, tool, instrument, or livestock.	
Facility	A physical structure or environment.	
Communication Network	A set of IT structures for transmission, routing, and reception of data.	
Distribution Network	A physical network used to transport materials or energy.	
Artifact	A piece of data that is used or produced in a software development process, or by deployment and operation of an IT system.	
Material	Tangible physical matter or energy.	

Chapter 11. Relationships Between Core Domains

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

Figure 11-1 shows the relationships between the Business, Application, and Technology Domain elements.

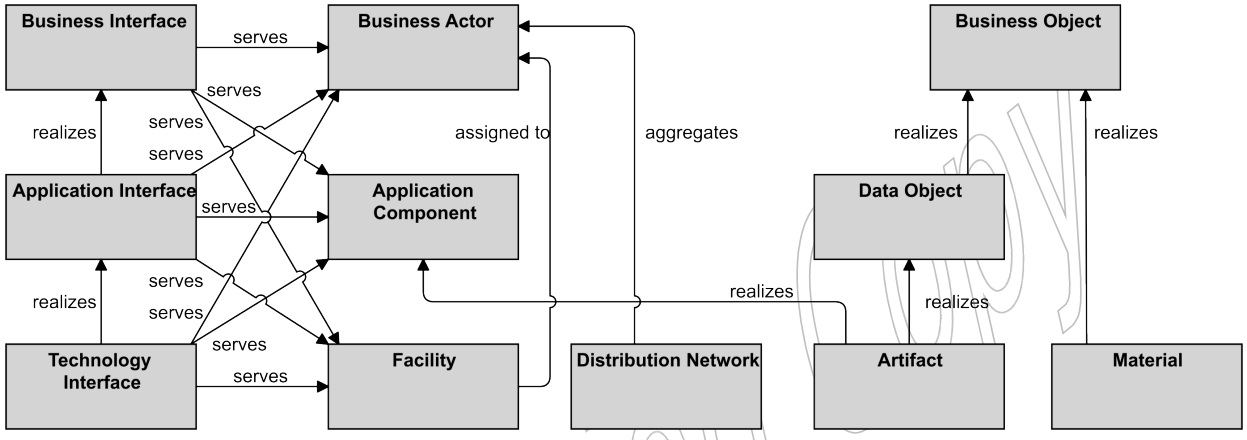


Figure 11-1: Relationships Between Business, Application, and Technology Domain Elements

There are three types of relationships between these domains; as specified here:

1. *Assignment*: From facility to business actor, typically used to indicate that people, teams, or other organization units work in a building
2. *Realization* relationships: For example, to indicate that a data object is a digital representation of the business object it realizes; that a business object is realized by material; that a data object is realized by a physical data file; or that a physical data file is an executable that realizes an application component
3. *Aggregation*: Business actors may be aggregated in distribution networks; for example, postal workers delivering mail as part of a mail distribution network

   (NOTE: In this case, an artifact represents a “physical” component that is deployed on a device or system software element; this is modeled with an assignment relationship. A (logical) application component is realized by an artifact and, indirectly, by the element on which the artifact is deployed.)

Next to these, many other relationships can be derived. For example, if a business object is realized by a data object, which in turn is realized by an artifact, this artifact indirectly realizes the business object. Also, since all three types of interfaces can be assigned to services and those services in turn may serve all active structure elements (e.g., business actors, application components, or nodes), it can be derived that these interfaces can serve all those active structure elements. Perhaps the most common case of this is an application interface serving a business actor, to model that a user accesses the Graphical User Interface (GUI) of some application.

11.1. Example

Example 11-1 illustrates the insurance claims processing system at “ArchiSurance”; how customer claims, from initial registration through final payment, are handled.

The process begins when the “Customer”, acting as an “Insurant”, submits a claim through the “Claims Registration” system. This triggers a sequential workflow: “Register” → “Accept” → “Adjudicate” → “Pay”. Three of these steps offer a corresponding service (“Claim Registration”, “Claim Acceptance”, and “Claim Payment”) with which the “Insurant” interacts throughout the process.

The process is supported by two main applications: a “Financial Application” that handles “Payment Processing”, and a “CRM System” for “Customer Data Management”. These applications run on the “Blade System” hardware platform, which hosts “Application Server” and “Database Management System” system software.

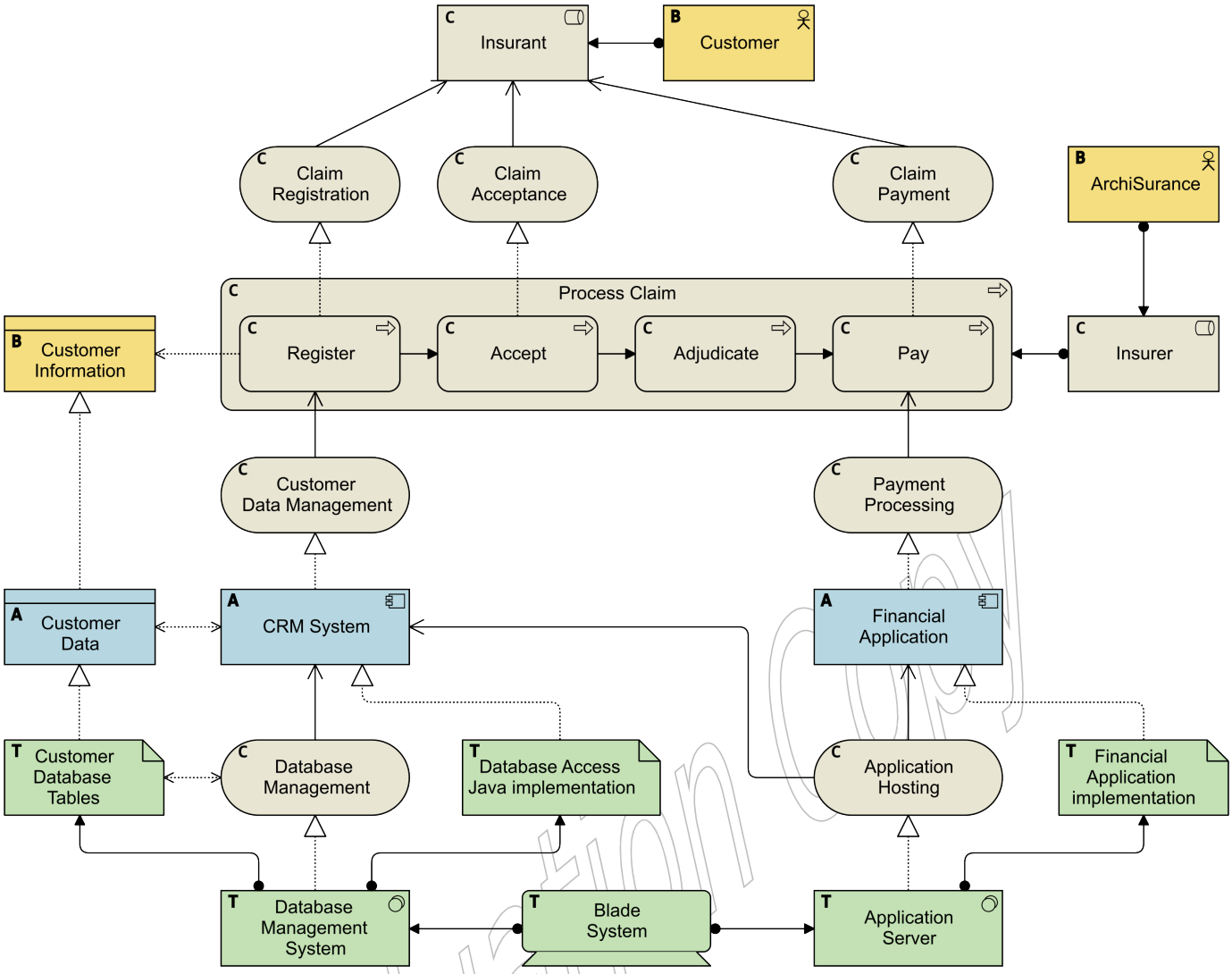
The “Customer Information” flows through multiple layers — from “Customer Database Tables” through application-level “Customer Data” objects, to business-level “Customer Information” that the registration process can update. The “CRM System” has read-write access to “Customer Data”, while the “Database Management” service maintains the underlying “Customer Database Tables”.

Key technology artifacts include the “Database Access Java Archive” implementation for database access (supporting the “CRM System”) and the “Financial Application” implementation, where the first is deployed on the “Database Management System” and the second on the “Application Server” infrastructure. The “Application Hosting” and “Database Management” services provide the foundational technology services that support the application components.

The entire system enables the “Insurer” role (fulfilled by ArchiSurance staff) to process a claim efficiently while maintaining proper data access controls and process flow from initial customer contact through final payment processing.

Note

This example is only intended to show how elements can be connected in different domains. It does not provide a full picture of handling insurance claims. For example, insurance policies and claims themselves are not represented, nor is the application that manages them.



Example 11-1: Cross-Domain Relationships

Letters in the top-left corner of elements are used here to denote to which domain those elements belong, as described in Section 3.8.

Chapter 12. Implementation and Migration Domain

The implementation and migration elements support the implementation and migration of architectures. This includes modeling implementation programs and projects to support program, portfolio, and project management. It also includes support for migration planning.

12.1. Implementation and Migration Elements Metamodel

The implementation and migration metamodel (see Figure 12-1) gives an overview of the implementation and migration elements and their relationships.

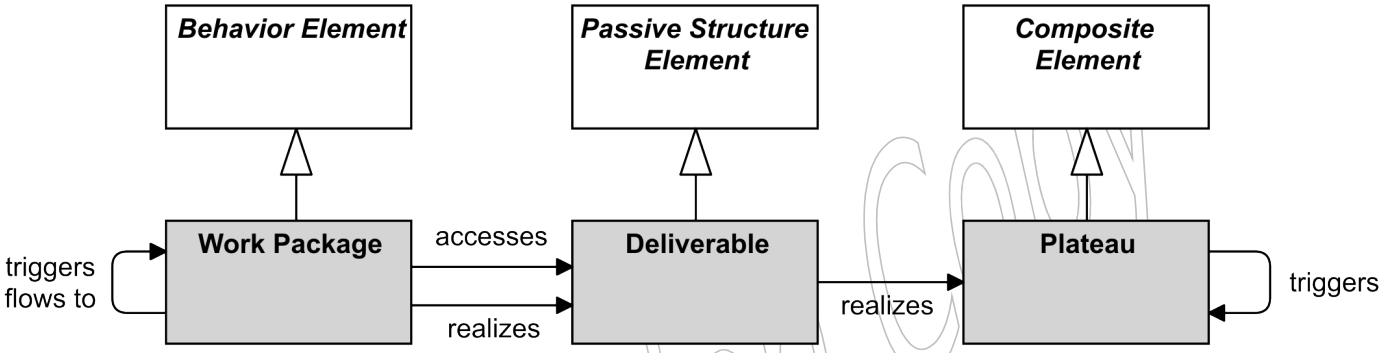


Figure 12-1: Implementation and Migration Metamodel

12.2. Implementation and Migration Elements

12.2.1. 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 typically not a continuously ongoing activity, but has a start and an end. It produces a well-defined set of results, typically modeled as goals, outcomes, or deliverables. The work package element can be used to model sub-projects or tasks within a project, complete projects, programs, or project portfolios. In an Agile context, a work package can be used to model the work performed in an Agile iteration (e.g., sprint) or higher level increment.

Conceptually, a work package is similar to a 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.

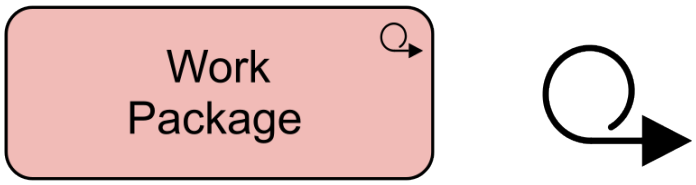


Figure 12-2: Work Package Notation

12.2.2. Deliverable

A deliverable represents a result 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. 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 Standard [C220].

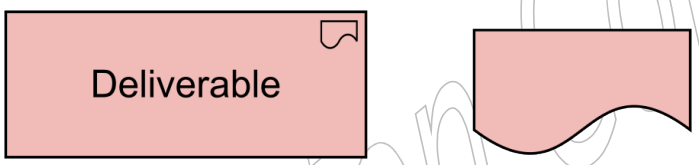


Figure 12-3: Deliverable Notation

12.2.3. 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 Architecture Development Method (ADM), Baseline and Target Architecture Descriptions 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 and expressing the step-by-step approach to migration.

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

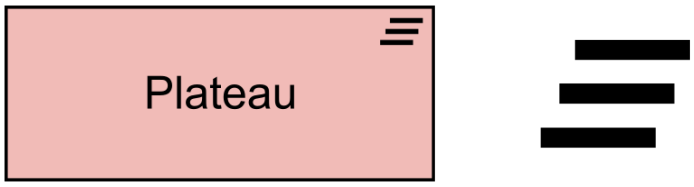
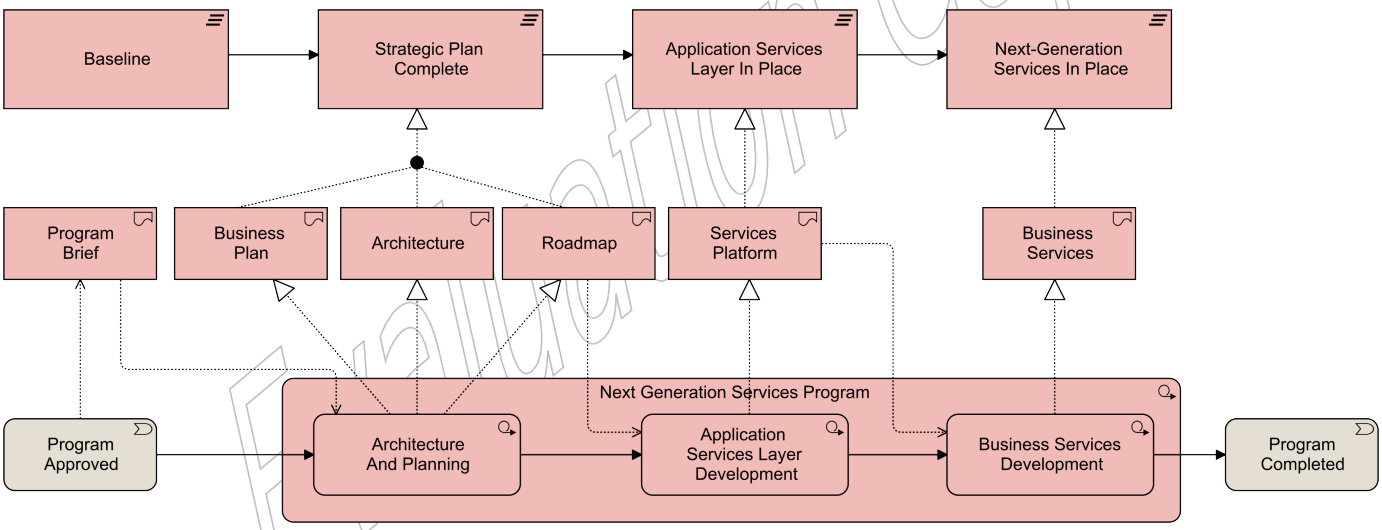


Figure 12-4: Plateau Notation

12.2.4. Example

The “Next Generation Services Program” work package aggregates three other work packages. An 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 in turn triggers the event “Program Completed”. The “Program Approved” 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”. Similarly, the other work packages realize other deliverables that realize the subsequent plateaus.

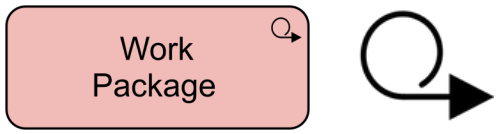
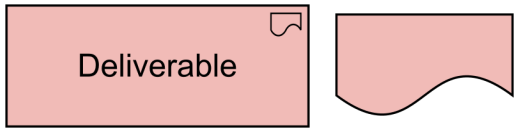
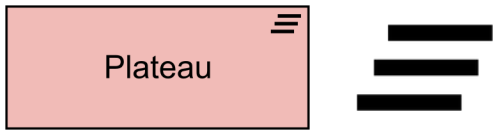


Example 12-1: Implementation and Migration Elements

12.3. Summary of Implementation and Migration Elements

Table 12-1 gives an overview of the implementation and migration elements, with their definitions.

Table 12-1: 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.	
Deliverable	A result of a work package.	
Plateau	A relatively stable state of the architecture that exists during a limited period of time.	

12.4. Relationships with Domains

Figure 12-5 shows how the implementation and migration elements can be related to other concepts.

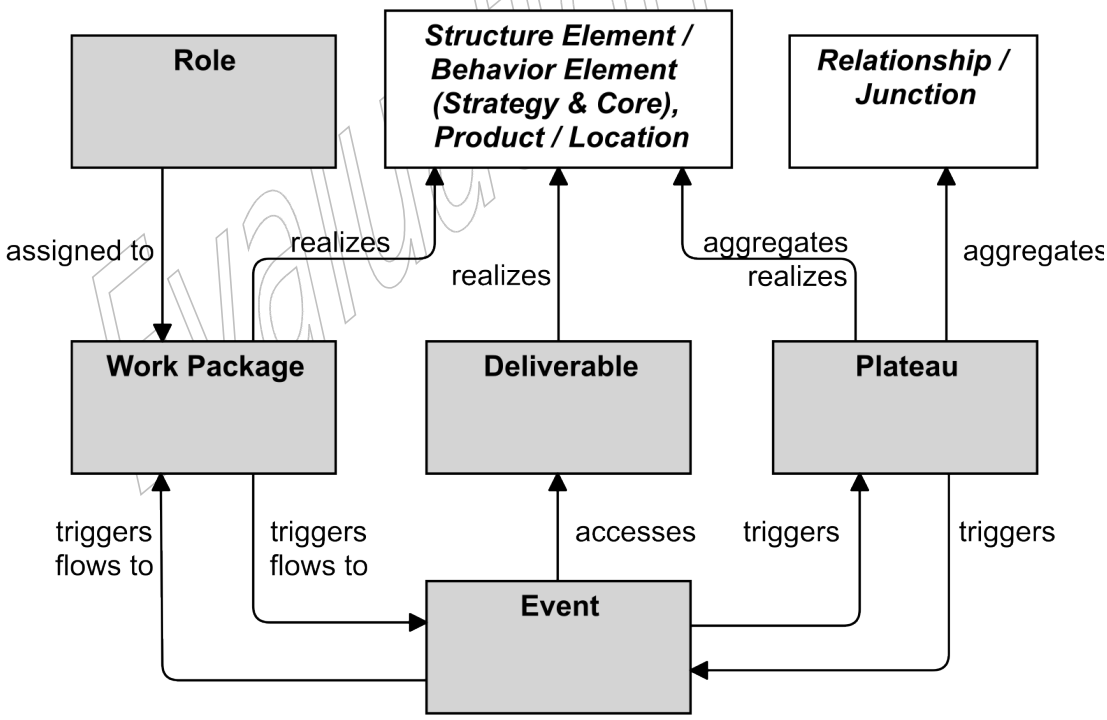


Figure 12-5: Relationships of Implementation and Migration Elements with Other Concepts

A role may be assigned to a work package. This is used to model, for instance, a Project Manager who is responsible for the 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 any of the elements of the Strategy and Core domains, and all relationships and junctions. Realization from a plateau to part of the architecture is also permitted. For example, a capability may be realized by a plateau, signifying that a certain capability increment is valid only during the time span of that plateau.

A deliverable may realize, among others, the implementation of an architecture or a part of an architecture. Therefore, a deliverable may realize any of the elements of the Strategy and Core domains.

Like most of the core language concepts, a composite element may 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, process, or service). However, it is still useful to make these relationships explicit to show that a deliverable is needed to realize certain requirements and goals.

Also, motivation elements can be related to 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 relationship.

Figure 12-6 summarizes the relationships between implementation and migration elements and motivation elements. Goals, outcomes, and requirements can be aggregated in plateaus. Requirements can be realized by deliverables. Since outcomes and goals can be realized by requirements, they can of course be realized indirectly by deliverables as well.

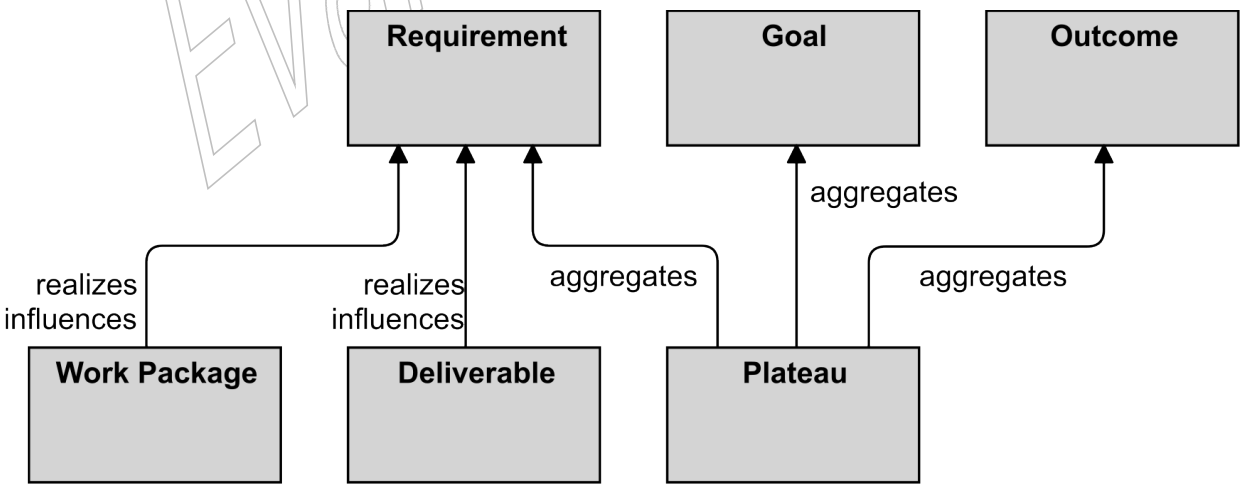


Figure 12-6: Relationships of Implementation and Migration Elements with Motivation Elements

Chapter 13. Stakeholders, Architecture Views, and Viewpoints

13.1. 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 [Sowa & Jackman, 1992; UML, 2011]).

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, architecture views are specified by architecture viewpoints. Architecture 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 also to relate two or more aspects.

In the domain of Enterprise Architecture, the TOGAF Standard [C220] describes a taxonomy of architecture 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 architecture viewpoints and views in Enterprise Architecture models.

The architecture 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 and incomplete depiction of the system.

13.2. 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 [ISO/IEC 42010], which provides a model for Architecture Description. Stakeholders, concerns, viewpoints, and views are important elements in this model, as depicted in Figure 6-1.

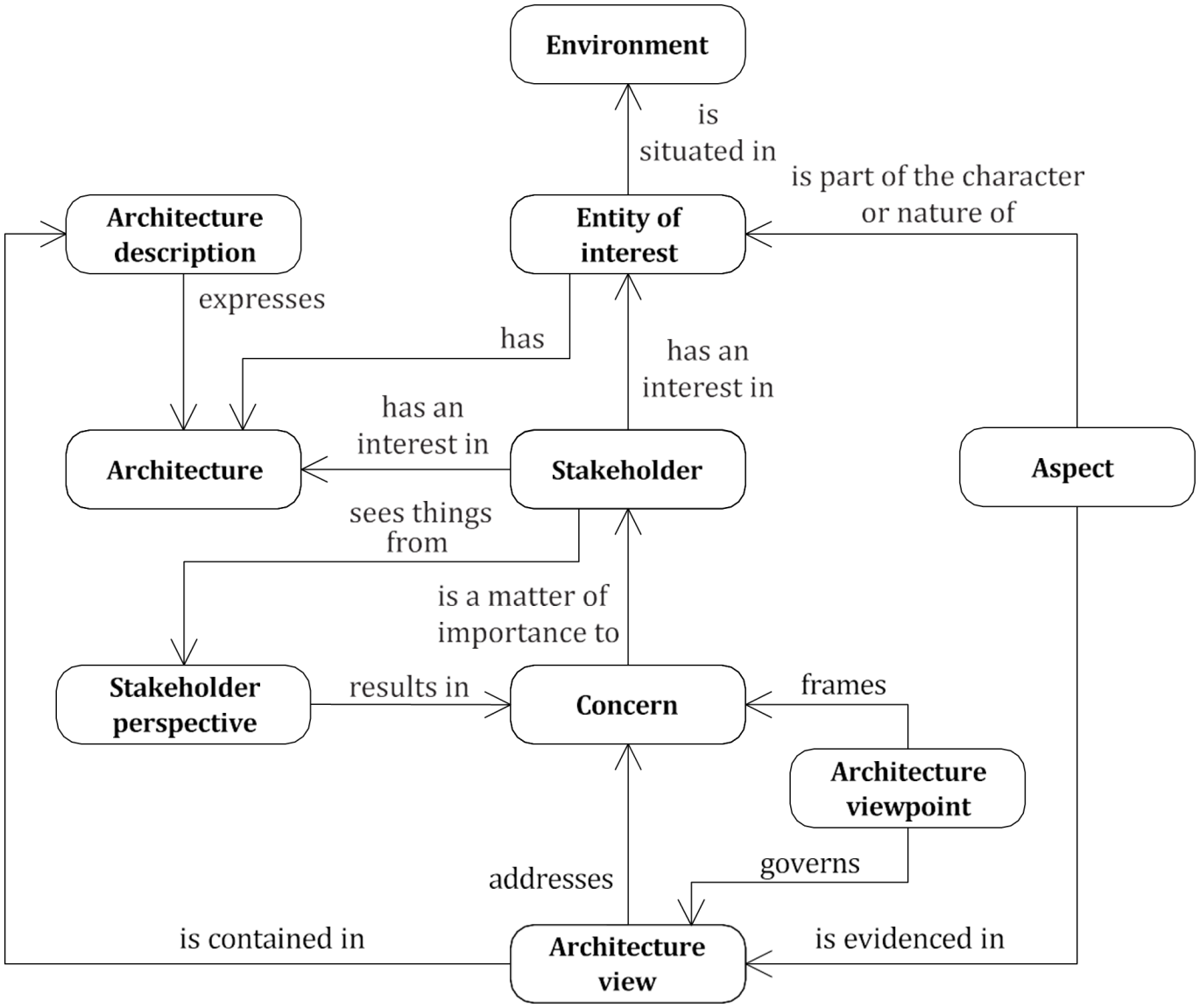


Figure 13-1: Conceptual Model of an Architecture Description [ISO/IEC/IEEE 42010]

The ArchiMate language with the viewpoint mechanism described in Section 13.4 assists and guides the architect in the definition and classification of governing viewpoints. The architect will use this mechanism in their work to construct and design views for stakeholder communication.

13.3. Architecture Views and Viewpoints

Architecture 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 an architecture 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.

Architecture viewpoints are a means to focus on particular aspects and domains of the architecture. These aspects and domains 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 domains 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 domains. What is and what is not shown in an architecture 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.

13.4. Viewpoint Mechanism

An architect is confronted with many different types of stakeholders and concerns. To help 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 13-2 shows how the viewpoint mechanism is used to create views addressing stakeholder concerns.

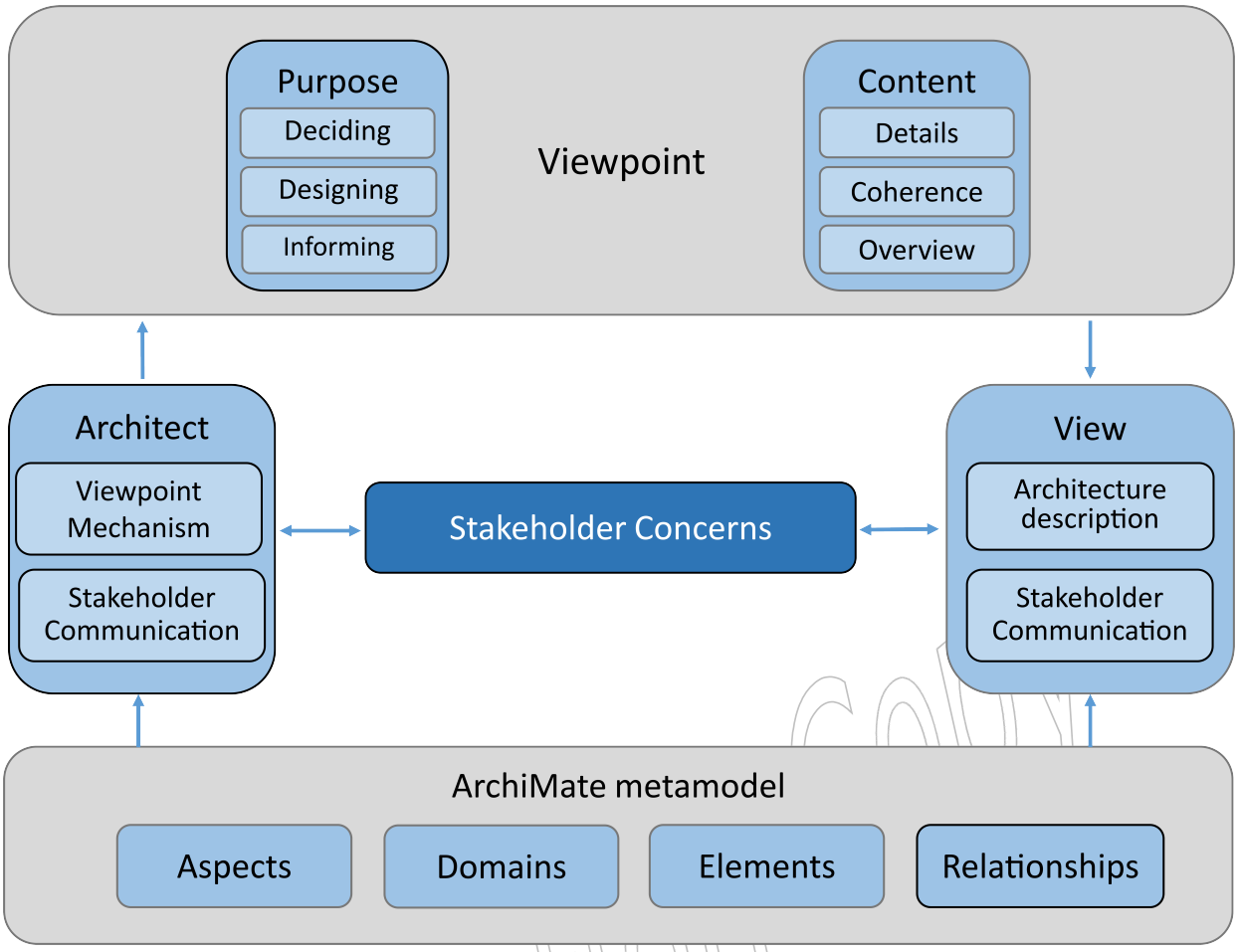


Figure 13-2: Framing Stakeholder Concerns using the Viewpoint Mechanism

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.

13.4.1. Defining and Classifying Viewpoints

To help define and classify 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.

- *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 language to select relevant aspects and domains. This is supported by the following three categories:

- *Details*: Views on the detailed level typically consider one domain and one aspect from the ArchiMate core language

  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 domains or multiple aspects are spanned

  Extending the view to more than one domain or aspect enables the stakeholder to focus on architecture relationships like process-uses-system (multiple domain) 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 domains and multiple aspects

  Typically, such overviews are addressed to Enterprise Architects and decision-makers, such as CEOs and CIOs.

13.4.2. 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 14.1 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.

13.5. Example Viewpoints

See Appendix C for a set of example viewpoints.

Chapter 14. 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 12, 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[1] mechanisms, specialized, or domain-specific purposes, such as:

- Support for specific types of model analysis
- Support for 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 toward 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.

14.1. Adding Attributes to ArchiMate Concepts

As stated earlier in this document, the ArchiMate language contains only the elements and relationships that are necessary for general architecture modeling. However, users might want to perform model-based performance or cost calculations to attach supplementary information (textual, numerical, etc.) to the model concepts. Every concept in an ArchiMate model can have attributes attached to it. ArchiMate concepts can be enriched in a generic way with supplementary information by means of a “profiling” specialization mechanism (see also [Eertink et al., 1999]).

A *profile* is a data structure which can be defined separately from the ArchiMate language but can be dynamically coupled with concepts; i.e., the user of the language is free to decide whether and when the assignment of a profile to a model concept 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:

1. *Predefined 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 concepts that have to be specified in order to execute common types of analysis.

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

At least the following basic data types are allowed for these attributes:

- String
- Integer
- Real
- Boolean
- Currency
- Date
- URL

In addition, the following complex types are supported:

- Structure, consisting of one or more fields of a basic type
- List, consisting of elements of one of the other types

The ArchiMate Model Exchange File Format [C19C] defines how these types are encoded and exchanged.

Examples

Table 14-1 shows possible profiles with input attributes needed for certain types of cost and performance analysis of architecture models [Jonkers & Iacob, 2009]. 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., role, business actor, application component, device) may have fixed and variable costs and a capacity.

Table 14-1: Profile Examples

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

14.2. Specialization of Concepts

The specification of the ArchiMate language metamodel uses the notion of specialization to define that some concept is a special version of another. This tree of specializations has the abstract concept “Concept” at the root, and “Element” is a specialization of that. The leaves of this tree are the concrete concepts used by modelers in actual models. (Note that this specialization of concepts at the language definition level is not to be confused with the specialization relationship used at the model level, where you can model that an individual model element represents a specialization of another model element; e.g., “Car Insurance” being a specialization of “Insurance”.) This metamodel-level specialization is not accessible to users of the language, but is only used in its definition; otherwise it would be possible to add arbitrary concepts anywhere in the abstract language structure and we would no longer have a standard modeling language.

However, the ArchiMate Specification does support the specialization of its concrete concepts (i.e., the ones used in models). This allows you to define, for instance, Social Network as a special version of the concept “Collaboration”, if you want to distinguish such collaborations of social actors in your models. This does not take you outside the boundaries of the language itself; it merely lets you add a level of refinement to existing concepts. This is sometimes called “stereotyping” of concepts.

Specialized elements inherit the properties of the elements they specialize (including the relationships that are allowed for the element). Also, new graphical notation can 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. For a specialized concept, certain attributes may also 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 concepts 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 concepts are used.

Specialization of concepts is done by using the profile mechanism described in Section 14.1. 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 general 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»").

14.2.1. Examples of Specializations of Common Domain Elements (Informative)

Table 14-2 shows examples of specializations of Common Domain elements.

Table 14-2: Example Specializations of Common Domain Elements

Parent Concept	Specialized Concept	Description
Role	Business Role	Represents the responsibility for performing specific behavior, to which a business actor can be assigned, or the part an actor plays in a particular action or event.
Collaboration	Business Collaboration	Represents an aggregate of two or more business internal active structure elements that work together to perform collective behavior.
	Application Collaboration	Represents an aggregate of two or more application internal active structure elements that work together to perform collective application behavior.
	Technology Collaboration	Represents an aggregate of two or more technology internal active structure elements that work together to perform collective technology behavior.
	Social Network	A social structure made up of social actors (individuals or organizations) and the connections between these actors.

Parent Concept	Specialized Concept	Description
Service	Business Service	Represents explicitly defined behavior that a business role, business actor, or business collaboration exposes to its environment.
	Application Service	Represents an explicitly defined exposed application behavior.
	Technology Service	Represents an explicitly defined exposed technology behavior.
	Business Decision	A conclusion that a business arrives at through business logic and which the business is interested in managing.
	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.
Process	Business Process	Represents a sequence of business behaviors that achieves a specific result such as a defined set of products or business services.
	Application Process	Represents a sequence of application behaviors that achieves a specific result.
	Technology Process	Represents a sequence of technology behaviors that achieves a specific result.
	Business Interaction	Represents a unit of collective business behavior performed by (a collaboration of) two or more business actors, business roles, or business collaborations.
	Application Interaction	Represents a unit of collective application behavior performed by (a collaboration of) two or more application components.
	Technology Interaction	Represents a unit of collective technology behavior performed by (a collaboration of) two or more technology internal active structure elements.
	Activity	Atomic internal behavior element (at the considered abstraction level) that will not be decomposed any further.

Parent Concept	Specialized Concept	Description
Function	Business Function	Represents a collection of business behavior based on a chosen set of criteria such as required business resources and/or competencies, and is managed or performed as a whole.
	Application Function	Represents automated behavior that can be performed by an application component.
	Technology Function	Represents a collection of technology behavior that can be performed by a technology internal active structure element.
Event	Business Event	Represents a business-related state change.
	Application Event	Represents an application state change.
	Technology Event	Represents a technology state change.
	Implementation Event	Represents a state change related to implementation or migration.
	Threat Event (Risk & Security Overlay)	Event with the potential to adversely impact an asset. An *attack* 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.

14.2.2. Examples of Specializations of Business Domain Elements (Informative)

Table 14-3 shows examples of specializations of Business Domain concepts.

Table 14-3: Example Specialization of Business Domain Elements

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	Any 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 Object	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.

14.2.3. Examples of Specializations of Application Domain Elements (Informative)

Table 14-4 shows examples of specializations of Application Domain elements.

Table 14-4: Example Specializations of Application Domain 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 (GUI)	On-screen interface (GUI) with which a human user can interact with an application component.

14.2.4. Examples of Specializations of Technology Domain Elements (Informative)

Table 14-5 shows examples of specializations of Technology Domain elements.

Table 14-5: Example Specializations of Technology Domain 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	Wi-Fi Network	Wireless Local Area Network (WLAN).
	Wide Area Network	Long-range data communication network.
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.
	Representation	Represents a perceptible form of the information carried by a business object.
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.

14.2.5. Examples of Specializations of Motivation Elements (Informative)

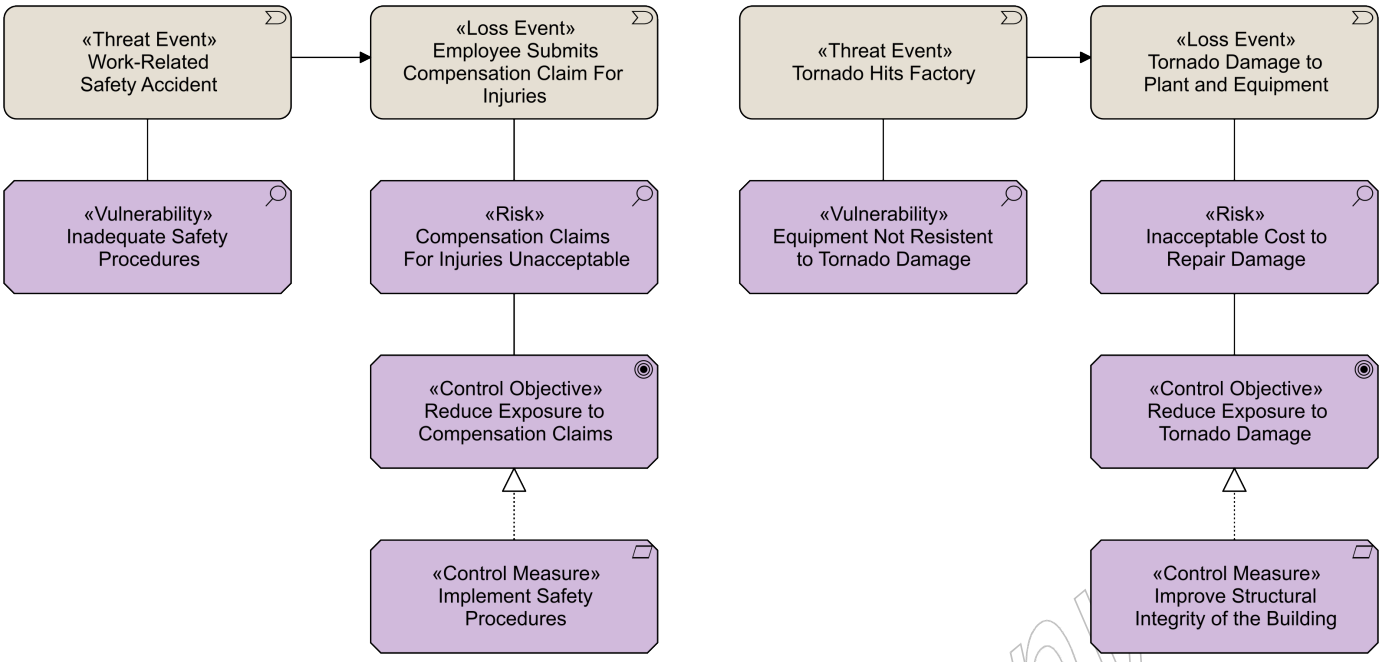
Table 14-6 shows examples of specializations of motivation elements.

Table 14-6: Example 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 toward 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.
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.
	Constraint	Represents a limitation on aspects of the architecture, its implementation process, or its realization.

Example

Example 14-1 illustrates the use of specializations of Business Domain 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 14-1: Specializations of Business Domain and Motivation Elements

14.2.6. Examples of Specializations of Strategy Elements (Informative)

Table 14-7 shows examples of specializations of strategy elements.

Table 14-7: Example 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.

14.2.7. Examples of Specializations of Implementation and Migration Elements (Informative)

Table 14-8 shows examples of specializations of implementation and migration elements.

Table 14-8: Example 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.
Deliverable	Gap	Represents a statement of difference between two plateaus.

14.2.8. Examples of Specializations of Composite Elements (Informative)

Table 14-9 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.

Table 14-9: Example Specializations of Composite 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, Function, and Data Object	Data Store	A repository for persistently storing and managing collections of data.

14.2.9. Examples of Specializations of Relationships and Junctions (Informative)

Table 14-10 shows examples of specializations of relationships and junctions.

Table 14-10: Example Specializations of Relationships and Junctions

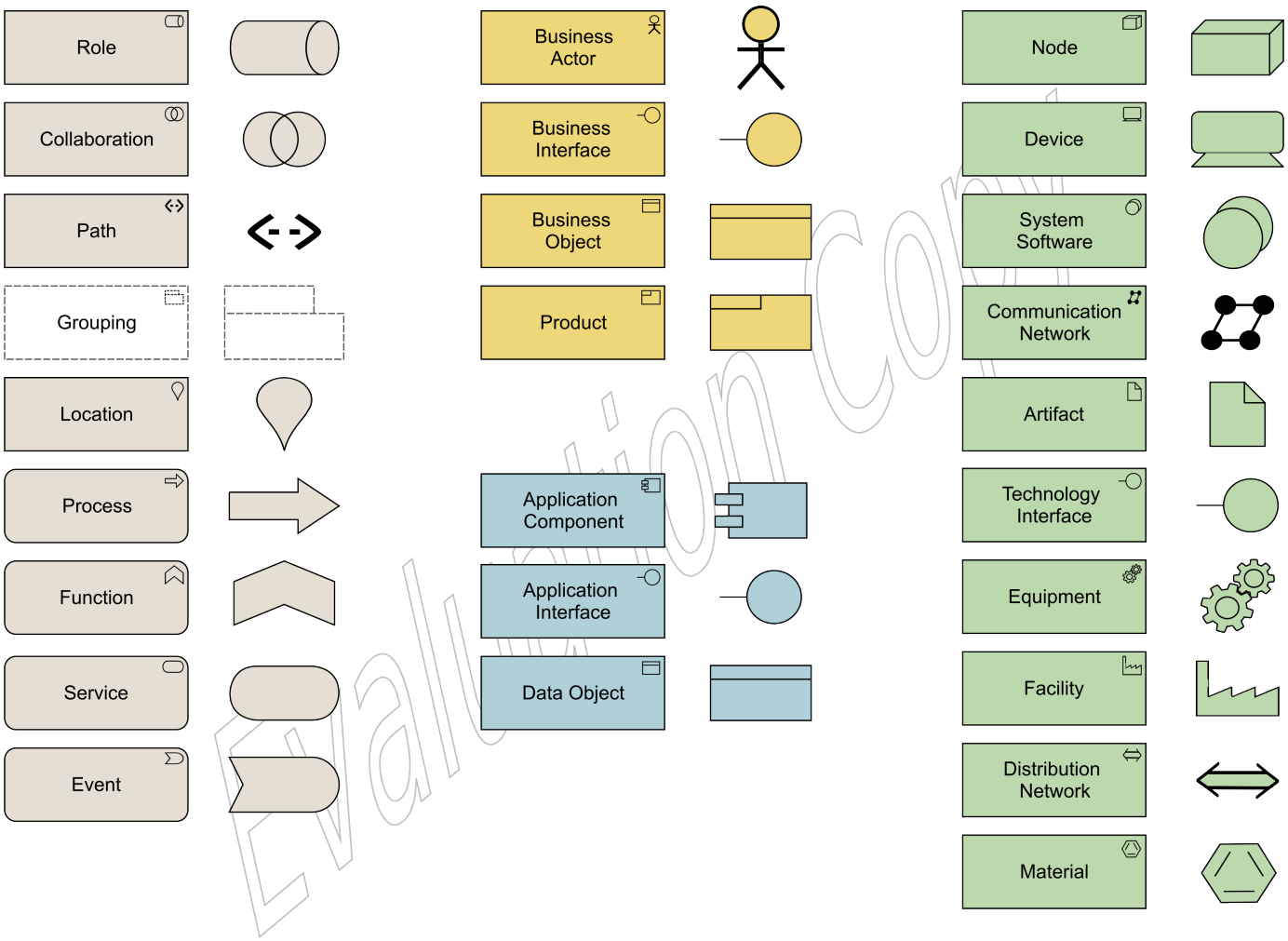
Parent Concept	Specialized Concept	Description
Flow	Money Flow	A flow of money between behavior elements.
Assignment	Responsibilities Assignment	Assignment from a business actor to a role.
	Behavior Assignment	Assignment from an active structure to a behavior element.
Or-junction	Or-join	A junction with 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.

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

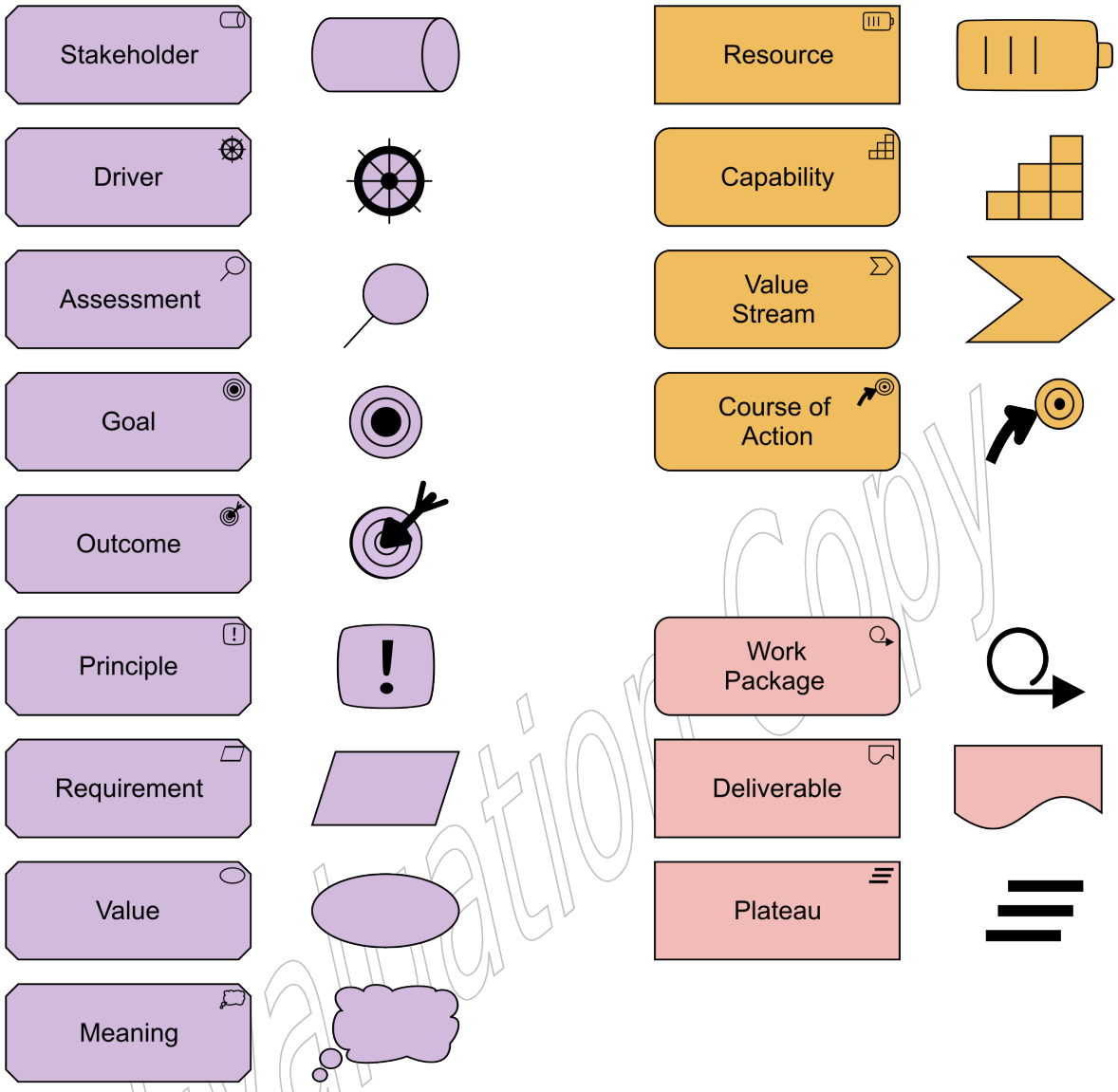
Appendix A: Summary of Language Notation

This appendix describes the default iconography of the ArchiMate language. Modelers can choose to use a different iconography on any architecture view if it helps to communicate better with stakeholders. However, it is 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.

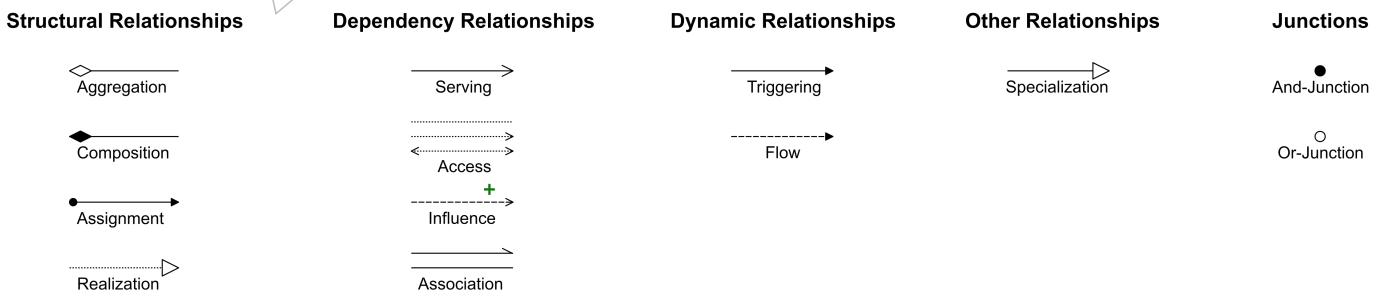
A.1. Core Elements



A.2. Motivation, Strategy, Implementation and Migration Elements



A.3. Relationships and Junctions



Appendix B: Relationships (Normative)

This appendix details the normative requirements for relationships between elements of the ArchiMate modeling language. This is mainly intended for tool implementation purposes.

B.1. Specification of Derivation Rules

The following sections specify the formal rules for deriving relationships. The input relationships used for derivation must be allowed by the tables in this appendix. The resulting relationship will then always be allowed by definition and is listed in these tables as well. Note that these derivation rules do not work on relationships between core elements and other elements such as motivation, strategy, or implementation and migration elements, with the exception of the realization and influence relationships. This appendix states in more detail the restrictions that were applied to the use of the derivation rules to arrive at the relationship tables. Applying these rules and restrictions together results in the tables in this appendix, which contain all allowed relationships in the language.

We distinguish between two types of derivations: those that are certainly true in any model where these rules apply, and those that are potentially true but uncertain, depending on the specifics of the model concerned.

Notation of Derivation Rules

In the description of the derivation rules, a shorthand is used to describe relations: $p(a,b)$: R is used to describe the relationship with name p that has concept a as source, concept b as target, and R as its relationship type.

The source and target concepts may be of any type. The relationship type can be restricted by the definition.

By convention, concepts are named a, b, and c in order of appearance, relationships are named p, q, and r in order of appearance, and relationship types are named S, T, and U in order of appearance.

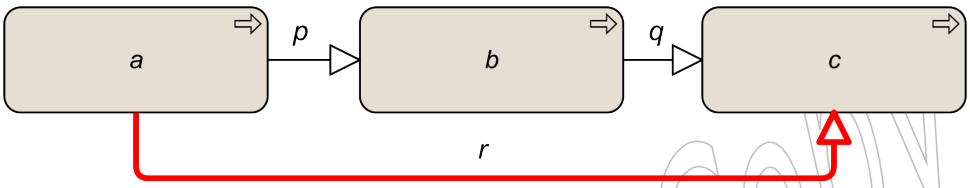
B.2. Derivation Rules for Valid Relationships

This section states the Derivation Rules (DRs) for derivations that are valid in any model where these rules apply.

B.2.1. Valid Derivations for Specialization Relationships

DR 1: Transitivity of Specialization

If two relationships $p(a,b) : S$ and $q(b,c) : S$ exist, with S being *Specialization*, then a relationship $r(a,c) : S$ can be derived.



Example B-1: Transitivity of Specialization

B.2.2. Valid Derivations for Structural Relationships

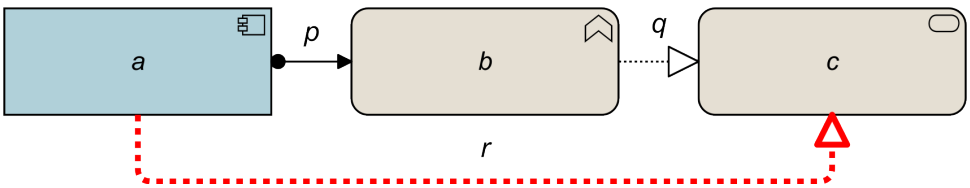
The structural relationships can be ordered by "strength":

- Realization (weakest)
- Assignment
- Aggregation
- Composition (strongest)

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

DR 2: Derivation Between Structural Relationships

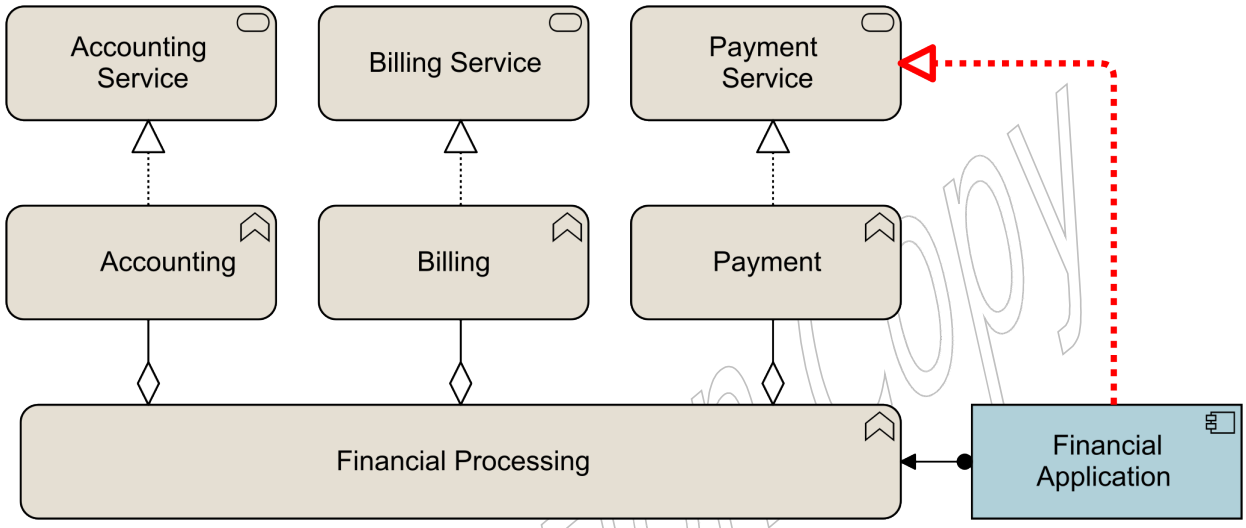
If two relationships $p(a,b) : S$ and $q(b,c) : T$ exist, with S and T being structural relationships, then a relationship $r(a,c) : U$ can be derived, with U being the weakest of S and T.



Example B-2: Derivation of Structural Relationships

Informally, this means that if two structural relationships are “in line” (the target of one relation joins at the source of the other relation) they can be replaced by the weakest of the two. Transitively applying this property allows us to replace a “chain” of structural relationships that are in line (with intermediate model elements) by the weakest structural relationship in the chain.

In Example B-3, assume that the goal is to omit the functions, sub-functions, and services from the model. In this case, an indirect realization relationship (the relationship labeled “Derived Relationship” (thick arrow on the right) can be derived from “Financial Application” to the “Payment Service” (from the chain assignment – aggregation – realization).



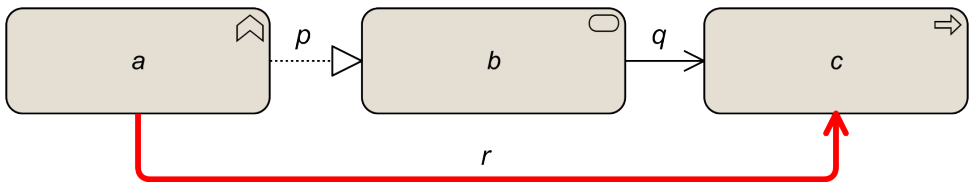
Example B-3: Derivation from a Chain of Structural Relationships

B.2.3. Valid Derivations for Dependency Relationships

Part of the language definition is an abstraction rule that states that a structural relationship and a dependency relationship that join at an intermediate element under certain conditions can be combined and replaced by the dependency relationship. This rule is split into two parts for both the source and target side of the dependency.

DR 3: Derivation Between Structural and Dependency Relationships

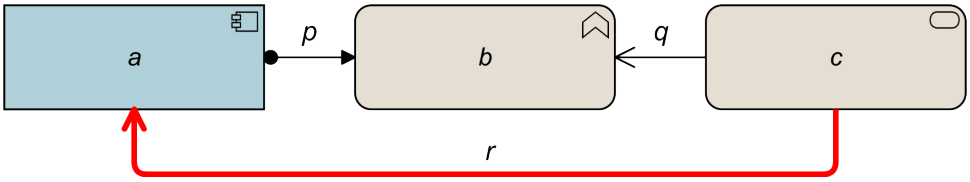
If two relationships $p(a,b) : S$ and $q(b,c) : T$ exist, with S being a structural relationship and T being a dependency relationship, then a relationship $r(a,c) : T$ can be derived.



Example B-4: Derivation from a Dependency and a Structural Relationship in Line

DR 4: Derivation Between Opposing Structural and Dependency Relationships

If two relationships *p(a,b)* : *S* and *q(c,b)* : *T* exist, with *S* being a structural relationship and *T* being a dependency relationship, then a relationship *r(c,a)* : *T* can be derived.



Example B-5: Derivation from a Dependency and a Structural Relationship in the Opposite Direction

These rules may be combined with the derivation rule for structural relations (DR2), allowing to replace a "chain" of structural relationships and a dependency relationship (with intermediate model elements) by the dependency relationship in the chain, given that the chain does satisfy the restrictions for structural and dependency relationships. Informally, this means that the beginning and/or endpoint of a dependency relationship can be transferred "backwards" in a chain of elements connected by structural relationships.

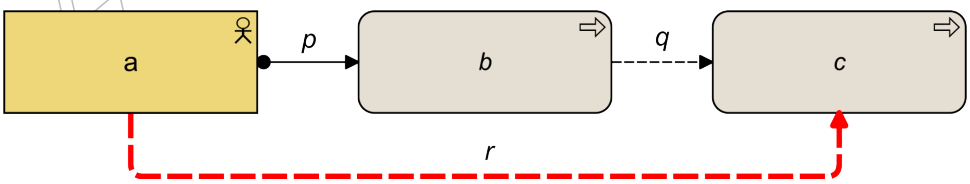
B.2.4. Valid Derivations for Dynamic Relationships

Part of the language definition is an abstraction rule that states that a structural relationship and a dynamic relationship that join at an intermediate element under certain conditions can be combined and replaced by the dynamic relationship. This rule is split into a generic rule and rules specific for flow and triggering.

For the two dynamic relationships, the following rules apply.

DR 5: Derivation Between Structural and Dynamic Relationships

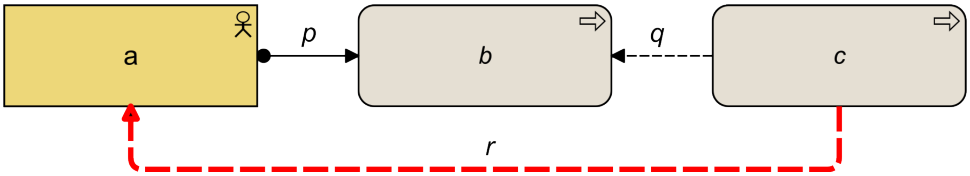
If two relationships *p*(*a,b*) : *S* and *q*(*b,c*) : *T* exist, with *S* being a structural relationship and *T* being a dynamic relationship, then a relationship *r*(*a,c*) : *T* can be derived.



Example B-6: Derivation from a Dynamic and a Structural Relationship in Line

DR 6: Derivation Between Structural and Flow Relationships

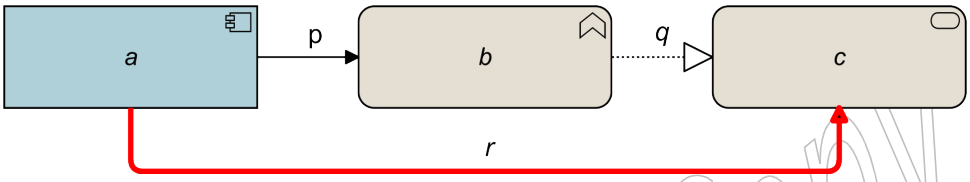
If two relationships *p*(*a,b*) : *S* and *q*(*c,b*) : *T* exist, with *S* being a structural relationship and *T* being *Flow,* then a relationship *r*(*c,a*) : *T* can be derived.



Example B-7: Derivation from a Flow and a Structural Relationship in the Opposite Direction

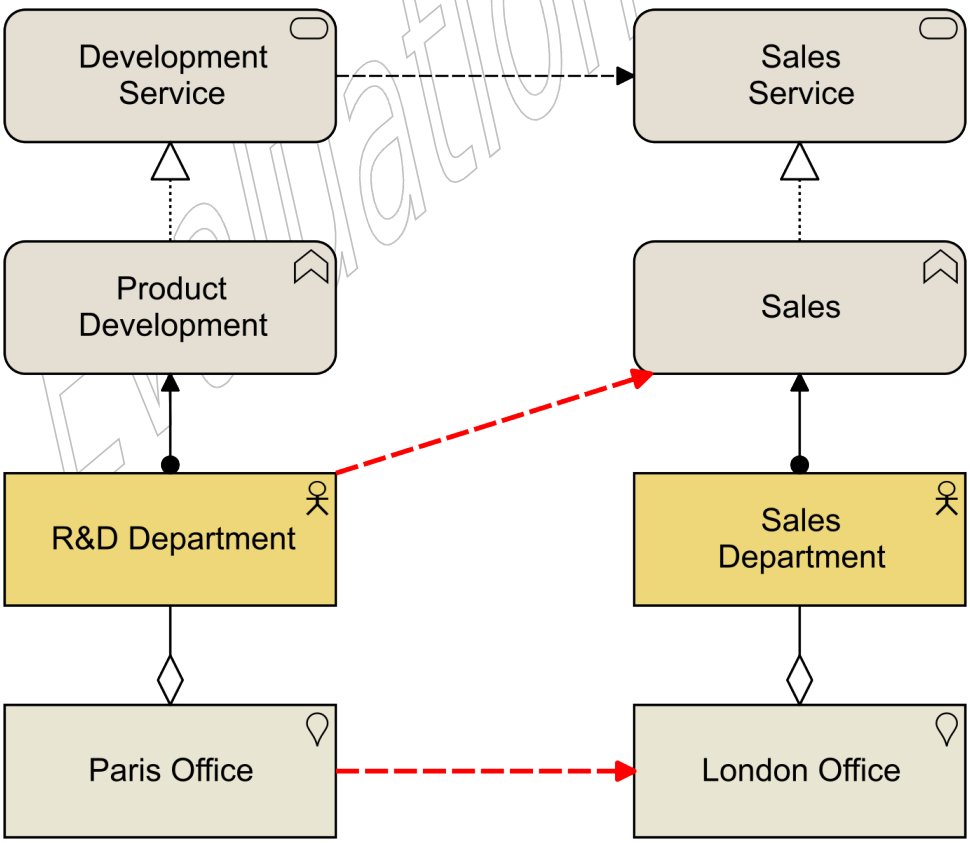
DR 7: Derivation Between Structural and Triggering Relationships

If two relationships $p(a,b) : S$ and $q(b,c) : T$ exist, with S being *Triggering* and T being a structural relationship, then a relationships $r(a,c) : S$ can be derived.



Example B-8: Derivation from a Triggering and a Structural Relationship in Line

These rules can be applied repeatedly. Informally, this means that the beginning and/or endpoint of a flow relationship can be transferred "backwards" in a chain of elements connected by structural relationships. Example B-9 shows two of the possible flow relationships that can be derived with these rules, given a flow relationship between the two services.

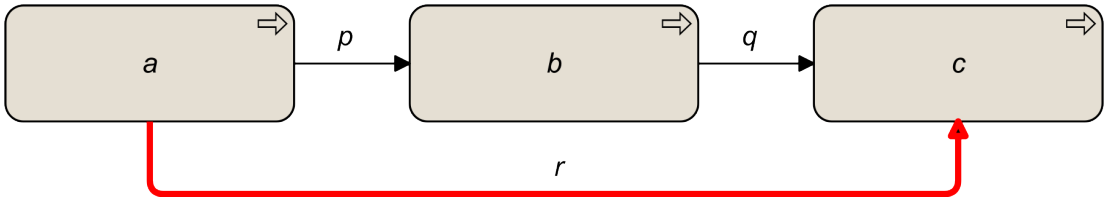


Example B-9: Derivation from Dynamic Relationships

Moreover, triggering relationships are *transitive*, as expressed in the next rule.

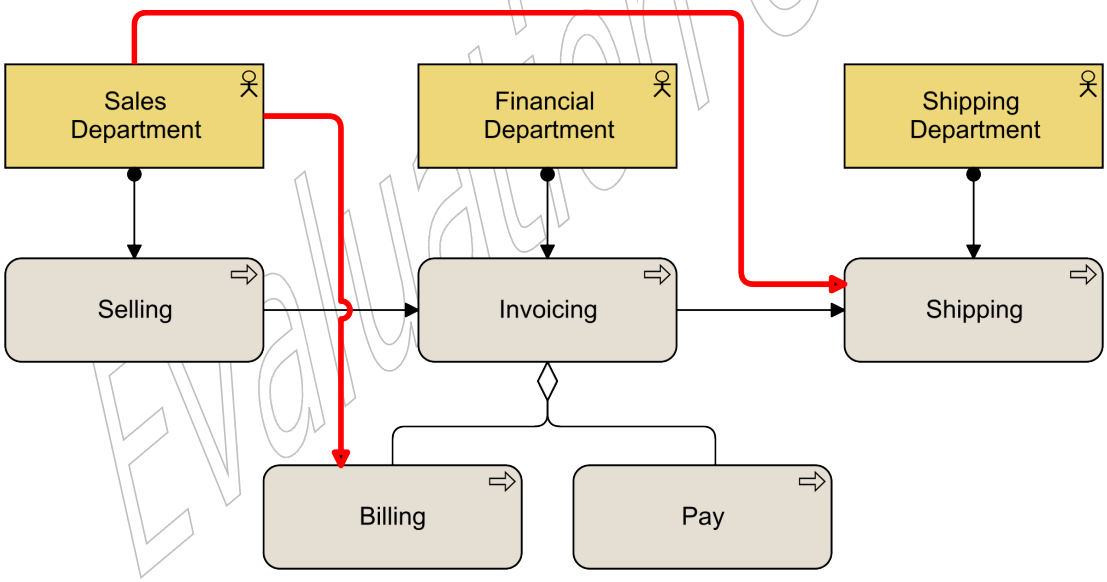
DR 8: Derivation Between Triggering Relationships

> If two relationships $p(a,b) : S$ and $q(b,c) : S$ exist, with S being *Triggering*, then a relationship $r(a,c) : S$ can be derived.



Example B-10: Derivation from Triggering Relationships

This rule may be combined with the rules for deriving dynamic relations and structural relationships, thus allowing discovery of triggering relations. Example B-11 shows how the "Sales Department" is assigned to a process "Selling" that triggers a process "Invoicing", which aggregates the processes "Billing" and "Payment". "Invoicing" in turn triggers the process "Shipping", to which the "Shipping Department" is assigned. The derivation rules allow that the "Sales Department" triggers the "Shipping" process, but also the process "Billing".



Example B-11: Derivation from Triggering and Structural Relationships

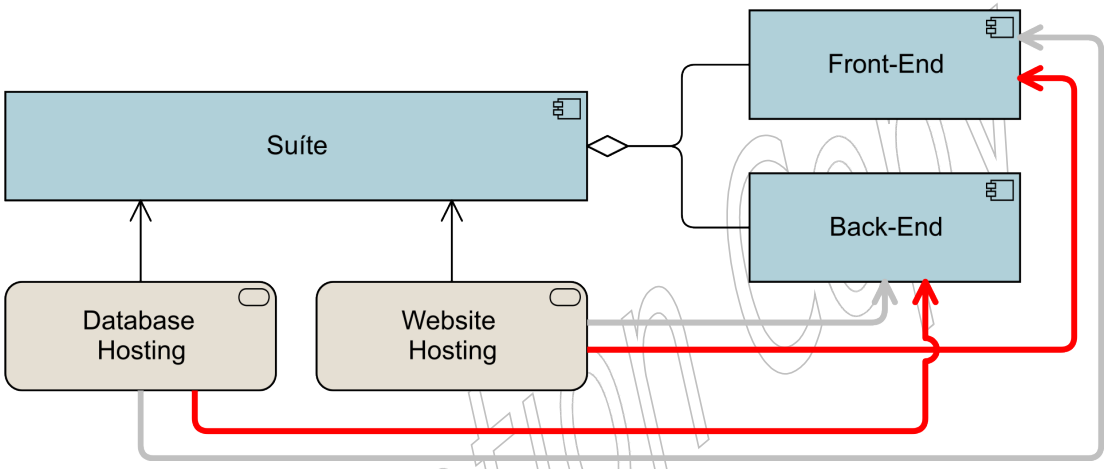
B.3. Derivation Rules for Potential Relationships

The derivation rules defined so far lead to relationships that are valid with high certainty. If a model is well designed and describes a stable state of the enterprise, these derived relationships can be trusted.

This section describes derivation rules for relationships with lower certainty. They might be relevant but may also be wrong, depending on the specifics of the model. It is up to the modeler to decide on this.

The derivation rules for potential relationships, or Potential Derivation Rules (PDRs), are used to enrich the metamodel with relationships that otherwise would not be allowed and can be used to discover relationships in a model that otherwise might not show.

Example B-12 shows a potential derivation in which some relationships are valuable, and some are not. In this example, an architect first modeled an application component called “Suite” that uses two infrastructure services called “Website Hosting” and “Database Hosting”. Later, the application component “Suite” was detailed by adding two aggregated application components “Front-End” and “Back-End”. The architect in this case did not reconsider the serving relations. Potential derivation rule PDR 5 allows the red and gray relationships. In this case, the architect determines that only the red relationships are valuable.



Example B-12: Examples of Potential Derivation

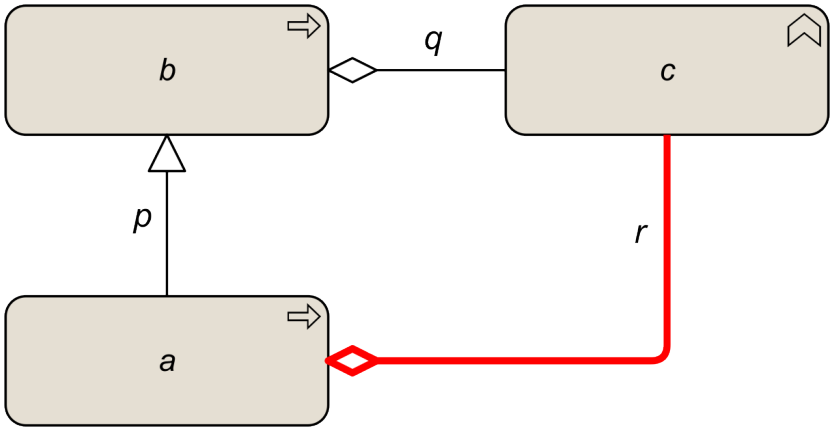
B.3.1. Potential Derivation for Specialization Relationships

Part of the language definition is a rule that states that every relation from or to a generic element is inherited by the specialized element. This leads to a number of potential derivations.

The first two rules apply in the case where the target of a specialization relationship is the source or target of any other relationship. In this case, the source of the specialization *could* have the same relationship.

PDR 1: Derivation with Specialization and Other Relationships

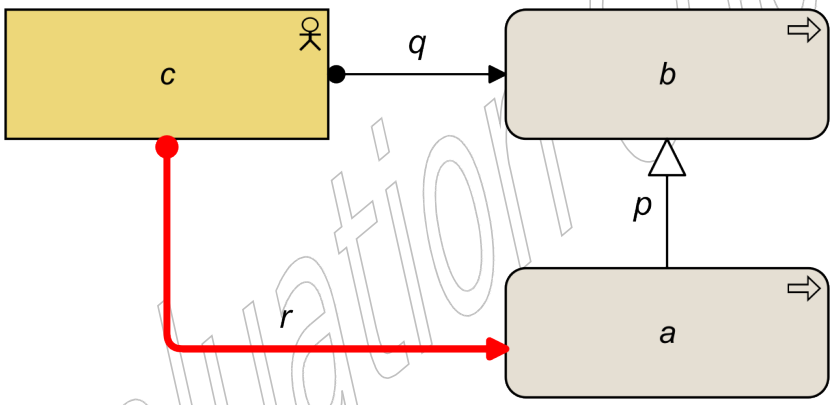
If two relationships $p(a,b) : S$ and $q(b,c) : T$ exist, with S being *Specialization* and T being a structural, dependency, or dynamic relationship, then a relationship $r(a,c) : T$ might be derived.



Example B-13: Potential Derivation from a Specialization and Another Relationship in Line

PDR 2: Derivation with Specialization and Other Relationships

If two relationships $p(a,b) : S$ and $q(c,b) : T$ exist, with S being *Specialization* and T being a structural, dependency, or dynamic relationship, then a relationship $r(c,a): T$ might be derived.

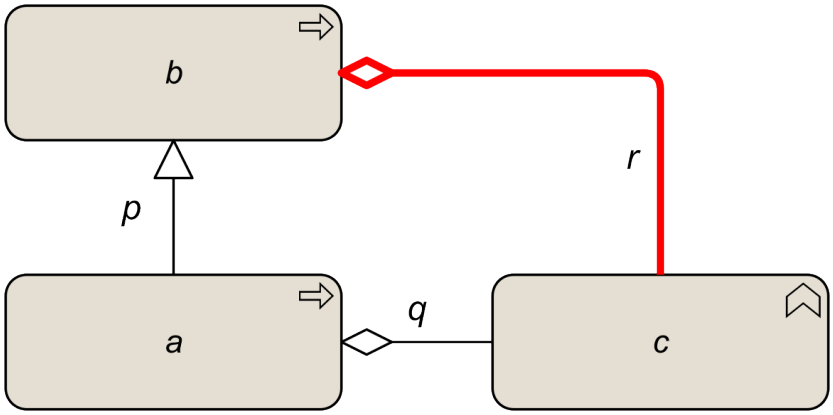


Example B-14: Potential Derivation from a Specialization and Another Relationship in the Opposite Direction

The next two rules apply in the case where the source of a specialization relationship is joining the source or target of any other relationship. In this case, the target of the specialization *could* have the same dependency.

PDR 3: Potential Derivation Between Specialization and Any Other Relationship

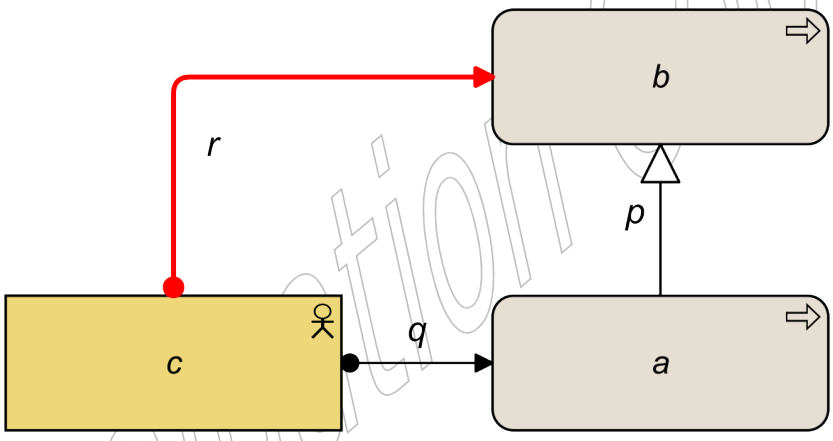
If two relationships $p(a,b) : S$ and $q(a,c) : T$ exist, with S being *Specialization* and T being a structural, dependency, or dynamic relationship, then a relation $r(b,c) : T$ might be derivable.



Example B-15: Potential Derivation from Another Relationship and a Specialization in the Opposite Direction

PDR 4: Potential Derivation Between Specialization and Any Other Relationship

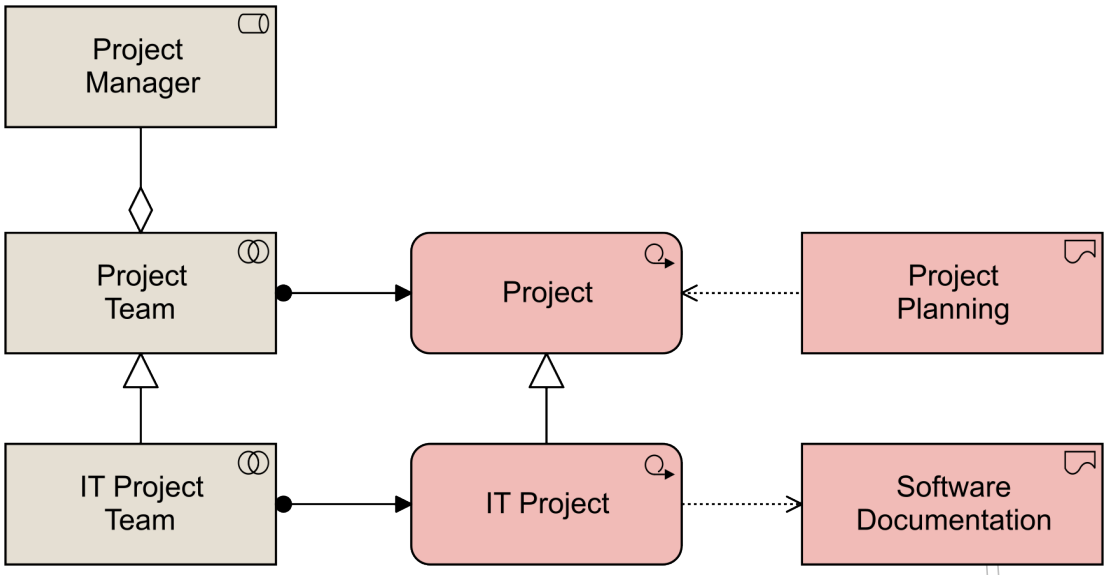
If two relationships $p(a,b) : S$ and $q(c,a) : T$ exist, with S being *Specialization* and T being a structural, dependency, or dynamic relationship, then a relationship $r(c,b) : T$ might be derivable.



Example B-16: Potential Derivation from Another Relationship and a Specialization in Line

The potential relationships derived with the rules from this section may vary a lot in likelihood, depending on the direction of the derivation (from generalized to specialized element or *vice versa*), the type of relationship, and the specific interpretation of the relationship. Also, a chain of multiple potential derivations usually leads to a lower probability.

Consider a model, as shown in Example B-17, with a “Project Team” assigned to a “Project”, and an “IT Project Team”, as a specialization of “Project Team”, assigned to an “IT Project”, as a specialization of “Project”. A “Project Team” aggregates a “Project Manager”, a “Project” accesses (reads) “Project Planning”, and an “IT Project” accesses (writes) “Software Documentation”).



Example B-17: Specializations Used in Potential Derivations

- Aggregation relationships often lead to derivations that are (almost) certain when moving the source of the relationship to a specialized element (PDR 1), or the target of the relationship to a generalized element (PDR 4)

  In this example, “IT Project Team” aggregates “Project Manager” is a derived relationship that is almost certain.

- For the assignments, this depends on the perspective: “IT Project Team” assigned to “Project” (PDR 1) is probably true in the sense that the “IT Project Team” always performs a “Project”, but uncertain in the sense that a “Project” is not always performed by an “IT Project Team” (e.g., if it is a business project)

  For “Project Team” assigned to “IT Project” (PDR 2), this is the other way around.

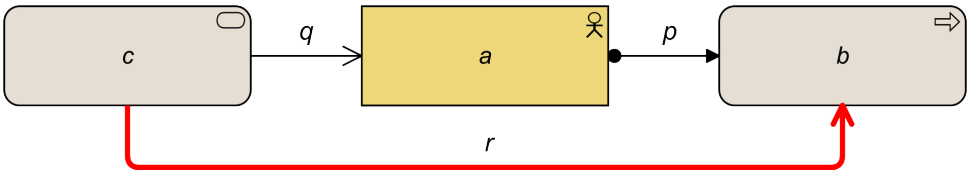
- “IT Project” accesses (reads) “Project Planning” (PDR 1) is almost certainly a derived relationship, while “Project” accesses (writes) “Software Documentation” (PDR 3) is only valid for a subset of “Projects”

B.3.2. Potential Derivation for Structural and Dependency Relationships

The next two rules apply in the case where a structural and dependency relation are joining at the source of the structural relation. In this case, the target of the structural relation *could* have the same dependency.

PDR 5: Potential Derivation Between Structural and Dependency Relationships

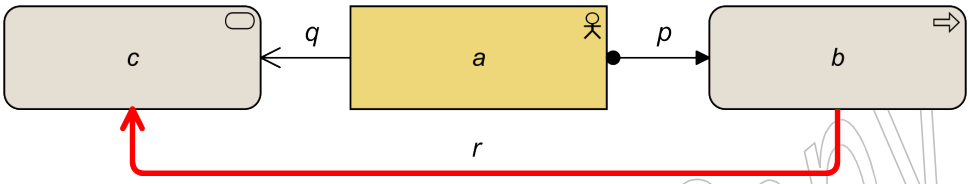
If two relationships $p(a,b) : S$ and $q(c,a) : T$ exist, with S being a structural relationship and T being a dependency relationship, then a relationship $r(c,b) : T$ might be derivable.



Example B-18: Potential Derivation from a Dependency and a Structural Relationship in Line

PDR 6: Potential Derivation Between Structural and Dependency Relationships

If two relationships $p(a,b) : S$ and $q(a,c) : T$ exist, with S being a structural relationship and T being a dependency relationship, then a relationship $r(b,c) : T$ might be derivable.



Example B-19: Potential Derivation from a Dependency and a Structural Relationship in the Opposite Direction

B.3.3. Potential Derivation for Dependency Relationships

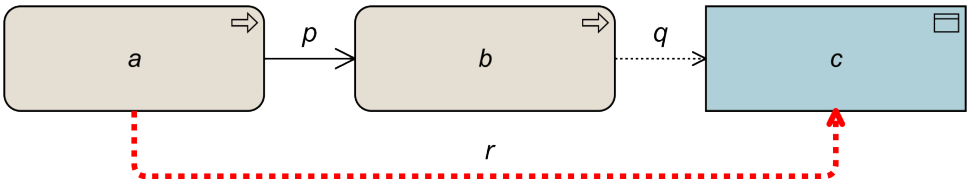
The next rule applies in the case where two dependency relationships are joining at an intermediate element. In this case, the two relations *could* be replaced by one, being the weaker of the two.

The dependency relationships are ordered by "strength":

- Association (weakest)
- Influence
- Access
- Serving (strongest)

PDR 7: Potential Derivation Between Dependency Relationships

If two relationships $p(a,b) : S$ and $q(b,c) : T$ exist, with S and T being a *Dependency Relationship*, then a relationship $r(a,c) : U$ might be derivable, with U being the weakest of S and T.



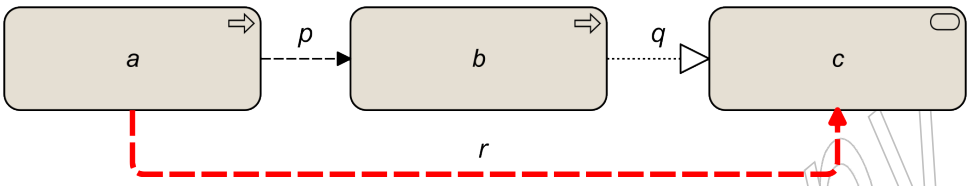
Example B-20: Potential Derivation from Two Dependency Relationships

B.3.4. Potential Derivation for Dynamic Relationships

The next rules apply in the case where a structural and dynamic relation are joining at an intermediate element. In this case, the target of the structural relation *could* have the same dependency.

PDR 8: Potential Derivation Between Structural and Dynamic Relationships

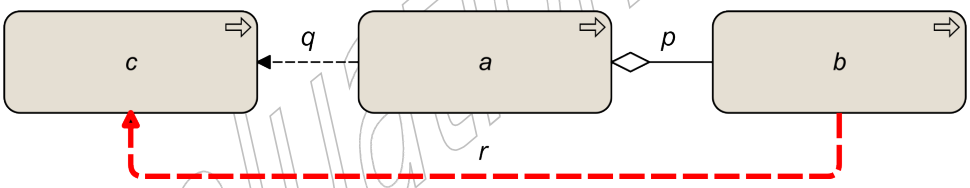
If two relationships $p(a,b) : S$ and $q(b,c) : T$ exist, with S being *Flow* and T being a structural relationship, then a relation $r(a,c) : S$ might be derivable.



Example B-21: Potential Derivation from a Dynamic and a Structural Relationship in Line

PDR 9: Potential Derivation Between Structural and Dynamic Relationships

If two relationships $p(a,b) : S$ and $q(a,c) : T$ exist, with S being a structural relationship and T being a dynamic relationship, then a relationship $r(b,c) : T$ might be derivable.

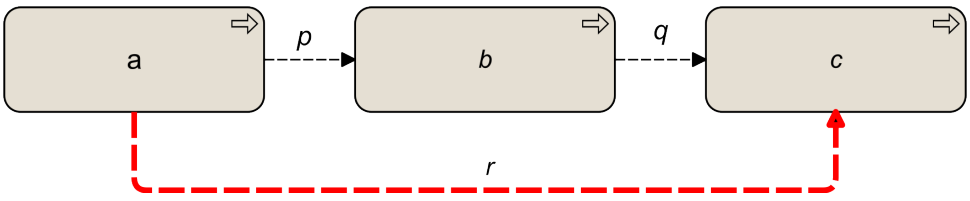


Example B-22: Potential Derivation from a Dynamic and a Structural Relationship in the Opposite Direction

The next rule applies in the case where two flow relationships join at an intermediate element. In this case, the flow relations *could* be replaced by one relation.

PDR 10: Potential Derivation Between Flow Relationships

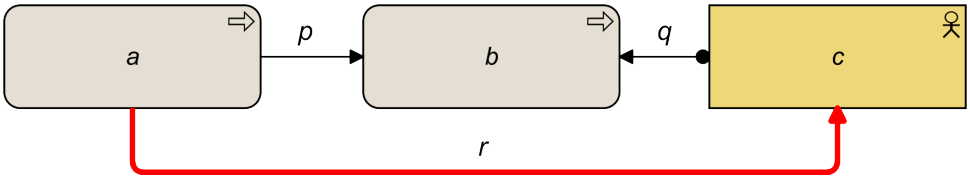
If two relationships $p(a,b) : S$ and $q(b,c) : S$ exist, with S being *Flow,* then a relationship $r(a,c) : S$ might be derivable.



Example B-23: Potential Derivation from Two Flow Relationships

PDR 11: Potential Derivation Between Triggering and Structural Relationships

If two relationships $p(a,b) : S$ and $q(c,b) : T$ exist, with S being a Triggering relationship and T being a structural relationship, then a relation $r(a,c) : S$ might be derivable.



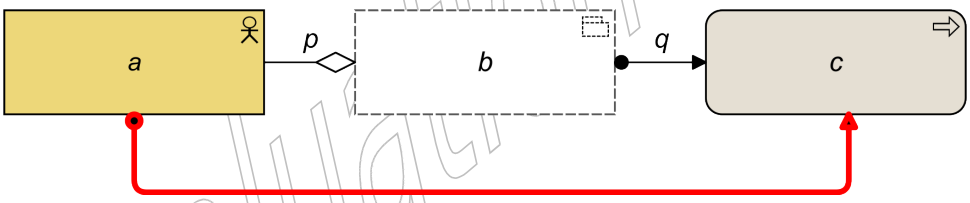
Example B-24: Potential Derivation from a Triggering and Structural Relationship

B.3.5. Potential Derivation Rule for Grouping

The next rule applies specifically for the grouping element.

PDR 12: Potential Derivation with Elements Aggregated in a Grouping Element

If two relationships $p(b,a) : S$ and $q(b,c) : T$ exist, with S being *Aggregation*, b being a Grouping element and T being a *Realization* or *Assignment,* then a relationship $r(a,c) : T$ might be derivable, only if the metamodel allows T from a to c.



Example B-25: Potential Derivation with Grouping Element

B.4. Restrictions on Applying Derivation Rules

This section describes a number of restrictions that apply when using the derivation rules to infer the set of allowed relationships. In this context, the following are called “domains”:

- Motivation (see Chapter 6)
- Strategy (see Chapter 7)
- Core, which includes the Business Domain (see Chapter 8), Application Domain (see Chapter 9), Technology Domain (see Chapter 10), and the Common Domain (see Chapter 4)
- Implementation and Migration (see Chapter 12)

Given a relation *p*(*a*,*b*) : *S* that can be derived through one of the derivation rules of Section B.2 or Section B.3, then that derivation is not allowed if any of the following is true:

- *a* is in one of the domains *Implementation and Migration*, *Core*, or *Strategy*, and *b* is in the domain *Motivation*, and *S* is not *Assignment*, *Realization*, *Influence*, or *Association*
- *a* is in the domain *Motivation*, and *b* is in one of the domains *Implementation and Migration*, *Core*, or *Strategy,* and *S* is not *Association*
- *a* is in one of the domains *Implementation and Migration* or *Core*, and *b* is in the domain *Strategy,* and *S* is not *Realization* or *Association*
- *a* is in the domain *Strategy*, and *b* is in one of the domains *Implementation and Migration* or *Core*, and *S* is not *Association*
- *a* is in the domain *Implementation and Migration*, and *b* is in the domain *Core,* and *S* is not *Realization* or *Association*
- *a* is in the domain *Core*, and *b* is in the domain *Implementation and Migration*, and *S* is not *Assignment* or *Association*
- *a* is a *Grouping*, *Location* or *Plateau*, and *b* is in the domain *Relationships*, and *S* is not *Aggregation*, or *Association*
- *a* is not a *Grouping*, *Location*, or *Plateau*, and *b* is in the domain *Relationships*, and *S* is not *Association*
- *a* is in the domain *Relationships*, and *S* is not *Association*
- *S* is *Influence* and *b* is not in the domain *Motivation*
- *S* is *Access* and *b* is not a *Passive Structure Element*
- *a* is not a *Passive Structure Element*, and *b* is a *Passive Structure Element*, and *S* is not *Access*, *Assignment*, or *Association*
- *a* is a *Passive Structure Element*, and *b* is a *Passive Structure Element*, and *S* is not *Realization* or *Association*
- *a* is a *Passive Structure Element*, and *b* is not a *Passive Structure Element*, and *S* is not *Realization*, *Influence*, or *Association*

Given a relation *p*(*a*,*b*) : *S* that can be derived through one of the derivation rules of Section B.2 or Section B.3, with *c* being the third element on which the original relations joined (for instance *q*(*a*,*c*) : *T* and *r*(*c*,*b*) : *U*) than that derivation is not allowed if any of the following is true:

- The domain of *c* is different from the domains of both *a* and *b*, unless *a* is in domain *Implementation and Migration*, *c* is in domain *Core*, and *b* is in one of the domains *Motivation* or *Strategy*
- *a* is in domain *Implementation and Migration*, *b* is in one of the domains *Motivation* or *Strategy*, and *c* is *Location* or *Grouping*

Notes:

- These restrictions only apply to derived relationships, not to relationships explicitly defined in the metamodel diagrams, which are allowed by definition
- Product and plateau are composite elements, but can only aggregate the specific concepts depicted in their respective metamodel fragments Figure 8-5 and Figure 12-5

B.5. Relationship Tables

This section provides a set of tables with all allowed relationships. It is constructed from the metamodel figures in Chapter 3 through Chapter 12 and the derivation rules for relationships outlined in Section B.1.

The letters in the tables have the following meaning:

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

Upper case denotes a direct relationship, lower case denotes a derived relationship.

		Motivation								
	From → *To ↓*	Assessment	Driver	Goal	Meaning	Outcome	Principle	Requirement	Stakeholder	Value
Common	Collaboration	O	O	O	O	O	O	O	O	O
	Event	O	O	O	O	O	O	O	O	O
	Function	O	O	O	O	O	O	O	O	O
	Grouping	Ogns	Ogns	Ogns	Ogns	Ognrs	Ognrs	Ognrs	Ogns	Ogns
	Location	O	O	O	O	O	O	O	O	O
	Path	O	O	O	O	O	O	O	O	O
	Process	O	O	O	O	O	O	O	O	O
	Role	O	O	O	O	O	O	O	O	O
	Service	O	O	O	O	O	O	O	O	O
Motivation	Assessment	GNOS	NO	NO	NO	NO	NO	NO	NO	NO
	Driver	NO	GNOS	NO	NO	NO	NO	NO	NO	NO
	Goal	NO	NO	GNOS	NO	NOR	NOr	NOr	NO	NO
	Meaning	NO	NO	NO	GNOS	NO	NO	NO	NO	NO
	Outcome	NO	NO	NO	NO	GNOS	NOR	NOR	NO	NO
	Principle	NO	NO	NO	NO	NO	GNOS	NOR	NO	NO
	Requirement	NO	NO	NO	NO	NO	NO	GNOS	NO	NO
	Stakeholder	NO	NO	NO	NO	NO	NO	NO	GNOS	NO
	Value	NO	NO	NO	NO	NO	NO	NO	NO	GNOS
Strategy	Capability	O	O	O	O	O	O	O	O	O
	Course of Action	O	O	O	O	O	O	O	O	O
	Resource	O	O	O	O	O	O	O	O	O
	Value Stream	O	O	O	O	O	O	O	O	O
Business	Business Actor	O	O	O	O	O	O	O	O	O
	Business Interface	O	O	O	O	O	O	O	O	O
	Business Object	O	O	O	O	O	O	O	O	O
	Product	O	O	O	O	O	O	O	O	O
Application	Application Component	O	O	O	O	O	O	O	O	O
	Application Interface	O	O	O	O	O	O	O	O	O
	Data Object	O	O	O	O	O	O	O	O	O
Technology	Artifact	O	O	O	O	O	O	O	O	O
	Communication Network	O	O	O	O	O	O	O	O	O
	Device	O	O	O	O	O	O	O	O	O
	Distribution Network	O	O	O	O	O	O	O	O	O
	Equipment	O	O	O	O	O	O	O	O	O
	Facility	O	O	O	O	O	O	O	O	O
	Material	O	O	O	O	O	O	O	O	O
	Node	O	O	O	O	O	O	O	O	O
	System Software	O	O	O	O	O	O	O	O	O
	Technology Interface	O	O	O	O	O	O	O	O	O
Implementation & Migration	Deliverable	O	O	O	O	O	O	O	O	O
	Plateau	O	O	O	O	O	O	O	O	O
	Work Package	O	O	O	O	O	O	O	O	O

		Strategy			
	From → *To ↓*	Capability	Course of Action	Resource	Value Stream
Common	Collaboration	O	O	O	O
	Event	O	O	O	O
	Function	O	O	O	O
	Grouping	Ofgnrstv	Ofgnrstv	Ofginrstv	Ofgnrstv
	Location	O	O	O	O
	Path	O	O	O	O
	Process	O	O	O	O
	Role	O	O	O	O
	Service	O	O	O	O
Motivation	Assessment	On	On	On	On
	Driver	On	On	On	On
	Goal	Onr	Onr	Onr	Onr
	Meaning	On	On	On	On
	Outcome	Onr	NOR	Onr	Onr
	Principle	Onr	Onr	Onr	Onr
	Requirement	NOR	NOR	NOR	NOR
	Stakeholder	On	On	On	On
	Value	On	On	On	On
Strategy	Capability	FGOSTV	FOTV	IOftv	FOTV
	Course of Action	FORTV	FGOSTV	IOfrtv	FORTV
	Resource	Oftv	Oftv	GOSftv	Oftv
	Value Stream	FOTV	FOTV	IOftv	FGOSTV
Business	Business Actor	O	O	O	O
	Business Interface	O	O	O	O
	Business Object	O	O	O	O
	Product	O	O	O	O
Application	Application Component	O	O	O	O
	Application Interface	O	O	O	O
	Data Object	O	O	O	O
Technology	Artifact	O	O	O	O
	Communication Network	O	O	O	O
	Device	O	O	O	O
	Distribution Network	O	O	O	O
	Equipment	O	O	O	O
	Facility	O	O	O	O
	Material	O	O	O	O
	Node	O	O	O	O
	System Software	O	O	O	O
	Technology Interface	O	O	O	O
Implementation & Migration	Deliverable	O	O	O	O
	Plateau	O	O	O	O
	Work Package	O	O	O	O

		Common								
	From → *To ↓*	Collaboration	Event	Function	Grouping	Location	Path	Process	Role	Service
Common	**Collaboration**	GOSftv	Oftv	Oftv	GOfrstv	GOftv	Oftv	Oftv	Oftv	OVft
	Event	Ofirtv	GOSftv	FOTv	GOfirstv	GOfirtv	Oftv	FOTv	IOftv	FOTv
	Function	Ofirtv	FOTv	FGOSTv	GOfirstv	GOfirtv	Oftv	FGOTv	IOftv	OVft
	Grouping	Oafginrstv	Oafgnrstv	Oafgnrstv	GOSafinrtv	GOafinrstv	Oafgnrstv	Oafgnrstv	Oafginrstv	Oafgnrstv
	Location	Oftv	Oftv	Oftv	GOfrstv	GOSftv	Oftv	Oftv	Oftv	Oftv
	Path	ORftv	Oftv	Oftv	GOfrstv	GOfrtv	GORSftv	Oftv	ORftv	OVft
	Process	Ofirtv	FOTv	FGOTv	GOfirstv	GOfirtv	Oftv	FGOSTv	IOftv	OVft
	Role	GIOfrtv	Oftv	Oftv	GOfirstv	GOfirtv	Oftv	Oftv	GIOSftv	OVft
	Service	Ofirtv	FOTv	ORftv	GOfirstv	GOfirtv	Oftv	ORftv	Ofirtv	FGOSTv
Motivation	**Assessment**	On	On	On	GOns	GOn	On	On	On	On
	Driver	On	On	On	GOns	GOn	On	On	On	On
	Goal	Onr	Onr	Onr	GOnrs	GOnr	Onr	Onr	Onr	Onr
	Meaning	On	On	On	GOns	GOn	On	On	On	On
	Outcome	Onr	Onr	Onr	GOnrs	GOnr	Onr	Onr	Onr	Onr
	Principle	Onr	Onr	Onr	GOnrs	GOnr	Onr	Onr	Onr	Onr
	Requirement	NOR	NOR	NOR	GNORs	GNOR	NOR	NOR	NOR	NOR
	Stakeholder	Oin	On	On	GOinrs	GOin	On	On	On	On
	Value	On	On	On	GOns	GOn	On	On	On	On
Strategy	**Capability**	Or	O	OR	GOfirstv	GOr	Or	OR	Or	OR
	Course of Action	Or	O	OR	GOfirstv	GOr	Or	OR	Or	OR
	Resource	OR	O	O	GOfrstv	GOR	OR	O	OR	O
	Value Stream	Or	O	OR	GOfirstv	GOr	Or	OR	Or	OR
Business	**Business Actor**	GOfitv	Oftv	Oftv	GOfirstv	GOfitv	Oftv	Oftv	Oftv	OVft
	Business Interface	Ofgirtv	Oftv	Oftv	GOfirstv	GOfirtv	Oftv	Oftv	GOfirtv	OVft
	Business Object	Oai	AO	AO	GOairs	GOai	Oa	AO	Oa	AO
	Product	Oftv	Oftv	Oftv	GOfrstv	GOftv	Oftv	Oftv	Oftv	Oftv
Application	**Application Component**	GOfrtv	Oftv	Oftv	GOfrstv	GOfrtv	Oftv	Oftv	Oftv	OVft
	Application Interface	Ofgirtv	Oftv	Oftv	GOfirstv	GOfirtv	Oftv	Oftv	GOfirtv	OVft
	Data Object	Oai	AO	AO	GOairs	GOai	Oa	AO	Oa	AO
Technology	**Artifact**	Oai	AO	AO	GOairs	GOai	Oa	AO	Oa	AO
	Communication Network	Oftv	Oftv	Oftv	GOfrstv	GOftv	Oftv	Oftv	Oftv	OVft
	Device	GOfirtv	Oftv	Oftv	GOfirstv	GOfirtv	Oftv	Oftv	Oftv	OVft
	Distribution Network	Oftv	Oftv	Oftv	GOfrstv	GOftv	Oftv	Oftv	Oftv	OVft
	Equipment	GOfirtv	Oftv	Oftv	GOfirstv	GOfirtv	Oftv	Oftv	Oftv	OVft
	Facility	GOfitv	Oftv	Oftv	GOfirstv	GOfitv	Oftv	Oftv	Oftv	OVft
	Material	Oai	AO	AO	GOairs	GOai	Oa	AO	Oa	AO
	Node	GOfitv	Oftv	Oftv	GOfirstv	GOfitv	Oftv	Oftv	Oftv	OVft
	System Software	GOfirtv	Oftv	Oftv	GOfirstv	GOfirtv	Oftv	Oftv	Oftv	OVft
	Technology Interface	Ofgirtv	Oftv	Oftv	GOfirstv	GOfirtv	Oftv	Oftv	GOfitv	OVft
Implementation & Migration	**Deliverable**	O	AO	O	GOars	GO	O	O	O	O
	Plateau	O	OT	O	GOfrst	GO	O	O	O	O
	Work Package	Oi	FOT	O	GOfist	GOi	O	O	IO	O

		Business				Application		
	From → *To ↓*	Business Actor	Business Interface	Business Object	Product	Application Component	Application Interface	Data Object
Common	Collaboration	Oftv	Oftv	O	Oftv	Oftv	Oftv	O
	Event	Ofitv	Oftv	O	Ofrtv	Ofitv	Oftv	O
	Function	Ofitv	Oftv	O	Ofrtv	Ofitv	Oftv	O
	Grouping	Oafginrstv	Oafginrstv	Ognrs	Oafgnrstv	Oafginrstv	Oafginrstv	Ognrs
	Location	Oftv	Oftv	O	Oftv	Oftv	Oftv	O
	Path	ORftv	ORftv	O	Ofrtv	ORftv	ORftv	O
	Process	Ofitv	Oftv	O	Ofrtv	Ofitv	Oftv	O
	Role	IOftv	Oftv	O	Ofrtv	IOftv	Oftv	O
	Service	Ofirtv	IOftv	O	GOfrtv	Ofirtv	IOfrtv	O
Motivation	Assessment	On	On	On	On	On	On	On
	Driver	On	On	On	On	On	On	On
	Goal	Onr	Onr	Onr	Onr	Onr	Onr	Onr
	Meaning	On	On	On	On	On	On	On
	Outcome	Onr	Onr	Onr	Onr	Onr	Onr	Onr
	Principle	Onr	Onr	Onr	Onr	Onr	Onr	Onr
	Requirement	NOR	NOR	NOR	NOR	NOR	NOR	NOR
	Stakeholder	IOn	On	On	On	On	On	On
	Value	On	On	On	On	On	On	On
Strategy	Capability	Or	Or	Or	Or	Or	Or	Or
	Course of Action	Or	Or	Or	Or	Or	Or	Or
	Resource	OR	OR	OR	OR	OR	OR	OR
	Value Stream	Or	Or	Or	Or	Or	Or	Or
Business	Business Actor	GOSftv	Oftv	O	Oftv	Oftv	Oftv	O
	Business Interface	GOfirtv	GOSftv	O	Ofrtv	Ofirtv	ORftv	O
	Business Object	IOa	Oa	GOS	GOa	Oa	Oa	OR
	Product	Oftv	Oftv	O	GOSftv	Oftv	Oftv	O
Application	Application Component	Oftv	Oftv	O	Ofrtv	GOSftv	Oftv	O
	Application Interface	Ofirtv	Oftv	O	Ofrtv	GOfirtv	GOSftv	O
	Data Object	Oa	Oa	O	GOa	IOa	Oa	GOS
Technology	Artifact	Oa	Oa	O	GOa	Oa	Oa	O
	Communication Network	Oftv	Oftv	O	Oftv	Oftv	Oftv	O
	Device	Oftv	Oftv	O	Ofrtv	Oftv	Oftv	O
	Distribution Network	Oftv	Oftv	O	Oftv	Oftv	Oftv	O
	Equipment	Oftv	Oftv	O	Ofrtv	Oftv	Oftv	O
	Facility	Oftv	Oftv	O	Oftv	Oftv	Oftv	O
	Material	Oa	Oa	O	GOa	Oa	Oa	O
	Node	Oftv	Oftv	O	Oftv	Oftv	Oftv	O
	System Software	Oftv	Oftv	O	Ofrtv	Oftv	Oftv	O
	Technology Interface	Ofitv	Oftv	O	Ofrtv	Ofitv	Oftv	O
Implementation & Migration	Deliverable	O	O	OS	O	O	O	OS
	Plateau	O	O	O	O	O	O	O
	Work Package	Oi	O	O	O	Oi	O	O

		Technology									
	From → *To ↓*	Artifact	Communication Network	Device	Distribution Network	Equipment	Facility	Material	Node	System Software	Technology Interface
Common	**Collaboration**	O	Oftv	Oftv	Oftv	Oftv	Oftv	O	Oftv	Oftv	Oftv
	Event	Or	Ofirtv	Ofirtv	Ofirtv	Ofirtv	Ofirtv	Or	Ofirtv	Ofirtv	Oftv
	Function	Or	Ofirtv	Ofirtv	Ofirtv	Ofirtv	Ofirtv	Or	Ofirtv	Ofirtv	Oftv
	Grouping	Ognrs	Oafginrstv	Oafginrstv	Oafginrstv	Oafginrstv	Oafginrstv	Ognrs	Oafginrstv	Oafginrstv	Oafginrstv
	Location	O	Oftv	Oftv	Oftv	Oftv	Oftv	O	Oftv	Oftv	Oftv
	Path	Or	ORftv	ORftv	ORftv	ORftv	ORftv	Or	ORftv	ORftv	ORftv
	Process	Or	Ofirtv	Ofirtv	Ofirtv	Ofirtv	Ofirtv	Or	Ofirtv	Ofirtv	Oftv
	Role	Or	Ofirtv	IOfrtv	Ofirtv	IOfrtv	IOfrtv	Or	IOfrtv	IOfrtv	Oftv
	Service	Or	Ofirtv	Ofirtv	Ofirtv	Ofirtv	Ofirtv	Or	Ofirtv	Ofirtv	IOfrtv
Motivation	**Assessment**	On	On	On	On	On	On	On	On	On	On
	Driver	On	On	On	On	On	On	On	On	On	On
	Goal	Onr	Onr	Onr	Onr	Onr	Onr	Onr	Onr	Onr	Onr
	Meaning	On	On	On	On	On	On	On	On	On	On
	Outcome	Onr	Onr	Onr	Onr	Onr	Onr	Onr	Onr	Onr	Onr
	Principle	Onr	Onr	Onr	Onr	Onr	Onr	Onr	Onr	Onr	Onr
	Requirement	NOR	NOR	NOR	NOR	NOR	NOR	NOR	NOR	NOR	NOR
	Stakeholder	On	On	On	Oin	On	Oin	On	Oin	On	On
	Value	On	On	On	On	On	On	On	On	On	On
Strategy	**Capability**	Or	Or	Or	Or	Or	Or	Or	Or	Or	Or
	Course of Action	Or	Or	Or	Or	Or	Or	Or	Or	Or	Or
	Resource	OR	OR	OR	OR	OR	OR	OR	OR	OR	OR
	Value Stream	Or	Or	Or	Or	Or	Or	Or	Or	Or	Or
Business	**Business Actor**	O	Oftv	Oftv	GOfitv	Oftv	IOftv	O	Ofitv	Oftv	Oftv
	Business Interface	Or	Ofirtv	Ofirtv	Ofgirtv	Ofirtv	Ofirtv	Or	Ofirtv	Ofirtv	Ofrtv
	Business Object	OSr	Oa	Oa	Oai	Oa	Oai	ORS	Oai	Oa	Oa
	Product	O	Oftv	Oftv	Oftv	Oftv	Oftv	O	Oftv	Oftv	Oftv
Application	**Application Component**	OR	Ofrtv	Ofrtv	Ofrtv	Ofrtv	Ofrtv	Or	Ofrtv	Ofrtv	Oftv
	Application Interface	Or	Ofirtv	Ofirtv	Ofirtv	Ofirtv	Ofirtv	Or	Ofirtv	Ofirtv	ORftv
	Data Object	ORS	Oa	Oa	Oa	Oa	Oa	OSr	Oa	Oa	Oa
Technology	**Artifact**	GOSr	Oai	IOa	Oai	Oai	Oai	OSr	Oai	IOa	Oa
	Communication Network	O	GOSftv	Oftv	Oftv	Oftv	Oftv	O	Oftv	Oftv	Oftv
	Device	OR	GOfrtv	GOSfrtv	Ofgirtv	GOfrtv	Ofgirtv	OR	GOfirtv	Ofrtv	Oftv
	Distribution Network	O	Oftv	Oftv	GOSftv	Oftv	Oftv	O	Oftv	Oftv	Oftv
	Equipment	O	Oftv	Oftv	GOfirtv	GOSfrtv	GOfirtv	OR	GOfirtv	Oftv	Oftv
	Facility	O	Oftv	Oftv	GOfitv	Oftv	GOSfitv	O	GOfitv	Oftv	Oftv
	Material	OS	Oa	Oa	Oai	IOa	Oai	GOSr	Oai	Oa	Oa
	Node	O	Oftv	Oftv	Ofgitv	Oftv	GIOftv	O	GOSfitv	Oftv	Oftv
	System Software	OR	GOfirtv	IOfrtv	Ofgirtv	Ofirtv	Ofgirtv	Or	GOfirtv	GIOSfrtv	Oftv
	Technology Interface	Or	Ofgirtv	GOfirtv	Ofgirtv	GOfirtv	GOfirtv	Or	GOfirtv	GOfirtv	GOSftv
Implementation & Migration	**Deliverable**	OS	O	O	O	O	O	OS	O	O	O
	Plateau	O	O	O	O	O	O	O	O	O	O
	Work Package	O	Oi	Oi	Oi	Oi	Oi	O	Oi	Oi	O

		Implementation & Migration		
	From → *To ↓*	Deliverable	Plateau	Work Package
Common	Collaboration	OR	GOR	OR
	Event	OR	GORT	FORT
	Function	OR	GOR	OR
	Grouping	Ognrs	Oafginrst	Oafgnrst
	Location	OR	GOR	OR
	Path	OR	GOR	OR
	Process	OR	GOR	OR
	Role	OR	GOR	OR
	Service	OR	GOR	OR
Motivation	Assessment	On	On	On
	Driver	On	On	On
	Goal	Onr	GOnr	Onr
	Meaning	On	On	On
	Outcome	Onr	GOnr	Onr
	Principle	Onr	Onr	Onr
	Requirement	NOR	GNOR	NOR
	Stakeholder	Onr	Oinr	Onr
	Value	On	On	On
Strategy	Capability	OR	GOR	OR
	Course of Action	OR	GOR	OR
	Resource	OR	GOR	OR
	Value Stream	OR	GOR	OR
Business	Business Actor	OR	GOR	OR
	Business Interface	OR	GOR	OR
	Business Object	OR	GOR	OR
	Product	OR	GOR	OR
Application	Application Component	OR	GOR	OR
	Application Interface	OR	GOR	OR
	Data Object	OR	GOR	OR
Technology	Artifact	OR	GOR	OR
	Communication Network	OR	GOR	OR
	Device	OR	GOR	OR
	Distribution Network	OR	GOR	OR
	Equipment	OR	GOR	OR
	Facility	OR	GOR	OR
	Material	OR	GOR	OR
	Node	OR	GOR	OR
	System Software	OR	GOR	OR
	Technology Interface	OR	GOR	OR
Implementation & Migration	Deliverable	GOS	Oa	AOR
	Plateau	OR	GOSTf	Ofrt
	Work Package	O	Oft	FGOST

B.6. Grouping, Plateau, and Relationships Between Relationships

In addition to the relationships derived using the rules specified in Section B.2 or Section B.3, the following relationships are also allowed for grouping, plateau, relationships, and junctions:

- Grouping and location elements may have an aggregation relationship to any concept (element, relationships, or junctions)
- A grouping element may be the source of any relationship with any element (provided that the element is a possible target element for the relationship)
- A grouping element may be the target of any relationship with any 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

The resulting relationships between elements have been added to the tables in Section B.5 and are summarized in Table B-1.

Table B-1: Grouping, Plateau, and Relationships Between Relationships

From → ↓ To	Grouping	Plateau / Location	Element other than Grouping, Plateau, Location	Relationship	Junction
Grouping	any	g + any**	any**	o	afinortv*
Element other than Grouping	any*	See metamodel	See metamodel	o	afinortv*
Relationship	g + o	g + o	o		o
Junction	afinortv	afinortv**	afinortv**	o	afinortv

* Provided that element is a possible target element of the relationship (see Section B.5).

** Provided that element is a possible source element of the relationship (see Section B.5).

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

Appendix C: Example Viewpoints

C.1. Basic Viewpoints in the ArchiMate Language

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.

A basic type of viewpoint, is a simple selection of a relevant subset of ArchiMate concepts. The expressed representation of that part of an architecture in this section is geared toward the stakeholders that will utilize 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 their 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, Information and Communication Technology (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 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 domains of Business, Application, and Technology may be used. However, not every combination of these would give meaningful results. In some cases, separate viewpoints for the different domains 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 C-1. The table also shows the perspective for the viewpoint. Some viewpoints have a scope that is limited to a single domain or aspect, when others link multiple domains 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 define the internal structure of elements
2. *Support*: Viewpoints where you are looking at elements that are supported by other elements
3. *Cooperation*: Toward peer elements that cooperate with each other, typically across aspects
4. *Realization*: Viewpoints where you are looking at elements that realize other elements

Table C-1: Basic Viewpoints

Category: Composition		
Name	**Perspective**	**Scope**
Organization	Structure of the enterprise in terms of roles, departments, etc.	Single domain, single aspect
Application Structure	Shows the structure of a typical application in terms of its constituents	Single domain, multiple aspects
Information Structure	Shows the structure of the information used in the enterprise	Multiple domains, single aspects
Technology	Infrastructure and platforms underlying the enterprise's information systems in terms of networks, devices, and system software	Single domain, multiple aspects
Layered	Provides overview of architecture(s)	Multiple domains, multiple aspects
Physical	Physical environment and how this relates to IT infrastructure	Multiple domains, multiple aspects
Category: Support		
Name	**Perspective**	**Scope**
Product	Shows the contents of products	Multiple domains, multiple aspects
Application Usage	Relates applications to their use in, for example, processes	Multiple domains, multiple aspects
Technology Usage	Shows how technology is used by applications	Multiple domains, multiple aspects

Category: Cooperation		
Name	**Perspective**	**Scope**
Process Cooperation	Shows the relationships between various processes	Multiple domains, multiple aspects
Application Cooperation	Shows application components and their mutual relationships	Application Domain, multiple aspects
Category: Realization		
Name	**Perspective**	**Scope**
Service Realization	Shows how services are realized by the requisite behavior	Multiple domains, multiple aspects
Implementation and Deployment	Shows how applications are mapped onto the underlying technology	Multiple domains, multiple aspects

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 stakeholder 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 *Enterprise Architecture at Work: Modeling, Communication, and Analysis* [Lankhorst et al., 2017].

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.

As outlined before, a viewpoint's representation should be geared toward 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 C-2: 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 domain/Single aspect

Elements

- Business actor
- Role
- Collaboration
- Location
- Business interface

C.1.2. Application Structure Viewpoint

The application structure viewpoint shows the structure of one or more applications or components. This viewpoint is useful in designing or understanding the main structure of applications or components and the associated data; e.g., to break down the structure of the system under construction, or to identify legacy application components that are suitable for migration/integration.

Table C-3: Application Structure Viewpoint Description

Application Structure Viewpoint	
Stakeholders	Application and Solution Architects
Concerns	Application structure, consistency and completeness, reduction of complexity
Purpose	Designing
Scope	Single domain/Multiple aspects

Elements

- Application component
- Application interface
- Collaboration
- Data object

C.1.3. 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 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 C-4: 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 domains/Single aspects

Elements

- Business object
- Data object
- Artifact
- Meaning

C.1.4. Technology Viewpoint

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

Table C-5: Technology Viewpoint Description

Technology Viewpoint	
Stakeholders	Infrastructure Architects, Operational Managers
Concerns	Stability, security, dependencies, costs of the infrastructure
Purpose	Designing
Scope	Single domain/Multiple aspects

Elements

- Location
- Node
- Collaboration
- Device
- System software
- Technology interface
- Communication network
- Path
- Process
- Function
- Service
- Event
- Artifact

C.1.5. 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 in Table C-6. 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 C-6: 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 domains/Multiple aspects

Elements

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

C.1.6. 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 C-7: 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 domains/Multiple aspects

Elements

- Location
- Node
- Device
- Equipment
- Facility
- Path
- Communication network
- Distribution network
- Material

C.1.7. Product Viewpoint

The product viewpoint depicts the value that these products offer to the customers or other external parties involved. It shows the composition of one or more products in terms of the constituting (business, application, or technology) services and passive elements. 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 aggregating existing services and/or passive elements, or by identifying which new services or passive elements have to be created for this product, given the value a customer expects from it. It may then serve as input for process architects and others that need to design the processes and ICT realizing these products.

Table C-8: 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 domains/Multiple aspects

Elements

- Business actor
- Role
- Collaboration
- Business interface
- Process
- Function
- Event
- Service
- Business object
- Product
- Application component
- Application interface
- Data object
- Artifact
- Material
- Value

C.1.8. Application Usage Viewpoint

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

Table C-9: 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 domains/Multiple aspects

Elements

- Business actor
- Role
- Collaboration
- Process
- Function
- Event
- Business object
- Application component
- Application interface
- Service
- Data object

C.1.9. 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 C-10: Technology Usage Viewpoint Description

Technology Usage Viewpoint	
Stakeholders	Application, Infrastructure Architects, Operational Managers
Concerns	Dependencies, performance, scalability
Purpose	Designing
Scope	Multiple domains/Multiple aspects

Elements

- Application component
- Service
- Process
- Function
- Event
- Data object
- Node
- Device
- Collaboration
- System software
- Technology interface
- Communication network
- Path
- Artifact

C.1.10. Process Cooperation Viewpoint

The process cooperation viewpoint is used to show the relationships of one or more processes with each other and/or with their environment. It can be used both to create a high-level design of 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 process cooperation are:

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

Each of these can be regarded as a “sub-viewpoint” of the process cooperation viewpoint.

Table C-11: Process Cooperation Viewpoint Description

Process Cooperation Viewpoint	
Stakeholders	Process and Domain Architects, Operational Managers
Concerns	Dependencies between processes, consistency and completeness, responsibilities
Purpose	Designing, deciding
Scope	Multiple domains/Multiple aspects

Elements

- Business actor
- Role
- Collaboration
- Location
- Business interface
- Process
- Function
- Event
- Service
- Business object
- Application component
- Application interface
- Data object

C.1.11. Application Cooperation Viewpoint

The application cooperation viewpoint describes the relationships between application 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 process.

Table C-12: 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	Application Domain/Multiple aspects

Elements

- Location
- Application component
- Collaboration
- Application interface
- Process
- Function
- Event
- Service
- Data object

C.1.12. 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 process view. It provides a “view from the outside” on one or more processes.

Table C-13: Service Realization Viewpoint Description

Service Realization Viewpoint	
Stakeholders	Process and Domain Architects, Product and Operational Managers
Concerns	Added value of processes, consistency and completeness, responsibilities
Purpose	Designing, deciding
Scope	Multiple domains/Multiple aspects

Elements

- Business actor
- Role
- Collaboration
- Business interface
- Process
- Function
- Event
- Service
- Business object
- Application component
- Application interface
- Data object

C.1.13. 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 C-14: 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 domains/Multiple aspects

Elements

- Application component
- Collaboration
- Application interface
- Process
- Function

- Event
- Service
- Data object
- System software
- Technology interface
- Path
- Artifact

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 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 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

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 *Enterprise Architecture at Work: Modeling, Communication, and Analysis* [Lankhorst et al., 2017].

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 C-15: 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. The refinement of tangible goals into requirements, describe the properties that are then 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 C-16: 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
- 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, processes, application services, application components, etc. Typically, the requirements result from the goal refinement viewpoint.

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

Table C-17: 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
- Principle
- Requirement
- Outcome
- Value
- Meaning

- Core element
- Course of action
- Resource
- Capability
- Value stream

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 C-18: 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
- 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 that 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 *strategy viewpoint* provides a high-level strategic overview of the strategies of the enterprise, its capabilities, value streams, and resources, and the envisaged outcomes
- The *capability map viewpoint* provides an overview of the capabilities of the enterprise
- The *value stream viewpoint* shows an overview of value-creating steps in the enterprise and the capabilities that support these
- 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 *Enterprise Architecture at Work: Modeling, Communication, and Analysis* [Lankhorst et al., 2017].

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, value streams, and resources supporting those, and the envisaged outcomes.

Table C-19: 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
- Value stream

- 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 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 C-20: 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. Value Stream Viewpoint

The value stream viewpoint allows the Business Architect to create a structured overview of a value stream, the capabilities supporting the stages in that value stream, the value created, and the stakeholders involved.

Table C-21: Value Stream Viewpoint Description

Value Stream Viewpoint	
Stakeholders	Business Managers, Enterprise and Business Architects
Concerns	Architecture strategy and tactics, motivation
Purpose	Designing, deciding
Scope	Strategy

Elements

- Value stream
- Capability

- Value
- Outcome
- Stakeholder

C.3.4. 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 C-22: 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
- Value stream
- Resource
- Outcome
- Value
- Meaning
- Core element

C.3.5. 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 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 C-23: 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 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 *Enterprise Architecture at Work: Modeling, Communication, and Analysis* [Lankhorst et al., 2017].

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 addressed 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 C-24: 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
- Outcome
- Work package
- Event
- Deliverable
- Business actor
- 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 element has been quite extensively presented in Chapter 12, here the migration viewpoint is only briefly described and positioned by means of Table C-25.

Table C-25: Migration Viewpoint Description

Migration Viewpoint	
Stakeholders	Enterprise Architects, Process Architects, Application Architects, Infrastructure Architects, Domain Architects, employees, shareholders
Concerns	History of models
Purpose	Designing, deciding, informing
Scope	Implementation and Migration

Elements

- Plateau
- Deliverable
- Work package

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, and 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 C-26: 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 domains/Multiple aspects

Elements

- Goal
- Requirement
- Work package
- Event
- Deliverable
- Plateau
- Business actor
- Role
- Location
- Core element

Appendix D: Relationship to Other Standards, Specifications, and Guidance Documents

This appendix describes the relationship of the ArchiMate language to other standards and documents, including the TOGAF Standard, the BIZBOK® Guide, UML, BPMN, and BMM™.

D.1. The TOGAF Standard

The ArchiMate language, as described in this document, complements the TOGAF Standard [C220] 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 D-1. This correspondence indicates a fairly easy mapping between TOGAF views and the ArchiMate viewpoints. A more detailed description of this correspondence is given in The Open Group Guide *How to Use the ArchiMate® Modeling Language to Support the TOGAF® Standard* [G21E].

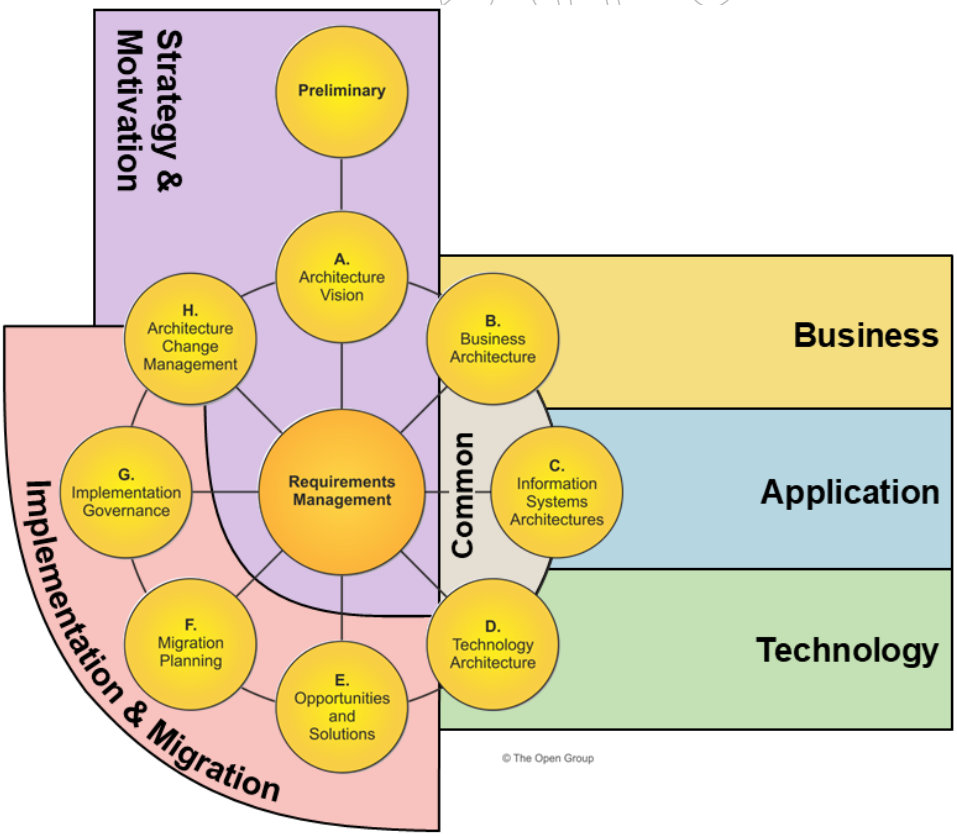


Figure D-1: Correspondence Between the ArchiMate Language and the TOGAF ADM

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 and many viewpoints from both address largely the same issues. Moreover, the viewpoint mechanism described in Section 13.4 lends itself well to define TOGAF viewpoints using ArchiMate concepts.

It is important to reiterate that the ArchiMate Standard is a modeling language and not a framework. Therefore, the viewpoint definitions are more detailed and specify the stakeholders, concerns, level of detail, or abstraction level, and the entity types involved in the viewpoints. In the TOGAF Standard this is presented in a more general way. Therefore, 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 The Open Group Guide *How to Use the ArchiMate® Modeling Language to Support the TOGAF® Standard* [G21E] for a detailed explanation of how the TOGAF and ArchiMate Standards can be used together, including how to create a minimal ArchiMate based metamodel, which covers the whole ADM cycle.

D.2. The BIZBOK Guide

The ArchiMate language provides many concepts that are suitable for modeling Business Architectures. The core domains outlined in the BIZBOK Guide [BIZBOK Guide] — capabilities, value streams, organization, and information — are explicitly covered by relevant concepts in the Business Domain and strategy elements of the ArchiMate language. The other domains — stakeholders, strategies, policies, products, initiatives, and metrics — can also be described easily using appropriate ArchiMate concepts. The language supports key Business Architecture techniques such as capability mapping, organization mapping, information mapping, and value stream mapping, and with its extensive set of relationships it also covers the interconnections between these domains such as value stream — capability cross-mapping; e.g., see Example 7-1.

More advanced descriptions are also possible. In the *TOGAF Series Guide: Value Streams* [G178] and the BIZBOK Guide, value streams are decomposed in a specific way. Stages in a value stream are not connected via relationships where value is exchanged, as in Example 7-1. Rather, the stages produce value items that are aggregated at a higher level into an overall value proposition, and the entry and exit conditions for each stage are specified explicitly. In the ArchiMate language, the former can be modeled with aggregation relationships and the latter using requirements.

D.3. Other Modeling Languages

The general approach in the design of the ArchiMate language is that it has some overlap with other modeling languages. Concepts that are shared between two languages can be used to bridge the gap and create integrated sets of models. This idea is illustrated in Figure D-2.

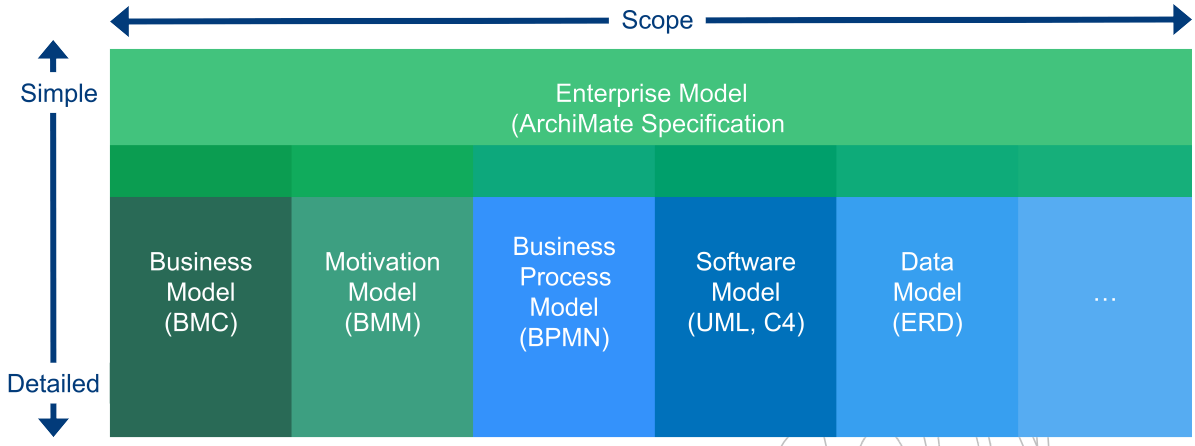


Figure D-2: Correspondence Between the ArchiMate Language and Other Modeling Languages

This way, an ArchiMate model functions as a high-level "umbrella" that ties together different models at more detailed levels. From the Enterprise Architecture level, it is possible to drill down into detailed models of specific aspects, and *vice versa*. The next sections illustrate this in more detail for the BPMN, UML, and BMM standards.

D.4. BPMN

Both the ArchiMate language and BPMN [BPMN] 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 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 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 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.5. UML

The ArchiMate language has derived a number of concepts from UML [UML, 2011]. For other concepts, straightforward correspondences can be defined.

In the Business Domain, the ArchiMate 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. The collaboration concept has been inspired by collaborations as defined in the UML standard [UML, 2011].

In the Application Domain, 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 a less direct manner, the ArchiMate process and function concepts can be mapped onto UML activity diagrams, and a service to a use-case diagram.

Many of the elements of the ArchiMate Technology Domain 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, 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 are 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 The Open Group White Paper, *Using the ArchiMate® Language with UML®* [W134] for a more detailed explanation about how the UML language and the ArchiMate Specification can be used together.

D.6. BMM

The ArchiMate strategy and motivation elements have been inspired partly by the Business Motivation Model™ [BMM™]. 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 and requirement 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.

Appendix E: Changes from Version 2.1 to This Document

E.1. Changes from Version 2.1 to Version 3.0.1

The main changes between Version 2.1 and Version 3.0.1 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 Standard
- 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
- 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 and 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 Specification 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 3.0.1. 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.

E.2. Changes from Version 3.0.1 to Version 3.1

The main changes between Version 3.0.1 and Version 3.1 of the ArchiMate Specification are listed below. In addition to these changes, various other minor improvements in definitions and other wording have been made:

- Introduced a new strategy element: value stream
- Added an optional directed notation for the association relationship
- Improved the organization of the metamodel and associated figures
- Further improved and formalized the derivation of relationships

The formalization of the derivation rules as mentioned above has had a minor impact on the metamodel structure, since some relationships can now be derived that formerly had to be specified explicitly in the metamodel. It has also led to the removal of a small number of spurious relationships. To transform an ArchiMate 3.0.1 model to ArchiMate 3.1, any such relationship may be replaced by a directed association.

E.3. Changes from Version 3.1 to Version 3.2

The main changes between Version 3.1 and Version 3.2 of the ArchiMate Specification are listed below. In addition to these changes, various other minor improvements in definitions, explanations, and examples have been made:

- Improved the definitions of several concepts, most notably outcome, constraint, business function, and product
- Made the physical elements part of the Technology Layer chapter instead of a separate chapter
- Changed the Technology Layer metamodel, making device, system software, facility, and equipment no longer subtypes of node but of technology internal active structure element, and adding composition and aggregation relationships with node
- Added composition and aggregation relationships from plateau to outcome
- Added realization from material to equipment
- Improved the restrictions on derivation rules described in Section B.3.5
- Added a derivation rule for grouping
- Changed the icon notation of the meaning and communication network elements, added an icon notation for work package, and added new box notations for meaning, value, business object, contract, representation, work package, and deliverable
- Changed the default color of the plateau and gap elements to the same pink as the other Implementation and Migration elements

E.4. Changes from Version 3.2 to This Document

The main changes between Version 3.2 and this version of the ArchiMate Specification are listed below. In addition to these changes, various other minor improvements in definitions, explanations, and examples have been made:

- Business interaction, application interaction, technology interaction, constraint, contract, gap, and representation have been removed
- Behavior elements have been merged across layers, leading to a single set of service, process, function, and event
- Implementation event has been replaced by the now generic event element

- Business, application and technology collaborations have been merged into a single collaboration element
- Business role has been replaced by a generic role element to which any internal active structure element can be assigned
- The depiction of the language has been updated and is no longer a matrix combining aspects and layers, but the “ArchiMate Hexagonion”
- The term *layer* has been replaced by the more generic term *domain*
- The “Generic Metamodel” chapter has been replaced by a chapter describing all the previously mentioned generic elements; see Chapter 4
- Path is now a part of the Common Domain
- Aggregation from path to technology internal active structure element has been replaced by a realization from active structure element to path
- Relationships can now have multiplicity to express the constraints put on the instances of elements on their ends

ArchiMate 3.2 models can easily be translated to the new version by applying the following transformations (see also Section 14.2):

- Constraint can be replaced by a (specialization of) requirement
- Contract can be replaced by a (specialization of) business object
- Gap can be replaced by a (specialization of) assessment or deliverable
- Representation can be replaced by a (specialization of) data object, artifact, or material
- Business, application, and technology interactions can be replaced by a (specialization of) function or process
- Other business, application, and technology behavior elements can be replaced by (a specialization of) their generic counterpart
- Implementation event can be replaced by a (specialization of) event
- 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 aggregation from a path to a technology internal active structure element, this may be replaced by a realization from the technology active structure element to the path. If it concerns a realization between services of different layers, this may be replaced by a specialization (from the realizing service to the realized service) or an aggregation (from the realized service to the realizing service).

Appendix F: Acronyms

ABB

Architecture Building Block

ADM

Architecture Development Method

AI

Artificial Intelligence

ASCII

American Standard Code for Information Interchange

B2B

Business-to-Business

BMM

Business Motivation Model

BPMN

Business Process Model and Notation

CEO

Chief Executive Officer

CFO

Chief Financial Officer

CIO

Chief Information Officer

CMO

Chief Marketing Officer

CRM

Customer Relationship Management

DR

Derivation Rule

ERD

Entity Relationship Diagram

GUI

Graphical User Interface

HTML

HyperText Markup Language

IaaS

Infrastructure as a Service

ICT

Information and Communication Technology

IoT

Internet of Things

LAN

Local Area Network

JEE

Java, Enterprise Edition (was J2EE)

PDF

Portable Document Format

PDR

Potential Derivation Rule

RTF

Rich Text Format

SBB

Solution Building Block

SWOT

Strengths, Weaknesses, Opportunities, and Threats

UML

Unified Modeling Language

VPN

Virtual Private Network

WAN

Wide Area Network

WLAN

Wireless Local Area Network

XML

eXtensible Markup Language

Index

T

V

W