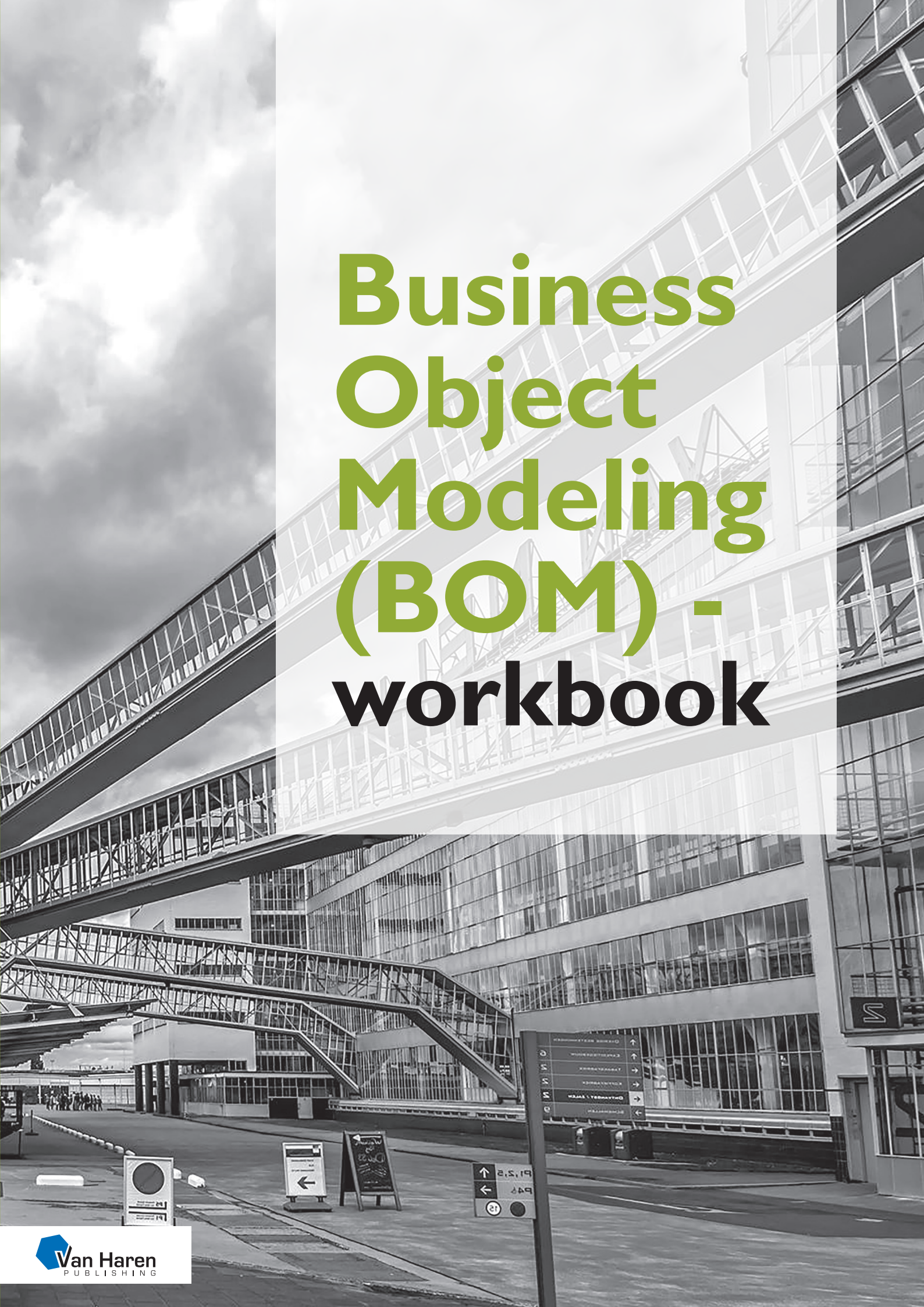


Business Object Modeling (BOM) - workbook

Van Haren PUBLISHING

Business Object Modelling (BOM) Workbook

A pattern-based approach to creating, managing and using an enterprise data model

Colophon

Title:	Business Object Modelling (BOM) Workbook
Subtitle:	A pattern-based approach to creating, managing and using an enterprise data model
Authors:	Martine Alaerts Patrick Derde
Reviewers :	René De Vleeschauwer (Partner at Envizion,) Laleh Rafati (Information Architect at Envizion)
Publisher:	Van Haren Publishing, 's-Hertogenbosch
ISBN Hard Copy:	9789401810098
Edition:	First edition, first print, April 2023
Design:	Van Haren Publishing, 's-Hertogenbosch
Copyright:	© Van Haren Publishing 2023
	For further information about Van Haren Publishing please e-mail us at: info@vanharen.net or visit our website: www.vanharen.net

All rights reserved. No part of this publication may be reproduced, distributed, stored in a data processing system or Published in any form by print, photocopy or any other means whatsoever without the prior written Consent of the authors and publisher.

Although this publication has been composed with most care, author nor editor can accept any liability for damage caused by possible errors and/or incompleteness in this publication.

Trademark notice:

- ArchiMate® is a trademark of The Open Group.
- UML® is a trademark of the OMG.
- The BOM and Business Object Modeling are copyrighte

Contents

Welcome to this book about the Business Object Modeling approach (BOM approach) and the capabilities of an enterprise data model, elaborated through this approach.

As you are reading this book, we assume you share our concern for qualitative information, requiring a qualitative data architecture[1].

The importance of information as an enterprise asset cannot be underestimated. However, the flexibly exploitable information provided by an agile data architecture, continues to be a challenge. Often, information has only been looked at from the point of view of one "business silo". Different business organizations do not share a common view on business concepts and lack the common language to share their information. Data has been managed and stored only in view of the process and/or application that requires it – the application being designed to support a business silo. This results in a fragmented data solution landscape. Data are duplicated endlessly (preferably registered in different formats), data conflicts occur regularly and data quality does not reach the criteria of all users. Data integration requires a huge effort, not in the least due to the lack of an unambiguous business meaning.

This book treats the BOM approach, an approach that supports an unambiguous definition of all business concepts in a common language. It supports a step-by step elaboration of the enterprise data model, describing the data that are "the raw material" to produce the information that a business requires about these concepts. This model supports information governance and the management of an enterprise data architecture. The BOM approach and the enterprise data model, support information requirements analysis and data architecture decisions on a system level – which in its turn contributes to a further elaboration and enrichment of the enterprise data model.

The value of the BOM approach and an enterprise data model, is universal. Or to be more modest, the BOM approach is valid for all types of industries, but more so for the "tertiary sector", selling goods and/or services.

This book has originally been conceived to support members of the Banking Industry Architecture Network (BIAN). BIAN, recognizing from the start that "information" and "behavior" are equal partners, both necessary to provide financial services, uses the BOM approach to create its "BIAN BOM". The BIAN BOM, as an enterprise data model, is part of BIAN's Financial Industry Architecture Model, that also provides architecture building blocks for functionality and services.

This book has been "promoted" to a more general use, since the value of the BOM approach and an enterprise data model are not limited to the financial sector. A lot of the examples are not from the financial industry and the chapters about the application of the approach are industry-independent. But the synergy between BIAN's functionality, service and data architecture building blocks is of such value, that we did not want to keep this from our readers.

This is why we start this book with a chapter introducing BIAN and its Framework.

Part I of this book explains the BOM approach.

[1] We use the term "information" when we talk about what business wants to know, and "data" when we talk about the facts that need to be registered and managed in order to provide business with that information. More about this in Chapter 2.

We start with some documentation conventions, to enable you to read the diagrams that illustrate the examples. Then, we explain the patterns on which the BOM approach relies, and how they cooperate to create an unambiguous understanding of the business concepts and the detection of the building blocks of the enterprise data model.

The BOM approach leads (if you choose so) to an enterprise data model. Managing such a model on an enterprise-level is a challenge. We finish the second part of the book with techniques BIAN uses to document the BIAN BOM, in view of managing it as an enterprise data model.

In Part II of the book, we examine different ways to exploit the BOM approach and an enterprise data model to their full potential.

Disclaimer:
This book is larded with examples, in text and in diagrams.
None of these examples are real – although most of them are based on real situations in real companies. All of them are highly simplified.
"Snips" of BIAN diagrams are used in figures, to illustrate "something". These diagrams are often not readable. This is deliberate: the message is not the content of the diagram, but in its general appearance or (in most of the "unreadable cases"), its role in the illustrated context. You can consult these diagrams (or the version at the time of your consultation) on www.bian.org.

© Van Haren Publishing

1 A short Introduction to the BIAN architecture

This chapter introduces only those elements of BIAN and its Framework that are needed to understand the references made in this book. For a full overview of what BIAN has to offer and how its architecture is created, we refer to following book: "BIAN 2nd Edition – A framework for the financial services industry" (The BIAN association, 2021)[2].

The Banking Industry Architecture Network (BIAN) recognizes the problems of a financial industry, faced with a fragmented ICT platform, due to a long history of diverse views on functionality to be performed, services to be exchanged and information that is required. BIAN offers a Reference Architecture for the Financial Industry, that is aimed at supporting its members in their migration to a "coreless platform", open to and integrated in the Open API economy.
This Reference Architecture looks at information as an equal partner of "Behavior".

BIAN uses a pattern-based approach and agile principles to define its Reference Architecture for the Financial Industry.

This architecture – that can be consulted freely in the repository and API portal on the BIAN website[3]- consists of a MECE[4] collection of architecture building blocks and a series of archetypical examples of their usage to realize banking functionality. BIAN offers a MECE collection of architecture building blocks for three viewpoints on its architecture: functionality (the BIAN Service Domains), Services (the BIAN Service Operations) and data (the BIAN Business Objects).

1.1 (A selection of) BIAN's architecture building blocks

The "**BIAN Service Landscape**" (Figure 1-1) provides access to the MECE collection of Service Domains, the functionality building blocks.

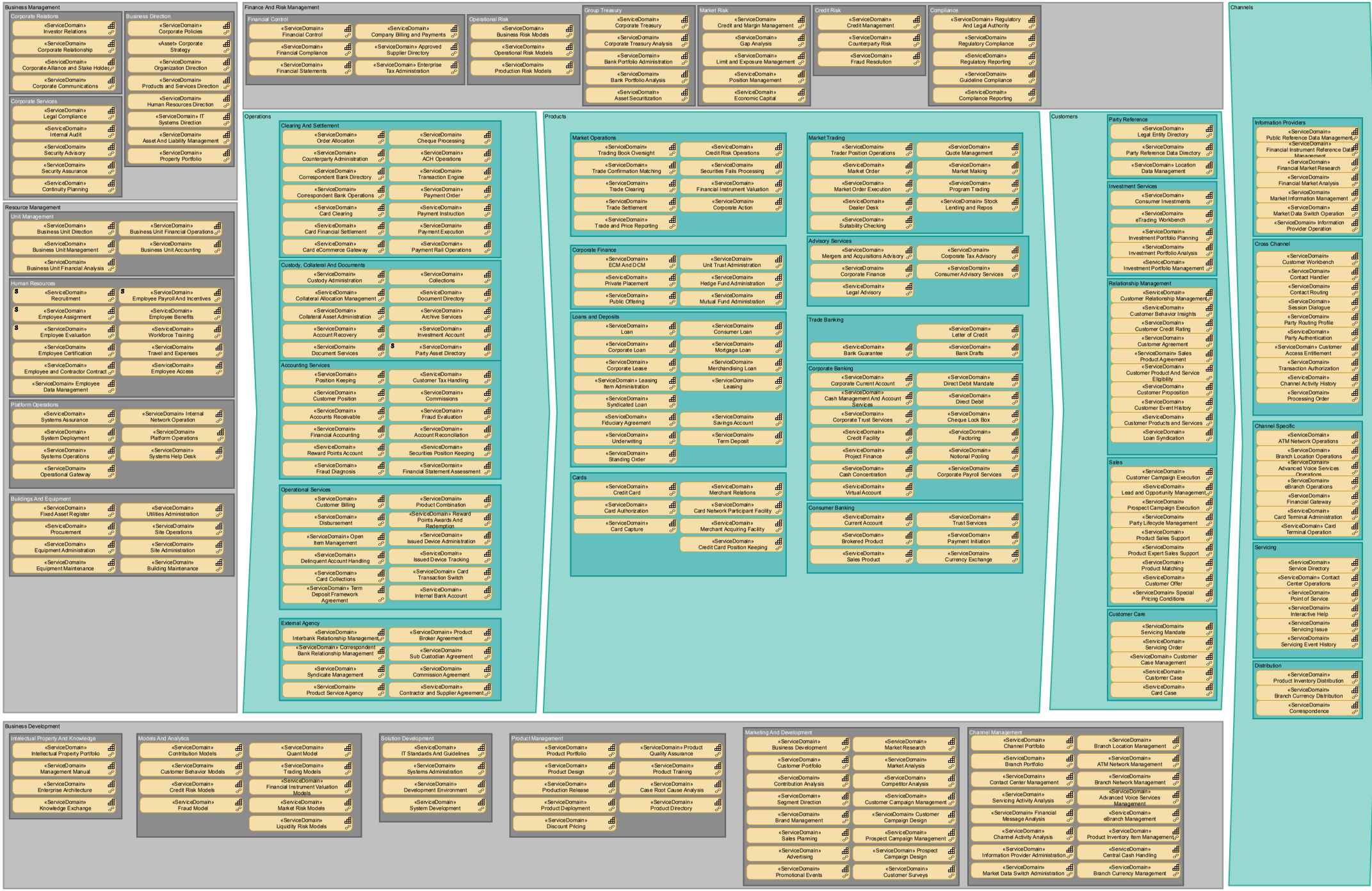
The **Service Domain** is a core concept in the BIAN architecture. A BIAN Service Domain represents the smallest functional partition that can be service-enabled as a discrete and unique business responsibility. All BIAN Service Domains taken together make up a "peer set" with each performing its own specific business function or purpose. Together, interacting through their Service Operations, they can cover all of the functionality required by a bank.

A Service Domain offers its services (Service Operations) to other Service Domains. This allows Service Domains to fulfil their role by delegating the execution of functionality to other Service Domains. The interaction between the Service Domains realizes the business activities that make up a bank.

Each Service Domain is responsible for managing and providing the data, created by the execution of its responsibilities. The Service Domain **Control Record** describes the main business information governed by the Service Domain. Each time a Service Domain fulfills its role, a Control Record instance is created or adapted. A Control Record represents "a set of business information that reflects all information created during the fulfillment of the role of a Service Domain".

[2] Published by Van Haren, ISBN *789401807685*
[3] www.BIAN.org
[4] Mutually Exclusive, Collectively Exhaustive

BIAN Service Domain Landscape version 11.0



A Control Record is represented in a Control Record diagram (example in Figure 1-2), as a hierarchic model. It is “decomposed” in a Control Record level and a “Behavior Qualifier” level. Each Behavior Qualifier represents the information governed by part of the Service Domain‘s responsibility.

Service Domains and their Control Records are defined, maintained and extended by Service Definition working groups, consisting of member-representatives with a solid experience in the responsibility areas of “their” Service Domains.

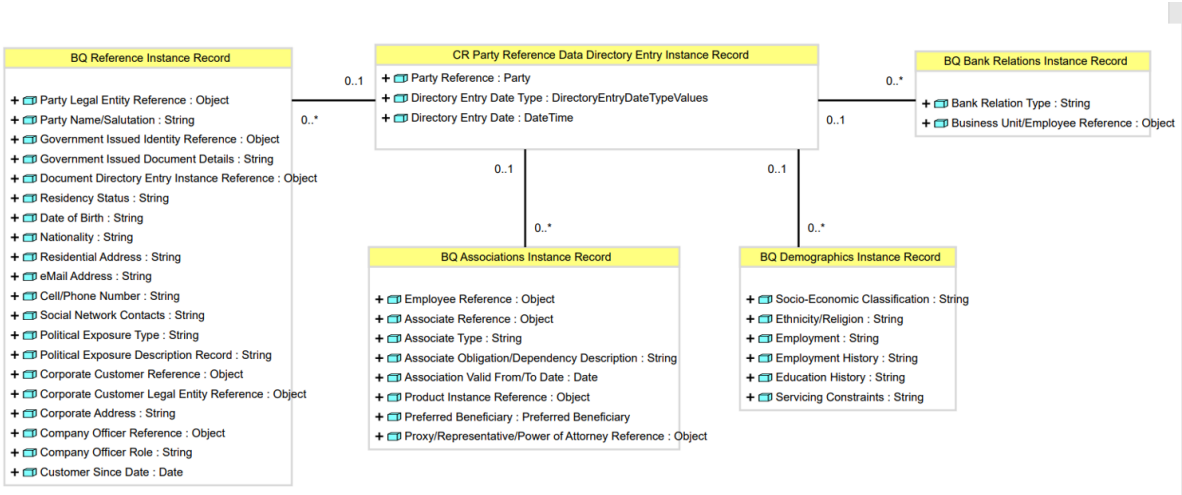


Figure 1-2 The Party Reference Data Directory Control Record diagram

Every Service Domain offers a collection of **Service Operations.** Service Operations support the challenges of interoperability within a bank and in the open finance ecosystem.

A Service Operation offered by a Service Domain, is made available to the environment through an access point called a **Semantic API Endpoint.** The Semantic API Endpoint **Message,** containing the information exchanged in a Service Operation, refers to the Control Record of the Service Domain, or any of its constituent parts.

The Control Record of a Service Domain represents the perspective of the Service Domain on information. Data Architecture building blocks need to represent an enterprise perspective and have a “model element” character. This is why the BIAN Business Object Model (**BIAN BOM**) is created.

This model and its building Blocks, the **Business Objects**, are the result of re-modeling the Control Records, using the **BOM approach**. This approach is the core of this book. It is explained in Chapter 3.

Each Control Record is re-modeled into a “**Service Domain BOM**” (example in Figure 1-3). The Content and Structure Pattern of the BOM approach (and a series of instruments explained in Chapter 4) ensure that these Service Domain views are consistent and compatible. The consolidation of the Service Domain BOM views is BIAN’s enterprise data model: the BIAN BOM. The BIAN BOM is managed by “data specialists” in the BIAN’s Information Architecture working group.

© Van Haren Publishing

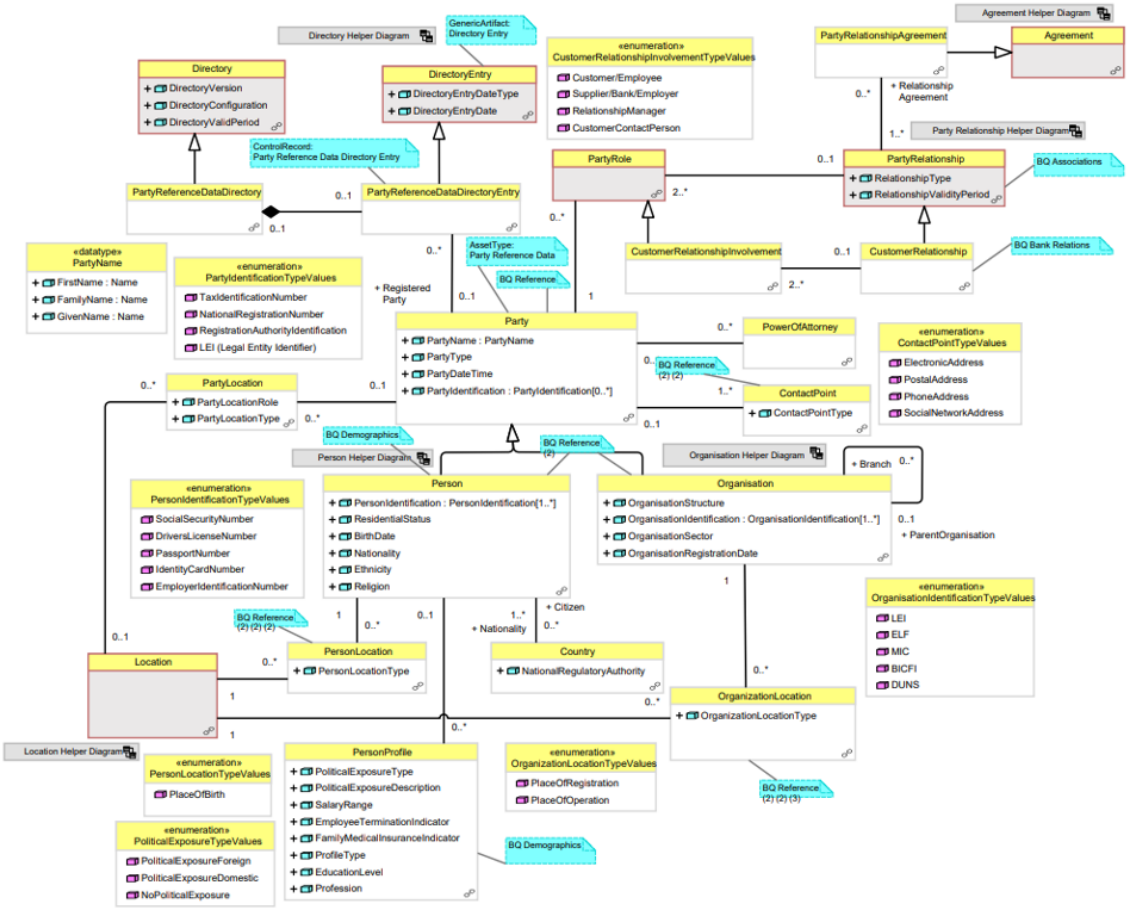


Figure 1-3 The Party Reference Data Directory Service Domain BOM diagram

Figure 8-3 and Figure 8-4 illustrate how the Control Record model is remodeled into a Service Domain BOM diagram. To fully understand this example, you will have to read Part I of this book. It explains how the BOM approach works and what instruments BIAN uses to manage the resulting enterprise data model.

1.2 Using the architecture building blocks to provide banking functionality

BIAN Business Scenarios and Wireframes provide practical examples of how Service Domains can interact. They provide a powerful mechanism to illustrate, by example, the roles and interactions supported by the Service Domains.

A **Business Scenario** depicts how BIAN Service Domains might work together through Service Operations in response to an event. These Service Operations realize the Service Connections between the Service Domains that are required to produce the desired outcome of the scenario.

BIAN stresses that its Business Scenarios are not process designs. But the technique *can* be used in (high level) process design, where each process step is represented by a Service Domain.

© Van Haren Publishing

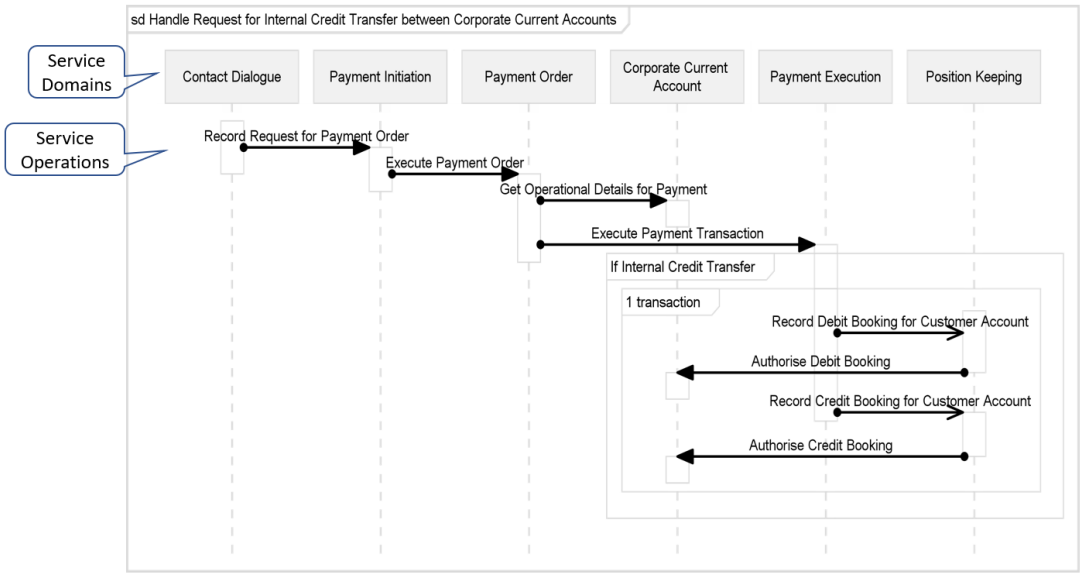


Figure 1-4 Example of a BIAN Business Scenario

Figure 1-4 represents an example of a Business Scenario. The “swim-lanes” are Service Domains, the “messages” are Service Operations. They actually *are*: in the BIAN repository, all representations of model elements are linked to one single “master element”, so each building block is defined only once (see Chapter 4).

A **Wireframe** is a representation that shows all Service Connections between a selection of Service Domains. *One could compare a Wireframe to a city plan, and a Business Scenario to one walk through the city*.

The Wireframe technique can be used -for example - to delimit an application component, finding clusters of “functionality building blocks" (Service Domains) that work closely together in different contexts[5].

Figure 1-5 contains an example of a BIAN Wireframe. That each “rectangle” represents a Service Domain goes without saying (the “master” representation of a Service Domain is an ArchiMate business capability). The Service Operations are not shown on the diagram, as there are too many.

Looking at Part II of this book, it is interesting to note that, by using Service Domains as building blocks to, for example, model processes, you automatically get a picture of the data each process step is responsible for: the Control Record of the Service Domain.

[5] Again, if you want to learn more about the ways these deliverables and techniques can be used: “BIAN 2nd Edition – A framework for the financial services industry” (The BIAN association, 2021)

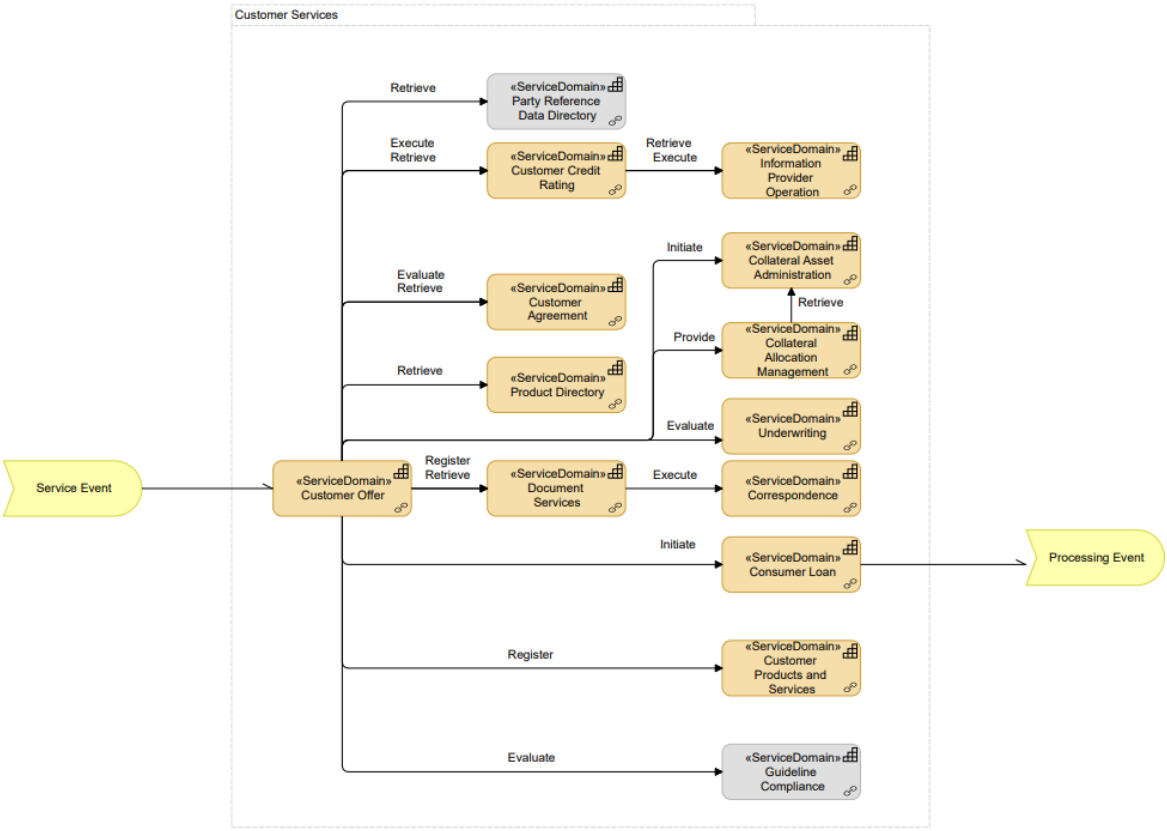


Figure 1-5 Example of a BIAN Wireframe

© Van Haren Publishing

PART I Understanding the BOM approach

The Business Object Modeling approach is an approach to identifying, formalizing and visualizing Business Objects and the relationships between them. It is a pattern-based approach where unambiguous definitions of **business concepts** lead to the detection of their building blocks: the **Business Objects**.

Business expresses its information requirements in terms of "**business concepts**". A business concept is "a concept that is of importance to the business". To fulfill the information requirements of the business, business concepts need to be identified and defined unambiguously.

To inform businesses concerning the concepts in which they are interested, data needs to be captured and managed. Business concepts are not the building blocks for the data architecture required to provide business flexibly and qualitatively with information. The building block of the data model upon which such a data architecture can be based, is the **Business Object**. These are the mutually exclusive, collectively exhaustive (MECE) units of data. Business Objects relate to each other, and thus constitute a **Business Object Model** (BOM).

The BOM approach leads to a data model and has the ability to support a step-by-step elaboration of an enterprise data model. It does not want to replace the widely used data modeling techniques. It wants to complement them. It adds business meaning to these techniques. It provides business patterns for looking at reality. Experience shows that, if the BOM approach is applied, very little to no extra "normalization techniques" are required to result in a "third normal form".

This part of the book seeks to help you master the BOM approach, by explaining its Patterns and illustrating, through many examples, how they are used.

It wants to inspire you on how to manage your enterprise data model by explaining BIAN's documentation approach, that helps in managing the BIAN BOM as an Enterprise Data Architecture Model.

The documentation conventions, explained in this chapter, are meant to help you in "reading" the diagrams that are used as examples. As a win-win, they also enable you to use BIAN's data-related deliverables.

© Van Haren Publishing

2 Being able to read the diagrams: Documentation conventions

Before we start explaining how the BOM approach works, we need to clarify some general terminology. Terminology represents concepts, and (as the BOM approach stresses) we need to ensure a mutual understanding of what concepts are addressed with which term.

The first section of this chapter contains the terms used for, and descriptions describing the meaning *we* give to those terms. *In this particular book. Based on literature, but undoubtedly not consistent with all literature.*

In order for you to read and understand the examples in the next chapters correctly, we will describe which modeling languages on an architecture and design level and what language elements we use in order to document a data model, resulting from the BOM approach.

2.1 Terminology: defining concepts

2.1.1 Information or data?

Information is what business needs/wants to know - about "business concepts", about "interesting things in reality". Information is aimed at a business purpose.

A **data element** is a property of "a thing". **Data** is the value of a data element. It represents facts about the world. Data is "the raw material" to create information.

A data value is basic or can be derived from other data values.
Basic means the value is not derived from other values e.g., Date of birth = September 20th 2009, Last Name = Johnson.
Derived means the value is calculated using a formula e.g., Age = Today – Date of birth, Gross Invoice Amount = Invoice Net Amount + Tax Amount.

A data element, as a property of "a thing", can be elemental or structured.
Elemental means that the data element contains one data value e.g., Last Name, Age, Gross Invoice Amount.
Structured means that the data element contains a group of data elements. e.g., Address contains Street, House Number, Postal Code, City.

A **data type** is an expression of the notation conventions and internal structure of a data element. A data type is elemental or structured.

Elemental means that the data element is generally accepted to be a basic structure e.g., Date, Integer, Floating Point, Character, String, Boolean, Enumeration.
Structured means that the data type is expressed as a collection of attributes, each with their own data type and/or a subset of allowed values e.g., Postal Address: (Street, House Number, Postal Code, City); Currency Amount: (Amount (real), Currency Code (ISO 4217 Currency Code)), ISO 4217 Currency Code: Enumeration

An **enumeration** is a data type that consists of a set of named values that represent possible values of a data element. E.g., the ISO 4217 Currency Codes.

© Van Haren Publishing

2.1.2 Model vs diagram, data model

Modeling is the activity of creating and maintaining models.
A **model** is an abstract representation of reality.

The concepts that are used to model the reality are "**model elements**" - representing "something that plays a role in the modeled reality", and "**relations"** between the model elements.

A model is not equal to a **diagram**. A diagram is visual representation of (part of) a model for the purpose of communication. Diagrams are a means to create, maintain and communicate (part of) a model.

Data modeling is the activity of creating and maintaining **data models**.
A model is an abstraction of the real world. It is made with a certain purpose and hence it represents a certain point of view on reality. A data model is made to capture the data, needed to provide business with information.
A data model is an abstraction of the real-world things and their relationships, and the data elements describing these.

There are two levels of "model elements" in a data model: the data model building block and the data element, describing this building block.
In the BOM approach, the data model building block is called a "**Business Object"**. The data element is called a "**Business Object Descriptor**".
A Business Object is defined as "something that exists in reality, concrete or abstract, and participates and/or influences the nature of the business.
A Business Object Descriptor is defined as "A data element that describes a Business Object".

So, reality, abstracted from the point of view of information, is captured in a data model, consisting of data building blocks and the relations among them. The data model building blocks are described by data elements. The notation conventions for such data element are described by a data type.
Facts about reality are captured in data – in a structure according to the data model, and according to notation conventions defined by the data types.

2.1.3 An example

Figure 2-1 shows a *diagram* of the *data model* that catches following "reality":
People can be married to each other. They can have children.
John and Mary are married and have three children: April, May and June.

Figure 2-1 Example of catching reality in a data model

"People" are reflected in the data model by the Party *Business Object*. People are described by (for example and among other) following *Business Object Descriptors* (*data elements)*: Identification (Party Identification) and name (Party Name). The *data* (facts) describing Mary and her family, would

be each family member's passport numbers and the names "John Johnson, Mary Smith[6], April, May and June Johnson."
The fact that people can be married and have children is expressed by the Business Object Relationship[7] "Party-Party Relationship". One of its data elements is "Relationship Type", with data type "Party_PartyRelationshipTypeValues (an enumeration).
The fact that Mary and John are married, would be expressed by a "Party_Party Relationship" instance, with a reference to John and Mary (as *data* in the *data elements* 'From Party" and "To Party"). The *data value* of the *data element* "Relationship Type" would be "Party is married to Party".
The fact that John is parent to April, would be expressed by a "Party_Party Relationship" instance, with a reference to John and April (as *data* in the *data elements* 'From Party" and "To Party"). The *data value* of the *data element* "Relationship Type would be "Party is parent of Party".
John would have this relationship with May and June too, and Mary would also have such a relationship with April, May and June.

This *data model* provides lots of *information*. It can answer questions such as "who is married to whom", how many children does Mary have, do May and June have the same parents …and many more.

2.1.4 Architecture, design, solution landscape

Architecture is "the desired structure of the components of a system, their interrelationships and the principles and guidelines governing their design and evolution over time".

Architecture stays on a certain abstraction level: it describes the "black box" of components and their interrelations. It describes those characteristics of components and relations that are required to understand what they are about. It describes (only) those details that are important to steer design. Architecture supports reasoning about the structures and behaviors of the system.

Think of the sketches your architect made, showing where the house would be on the plot, where the main functionalities would be (like kitchen, bedrooms, living space) and how these spaces would connect. These sketches become more detailed during the "architecture phase". At a certain point in time, you are satisfied with the plans. The "design phase" can start.

How detailed your architecture needs to be, depends on the importance of the detail for governing the resulting designs – hence implementations.

Architecture is always more abstract than **design**, which needs to contain all of the detail required for implementation.

Design describes the components "white box", the relations and interactions, in full detail, so they can be understood unambiguously by the implementor.

Think about the detailed plans your architect[8] draws, including where the bath and toilet will be, where the wall sockets will be… Think of the spreadsheet with detailed specifications and cost estimations for everything, from roof to walls and floors, to electricity and plumbing.

[6] Mary did not adopt her husband's name.
[7] See Section 3.2.
[8] The architect is now a "designer".

© Van Haren Publishing

The **solution landscape** is the structure of the implemented solutions: the actual components of a system, and their interrelationships.

Think of your house, as it is built on your plot, with garden, driveway, rooms with different functionality…and wall sockets as specified.

2.1.5 Conceptual, logical, physical

The enterprise data model, created through the BOM approach, is a conceptual data model.
It depicts the structure of the data, totally independent of how they could be exploited.

Conceptual data model: A data model reflecting solely the structure of the data, independent of how they are to be used. A conceptual data model does not take into account any implementation considerations.

A conceptual data model should not be confused with a logical data model (although they may be very similar and more similar as technology enables less "exploitation optimizations").

Logical data model: A data model that takes into account the data's structure as well as its exploitation. A logical data model depicts how the data are presented to the user by a solution.

Physical data model: A model depicting the structure according to which data will be stored in the storage system of choice.

The physical data model is about the storage medium (a notebook, an Excel spreadsheet, an Oracle database…).

2.2 Documentation conventions in ArchiMate and UML

BIAN uses the ArchiMate and UML languages to document its architecture.

The ArchiMate language is used for deliverables on an architecture level, the UML language for deliverables on a design-level.

> An important note: The BOM approach results in conceptual data models. The architecture and design levels differ in the amount of detail they offer, not in how close they come to implementation specifications.
> *In the "building a house" example, there would be nothing about the choice of material (e.g. roof-and floor tiles, bricks, isolation material, no choices in bathroom furniture…)*

ArchiMate is a graphical modeling language. It is used for describing enterprise architectures.
It is an Open standard developed and maintained by The Open Group[9].

UML (Unified Modeling Language) is a language for describing a software system. It is a standard developed and maintained by the Object Management Group (OMG)[10].

Figure 2-2 gives an overview of the language elements of ArchiMate and UML that are used to document the BIAN BOM and the examples in this book.

[9] Learn more in ArchiMate on https://www.opengroup.org/archimate-forum/archimate-overview.
[10] Learn more about UML on About the Unified Modeling Language Specification Version 2.5.1 (omg.org).

In this chapter, we explain “the words” of these languages: the elements of these languages, used to document “a BOM”. In Chapter 3, we will look at “the sentences”: how we use these language elements to convey a more specific meaning, related to the BOM approach.

The way we describe the elements of the ArchiMate and UML languages that we use is based on their definition in these languages, but does not coincide fully.

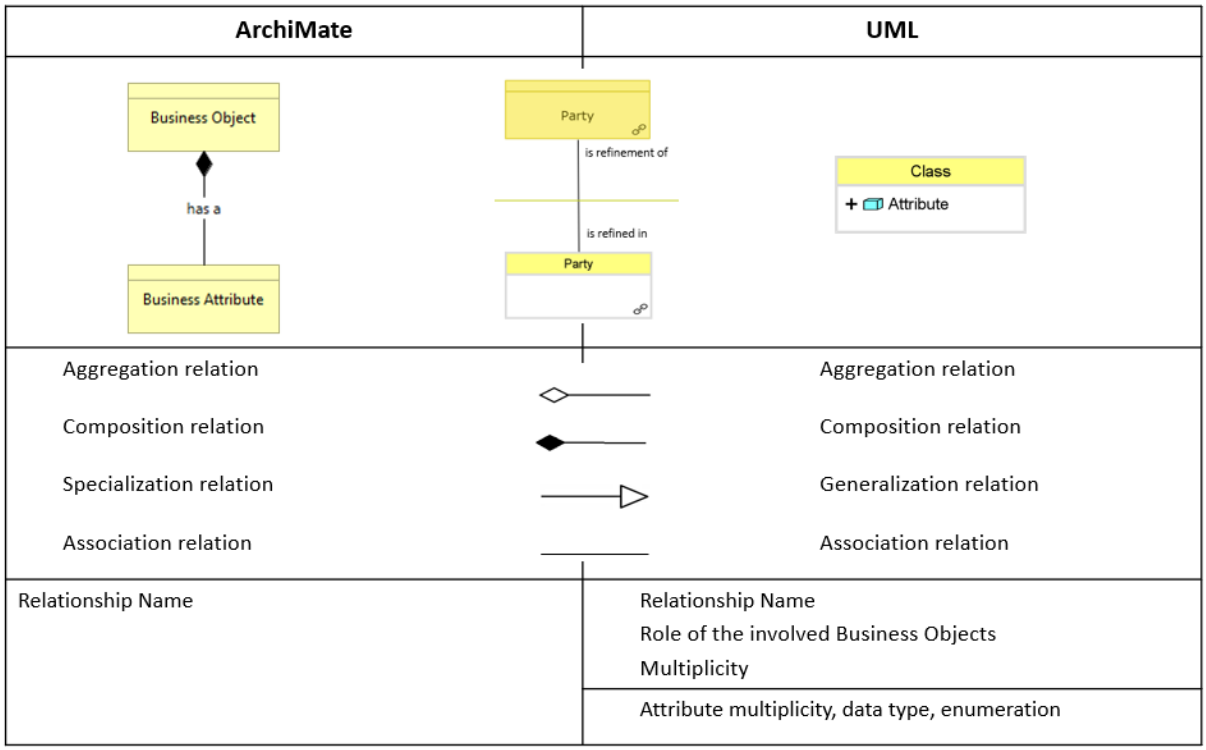


Figure 2-2 Language elements of ArchiMate and UML, used to document the BIAN BOM

2.2.1 Business Objects and their Descriptors.

As shown in Figure 2-2, the “business object[11]” icon of ArchiMate’s business layer, is used to represent the **Business Objects** at an architecture-level. UML classes represent Business Objects at the level of design.

Business Objects are NOT described twice in the BIAN repository: A Business Object at the level of architecture is refined in a Business Object at the level of design. Thus, there is only one definition. There are more Business Objects in UML than in ArchiMate, as they are less abstract (they are decomposed, specialized…). The refinement relation is laid between the ArchiMate Business Object and the “core” UML Object.

A **Business Object Descriptor** is also represented by the business object icon in ArchiMate. It is connected to the Business Object it describes by a composition relation. Indeed, if the Business Object is no longer relevant, neither is its Descriptor.
In UML, we use the attributes of the class.

In UML, a Business Object Descriptor can be described further. We can add:

- A multiplicity, expressing the number of times a Business Object Descriptor can be present.

[11] Note the “lowercase” of the business object *icon* versus the “uppercase” of Business Object *data model building block*.

© Van Haren Publishing

- A data type, expressing the notation conventions and structure of a Descriptor. If this is a structured data type, it describes a “group of attributes”, each with their own data type.
- Data types can be enumerations, listing the possible/allowed values of the Descriptor.

Figure 2-3 represents some examples:
There can be zero to many “Account Identifications”.
“Account Balance” is a structured data element: it consists of three attributes (“Account Balance”, “Account Balance Type “and “Account Balance Date”.
“Account Balance” has in its turn a structured data type (“Amount”): it consists of “Amount Value” and “Amount Currency.” – the latter is described by an enumeration not shown in the figure, but well known: a Currency Code, according to an international standard.
“Account Balance Type” can have the values detailed in the enumeration “AccountBalanceTypeValues“ (OpeningBalance, ClosingBalance, CurrentBalance).
“Account Balance Time” is written down according to the conventions specified in the data type “DateTime”.

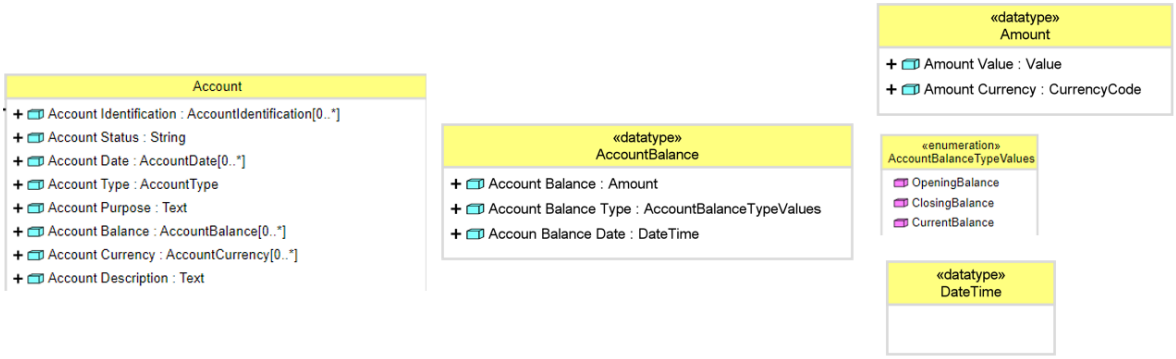


Figure 2-3 Examples of multiplicity, data type and enumeration

2.2.2 Relations

Analogous types of relations are used in ArchiMate as in UML. They have the same graphical appearance and are used to convey the same meaning.

The **composition** relation represents the fact that an Object consists of one or more Objects, in an “existence dependency” relationship. This means that (normally), the part can only exist if the whole exists and it cannot belong to another “whole”. The parts in a composition relationship, are expected to “live and die” with the whole.

In documenting the BOM, the composition relation is used where there is a “live and die dependency” between one Business Object and one or more other Objects.

The **Aggregation** relation represents the fact that an Object relates to one or more other Objects. Unlike composition, aggregation does not imply a “live and die” dependency between the aggregating and aggregated concepts. The parts can exist outside the whole. The parts and the whole may be created and destroyed at different times. The parts can be changed or replaced whenever it is convenient. The parts may be shared with other objects.

© Van Haren Publishing

ArchiMate's **Specialization** relation, representing that one Object is a particular kind of another Object, and UML's **Generalization** relation, representing a taxonomic relation between a more general thing and a more specific thing, are used for the same purpose in the BOM documentation. The Generalization/Specialization relations are used for abstracting concepts / making concepts more concrete, possibly in several layers.

The **Association** relation: specifies that Objects are related to one another.

Relations can be described further.

In ArchiMate, a relationship can be named.

In UML, the relation can also be named, but more properties can be added. (On the ArchiMate *diagram*, we can also show these properties, but they are not part of the *model*…)
An UML relation can receive following properties:

- Relationship name;
- A role for each Business Object involved in the relationship;
- A multiplicity, expressing how many Business Object instances can be involved in this relationship.

In Figure 2-4, the association between "Party" and "Location" expresses the relationship "Address". The Party's role in this relationship is "Located Party" and the Location's is either "Residence" "or "Domicile". A Party can have zero to many addresses and a Location can be the address of zero to many Parties.

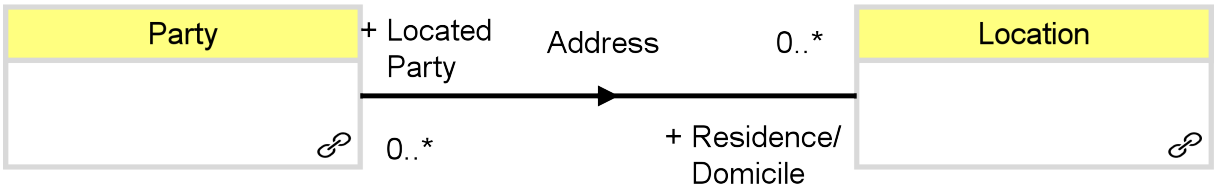


Figure 2-4 Example of describing a relation (in UML)

Multiplicity is implicit in a composition, aggregation and specialization/generalization relation, so it is used (mainly) to specify association relations.

An association relation with multiplicities is a powerful instrument, that is being used more and more in the design level of the BIAN BOM. Association relations with 0…1 to many multiplicity are replacing the aggregation relations, association relations with 1 to many multiplicity are replacing composition relations in UML.

BIAN uses the ArchiMate and UML languages to document its BOM – and other deliverables. This does not imply that BIAN requires its members to use these modeling languages or that BIAN's deliverables are only relevant for organizations using ArchiMate and/or UML.

© Van Haren Publishing

3 Explaining the Business Object Modeling Approach

Business expresses its information requirements in "**business concepts**".

A business concept is a concept that is of importance to the business. To fulfill the information requirements of the business, business concepts need to be identified carefully and defined unambiguously.

To inform businesses concerning the concepts in which they are interested, data needs to be captured and managed.

Business concepts are however not the building blocks for the data model that can capture the data, the "raw material", required to provide business with information of the expected quality, and that allows flexibility in business information demands.

The building block of such a data model is the **Business Object.**

These are the mutually exclusive, collectively exhaustive (MECE) units of data. Business Objects relate to each other, and thus constitute the **Business Object Model** (BOM).
The MECE collection of Business Objects provides the required flexibility in information provision: a combination of their data can support ever-changing information needs.

The **Business Object Modeling approach** (BOM approach) is an approach to identifying, formalizing and visualizing Business Objects and the relationships between them. It is a pattern-based approach where unambiguous definitions of business concepts lead to the detection of their building blocks: the Business Objects.

The core objective of the BOM approach is to detect the Business Objects and their relationships, that can provide business with the required information about business concepts. Thus, it provides the building blocks for an enterprise data model,

The BOM approach can contribute to solving speech confusion. By providing solid definitions inspired by the Content and Structure Patterns, business (and ICT organizations can more easily understand each other. Homonyms and synonyms – often a source of speech confusion - can be detected.

The Content and Structure Patterns can also be used to check and complete the information requirements.

The BOM approach combines three patterns, supporting unambiguous definitions of business concepts and the detection of the data building blocks, needed to fulfil the information requirements.

> The **Content Pattern**: An abstract data model, valid for any business. It contains the Business Objects and their relations, that make up any business, on a high abstraction level.
>
> The **Structure Pattern**: An abstract model that depicts the structure of the data that can describe a business concept.

Content and Structure Patterns are combined to search for the meaning of a business concept, and to unambiguously **define** it. This definition may need "decomposing", i.e., the business concepts used to define the original one may need to be defined too.

© Van Haren Publishing

This is supported by the Classification Pattern.

> The **Classification Pattern**: A classification scheme used to classify business concepts by nature.
> The Classification Pattern is on an abstraction level above both Content and Structure Patterns.

The classification pattern helps to decide whether a business concept's definition has been "decomposed" to the level of "data model building block". It also helps in detecting possible alternative meanings of the term used to address the business concept.

Figure 3-1 depicts the patterns, involved in the BOM approach, suggesting their cooperative and iterative contribution to defining business concepts and detecting Business Objects.

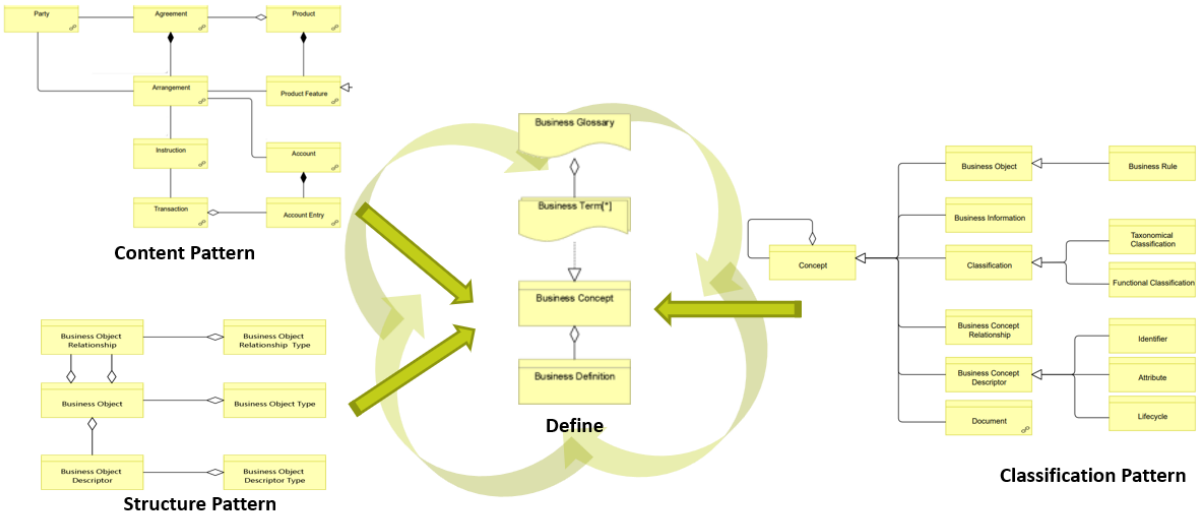


Figure 3-1 The BOM approach and its patterns

Flabbergasted? Don't worry, we will guide you through the BOM approach step-by-step, supported by many examples.

First (Section 3.1) we explain and illustrate the Content Pattern and how it relates to the reality of an organization, active in selling goods and /or providing services.

Then (Section 3.2), we do the same for the Structure Pattern.

Once we have mastered these two patterns, we are ready to start using them in defining business concepts (Section 3.3). We use the best practices for definitions, described in this chapter. We also learn how to "decompose" definitions if necessary.

In Section 3.4, we show you how the Classification Pattern helps to decide whether your definition has arrived at the desired "building block" level. And not least, if it is possible that the term we have been defining may be used for other meanings that are equally relevant for the information requirements.

In Section 3.5, we illustrate that the patterns used in the BOM approach have another welcome use: namely, helping to complete the formation requirements by scanning the Structure and Content Patterns.

© Van Haren Publishing

3.1 The Content Pattern

3.1.1 Discovering the Content Pattern

The Content Pattern is an abstract data model, valid for any business. It contains the Business Objects and their relations that make up any business, on a high abstraction level.
Before we show you this abstract model and the definitions of its constituents, we want to help you discover it yourself. Discover with us, the "pattern" in this story, describing any company in the business of selling goods and/or providing services.

Actors in the market (companies or individuals) offer products (goods and/or services). This is how they make money.

These products (as in "the product catalogue") specify:
- The offering of the company or individual *(e.g., providing cauliflowers and potatoes, providing a refrigerator with a warranty period and delivery at home, providing consultancy services, providing a loan…)*
- And the "compensation" expected from the customer(s*) (e.g., the price, the hourly fee, the interest rate, the penalty fees, the type of collateral…).*

Companies and individuals sell these products to "customers". Each time a product is sold, the seller and buyer negotiate and agree on mutual engagements:
- The seller promises what (how and when and where, under what conditions) she will provide: her goods and services mentioned in the product. *E.g., 20 cauliflowers and 50 kilo potatoes, a loan of 20.000€ over a period of 3 years.*
- The buyer promises what (how, when, where, under what conditions) he will provide the "compensations".
E.g., 1€ per cauliflower and per kilo potatoes, a reduction of 10 pct because of the volume of vegetables, monthly repayments and an interest rate of 3,5 %, to be calculated according to an annuity formula.

Both parties need to agree on that, or the deal is off. Both parties need to comply with their promises, or there is trouble. *For example, if the refrigerator is not delivered, the buyer will not pay. If the repayments are not made, the bank will repossess the car.*
The agreements between buyer and seller can be explicit. – *for example, in case of a refrigerator, consultancy services and a loan*. Or it can be implicit, *for example if you buy a cauliflower or have a drink in a bar.*

Now, how is the company able to provide its products and services?
Even if it makes them itself *(a baker bakes his bread, a manufacturer makes her refrigerators, the bank provides the funds for its own loan agreements*), a business will always need suppliers.
A retailer buys his cauliflowers and refrigerators from farmers and manufacturers. He has the refrigerators she sells delivered at home by a delivery company. A bank buys its computers from a manufacturer and has its loans managed by a service provider.

Suppliers also have a product catalogue, from which our company chooses. Our company negotiates a deal with its suppliers, them promising to deliver the required goods and services, our company promising to provide a "compensation" …

© Van Haren Publishing

So, what pattern do we see here?

Sellers are buyers when dealing with suppliers. Suppliers also have suppliers, so are also buyers, customers will sell thing second hand so are also sellers…
They are all **Parties**, systematically or incidentally offering **Products**.

Product are described by **Product Features**, detailing what can be bought as well as what compensation is expected. They describe what room for negotiation there is. *E.g., the Loan amount can be between x and y, it can be disbursed in full when I buy my house, or requested in parts as the building of my house progresses and bills of builders come in. The interest rate will depend on the market rate, the duration of the loan and the value of the collateral compared with the loan amount*.

When a product is sold, this means the involved parties have an **agreement** on the terms and conditions of the deal. The deal (the agreement) is only valid if all the **arrangements**, describing the engagements (promises) of each party, are agreed upon.
I will give my customer a cauliflower, he will give me money.
I *can order a variable number of cauliflowers every week on Saturday and you will deliver them on Monday before 7 am. I will pay you x€ per delivered cauliflower by Friday evening of the same week. My bank will give me a loan amount, I will pay back this amount piece by piece, plus some interest, with a certain frequency.*

Now, let us discover the second part of the Content Pattern.

I regularly sell a cauliflower to a customer, who pays me on the spot.

Each Saturday, as we agreed, I order my cauliflowers with you, my supplier. You deliver them early on Monday morning. I count the cauliflowers and register this delivery in my stock.
I pay you on Friday.
If you did not deliver, I will not pay. If I did not pay, the next delivery will not be made.

I have a mortgage loan for remodeling my house. Each time a bill comes in, I am allowed to ask for a part of the loan amount. My disbursement request is supported by this bill. The bank checks the bills and disburses the money on my bank account. They register this disbursement in their books.
Each month, I receive a due date request from the bank, requesting me to pay one twelfth of my annuity. I write a payment order, which causes the money to be taken from my bank account and transferred to the loan account. Both transactions will be registered in the bank's books.
If I do not pay, the bank will no longer honor my disbursement requests and, after some agreed upon period, start the process of selling my house.

What pattern do we see here?

The engagements taken in the arrangements that are part of the agreement can be honored immediately. The **transactions** accompany the agreement.
Me handing over the cauliflower is a transaction honoring my (implicit) promise to sell it to my customer, him handing me the money is a transaction honoring his (implicit) promise to pay me the price we agreed upon.

Or there can be a longer-term dimension to the agreement. The arrangements made in our agreement, create mutual obligations and rights that can be called upon in the future.
Instructions can be given, to honor these obligations.
E.g., a cauliflower order is an instruction to deliver a number of cauliflowers;
a disbursement request is an instruction to disburse a part of the loan amount;

© Van Haren Publishing

a due-date request is an instruction to pay the annuity. My payment order is an instruction to the bank to transfer money (an instruction I can give, because I have a "Bank Account Agreement" with the bank…).

Transactions honor the obligations taken in an arrangement. Immediately (*as in my individual cauliflower sale*) or called upon by instructions.
You delivering my 50 cauliflowers, the bank disbursing the amount I requested (after checking the constructor's bills), me giving the payment order, to execute my monthly repayment and interests.

If relevant, transactions can be "mirrored" by (an) entry(ies) in some registry(ies), or, as we call this: by **account entries** on **accounts**.
The bank's disbursement transaction is registered on my loan account, my bank account and in the bank's books. So is my repayment and interest payment.
The delivery of cauliflowers is entered in my stock.

3.1.2 The Content Pattern: definitions

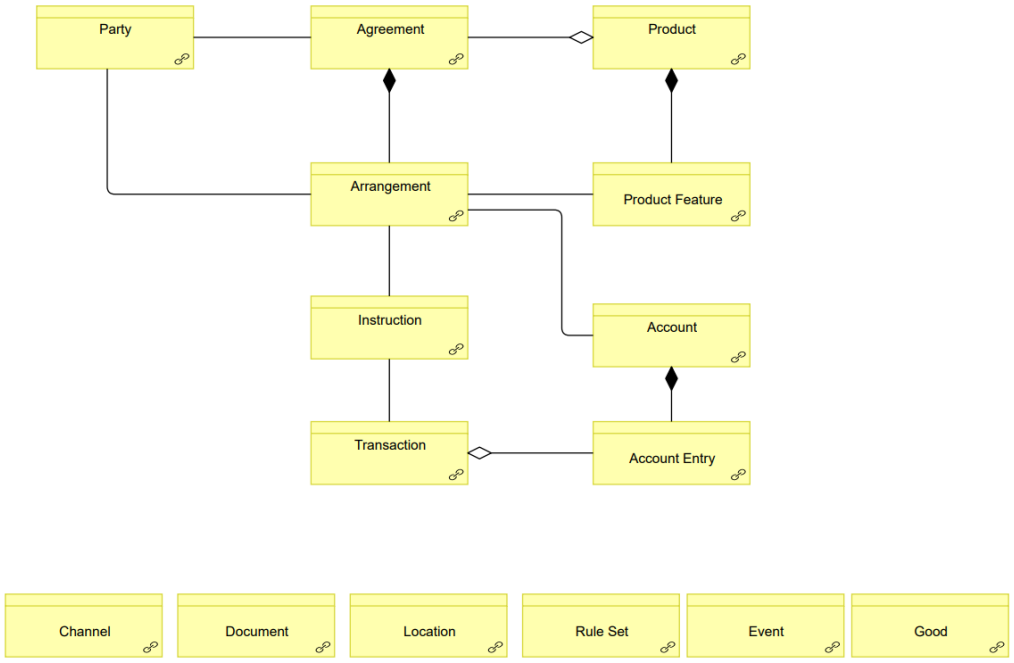


Figure 3-2 BOM Content Pattern

In Section 3.1.1; we discovered the Content Pattern (see Figure 3-2):
This is the story that describes the relationships between the core Business Object of the Content Pattern.

Businesses distinguish themselves from others by the **Products** they offer to the market. The sale of a product is concluded by an **Agreement** with the customer. Agreements with suppliers, and authorities enable the business to offer its products to the market.

An agreement is composed of a number of **Arrangements**, where one **Party** engages itself against another party to give or to do or not to do something (the subject of the Arrangement). The **Product Features** set the options/limits for these arrangements.

© Van Haren Publishing

The fulfilment of one or more arrangements of an Agreement can be triggered by giving an **Instruction** to do or to give something. The instruction will be accepted for execution only if the conditions, agreed upon in the Agreement, are met.

Instructions trigger *T***ransactions** needed to fulfil the Arrangements.

When a Transaction affects the position on an **Account***,* an **Account Entry** must be made on the appropriate Account.

What we have discovered in section 3.1.1 are the "core" Business Objects of the Content Pattern, defined as follows.

Party: A Person or an Organization.

Product: A type of Good or Service or a coherent package thereof, which is offered to internal or external customers.
The sale of a product is concluded by an Agreement.

A ***Service*** is a valuable functionality that fulfils a need or a requirement.

Product Feature: A characteristic of a Product, expressed in offered Services and/or Goods and the terms and conditions applicable to the delivery of the services**.**
*Product Features express the type of arrangements that can be the result of a product sale***.**

Agreement: A formal or informal common understanding between two or more Parties, concerning one or more subject matters, expressed in a set of Arrangements, terms, and conditions.
An Agreement is a collection of Arrangements between Parties. Together, they form a balance which these Parties commit themselves to achieve.

Arrangement: A promise between two or more Parties to do or not to do something, to give or not to give something.
An "Arrangement" always consists of:
- A "Subject Matter" which can be every"thing";
- A promise to "Act": to do, not to do, to give or not to give some"thing" (the subject matter), according to terms and conditions;
- Parties: it is a "Party" who makes the promises to do, not to do, to give or not to give something for/to a "Counter Party".

Instruction: A request to do "something."
This "something" is the commitment made in an Arrangement.

Transaction: The act of doing "something".
This "something" is the commitment made in an Arrangement.

Account: A measuring state on which movements in value or amounts of assets, rights and obligations are registered.
A financial account is a specialization of "account": an administrative financial state where the amounts of financial transactions are registered in debit or credit, resulting in a balance.

Account Entry: The record of a movement in value or amounts of assets, rights and obligations.
A Financial Account Entry (or booking) is the record of a financial transaction on a financial account. It results in an increase or decrease of a balance.

© Van Haren Publishing

As you noticed in Figure 3-2, there are a series of "supporting Business Objects" in the Content Pattern. These will often play a role in relation to the core Objects (or those concrete Business Objects "derived" from the core BusinessObjects).

Rule Set: A set of rules to guide, direct or operate a subject matter;
e.g., regulation, policy, law, guideline, principle, standard.

Event: Something that happens, has happened, can happen or is planned to happen.

Time: A dimension expressing when something happens or will happen.
Can be a period, date, a calendar, a more detailed time indication.

Location: An addressable space/position both in real and virtual environments.

Good: A tangible or intangible thing.

Channel: An interface through which parties can communicate or exchange goods or services.

Note: The diagram in and its subsequent story, may create the impression that all "core objects" need to be present in reality. This is NOT the case.
A Party can be relevant without any Agreements or Arrangements.
Arrangements exist only if the "overarching" Agreements exist and the same goes for Product Features and "their" Product. But an Agreement is not necessarily the result of a Product Sale. And (sadly,) a Product can exist without any (sales) Agreement.
Instructions do not necessarily find their motivation in an Arrangement and Transactions can happen without an Instruction or Arrangement.

3.1.3 Applying the Content Pattern in Sandra's Hair Studio

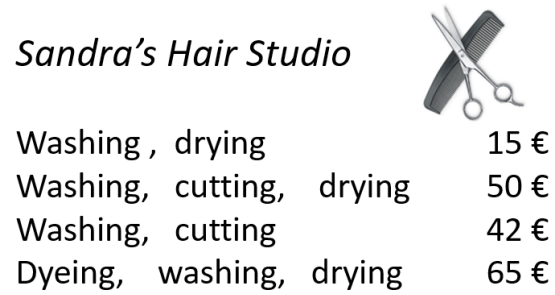


Figure 3-3 Sandra's Hair Studio's offering

Figure 3-3 shows my hairdresser's offering. I am telling no secret if I say that Sandra's Hair Studio, located at Tirlemont Road 16 at Hometown, is the Channel through which she offers her services, and Tirlemont Road 16, Hometown, is a Location.

What Products does Sandra offer? What are the Product Features? Figure 3-4 shows them.
Each line on Sandra's offering is a Product. Product A and C have three Product Features (two types of services Sandra provides, one compensation), Products B and D have four Product Features (three services and one compensation).

© Van Haren Publishing

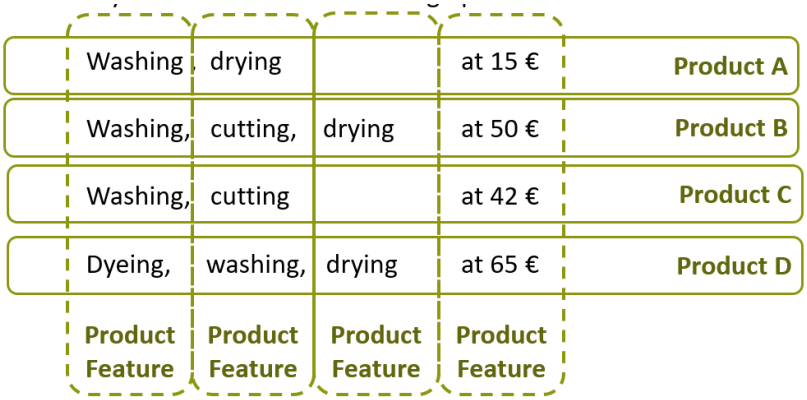


Figure 3-4 Sandra's products with product features

Now, let us look at the sale of one of these Products:

I go to my hairdresser and choose a Product. In Summer, I choose Washing and cutting (Product C), but since it is winter, I choose Washing, cutting, drying (Product B). This sales Transaction implies an Agreement. What are the Arrangements? Who promises what to whom?

Figure 3-5 shows the Agreement and the Arrangements. There are four: Sandra makes three promises to me, I make one promise to her.

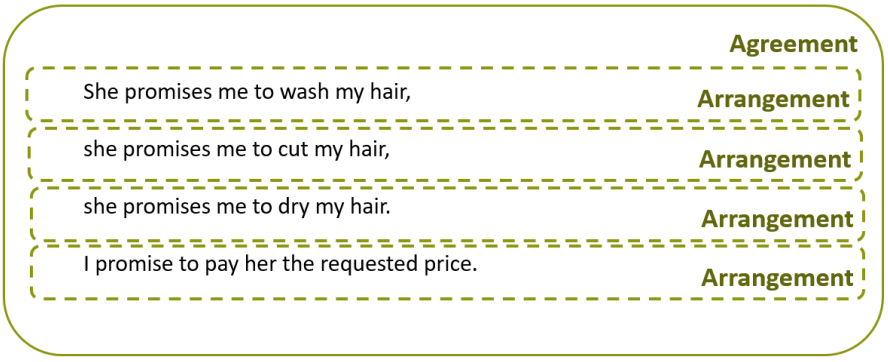


Figure 3-5 My Agreement with Sandra

An “Arrangement” always consists of following elements: - a “subject matter” - which can be every“thing”, a promise to “act” (to do, not to do, to give or not to give some”thing” - the subject matter) and Parties: one who makes the promise to “act” and a “Counter Party”, who is the beneficiary of the “act”.

Figure 3-6 shows the elements that need to be present in an Arrangement: “promising party”, the “counterparty” (or beneficiary), the “act” and the “subject matter” of the promise.
The “acts” are to wash, to cut and to dry and “to pay” respectively. The subject matter is three times “my hair” and once “the money”. Sandra is three times the “promiser” and me the “counterparty”, I am once the “promiser” and she the “counterparty”.

© Van Haren Publishing