Enterprise Architecture Modeling with ArchiMate

A Practical Introduction

(version 0.9 - October 2024)

© Pierluigi Plebani[†]
Barbara Pernici[†]

[†] Politecnico di Milano, Milan, Italy

This is a work in progress. Authors are constantly working to make this document more accurate. Nevertheless, if you notice any mistake or you want simply to leave a comment, it will be great if you can send an e-mail to pierluigi.plebani@polimi.it.

Contents

Chapter 1

Introduction

This manual aims to provide support for studying and improving the use of ArchiMate as a modeling tool for enterprise architectures. Specifically, this manual focuses on the ArchiMate core and the layered view. To this end, the structure of this text emphasizes the three main domains of enterprise architecture: business, application, and technology.

For each of these domains the text dedicates a chapter which, starting from examples that apply the basic pattern of ArchiMate, gradually add further elements of complexity. The sections end with some summary exercises.

It is emphasized that this text is not intended to replace texts which have by now become fundamental for the study of the ArchiMate model, but intends to act as a complementary tool. In this perspective, the text presents several references to two texts: the ArchiMate specifications provided by The Open Group* , and the book by G. Wierda†. For this reason, this text cannot be used to learn the basics of ArchiMate but to apply these basics principles in some common scenarios.

However, to make this document sufficiently self-contained, the next chapter provides a brief overview of ArchiMate, its goals and its main elements.

To complete what is reported in this volume, within the github repository accessible at https://github.com/ISGroup-Polimi/archimate-diagrams-public/ some exercises are published. These are taken from the material used in some of the courses in the Information Systems area offered by Politecnico di Milano.

* https://pubs.opengroup.org/architecture/archimate3-doc/_archimate_3_2_specification.html
† https://ea.rna.nl/mastering-ArchiMate-edition-3-1/

Finally, to better understand which is the role of ArchiMate, it is worthy to report a statement found in Reddit[‡]: "I think that one thing I've learned as I use ArchiMate more, is that you really can (and probably should) use it how you see fit, provided you are able to convey meaning clearly". And convey meaning is the keypoint in Enterprise Architecture.

‡ https://www.reddit.com/r/ArchiMate/comments/z9zjin/roles_vs_actors/?rdt=43283

Chapter 2

ArchiMate in a nutshell

ArchiMate is a graphic notation developed by The Open Group [*], a consortium of companies operating in the IT sector that aims to promote technological standards and open source initiatives through the involvement of more than 900 organizations that are currently members of the consortium.

Among the various initiatives, The Open Group promotes the definition and adoption of ArchiMate as an Enterprise Architecture (EA) modeling tool. Although ArchiMate is to be considered a design tool independent of the adopted EA Framework, it is still clear that there are many aspects in common with TOGAF, one of the most common EA Frameworks today as well as another product of The Open Group.

Considering that The Open Group periodically releases an update of the ArchiMate specification, it is worth remembering that, with respect to the objectives of this volume, the major version to be taken as a reference is version 3. All minor changes made in the definition of versions 3.1 and, last, 3.2 do not concern the portion of ArchiMate covered here.

As mentioned, ArchiMate is a graphical notation. Through this notation it is possible to define diagrams that can represent models. These models, as we will elaborate in Section 2.2, have the goal of describing a portion (i.e., view) of the EA under consideration, according to a given viewpoint.

In its full version, ArchiMate covers all domains that usually feature an EA, as shown in Figure 2.1. Here the domains are represented by the layers of the ArchiMate Framework and

[*] https://www.opengroup.org

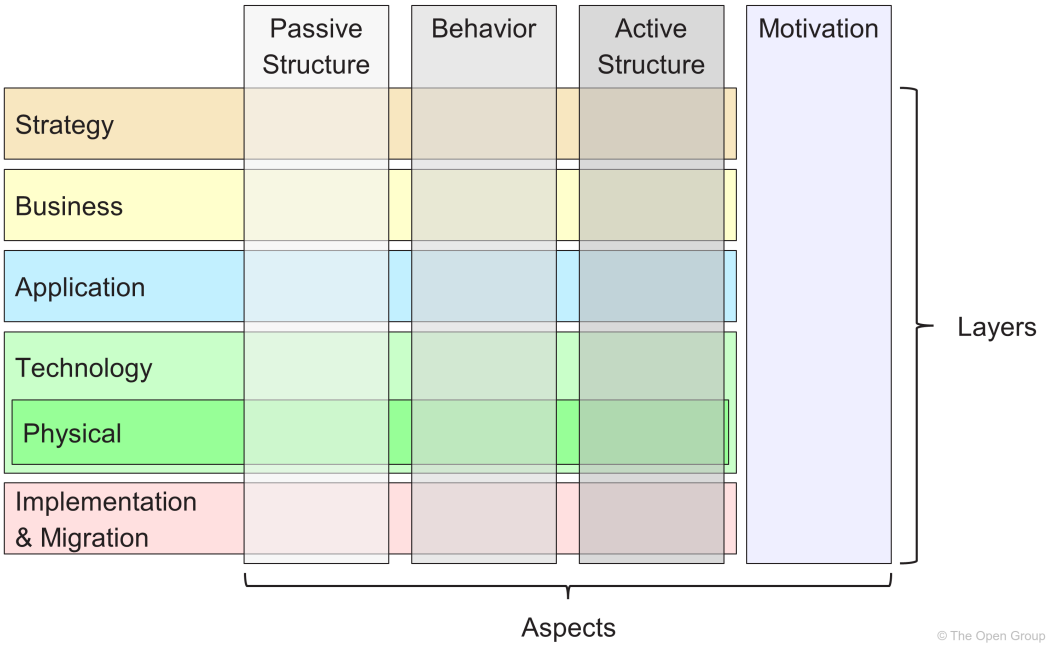

Figure 2.1: ArchiMate Full Framework

cover the most managerial levels, i.e., *Strategy* and *Business*, down to the most IT-related levels, i.e., *Application* and *Technology*. The last level is dedicated to specifying how, in the face of certain projects, the system or business services migrate from the current setting to the future setting, i.e., *Implementation & Migration*. Finally, as a cross-cutting element with respect to all other domains, ArchiMate also allows you to define the *Motivation* that guide the evolution of the EA.

With the exclusion of this vertical domain, all other domains can be represented in ArchiMate through a series of graphical elements that can be grouped into three macro classes: active elements, behavioral elements, and passive elements (see Figure 2.2). These represent the so-called aspects and they can be represented in two ways depending on whether the icon that specializes the element is at the top right (top of the figure) or is used to represent the entire element (bottom of the figure).

In the first case the active and passive elements have the particularity of being represented through a rectangular graphic element, with squared corners. Conversely, behavioral elements are represented by a graphic element that is, again, rectangular but with rounded corners. In the second case, while the active and behavioral take the shape of the icon, the passive elements are usually represented as a rectangle, with squared corners and a band on top. By remembering this distinctions it is possible, in a simple and immediate way, to visually

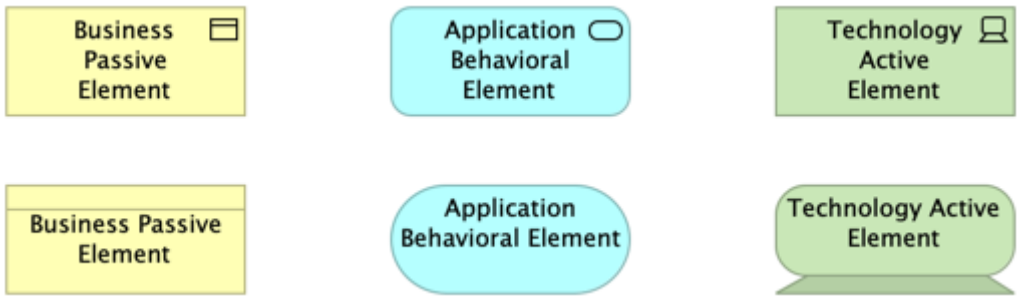

Figure 2.2: Typical representation of active, behavioral, and passive elements

identify which category any element within the diagram belongs to.

A further distinction between the graphic elements is given by the color of the element itself. Unlike the role of the icon that is part of the specification, the color is not actually defined in the specification (the specification is in fact defined as colorless). Therefore, it is only a common practice to indicate with the color yellow the elements that refer to the business level, the blue for the application level and the green for the elements of the technological level. In fact, as you can see by reading Weirda's text, this convention is not followed as different colors distinguished active, passive, and behavioral elements. For the sake of simplicity, in this text we adopt the most common adopted approach, thus different colors represent different layers.

2.1 ArchiMate core framework

Among the different domains with which an EA is usually divided, greater emphasis is often given to the business, application and technology domains. The first is concerned with describing the services offered both internally to other figures in the organization, and externally to the customers/users of the organization. In addition to this, the business level is concerned with defining the processes that underlie these processes and how they are organized within the company functions.

The application level, on the other hand, offers a view related to the set of software to support the business level. Therefore, at this level is the definition of the organization's application portfolio and characteristics of the applications that compose it. At this level, any integration solutions between the various applications are also defined in case it is necessary to put them in communication.

Finally, at the technological level, the description of the EA with respect to the platforms

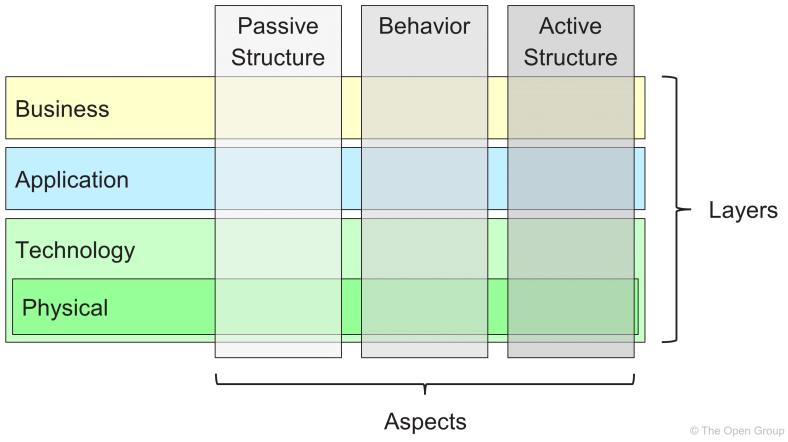

Figure 2.3: Enter Caption

on which the applications described at the previous level are deployed is included. The technologies used at the physical level to provide computational or network resources can also be defined here.

In this perspective, this volume focuses on the full ArchiMate framework, on a subset indicated by the term ArchiMate Core Framework (see Figure 2.3). Also for this portion all the rules described above apply with respect to the colors and shapes of the elements that are defined in it.

2.2 Viewpoints, Views and Layered viewpoint

Any architecture, and therefore even an EA, is a complex object that cannot be modeled exhaustively with a single diagram. As in fact defined in ISO 42010, which in the 2022 version also introduced the EA among the elements of interest in addition to software and system architectures [†], an architecture description provides for the definition of a set of diagrams capable of representing a model of the system according to a certain *viewpoint*.

What is a viewpoint? It is the set of conventions that allow you to specify an architecture that is relevant to one or more stakeholders and, therefore, can address their concerns. In fact, several stakeholders gravitate around an architecture. For example, if we are talking about a building, among the stakeholders we find who commissioned the work, who has to direct the construction, the workers who take care of the construction of the structure, who takes care of the electrical system, and who of the hydraulic system. To these are also added

[†] https://www.iso.org/standard/74393.html

those who, once the building is built, will have to take care of its maintenance. Well, each of them needs to have a description of the building that meets their interests. Therefore, the electrician must have the description of the electrical system map, the carpenter the schemes of how to steel should be placed in the concrete, the construction manager the entire executive project, and the client a vision of the cost estimate and the progress of the work. In some cases, the point of view could be partially shared: think, for example, of the construction manager and the client, both interested in the progress of the work. In other cases, the diagram of the architecture is specific to the stakeholder: for example, the electrical system map.

What is important to emphasize, for the objectives of this book, is that there are a set of rules – sometimes written, sometimes the result of practice – that define what are the notations, and therefore the diagrams, capable of describing the architecture according to a given viewpoint. In the case of Enterprise Architectures, we will have viewpoints that deal with describing the most strategic aspect, others that deal with defining the data available, others that only deal with collaboration at the application level between the different software available, or with collaboration at the business level between the different business processes. A viewpoint, therefore this set of rules, requires the drafting of a series of documents according to specific notations with respect to the needs of the stakeholders. We can find BPMN models for the detailed description of processes, or E-R models for data description, or SWOT analysis for the most strategic aspects.

Knowledge of the rules of a viewpoint allows the architecture designer to generate the so-called *view*. A view is therefore a concretization of a viewpoint obtained with the generation of the documents provided by the viewpoint according to the specified rules.

In the case of interest for this document, ArchiMate is one of the notations that can be used to describe some points of view. Typical examples of viewpoints associated with ArchiMate are described in https://pubs.opengroup.org/architecture/ArchiMate3-doc/ch-Example-Viewpoints.html. Among them, the focus is mainly on the viewpoint called *Layered*. This viewpoint allows you to describe, at a high level, the entire EA with the aim of describing not only the main characteristics of the business, application and technological domains, but also highlighting its dependencies.

A general view of the layered viewpoint is shown in Figure 2.4. The structure of the diagram makes the meaning of the name of this viewpoint evident, as the EA is represented in layers. At the macro level, these layers take up the main domains of the EA (business, application, and technology). In addition, following a service logic, each of these three layers is further broken down into two sub-layers: the upper one that describes the service offered and the lower one that describes how the service offered is realized. On this basis it is

possible to "read" the viewpoint in this way: a `Business Service` is usable by a `Business Actor` (which can be the final customer or a human being internal to the organization). To realize this service it is necessary to activate a certain `Business Process` [‡]. This `Business Process`, can find application-layer support for one or more of the activities that make it up. To this end, the `Business Process` will benefit from `Application Service` which represent the visible part of a set of `Application Function` capable of performing that service. As with the previous layer, the `Application Function` will use the `Technology Service` to run on the IT infrastructure and, for this purpose, `Technology Service` will rely on `Technology Function`.

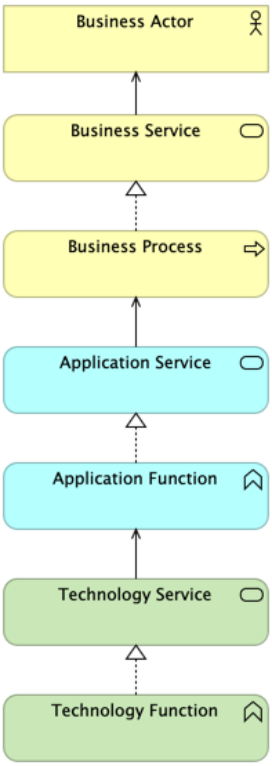

Figure 2.4: General representation of the layered viewpoint

A layered view therefore allows you to identify the services offered in each domain and link them together according to a rigid structure. In fact, the way the diagram is organized, the `Business Actor` will only be able to have visibility on the `Business Service`. Since no other arrow (which as we will see shortly indicates a relationship) points to the `Business Actor` this means that no other element is related to it. So, the `Business Actor` will be totally unaware of the applications and the IT system. Situation that, effectively, occurs in normality. We as users of a service such as the purchase of a property or the after-sales service of a product, we are only aware of the person who guides us in the operations. A separate mention must be made for all the services that are currently available online and that, to a certain extent, give a certain visibility of the application level also to the `Business Actor`. It will be discussed in the following chapters how this can be modeled.

The same type of visibility now discussed for the `Business Actor`, thus in the business layer, is actually also valid for the other layers. In fact, at the business layer only the `Application Service`s are visible but not their implementation. At the same model, only the `Technology Service`s are visible at

‡ As will also be described below, the realization of the service can also be associated with a `Business Function`. In this description of the viewpoint we only want to give a key to reading without being exhaustive with respect to the multiplicity of possible alternatives during modeling.

Structural Relationships		Notation
Composition	Represents that an element consists of one or more other concepts.	
Aggregation	Represents that an element combines one or more other concepts.	
Assignment	Represents the allocation of responsibility, performance of behavior, storage, or execution.	
Realization	Represents that an element plays a critical role in the creation, achievement, sustenance, or operation of a more abstract element.	
Dependency Relationships		**Notation**
Serving	Represents that an element provides its functionality to another element.	
Access	Represents the ability of behavior and active structure elements to observe or act upon passive structure elements.	
Influence	Represents that an element affects the implementation or achievement of some motivation element.	+/-
Association	Respesents an unspecified relationship, or one that is not represented by another ArchiMate relationship.	
Dynamic Relationships		**Notation**
Triggering	Represents a temporal or causal relationship between elements.	
Flow	Represents transfer from one element to another.	
Other Relationships		**Notation**
Specialization	Represents that an element is a particular kind of another element.	
Relationship Connectors		**Notation**
Junction	Used to connect relationships of the same type.	(And) Junction Or Junction

Figure 2.5: Set of relations in ArchiMate

the application layer but not their implementation.

2.3 ArchiMate Relations

To close this generic discussion of ArchiMates, it is now described the *relations*, which have been previously only mentioned as modeling elements that actally play an important role in a diagram. A relation allows you to connect two ArchiMate elements that can be active, passive or behavioral. This connection is semantically defined by the shape of the line representing the relation and the terminal as shown in Figure 2.5.

The relationships are grouped into four classes:

- Structural: they allow you to define the static view of a system, so what it is made of, who makes it, and who takes care of it.
- Dependency: allow you to define the use of one element relative to other elements.
- Dynamic: allow you to define the flow of events or actions within the system.
- Other: at the moment composed of the only specialization relationship that corresponds to the classic *is-a* relationship.

In addition to relations, there are also two operators called *junctions* that allow you to

connect two or more incoming relations and one outgoing relation. In this context, all the relations involved must necessarily be of the same type.

Not all relationships can be used to connect all elements. The ArchiMate specification clearly defines, in one of its appendices §, which connections are allowed.

The knowledge of relations and their belonging to a category is important to be able to manage the so-called *derived relations*. In fact, to make a diagram more readable or to provide a compact version, the chains of relations that are necessarily created when an element is in relation to another element and the latter in relation to a third element, can be revised to produce a version of the diagram that can represent the same model with a lower level of detail but without losing significance. In the following chapters, derived relations will often be used for this purpose.

§ https://pubs.opengroup.org/architecture/ArchiMate3-doc/ch-relationships-Normative.html

Chapter 3

Business Layer

This chapter focuses on the business layer modeling. The goal of this layer is to define the services made available by the organization. In particular, it is assumed in the following examples that these services are always offered to end consumers who are not necessarily internal to the organization itself.

The examples initially focus on the main pattern composed of `Business Service`, `Business Interface`, `Business Process`, `Business Role`, and `Business Object`. Then, derived relations and `Business Function` are also used.

It is worth remembering, with regard to the modeling of the processes underlying the services, that with ArchiMate it is not aimed at defining the process in its details but, on the contrary, its main elements. Therefore, it is plausible to hypothesize in case you also want to give a precise and detailed definition of the process, accompany the ArchiMate diagram with other defined diagrams using specific notations for business processes, e.g., BPMN (Business Process Management Notation).

Example 3.1 - Basic pattern

Among the services offered to its patients, the ACME hospital offers a booking service for medical visits through a Call Center, which allows the patients to book a new medical examination.

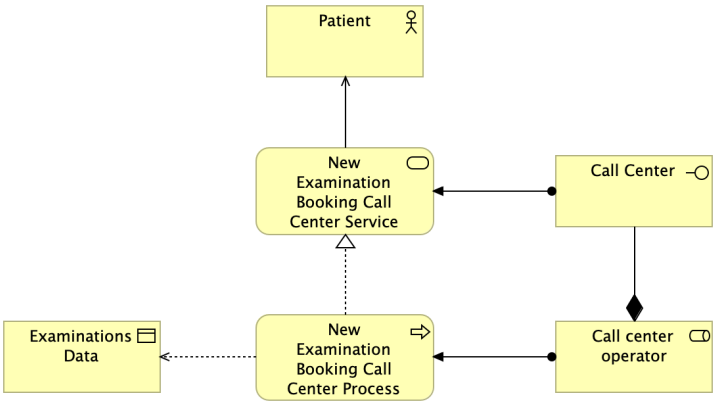

Figure 3.1: Example 3.1 diagram

The scenario described in Example 3.1 can be modeled with a typical pattern as shown in Figure 3.1. With respect to the given text of the example, the proposed diagram also included the *Call center operator* `Business Role`. This is required as each behaviour must be assigned (unless there is not an automation in place) to a role. It is reasonable to think that this operator is the required role. To provide a complete pattern, a generic `Business Object` *Examinations data* is also included in the diagram.

The naming convention adopted in this document assumes that the `Business Service` and the `Business Process` usually have in their labels the name of the `Business Interface` through which they are offered.

Example 3.2 - `Business Process` details

Referring to the Example 3.1, the new examination booking call center process is articulated in three main steps which are performed by the call center operator: the acquisition of the request for examination, the searching for the availabilities, and, depending on the result, the confirmation of the appointment or the cancellation of the request.

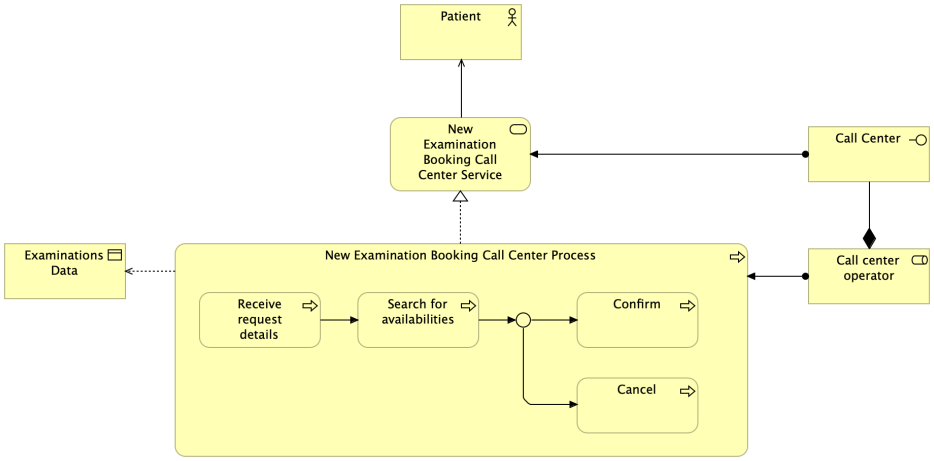

Figure 3.2: Example 3.2 diagram

The details of the `Business Process` described in the Example 3.2, are included in the diagram as `Business Process` components.

Following also the suggestion of Wierda (see par. 7.8), the `composition` relation holds between elements that are nested. In case a different relation is used, it will be explicitly specified.

Example 3.3 - Using `Business Function`

ACME Hospital has extended its reservation services for its patients. Now they are allowed not only to book a new doctor's appointment but also to cancel it. Both services are provided through a call center.

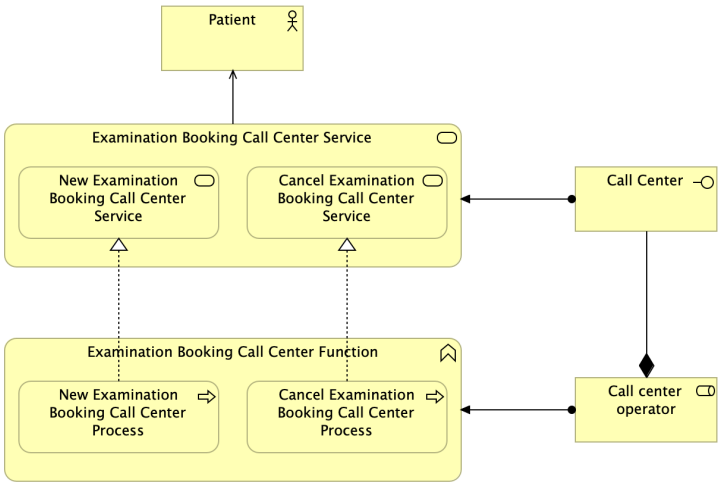

Figure 3.3: Example 3.3

In this example, we can see a possible use of `Business Functions` as the composition of different `Business Processes`. It is worth noticing that the *Examination Booking Call Center Service* is a combination of two sub-services which are realized by two independent `Business Processes`. As also suggested by Wierda in Sec. 14.22, "we should not model a Realization between each function [in our case process] and the overall services because they do not exist".

For the sake of simplicity, the *Examinations data* `Business Object` has been omitted. Details on how to manage this aspect will be detailed in Example 3.5.

As it is suggested by Wierda (see 17.2 page 93), a `Business Process` is the preferred option to realize a `Business Service`. For this reason, although admissible, the diagram does not have a `realization` relation between the `Business Function` and the `Business Service`. This will become clearer in the following when discussing the derived relations.

Using derived relations is possible to express additional information about the model. Derived relations are regulated as specified in https://pubs.opengroup.org/architecture/archimate3-doc/ch-relationships-Normative.html.

Notably, considering that `composition` relations exist between the nesting elements and the nested elements for both *Examination Booking Call Center Service* and *Examination Booking Call Center Process*, we are in the case discussed in section B2.2. As shown in Figure 3.4, with a chain of structural dependencies, i.e., `assigned-to` + `composition`, the resulting derived relation, reported in red, specifies that both the nested `Business Services` are offered through the same channel, i.e., *Call Center*, of the nesting `Business Service`. Similarly, the `Business Role` in charge of the nesting `Business Process` is also responsible for the nested ones.

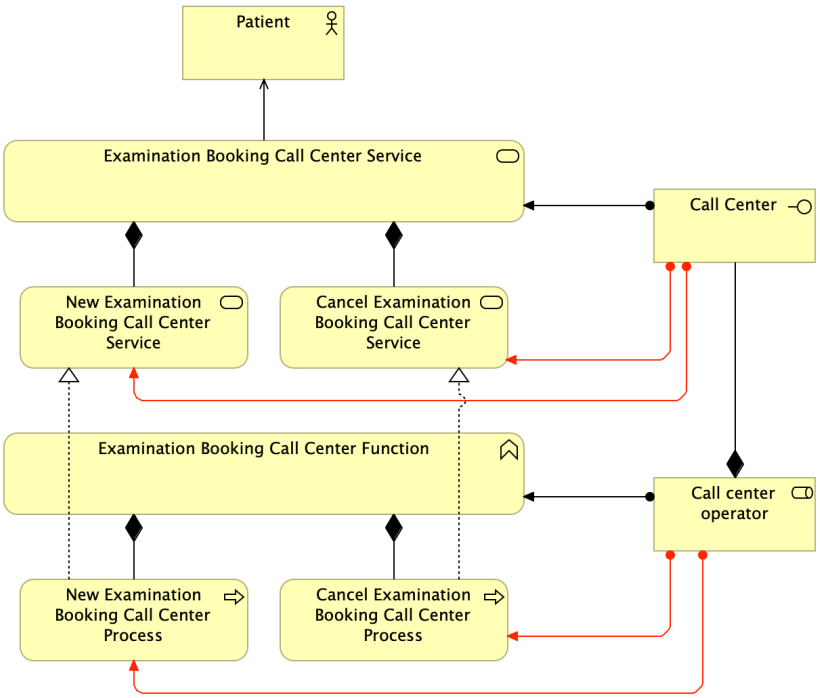

Figure 3.4: First set of derived relations from diagram in Figure 3.4

Another derived relation can be obtained considering the two chains `composition` + `realization` between *Examination Booking Call Center Function* and, respectively, the *New Examination Booking Call Center Service* and *Cancel Examination Booking Call Center Service*. As shown in Figure 3.5, two direct relations between the `Business Function` and the `Business Services` can be obtained.

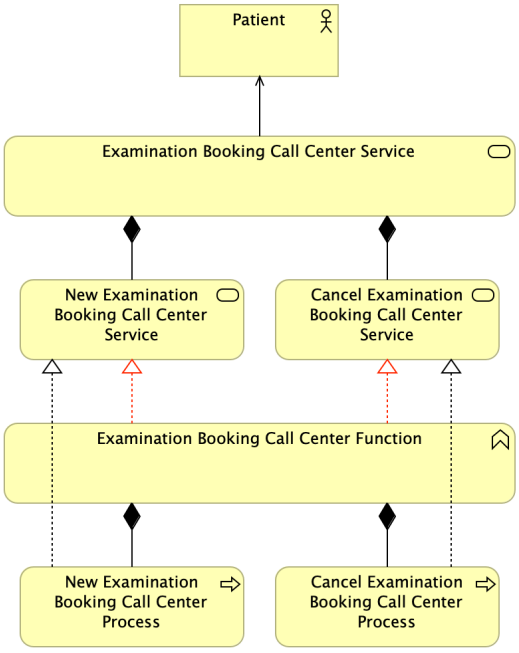

Figure 3.5: Second set of derived relations from diagram in Figure 3.5

Although admissible, in this second case, we do not think that the derived relation is as useful as for the first case. In fact, these new relationships do not add any meaningful information to the model

Example 3.4 - Business Process with multiple Business Roles

Focusing on the *Cancel Examination Booking Call Center Service* and the underlying process as modelled in Figure 3.2, when the *Call center operator* performs the composing `Business Process` *Analyze request* an agreement with the administration secretary is needed.

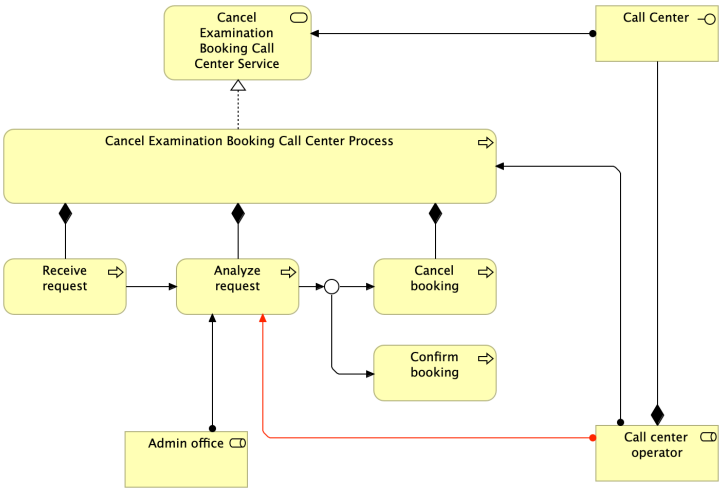

Figure 3.6: Diagram for Example 3.4

In a first attempt, the diagram related to this example could be the one shown in Figure 3.6. Considering the derived relation reported in red, it is clear that what is represented is in contrast with what mentioned above, i.e., a `Business Process` of which more than one `Business Roles` are in charge. To cope with this situation, ArchiMate offers two alternatives.

The first one is reported in Figure 3.7 takes advantage of the `Business Collaboration` as the aggregation of `Business Roles`.

The ArchiMate specification usually connects the active element `Business Collaboration` with `Business Interaction` as behavioural element. Actually, as also suggested by Wierda on page 33 (Section 8.2), the `Business Interaction` can be considered as equivalent to `Business Process`.

The second alternative, which has been introduced in version 3 of ArchiMate, takes advantage of a `junction` (see Figure 3.8). In this way, the *Analyse request* is connected to a single

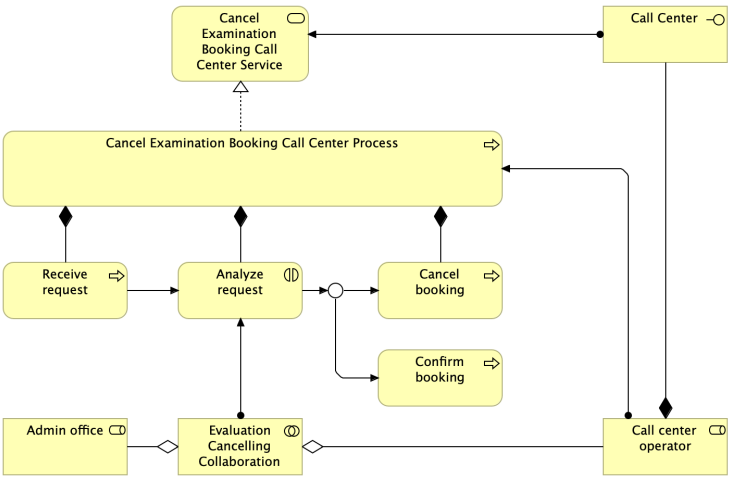

Figure 3.7: Diagram for Example 3.4

element which is obtained by the AND-junction of *Admin office* and *Call center operator*.

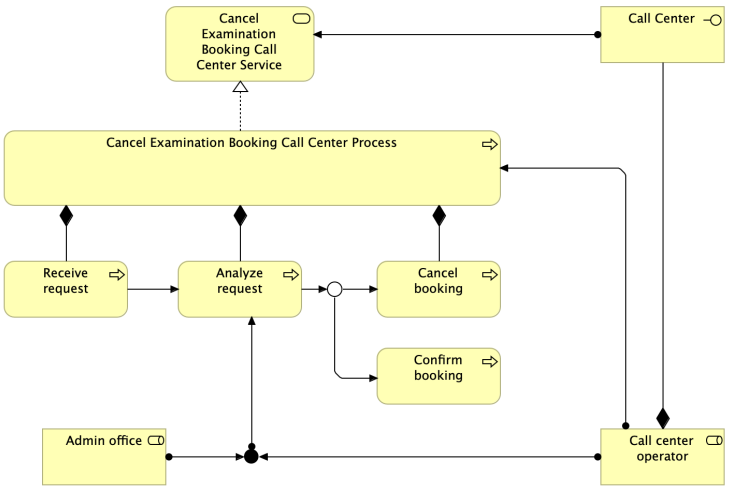

Figure 3.8: Diagram for Example 3.4

ArchiMate specification says (see 8.3.1) that "A complex business process may be an aggregation of other, finer-grained processes. To each of these, finer-grained roles may be assigned". This is a different situation than the one just described as the *Admin office* is not a fine-grained, i.e., `specialization`, of a *Call center operator*.

Example 3.5 - Business Objects

The examination data, that are used along the execution of the *Cancel Examination Booking Call Center Process*, contains the list of doctors, their availabilities, and the appointments already booked. The list of doctors is read by the *Receive request* to support the selection of the doctor, the availabilities are used by the *Search for availabilities*, while the *Appointments* is updated during the booking *Confirmation*.

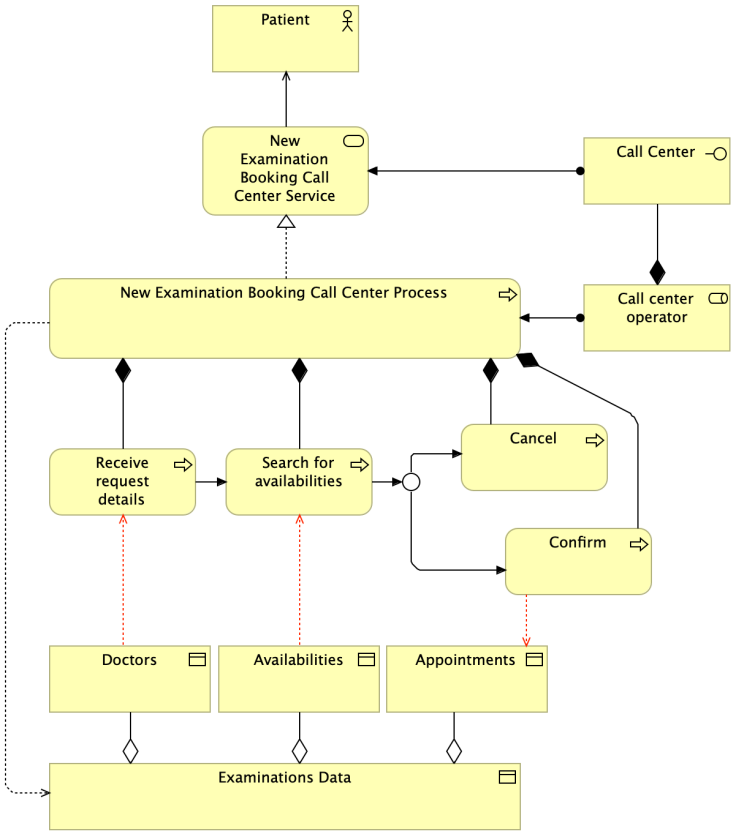

Figure 3.9: Diagram for Example 3.5

The diagram reported in Figure 3.9 enriches the diagram in Figure 3.2, where the `Examinations Data` is detailed in its components: i.e., *Doctors*, *Availabilities*, and *Appointments*.

Concerning the `Access` relationship, it is worth reminding what the ArchiMate specification says in Section 5.2.2: *"the direction of the relationship is always from an active structure element or a behavior element to a passive structure element, although the notation may point in the other direction to denote "read" access, and in both directions to denote read-write*

access. Care must be taken when using access with derived relationships because the arrow on the relationship has no bearing on its directionality.".

Following this specification, the goal of the initial diagram was only to indicate that the *New Examination Booking Call Center Process* access to the *Examinations Data*, thus, the direction of the `Access` relationship follows the direction from the behavior element towards the passive element, without specifying the type of access, i.e., read, write, or read/write.

Moving to this example, the text says that the steps in the `Business Process` has a specific access to the components of the *Examinations data* and they are reported explicitly in the diagram with the red arrows.

These access relationships have been reported in red, as they can be obtained as derived relationships. In fact, following the Potential Derivation Rule #6 in the specification (see Section B.3.2 of https://pubs.opengroup.org/architecture/archimate3-doc/ch-relationships-Normative.html#sec-Potential-Derivation-for-Structural-and-Dependency-Relationships), the path composed of `composed-of + access` between the *Receive request details* and the *Examinations Data*, can be replaced by a direct `Access` relationship between them. Then, the path composed of `access` (newly created) `+ aggregation` between the *Receive request details* and *Doctors* can be replaced by a direct `Access` relationship. This relationship will be directed from the `Receive request details` (the behavior element) to the *Doctors* (the passive element) but, once derived, as said in the specification, this can further specify the type of access: i.e., read mode, thus the direction is inverted. The same reasoning will be applied to the other two steps of the `Business Process`.

Example 3.6 - Automated `Business Process`

The ACME Hospital offers its patients the possibility to autonomously manage examination bookings through an advanced service offered via a dedicated Website.

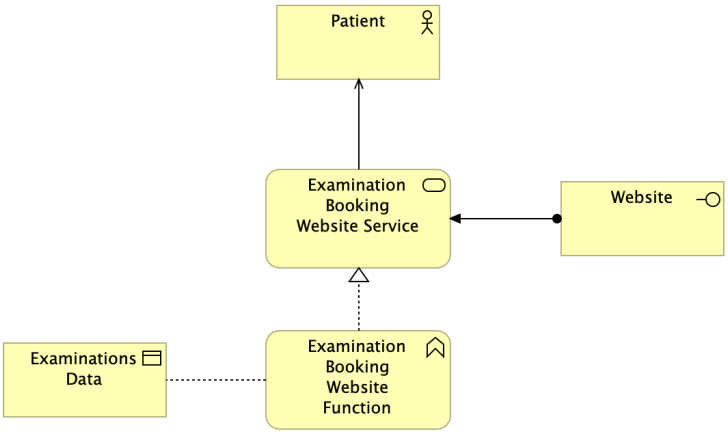

Figure 3.10: Example 3.6

The scenario described in the Example 3.6 can be represented with the typical pattern as shown in Figure 3.10. With respect to the normal case, the `Business Role` in charge of the *Examination Booking Website Function* simply does not exist as the process is automatic.

The modeling of automated business process is specified by Wierda in Section 7.14.

Example 3.7 - `Business Service` with multiple `Business Interfaces`

The ACME hospital also allows its patients to manage their bookings by interacting with a Website. The patients can book new examinations, get the list of existing bookings, and cancel an existing booking. About the latter, it cannot be used in a completely autonomous way, but also requires the intervention of a call center operator. In particular, we assume that a request for cancellation is performed via the Website, and then a confirmation is done after a call center operator calls the patient.

The diagram for Example 3.7 is very similar to what is described for Example 3.3. It is only noticeable, as shown in Figure 3.11, that there is an additional `Business Service` that is specific to the Website interface: *Browse Examination Booking Website Service*.

An important feature of this example is the need to specify how the *Cancel Examination Booking Website Service* requests integration between two `Business interfaces`. In the high-level specification shown in the figure, where we want to represent all the `Business Services`, it is sufficient to indicate *Website* as the `Business Interface` for all services, while *Call center* as a specific and additional interface for the cancellation.

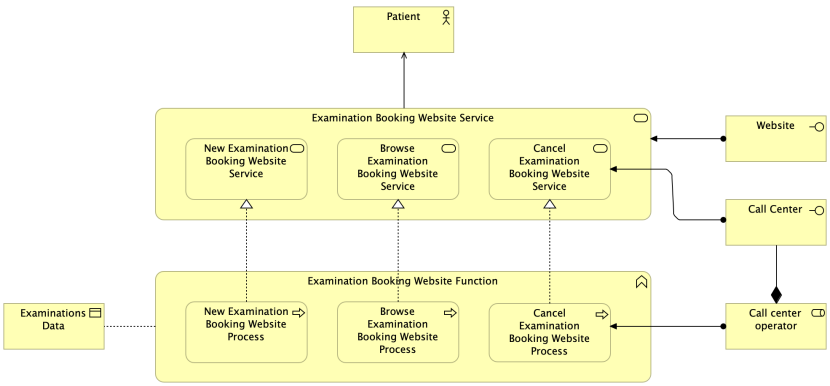

Figure 3.11: Example 3.7

ArchiMate specification, in defining a `Business Service`, explicitly admit – in Section 8.3.5 – the possibility to have more than one `Business Interfaces`. Conversely, as also described above, `Business Processes` or `Business Functions` must have a single *Business Role* assigned. In this specific example, this rule is not broken as the *Cancel Examination Booking Website Process* is assigned to a single role. By the way, this will became not true when considering the application layer.

.

Example 3.8 - Internal `Business Service`

An Italian company producing olive-oil company offers a service to its customers to buy its product online. This service offers the usual functionalities: browsing the catalog, adding/browsing/deleting the shopping cart, and confirming the order. Concerning the latter, the underlying process takes advantage of an internal service focused on the delivery management. This service will be driven by the information about the catalog, the products, and the orders.

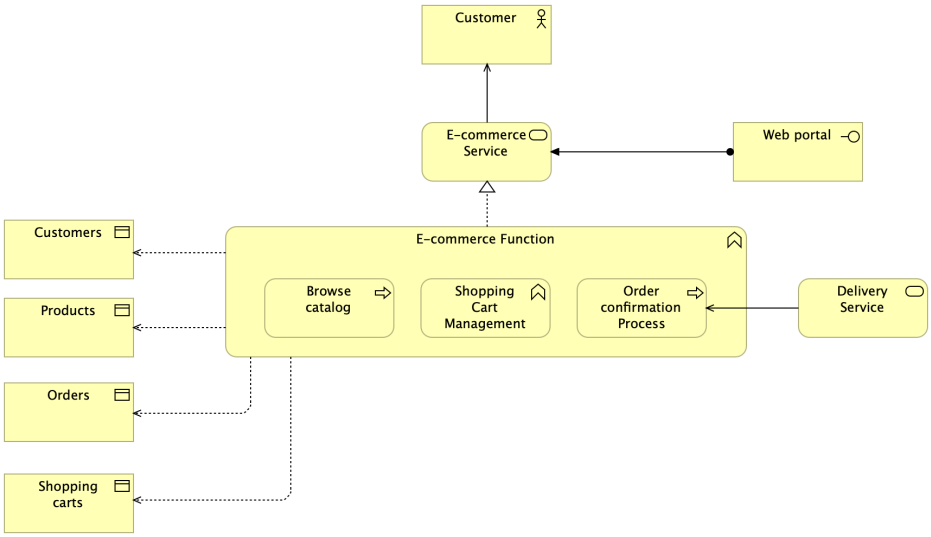

Figure 3.12: Diagram for Example 3.8

In this case, *Delivery Service* is a `Business Service` that has been declared as an internal service. This means that the consumer of this service will not be the final user (in this case the *Customer*) but another element behind the scene. In particular, the `serving` relationship is used to indicate that the *Delivery Service* is used during the *Order confirmation process*.

Concerning the `Business Objects`, they indicate the main concepts that are relevant for the given example. While it is clear that the *Customers*, *Products*, and *Orders* should be indicated in the diagram, for the *Shopping Cart* its presence could be questionable. In fact, this is an object relevant only during the execution of the process instance, as it will be destroyed as soon as the service is completed. Nevertheless, from a business perspective, this concept is visible to the *Customer*, thus it is worth being included.

The decision about the nature, i.e., `Business Process` or `Business Function` of the elements included in the *E-commerce function* is driven by the fact that, while the former and the latter are clearly composed of a series of steps (which are not actually defined in the text), the *Shopping Cart Management* groups a set of functions.

Although in the note of Example 3.3, we suggested using the `realization` relation between `Business Processes` and `Business Services`, in this case, it is indicated that the realization of the *E-commerce service* is given by the *E-commerce Function*. This is justified by the fact that the *E-commerce Service* is a generic service that is not really detailed. Expressing in the diagram that the *Browse catalog* and *Order confirmation process* realize the service is not correct as, not only the *Shopping Cart Management* is part of the realization, but also, taken singularly, they mean that a single element realizes the overall service. Which is not correct.

Example 3.9 - Usage of `Flow`

To improve the efficiency of the company and reduce costs, recently The ACME Insurance Company has decided to offer a new insurance service completely managed online. This service is dedicated only to protecting small objects (whose value is estimated as lower than 2000$) and it will be available only to reliable customers. This service will allow the customer to submit a new insurance request and, only if the customer is considered reliable by checking his/her past history, photos of the item are required to be uploaded along with some details (e.g., serial number, purchase date). The information about the object is passed to an employee of another business function for goods estimations. This function is used also for other activities in the company. This estimation is used to check whether the value of the item is really lower than 2000$. In that case, a contract to be signed is created, and it is sent to the customer who has to return it back to complete the procedure which terminates with the contract archival. In case the value exceeds the threshold, the customer is informed about other possible insurance policies that are more compliant with the type of object.

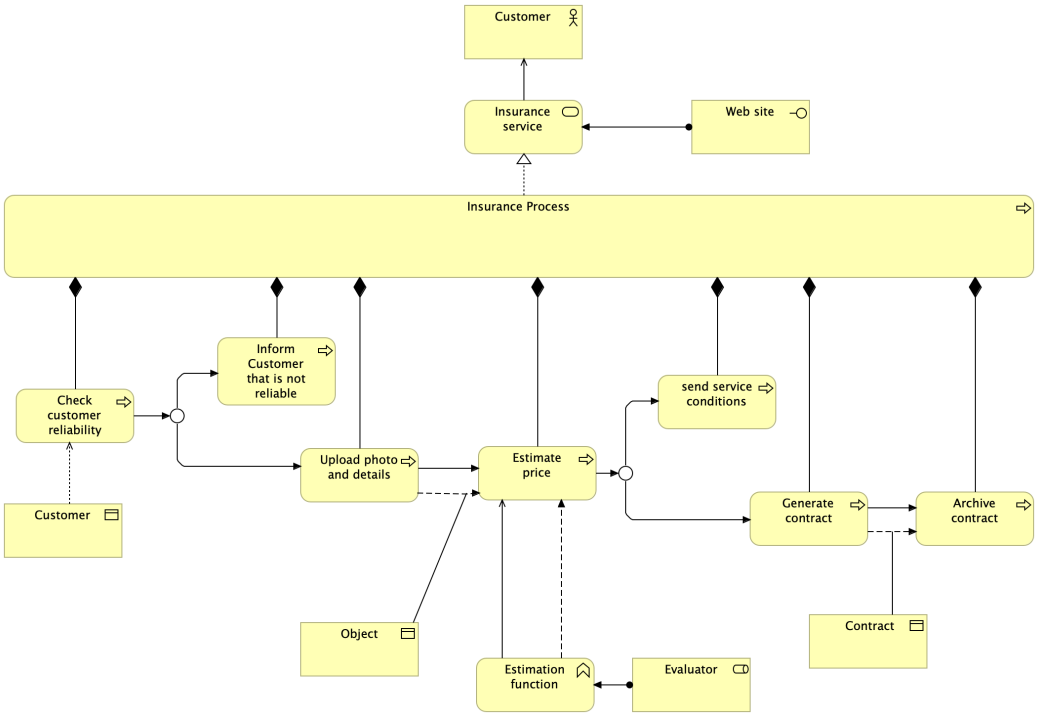

Figure 3.13: Solution for exercise 3.9

Reading the text, when defining the overall `Business Process`, it should be clear that the estimation function is an external element. In particular, this is a function that offers a service to the *Estimate price*. With respect to the previous example, there is no internal `Business Service`, but the `Business Function` *Estimation function* directly serves the *Estimate price*. The lack of `Business Service` indicates a strict coupling between the "servant" and the "served", while in the previous case, the mediation offered by the `Business Service` creates a more loose coupling.

With this setting, it is also possible to see how the `Flow` can be used to model the direction along which the information is passed between behavioural objects. Notably, when a `Trigger` relationship is used, as in the case of *Upload photo and details* and *Estimate price*, it is reasonable to assume that the data flows in the same direction as the trigger. Conversely, When a `serving` relation, it might happen that the data could flow in both directions. Nonetheless, having the `serving` relation suggests that one of the two connected elements is sending more valuable information. In this case, although it is reasonable that the *Estimate price* has to send the *object info* to the *Estimation function*, it is not worthy to be included in the model, leaving only the flow indicating the answer of the *Estimation function* towards the *Estimate price*.

A more detailed discussion about the relationship between `serving` relationship and `Flow` is given in Wierda in Section 24.2 page 128.

Example 3.10 - `Business Events`

The ACME Insurance company also provides a service to inform its customer in case the update of general terms of contracts in the company has an impact on the running contracts. In this case, the underlying process requires that the customer data and the data of the objects for which an insurance policy is active are recovered. These items are then re-evaluated, reviewed, and the related contracts sent to the customer for signature.

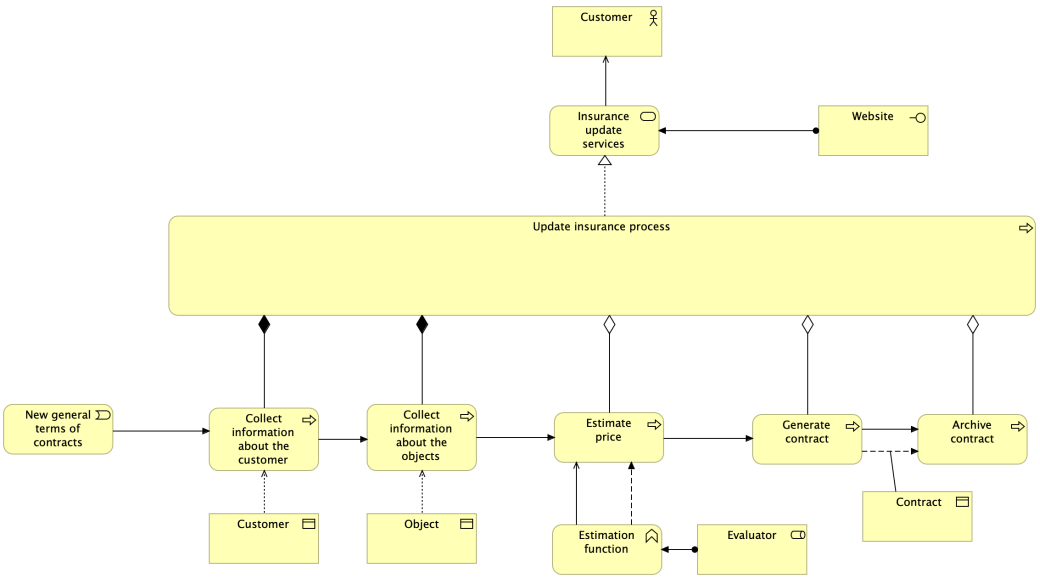

Figure 3.14: Diagram of example 3.10

In this example, a `Business Event` is introduced to indicate a specific state related to the update of the general terms. This event triggers the process where the last three elements are the same as in the previous example. For this reason, they are linked to the *Update insurance process* with an `aggregation` relationship, as it does not make sense an element is a component (i.e., it is a destination of a `composed-of` relationship) of more than one elements.

Example 3.11 - Multiple `realizations`

In addition to the possibility of booking examinations, the ACME hospital also offers the possibility to have information about the report related to the examinations when it is available. To access this service, the patient has to interact with an operator via the call center. If it is required, the call can be redirected to the phone of the medical doctor who has produced the report.

As shown in Figure 3.15, the *Medical Report Call Center Service* related to the new service offered by the hospital is realized by a combination of two `Business Processes`. One is managed by the *Call Center Operator*, while the second one is managed by the *Doctor*. Being the `Business Service` modelled as a realization of two `Business Processes` means that these `Business Processes` are involved but it is not taken for granted that both of them are required. In fact, only based on the specific case, the *Doctor* can be involved in the service.

Moreover, according to the discussion of Example 3.7, the *Medical Report Call Center Service* is modelled as offered by two interfaces, i.e., the *Call Center* and the *Phone call*, each of them related to the two different `Business Roles` involved, i.e., *Call Center Operator* and *Doctor*, respectively.

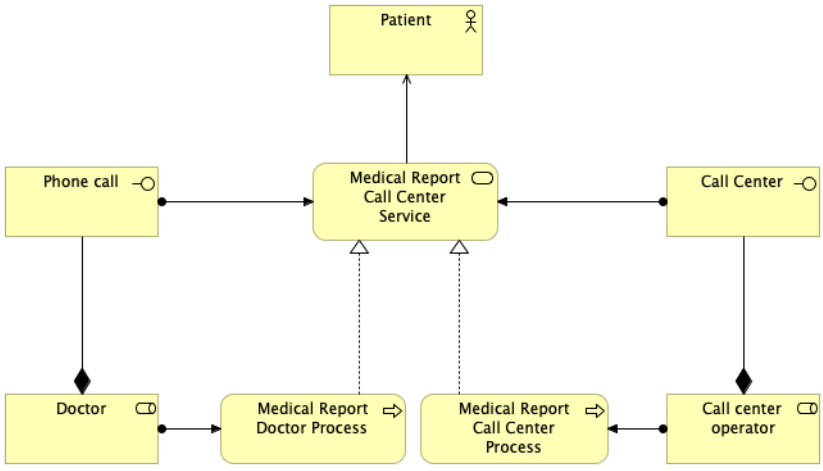

Figure 3.15: Diagram of example 3.11 - OR variant

As described by Wierda in Sec 14.22, when dealing with multiple realizations, the `junctions` can be used to indicate more precisely, if possible, how the `Business Processes` contribute in the realization of a `Business Service`. Notably, the variant represented in Figure 3.16 models the situation in which both the processes (there is a `AND-junction`) are required to offer the service. Furthermore, the variant represented in Figure 3.17 models the scenario in which the patient has to decide in advance either to communicate with the *Call Center Operator* or the *Doctor* (there is a `XOR-junction`).

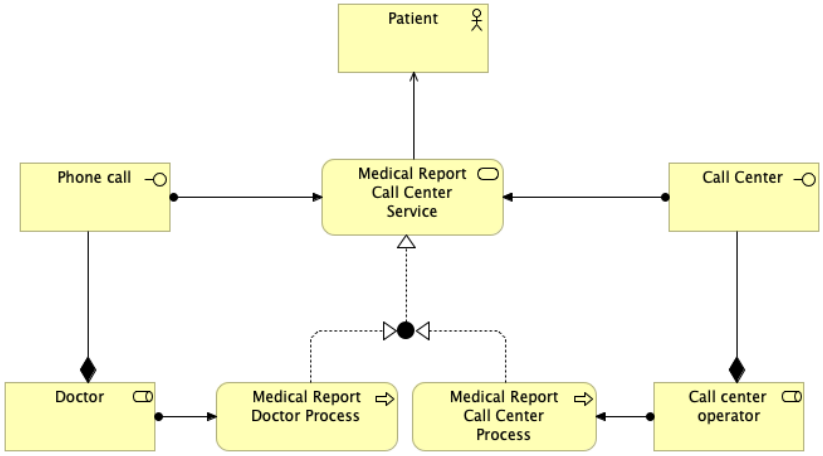

Figure 3.16: Diagram of example 3.11 - AND variant

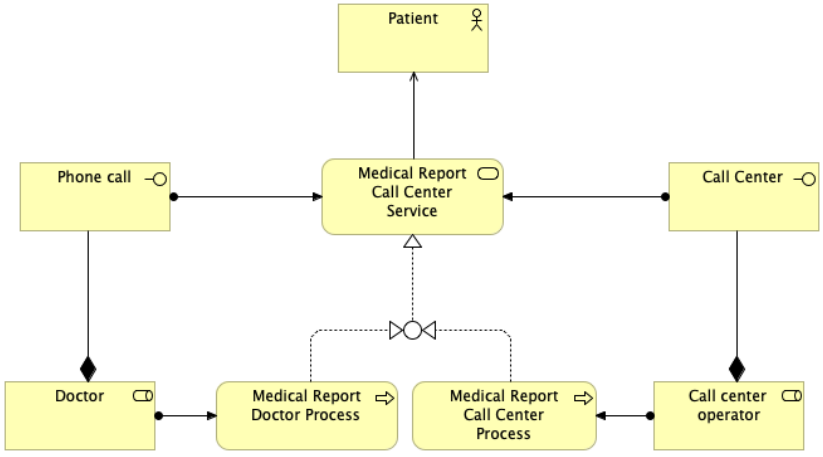

Figure 3.17: Diagram of example 3.11 - XOR variant

Exercises

Exercise 3.1. *Based on the diagram reported in Figure 3.18 where, instead of using the* `realization` *relation between* `Business Processes` *and* `Business Services`*, it has been used to connect the* `Business Function` *containing the* `Business Processes`*, check whether is possible to derive a model equivalent to the one in Figure 3.3.*

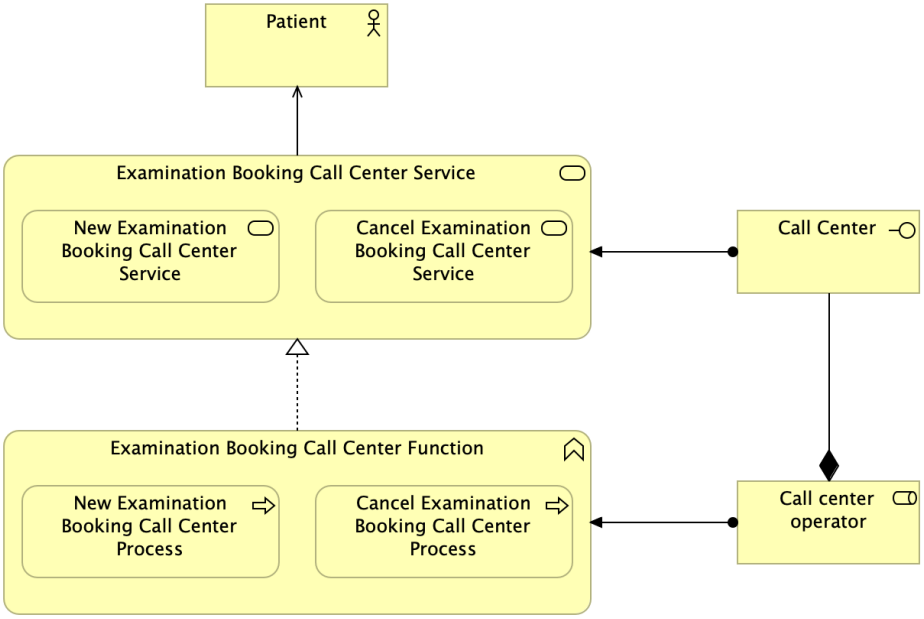

Figure 3.18: Diagram for Exercise 3.1

Exercise 3.2. *Considering the ArchiMate Diagram in Figure 3.6, describe why it is not possible to have a derived* `realized` *relationship from Admin office and Cancel Examination booking Call Center Service*

Exercise 3.3. *Considering the same scenario described in Example 3.9, assume that the generation of the contract is not automatic but is done by the same evaluator that estimated the price of the good and, in addition, requires that the contract needs to be revised and signed by the legal representative of the ACME Insurance company before sending the contract to the customer.*

Exercise 3.4. *To facilitate the use of its spaces, a Swiss university has decided to offer its students a new study room booking service. The service, accessible by contacting the secretariat, provides the possibility of booking a place in the study room, booking the entire*

study room, canceling one of the existing reservations, as well as requesting the catering service if the reservation includes a stay during lunchtime. In particular, this last activity requires the presence of a manager of the catering company.

Solutions

Solution 3.1. *The diagram in Figure 3.3 cannot be obtained from the diagram in Figure 3.18. In fact, only a partial equivalence can be obtained by considering the two* `realization + composition` *chains between the Examination Booking Call Center Function and, respectively, New Examination Booking Call Center Service and Cancel Examination Booking Call Center Service. This results in a* `realization` *relation between the* `Business Function` *and the two* `Business Services` *(see Figure 3.19). To have an equivalent diagram an additional step should be done to connect the two* `Business Processes` *and the* `Business Services` *but this is not possible as the* `realization` *and* `composition` *relations work in opposite direction. Being structural the two considered relations, no derived relation can be obtained.*

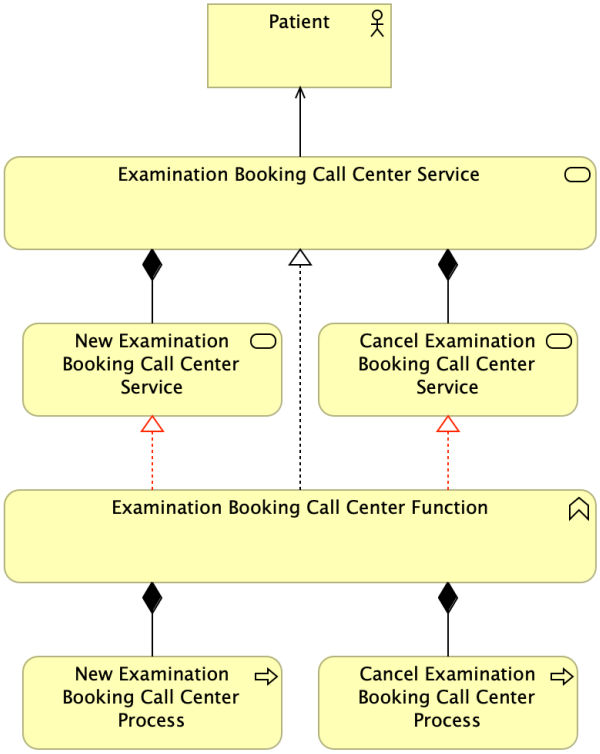

Figure 3.19: Partial solution for Exercise 3.1

Solution 3.2. *The Admin office and Cancel Examination Booking Call Center Service are*

linked through two structural relationships: i.e., `assigned-to` *and* `composition`*. Here, the problem is that these two relations do not have the same direction, thus it is not possible to obtain a derived relation.*

Solution 3.3. *In this case, a junction between the Evaluator and Legal representative is assigned to the Generate contract* `Business Process`*. It is worth noticing that the legal representative is an actor, while the evaluator is a role. In fact, it is reasonable to assume that in a company only a single representative exists.*

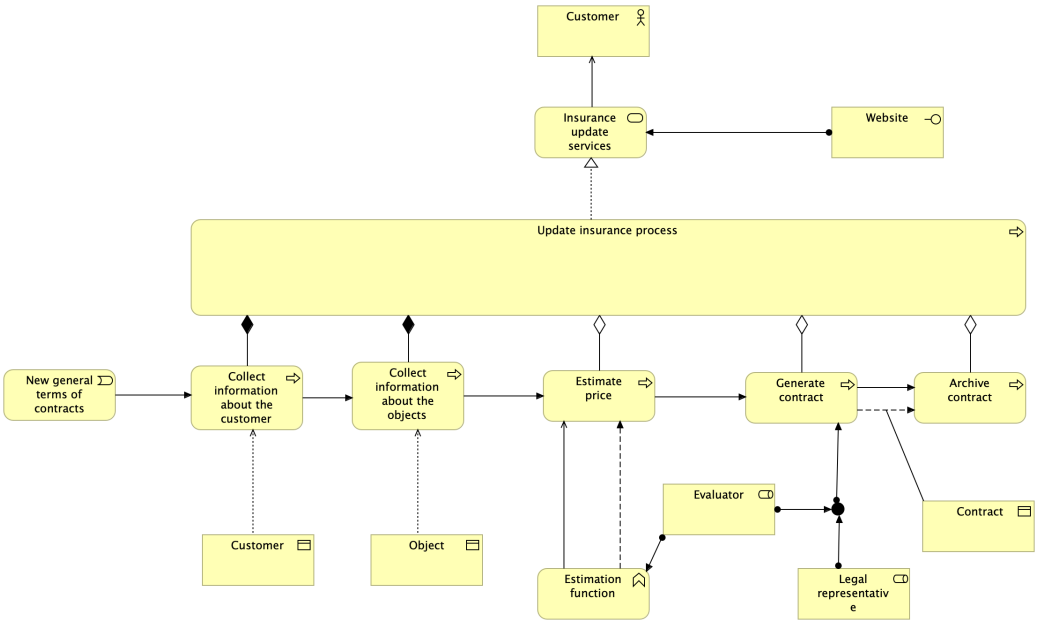

Figure 3.20: Solution for exercise 3.3

Chapter 4

Application Layer

This chapter focuses on the application layer modeling. The goal of this layer is to provide information on how the application portfolio is organized. Software applications are then described both vertically (what they are made of) and horizontally (what other applications can be used or used). Particularly important, in this layer, is the modeling of the integration between applications for which some representation patterns are presented.

Inspired by the scenarios described in the previous chapter, they have been investigated from an application perspective. In each case, at least two diagrams are proposed: one related to the application layer in which `Application Services`, `Application Functions`, `Application Interfaces`, `Application Modules`, and `Data Objects` are used; another diagram related to the connection with the `Business Layer` that clarifies how the application supports the business. In this way, the layered viewpoint begins to be outlined.

Example 4.1 - Basic pattern

The call center operator of the ACME hospital that has to assist the patients when booking new medical examinations is supported by a Web application offered by the CRM software.

Considering the basic pattern, which is valid also at application level, as shown in Figure 4.1, the *CRM* module represents the application in charge of offering the *Ass Examination Booking Function*. The *CRM* is offered through a *Web GUI* and, in this case, this interface is adopted to make the *New Examination Booking Service* available to the elements of the business layer. As there is no detail about the data managed to offer this service, the diagram include a generic `Data Object` to represent those data. A deep discussion on the data aspects will be introduced later in this chapter.

With respect to the business level (see Example 3.3, at the application level the `Application Function` is the preferred behavioral element in charge of realizing `Application Service`

.

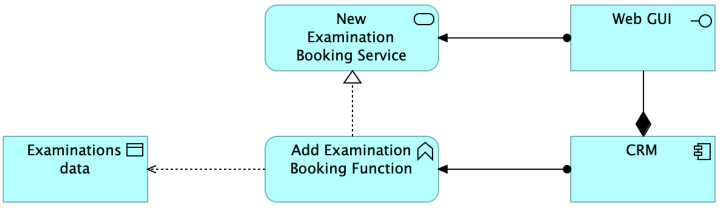

Figure 4.1: Example 4.1 diagram

At application layer, we have three possible interfaces: `API` which implies that the user is an application, `GUI` which implies that the user is a human being, and `CLI` which can be used by both: other applications or human beings.

Taking up the Example 3.1 that describe the new examination booking from a business perspective, the Figure 4.2 shows the connection with the application layer.

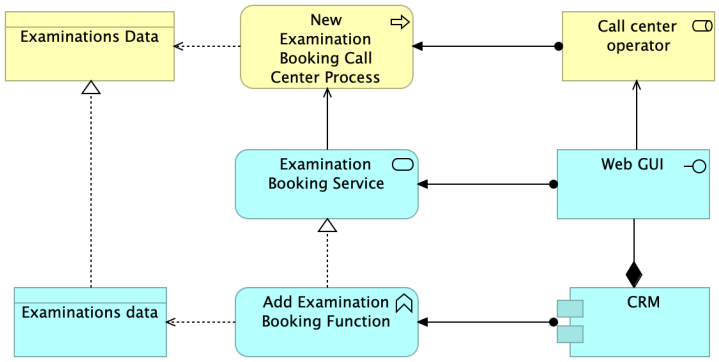

Figure 4.2: Connection between the Application Layer of Example 4.1 and related Business Layer in Example 3.1

Example 4.2 - Complex service

The CRM, yet via a Web interface, also offers the function to the call center operator to assist the patients in removing the examination booking. Due to a bad design, the CRM does not provide a good user interface, thus the user has to jump from one functionalities to another to perform a given task.

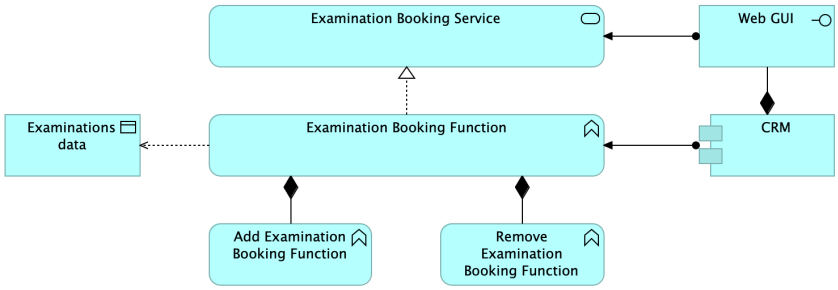

Figure 4.3: Diagram of Example 4.2

This new scenario implies that the *CRM* module offers an additional `Application Function` which contributes to the service offered at application level. As shown in Figure 4.3, this is modelled considering the that *CRM* is `assigned-to` the *Examination Booking Function*, a `Application Function` that acts as a container of the different functionalities supported. Consequently, a `realization` relationship holds between the *Examination Booking Function* and the *Examination Booking Service*. The advantage of using this approach to group behavioral elements (which is actually the goal of the functions in ArchiMate) is to make the diagrams more readable. As shown in the Figure 4.4, the relationships with the `Application Interface`, `Application Module`, and `Data Objects` need to be duplicated, thus uselessly making the diagram more complex. With respect to the similar situation at business level (see for instance, the Example 3.3 reported in Figure 3.3), in this case, there is no details concerning the structure of the `Application Service`, thus not being divided in two sub services, there is no need to have them connected with the related `Application Function`.

A peculiar element of this example relies on the bad user experience of the CRM which suggest that it is not easy to directly connect the functions offered by this module to the business level. For this reason, the *Examination Booking Service* is not modelled as a composition of two `Application Services` related to the creation and removal of an examination booking. In this way, the model expresses the case in which there is not a direct exposition of the `Application Functions` offered by the `Application Module` through `Application Services`.

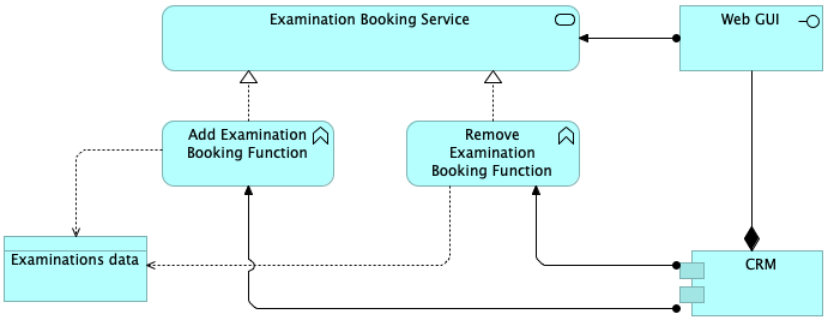

Figure 4.4: Diagram of Example 4.2 without grouping function

Leaving a generic service implies that it is not expressed in which way the two `Application Functions` are used at business level. In fact, as shown in Figure 4.5, the `Business Functions` are all served by the `Application Service` but it is not possible to define which `Application Function` contributes in serving which *Business Process*.

From now on, when it is not explictly required, the diagrams connecting different levels includes only the elements for which a relationship between the level holds.

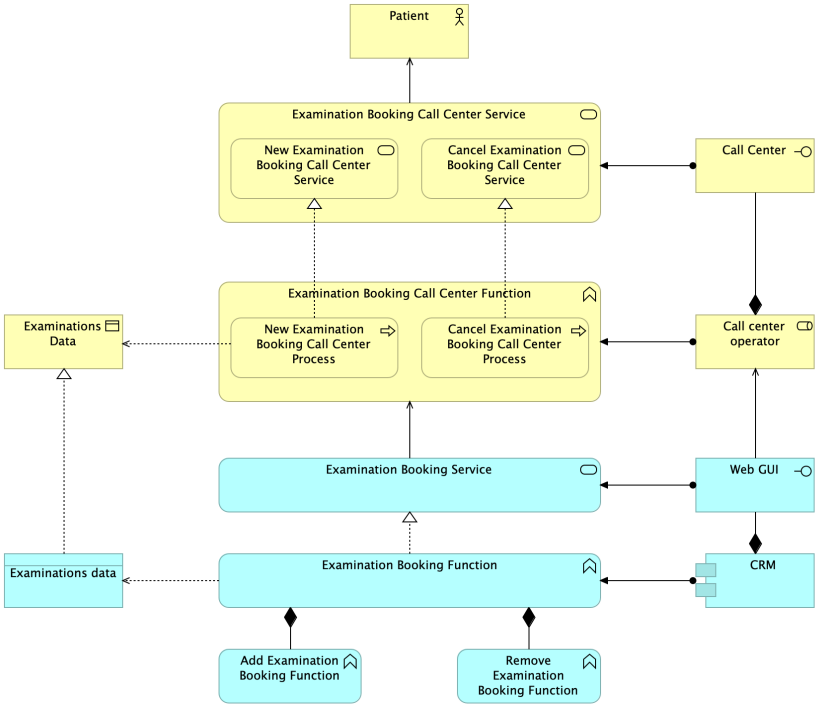

Figure 4.5: Connection between the Application Layer of Example 4.2 and related Business Layer in Example 3.3

Example 4.3 - `Data Objects`

To support the offering of the service related to a new booking (see Example 4.1), the CRM adopted by the ACME hospital stores information about the doctors and their availabilities in a dedicated system, while the appointments are stored in a different environment.

While, at business level, information is modelled from a conceptual standpoint, at application level it is important to capture how those concepts are represented to enable the automated processing (see ArchiMate Specification https://pubs.opengroup.org/architecture/archimate3-doc/ch-Application-Layer.html). Considering the diagram in Figure 4.6, and the case expressed in the text of the example, two `Data Objects` are considered: *Doctors* (which also contains their availabilities), and the scheduled `Appointments`.

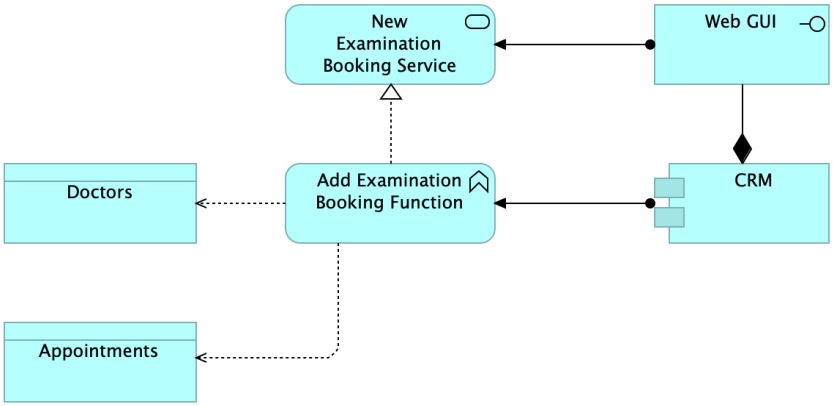

Figure 4.6: Diagram of Example 4.3

With respect to the previous example in this chapter, the *Examinations Data* could be no longer considered. Nor as a composition of these newly introduced `Data Objects`. This because there is no sense to have a `Data object` which is not resulting in any real information structure.

How the `Data Objects` contribute in the realization of the `Business Object` *Examination Data* as reported in Figure 3.1 is represented in Figure 4.7. In this case, the junction is used to express that both the `Data Objects` *Doctors* and *Appointments* contributes in the realization of the *Examinations Data*.

In case the `Business Objects` are defined more in details, as in the Example 3.5, the

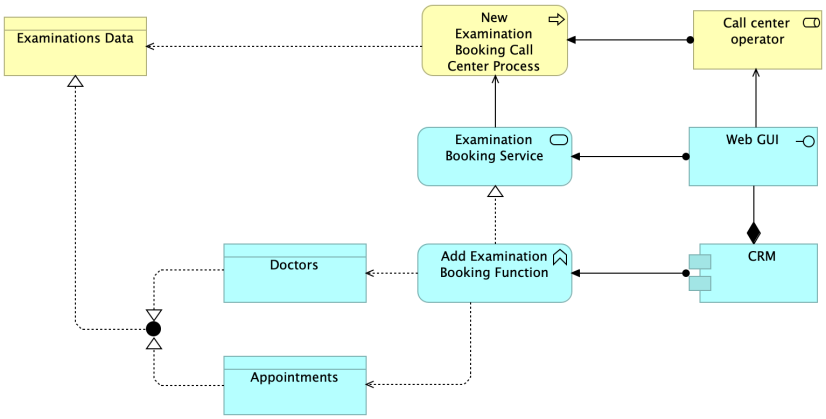

Figure 4.7: Connection between the Application Layer of Example 4.2 and related Business Layer in Example 3.1

diagram defining the link with the application layer can be the one shown in Figure 4.8. In this case, it is clear that the `Data Object` *Doctors* contributes to the realization of both the `Business Objects` *Doctors* and *Availabilities*. Conversely, the `Data Object` *Appointments* is in charge of the realization only of the related `Business Object`.

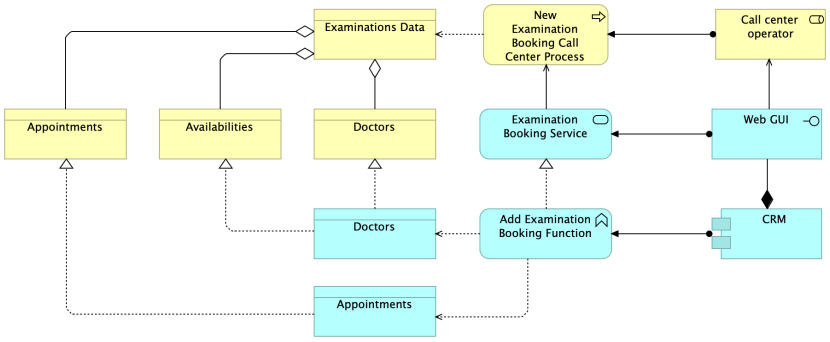

Figure 4.8: Connection between the Application Layer of Example 4.2 and related Business Layer in Example 3.5

Example 4.4 - Application integration

The functionalities offered by the CRM are obtained with an interaction with the ERP owned by the ACME Hospital. In particular, the information about the doctors are created by the ERP. The same information is used by the CRM to indicate the availabilities and to define the bookings for the patients.

This example concerns the possibility to model the integration that could occur at application level. In this case, the integration occurs by sharing the same `Data Objects` as shown in Figure 4.9.

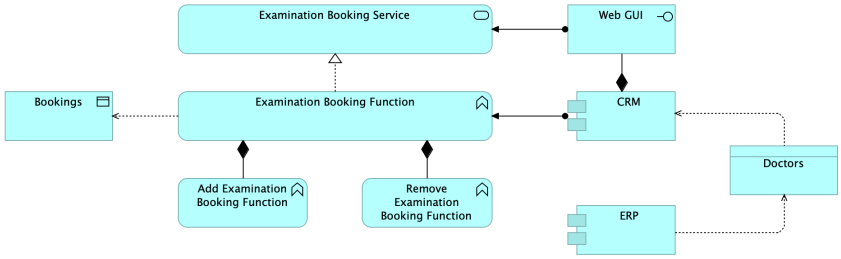

Figure 4.9: Diagram of Example 4.4

Another possible integration, that is common in the information system, implies a direct communication between applications. This assumes that a `Application Module` offers a `Application Function` exposed as a `Application Service` that is called by another `Application Function` offered by another `Application Module`. This situation is represented, in two versions by considering the derived relationship shown in red, in the diagram in Figure 4.10.

When introducing `Application Services` it is important to indicate who is calling what and this is can be done using the `flow` relationship. In this case, the *ERP* offers a service (which is not represented as it is hidden by derived relationships) that is called by the *CRM*. The direction of the flow, from the *ERP* to the *CRM*, indicates the direction of the main data, thus the *CRM* uses the data offered by the *ERP*. A deep discussion about who is calling what, is provided by Wierda in Section XX

Exploiting the derived relationships, it is possible to directly connect the *ERP* with a `serve` relationship to the *CRM*. This is indicated by the red arrow in the Figure. Although this is admissible, this implies that the *ERP API* `Application Interface` is no longer needed in the diagram. Depending on the case, removing this information could negatively affect the clarity of the diagram as it removes an information, i.e., integration via API, that could be important.

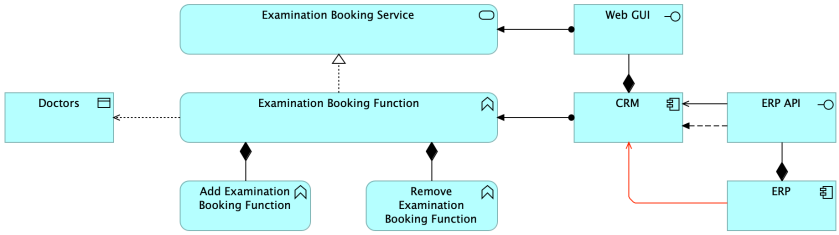

Figure 4.10: Diagram for a variant of Example 4.4

A further mode of integration between applications involves the use of a middleware, which could be, for instance, an enterprise service bus or a message-oriented middleware. In any cases, at the application level, it is only necessary to highlight the type of communication among the systems, without referring to the technology used for integration, which will then be the task of the diagrams at the technological level.

Therefore, focusing on the application level, in addition to integration based on data sharing (see Figure 4.9) or a direct call (see Figure 4.10) it is possible to define this integration in less detail through an application collaboration or through a flow of data between the parties (see Figure 4.10). On the left hand, it is shown how two systems dialogue with each other without necessarily requiring a clear definition of who produces information and who consumes the information. On the right hand, when you want to clearly express the direction of the information flow without saying that the mechanism is based on a direct call (as in the case of Figure 4.11), then it is sufficient to indicate the direction of the flow.

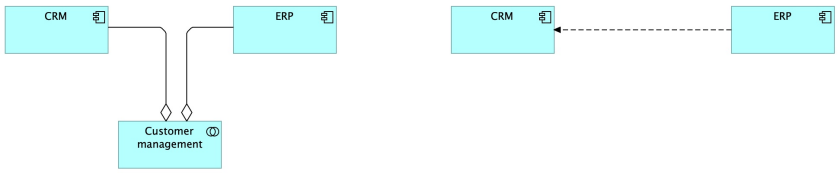

Figure 4.11: Additional integration methods

Example 4.5 - Automated `Business Service`

The service offered by the ACME Hospital to allow the patients to autonomously manage the bookings through a Web site (see Example 3.6), is natively offered by the CRM installed in the hospital using the same functions as accessed by the call center operators.

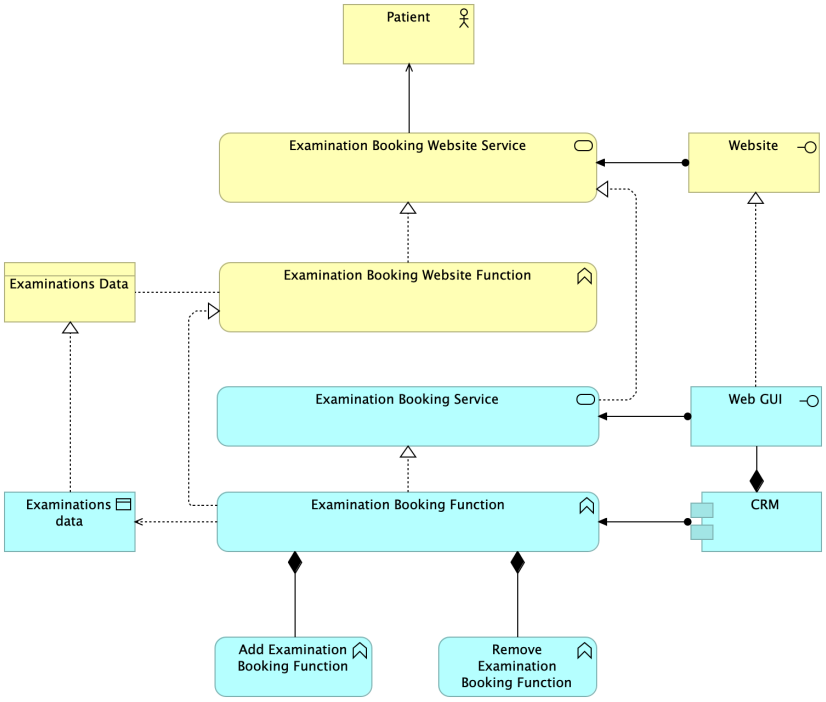

Figure 4.12: Diagram of Example 4.5

The diagram reported in Figure 4.12 shows the application of the pattern suggested by Wierda in Section 7.14. Being the *Examination Booking Website Service* fully automated (from a final user standpoint), the corresponding `Business Service` is directly realized by the underlying `Application Service`. To make this possible, a `realization` relationship between the related `Application Function` and `Business Function` must hold as well.

Example 4.6 - Multiple `Application Interfaces`

For historical reasons, in addition to the *Web GUI* offered by the CRM to offer the `Examination Booking Service`, the CRM also exposes its services through a `Desktop GUI`. Considering the Example 3.7, the *Web GUI* corresponds to the Web site with which the patients interact, while the *Desktop GUI* is used by the call center operator to assist the patients in case a booking cancellation is required.

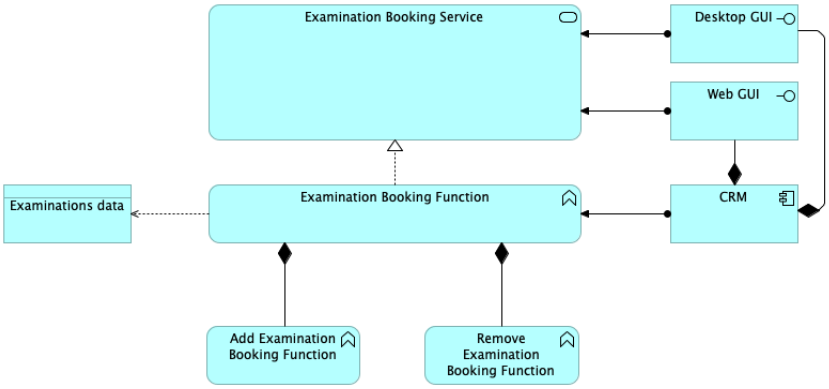

Figure 4.13: Diagram of Example 4.6

It is quite common that the same service could be offered through different interfaces, this is also supported by ArchiMate (see Section 9.2.3 of the specification) where "The same application service may be exposed through different interfaces, and the same interface may expose multiple services". At application level, shown in Figure 4.13, the availability of both the *Desktop GUI* and *Web GUI* are indicated, while also considering the connection with the business layer (see Figure 4.14), it is possible to show in which context the interfaces are used.

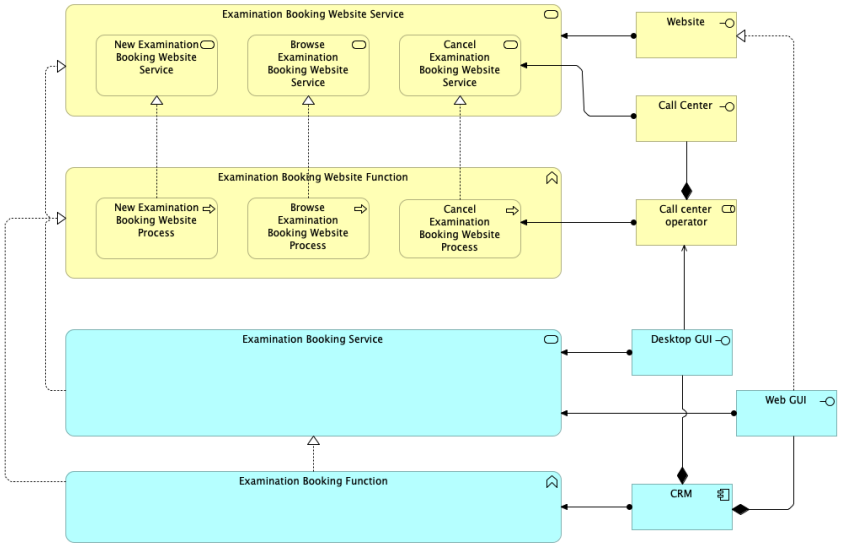

Figure 4.14: Connection between the Application Layer of Example 4.6 and related Business Layer in Example 3.7

Example 4.7 - User-driven application integration

Considering the Example 3.8, the functionalities required to support the *E-commerce function* are provided by two main software modules: a catalog system manager and an ERP. The former is in charge of allowing the customers to browse the catalog, the latter provides the functions for the shopping cart management and the order confirmation.

Focusing on the application layer, this example refers to a scenario in which two `Application Modules` are involved: *ERP* and *Catalog System*. As the `Business Service` is accessible with a web interface, we assume that both these two modules are exposed accordingly. Another assumption concerns the `Application Functions`: one exists for each of the `Business Function` or `Business Process` involved. The resulting diagram, as shown in Figure 4.15, is composed of two independent blocks, at least from the application standpoint. These blocks refers to the two `Application Modules` that in this context are not integrated. In fact, the integration between the two is performed, indirectly, by the user of the application.

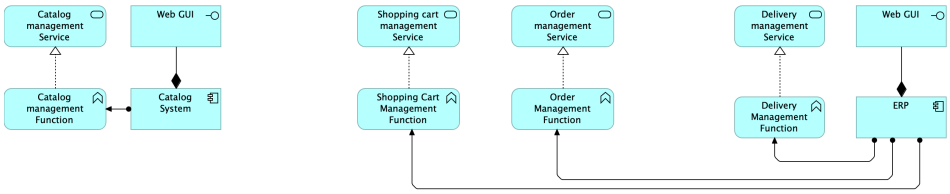

Figure 4.15: Diagram of Example 4.7

This type of user-driven integration is clear when considering the connection between the business and application layers shown in Figure 4.16. In this case, the realization of the `Business Interface` is obtained by the combination of both (in fact the realizations are joined with an AND-junction) the *Web GUI* offered by the two main `Application Modules`: i.e., the *ERP* and the *Catalog System*.

In this case, the AND junction has been used because, looking at the overall *E-commerce function*, it is reasonable to assume that all the `Application Functions` offered by the *Catalog System* and the *ERP* are used along the *E-commerce service* exploitation.

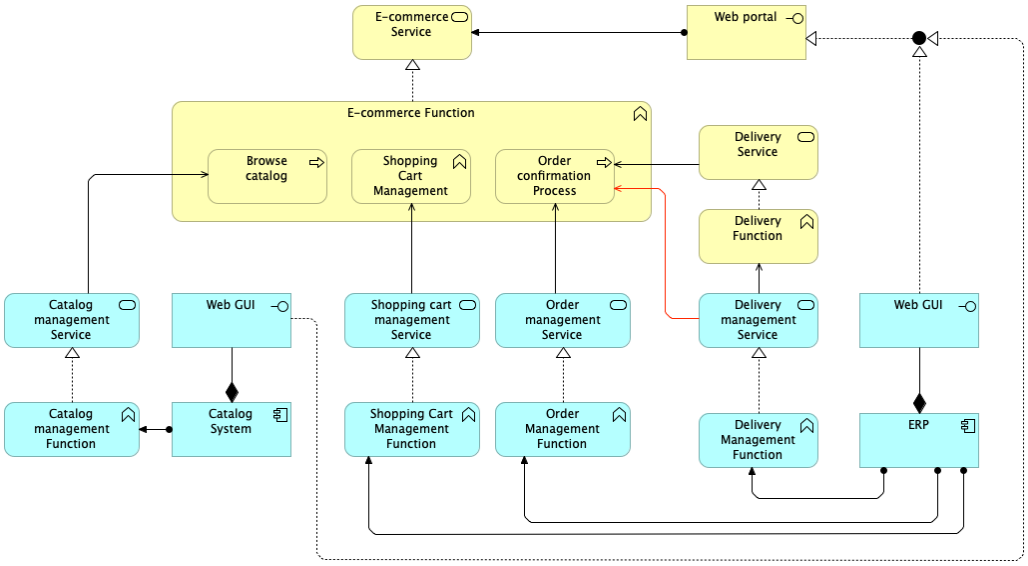

Figure 4.16: Connection between the Application Layer of Example 4.7 and related Business Layer in Example 3.8

Chapter 5

Technology Layer

This chapter focuses on the technology layer modeling which implies the modeling of the technology infrastructure on which the application has to be deployed and run. Adopting the approach followed by Wierda, which deeply leverages on the `node` as the element including the deployment resources available, in this chapter, we analyse how different types of deployment can be modelled. In particular, there is a specific focus on understanding the difference between system software and application software and how they are related to the application layer

In this way, typical patterns are introduced: from on-premise to cloud-based solutions. With respect to the previous chapters, we are not exploring all the possible elements made available by ArchiMate, but we focus only on `Device`, `System Software`, `Technology Service`, `Technology Function`, `Node`, and `Artefact`.

With respect to the previous layers, inspired by the work done by Wierda, a different basic pattern is proposed. For a better understanding, this layer is presented along with the relevant elements of the application layer to which it is connected.

Before introducing the pattern, it is firstly important to remind, from the application layer, that the `Application Module` is the logical element able to provide an `Application Function`. The executable file corresponds to its physical counterpart and is represented as an `Artifact` deployed on a `Device`. Please note that the concept of deployment (or installation) is represented by the `assigned-to` relationship.

As it usually happens, the executable file, alone, is useless as it requires an operating system

to run. For this reason, a specific `System Software` is assigned to the `Device` that also hosts the *CRM Executable file*.

An operating system is not mandatory. There are situations in which the application is installed on the bare metal. This is the case, as it will be discussed later, of the hypervisors in the bare metal configuration.

Most of the discussions in this chapter will be focused on the distinction between the `System Software` and the `Application Module` and the need to understand what to place on the technology layer rather than on the application layer.

Example 5.1 - Standalone, monolithic application

In its first implementation, the CRM offering the *Examination Booking Function* introduced in the Example 4.2, was not Web based, but it was implemented as a Window-based standalone monolithic application. This solution was sufficient as the call center operator was unique.

Considering the diagram shown in Figure 5.1, it is important to highlight that the application is: (i) standalone, i.e., it does not depend on any other software; (ii) monolithic, i.e., it is physically contained in a single artifact; (iii) windows-based, i.e., it runs on a specific OS.

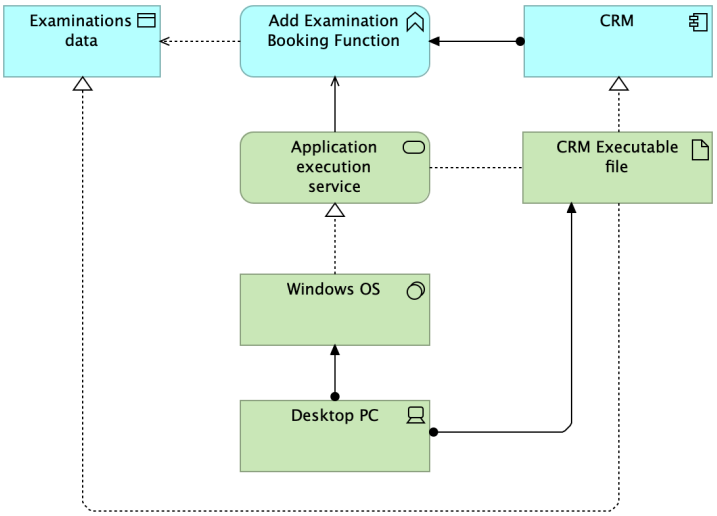

Figure 5.1: Diagram of Example 5.1

Starting from the latter point, a *Windows OS* is included as a *System Software* deployed on the *Desktop PC* where also the *CRM Executable file* is stored. Having a single `Artifact` corresponds to the monolithic nature of the CRM. The characteristics of being standalone, it is mapped in the diagram as the presence of a single `Artifact` in charge of everything, i.e., also the storage of the *Examination data* `Data Object`.

The last element of the diagram is the *Technology Service* that is required only to support the connection between the two layers. In fact, due to the layered modeling approach adopted by ArchiMate, a `System Software` should not be directly connected to the `Application Function`. Conversely, a `Technology Service` is required. In this way, the message delivered from the diagram is that the *CRM* `Application Module` is up and running on a Windows-based machine by means of the *Application execution services* offered by the OS.

The infrastructure provides only the tools (i.e., the OS calls) to start up the application leaving all the effort to manage it to the elements of the application layer.

The *Application execution service* is connected to the *CRM Executable file* through an `access` relationship. This reflects the need to operate on this file when the *Add Examination Booking Function* needs the functions offered by the *Application execution service*.

Example 5.2 - One-tiered application with external database

Due to the increase in the data to be managed, the CRM application has been improved accordingly. In particular, a Microsoft Access DB has been adopted to better manage and access the information about the examination. This DB can be accessed through ODBC protocol and must be located on the same machine as the CRM.

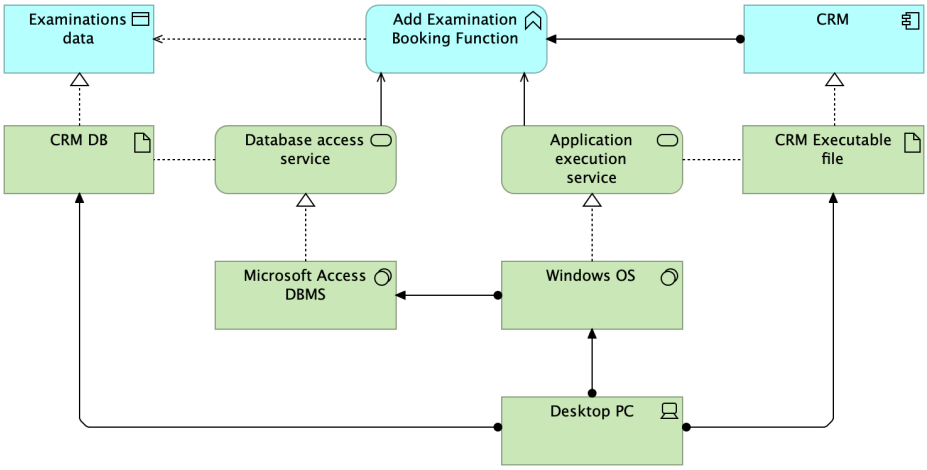

Figure 5.2: Diagram of Example 5.2

According to the new setting, a distinction between the application logic and the data logic is introduced. The former is offered by the *CRM* `Application Module`, while the latter is in charge of offering the *Examination Data* `Data Object`. This separation of duties is obtained at the technology layer by introducing a new `System Software` that corresponds to the *Microsoft Access*. This `System Software` is installed on the same machine and on top of the same OS on which the *CRM* `Application Module` is installed. The role of this `System Software` is to offer a typical `Technology Service`, i.e., *Database access service*, that can be used by the *Add Examination Booking Function* to support the operations at the business level. It is worth noticing that this `Technology Service` does not mention in its name the actual technology used (i.e., Microsoft Access) making the `Application Function` independent from the specific choice.

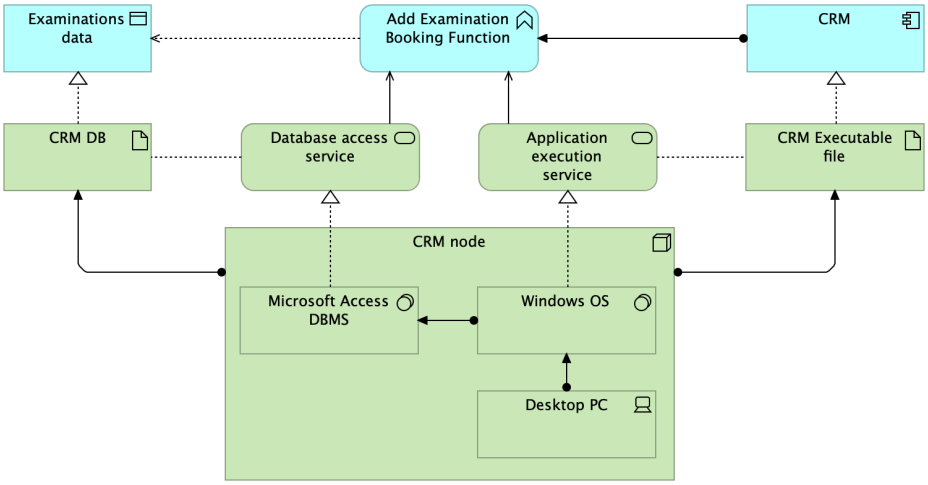

Figure 5.3: Variant of the Diagram in 5.2 with `Node`

The text of the example mentions ODBC protocol as a means of communication between the application logic and the data logic of the module. Anyway, this is not included in the diagram as we can consider it as an internal detail. In fact, as all the elements are deployed on a single machine, thus it can be classified as one-tiered and the ODBC represents an internal communication protocol that is not considered worth to be detailed.

For the sake of simplicity, differently from what is done by Wierda (see page 71), the reported diagrams do not use the `Technology Service` named *Exploitation* to group the different services that are used by a single `Application Function`.

With respect to the Example 5.1, the `Artifact` that realizes the *CRM* `Application Module`, i.e., *CRM Executable file*, is not in charge of managing also the data. In fact, the `Database access service` has this role and manages the *CRM DB* which realizes the `Examination Data`.

The `assigned-to` relation between the *Desktop PC* and the *CRM DB* indicates that the database has been stored on the same device as the *CRM Executable file*. Nevertheless, the current setting does not longer refer to a standalone application, as the *CRM* needs the presence of another application, i.e., *Microsoft Access*.

To make more evident how the applications can be considered one-tiered, thus deployed on a single node, it is convenient to introduce the `Node` that, based on the ArchiMate specification, is defined as "computational or physical resource that hosts, manipulates, or interacts with other computational or physical resource". Thus, it is considered as a logical representation of a set of elements able to provide some services. As reported in Figure 5.3, the `Device` and a `System software` installed on it are all included in a single `Node`.

Another alternative representation of this infrastructure is presented in Figure 5.4.

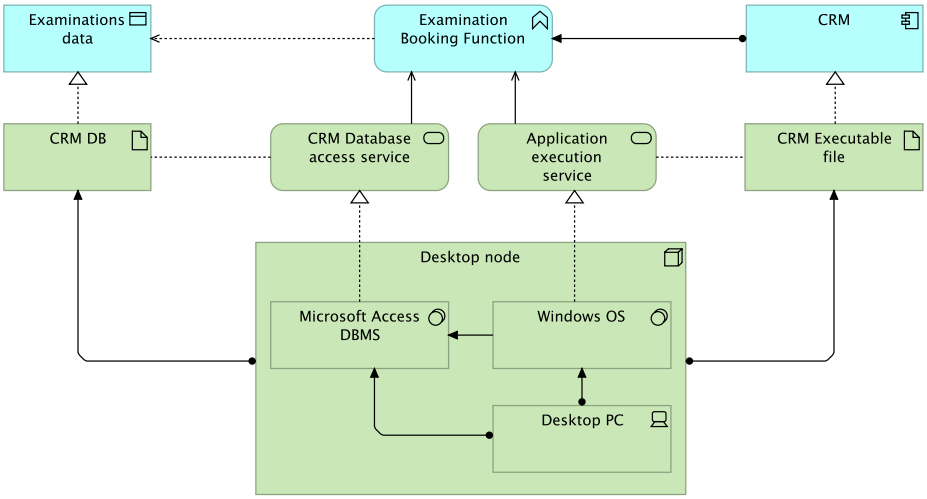

Figure 5.4: Alternative variant of the Diagram in 5.3

In this case, within the `Node`, both system software elements *Microsoft Access DBMS* and *Windows OS* are assigned to the `Device` *Desktop PC*. A serving relation between the two system software elements is also shown. This is an alternative view to represent this type of infrastructure. In the following, we adopt the representation shown in Figure 5.3 as it is more compact.

Example 5.3 - Two-tiered application with external database

Due to the increasing number of requests, the ACME hospital decided to employ additional call center operators. Accordingly, the CRM has been revised towards a desktop-based two-tiered application. All the call center operators can use a dedicated desktop application that share the same Microsoft Access database. The connection between the desktop application and the DB is based on ODBC.

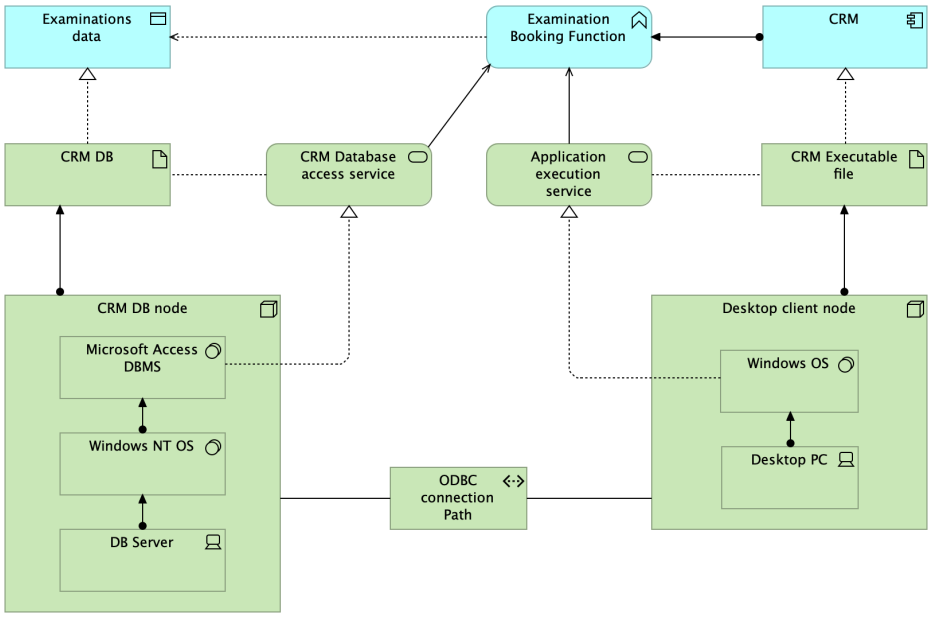

Figure 5.5: Diagram of Example 5.3

The new setting, where *Microsoft Access* runs on a separated device, is reported in Figure 5.5. This new configuration permits to understand the importance of the `Node` notation to make more evident how an application is distributed.

From a service perspective, i.e., the part visible to the application layer, nothing has changed. Conversely, the lower part of the diagram indicates that there are two nodes, i.e., *CRM DB node* and *Desktop client node*. The former provides the installation of the *Microsoft Access DBMS*, while the latter, the environment to host the *CRM executable file*. The two nodes are connected each other with a `Path` that indicates a logical link between the nodes, thus, without the need to indicate the exact structure of the network. In this case, it is found convenient to use the `Path` to indicate that the connection based on the *ODBC Protocol* that has been used for the communication between the CRM and the database.

In addition to the motivation related to the level of details expressed in the Example 5.2, the `Path` was not used because it can connect only two `Nodes` and in that case all the elements were included in a single node.

Example 5.4 - Web based three-tiered application

As additional step in its innovation, the CRM has been replaced with a JEE application running on Tomcat. For performance issues, also the DMBS has been substituted with a MySQL server. This permits to expose a Web GUI to the call center operators

With JEE technology, it is possible to implement a typical client-server web-based application where adding the database server results in a three-tiered deployment configuration. As shown in Figure 5.6, in addition to the *CRM DB node* already introduced in the previous examples, the middle layer is represented by the *CRM Web server* based on Tomcat, a JEE-compliant web server. This server offers the *Web app service execution* `Technology Service` that generates the *Web GUI* used by the call center operator. Finally, the *Desktop client node* represents the layer in which the Web GUI can be accessed.

The three nodes are connected through the ACME hospital *LAN*. This is reported in the diagram using the `Communication Network` element. With respect to the `Path` that indicates a logical connection, the `Communication Network` defines a physical infrastructure connecting two or more nodes.

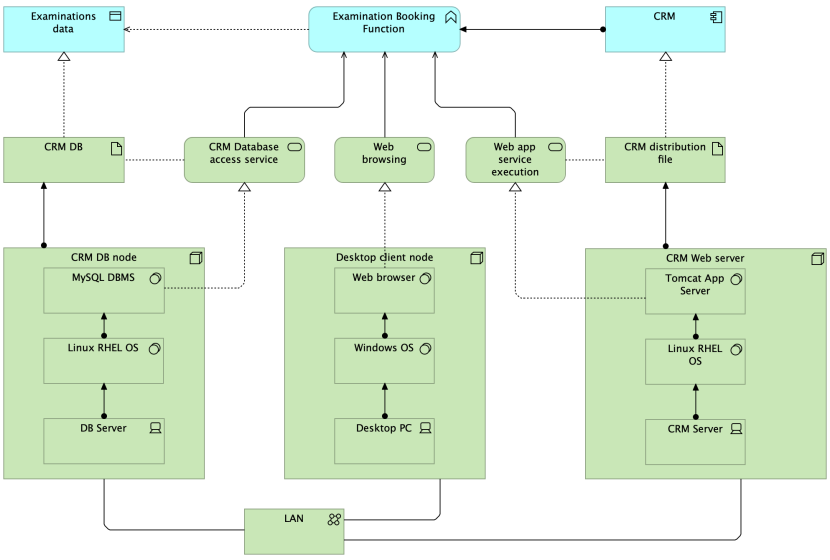

Figure 5.6: Diagram of the Example 5.4

Example 5.5 - Application integration

Considering the Example 4.4, the ERP of ACME Hospital that is used by the CRM to access to the *Doctors*, is based on a three-tiered configuration and the DB is running on a dedicated node.

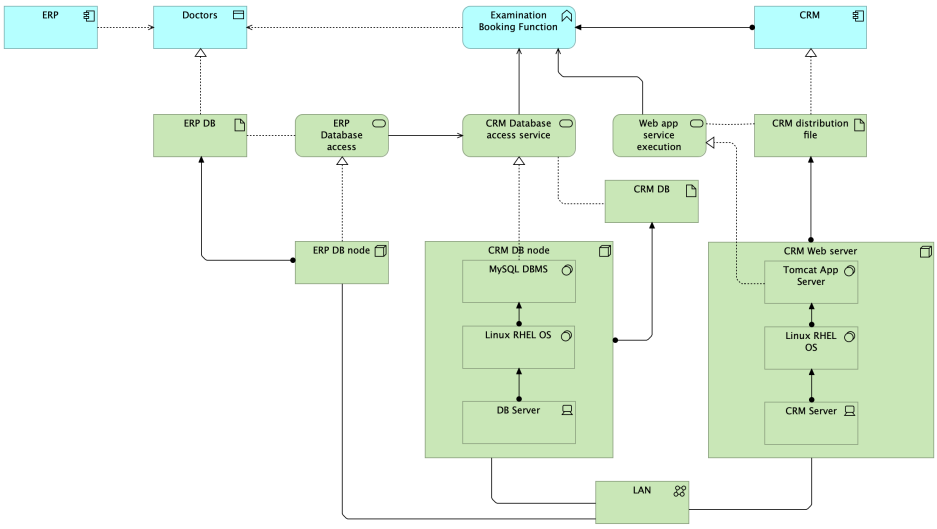

Figure 5.7: Diagram of the Example 5.5

Without specifying the details concerning the *ERP DB node* (no indication in this regard is given by the text), it is important to notice that the integration is possible at database level, as the *CRM Database access service* is served by the *ERP database access*. This indicates that when the data about the *Doctors* are required by the CRM, they are obtained by this connection.

The alternative to loosely integration based on data sharing, as discussed in the Example 4.4, involves the presence of direct communication between ERP and CRM. Specifically, the ERP offers a service called by the CRM and the information flow goes from the former to the latter. A plausible representation of what is happening at the technological level is shown in Figure 5.8. Here it is assumed that the *Node* on which the ERP is installed is a Web server (no detail in this case is provided with respect to what is present in the node) thanks to which it is possible for the ERP to offer a REST type service.

Considering the additional application integration modes seen in Example 4.4, Figure 5.8 shows the counterparty at the technological level. Specifically, the *Communication bus* (e.g., an Enterprise Service Bus) represents the `System Software` that is responsible for

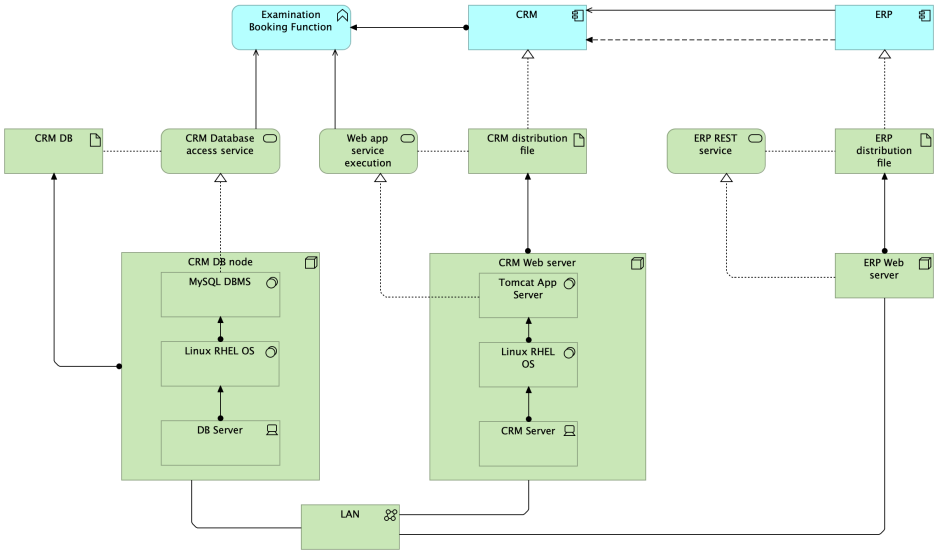

Figure 5.8: Diagram of the Example 5.5 considering a service-based integration

supporting collaboration between the parties. Regarding the communication between the parties, where the direction of the flow is clear but not the mechanism used, the right part of Figure 5.8 shows the use of a `Technology Service` to indicate the presence of a Message Oriented communication service, which is realized through *RabbitMQ*: a particular `System Software` that provides the required middleware features.

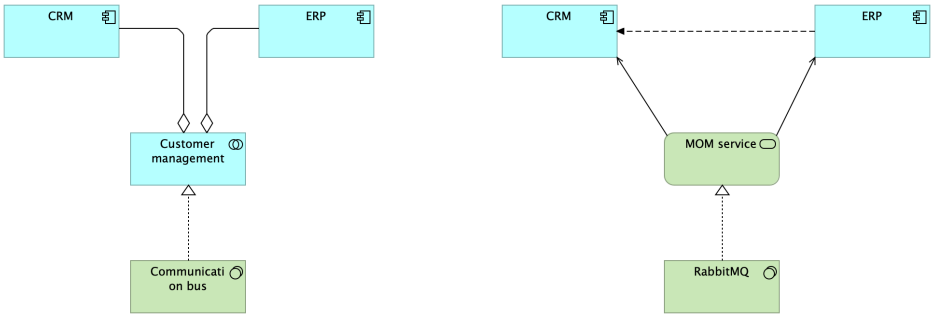

Figure 5.9: Technology view of the alternative integration methods introduced in Figure 4.11

Example 5.6 - Cloud-based solutions

The ACME hospital is investigating the possibility of adopting a cloud-based solution to implement the CRM. For this reason, it needs to model how the IT infrastructure is affected if an IaaS, PaaS, or SaaS solution is adopted.

Figure 5.10 shows the technology layer in case of an IaaS deployment where the ACME Hospital relies on a cloud provider to host the two nodes composing the CRM that are now called *CRM DB Virtual node* and *CRM Web Service Virtual node*. It is worth noticing that the two nodes do not contain the `Device`s as they are under control of the *IaaS provider*, thus they should not be included.

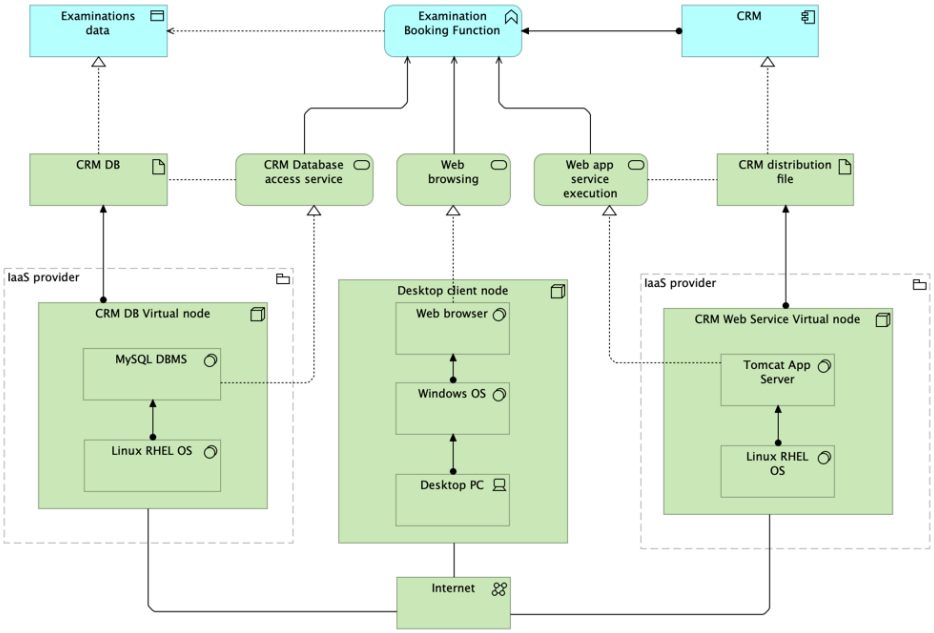

Figure 5.10: Diagram of IaaS deployment

In case the ACME Hospital decides to rely on a PaaS provider, Figure 5.11 shows the corresponding technology architecture. In this case, the *PaaS provider*s are in charge of offering the `Technology Service`s, i.e., *CRM Database access service* and *Web app service execution*. All the details concerning how those services are offered are hidden as they are under the responsibilities of the *PaaS provider*.

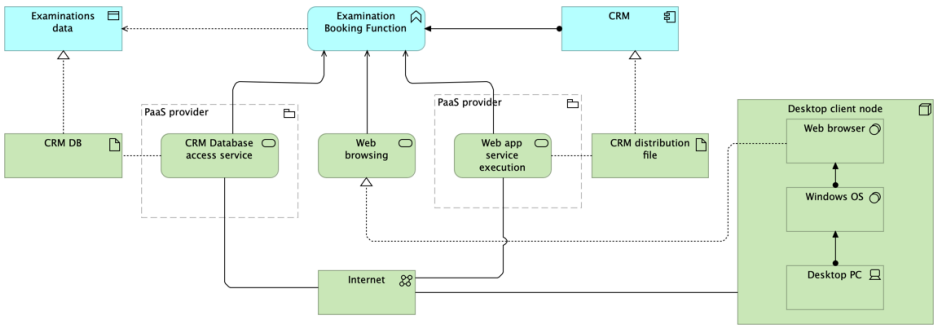

Figure 5.11: Diagram of PaaS deployment

Finally, in case of adoption of the SaaS provisioning model, Figure 5.12 shows how the technology layer is reduced to the *Desktop client node* as the technology details concerning the *Examination Booking Function* are under the realm of the *SaaS provider*.

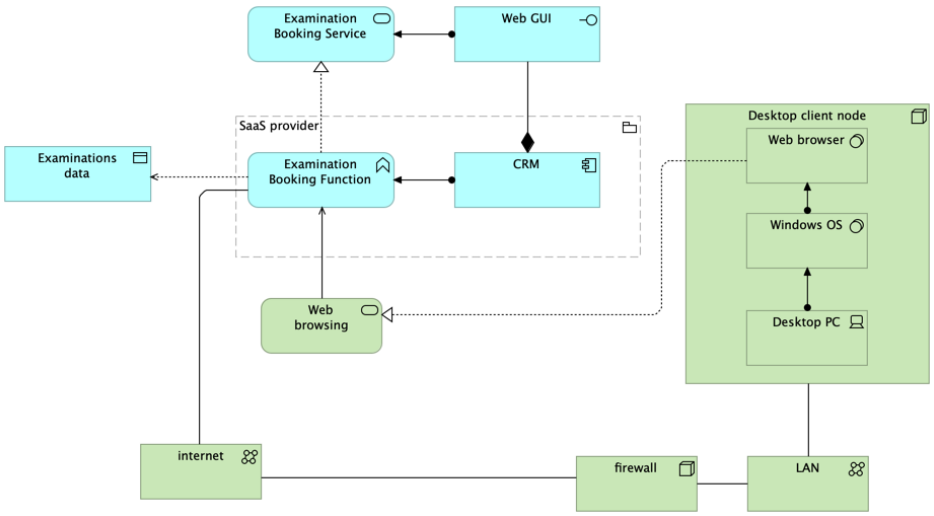

Figure 5.12: Diagram of SaaS deployment

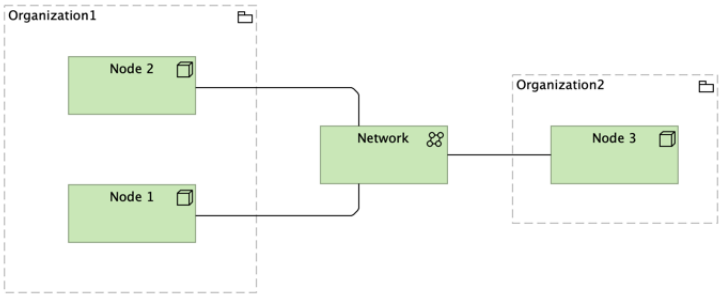

Figure 5.13: Examples of groups

The previous diagrams have introduced the `grouping` element, which is used to aggregate or compose one or more elements as shown in Figure 5.13. The use of grouping at the infrastructure level mainly serves to indicate who manages the various infrastructural elements that are being represented. For example, in Figure 5.13, it is highlighted that the infrastructure is provided by two different organizations, *Organization1* and *Organization2*, and that the elements are connected through a network.

Example 5.7 - Cloud-based provisioning

Keep considering the cloud technology, consider now the possibility for ACME Hospital to provide, thanks to its own private cloud, the *CRM DB Virtual Node* and the *CRM Web Service Virtual Node* provided in the configuration of Figure 5.10.

Compared to what was seen before, the model must represent not so much the cloud as seen by the user of its services, but the cloud as seen by the service provider and, in particular, the IaaS service.

As shown in Figure 5.14, the presence of a *Hypervisor* is assumed, whose task is to control the computational resources offered, in this case, by a *Blade System*. In this case, they also wanted to include in the diagram the `Technology Interface` offered by the hypervisor, called *Management Interface*, which can be used to manage the life cycle of the virtual machines for which it is responsible. The diagram also illustrates the presence of the two virtual machines *CRM DB Virtual Node* and *CRM Web Service Virtual Node*.

This example is inspired by the example discussed in the ArchiMate Specification at https://pubs.opengroup.org/architecture/archimate3-doc/ch-Technology-Layer.html (example 30).

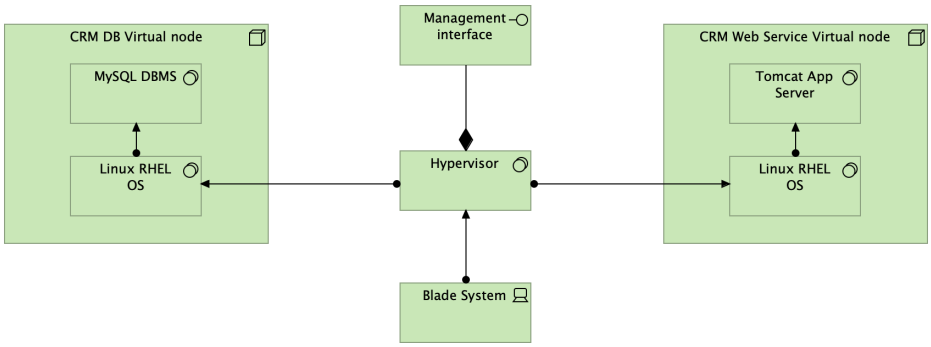

Figure 5.14: Diagram of Cloud provisioning

Exercises

Exercise 5.1. *To support a self-serve booking process supported by the automated business service described in Example 4.5, it is required to revise the diagram in Figure 5.6. In particular, it is needed to revise how the network is organized to allow external users to access the CRM functions.*

Exercise 5.2. *Referring to the Example 4.6 which implies the support of a multiple interface, revise the diagram reported in Figure 5.6 to map such a configuration. Consider that the solution is based on MySQL server also in case of the Desktop GUI-based alternative.*

Solutions

Solution 5.1. *To allow external users to access the functions offered by the CRM, it is needed to connect the LAN to the Internet. For security reasons, it is reasonable to include also a Firewall as an element in between the outer and inner networks.*

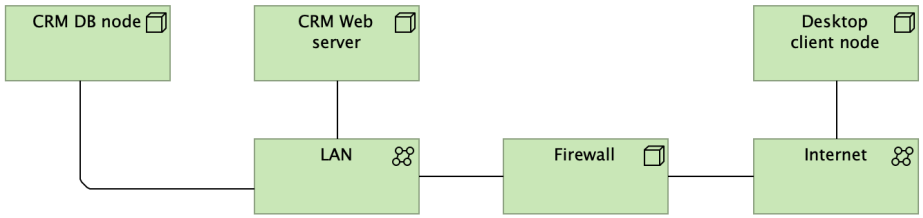

Figure 5.15: Variant of diagram 5.6 to allow a self-serve booking.

Solution 5.2. *In this case, the diagram represents a kind of merge of the two-tiered solution reported in Figure 5.5 and the three-tiered in Figure 5.6. The* `OR-junction` *indicates that only one* `Technology Service` *between the Application execution service and the Web browsing can be used at the same time.*

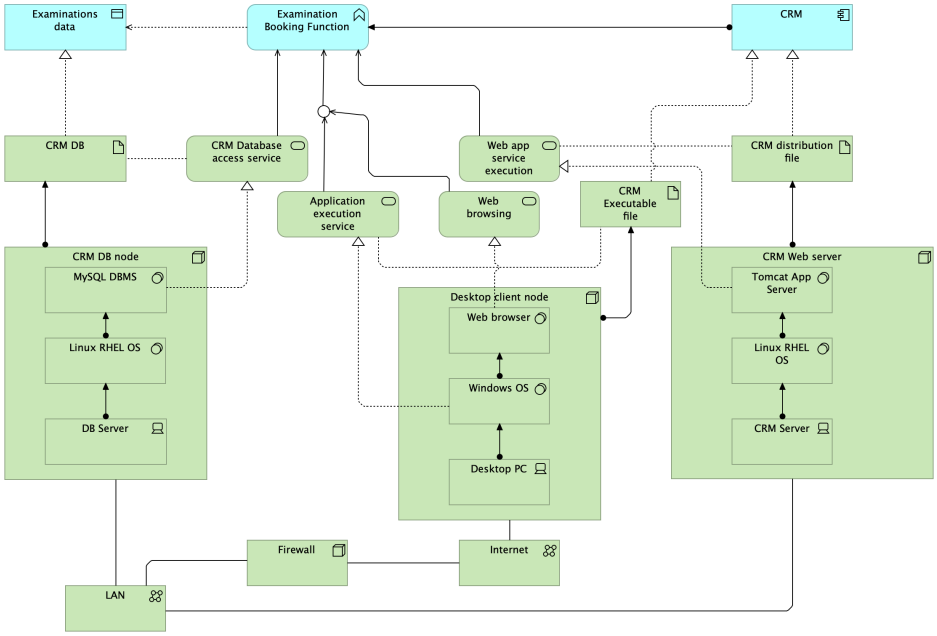

Figure 5.16: Variant of diagram 5.6 to manage double interface.

Solution 5.3. *The integration is here possible because the ERP Web server offers a*

`Technology Service` *corresponding to a REST service that can be called by the* `Technology Service` *offered by the CRM Web server. This connection corresponds to the* `serve` *relationship occurring at application layer between the ERP and CRM* `Application Module`*s.*

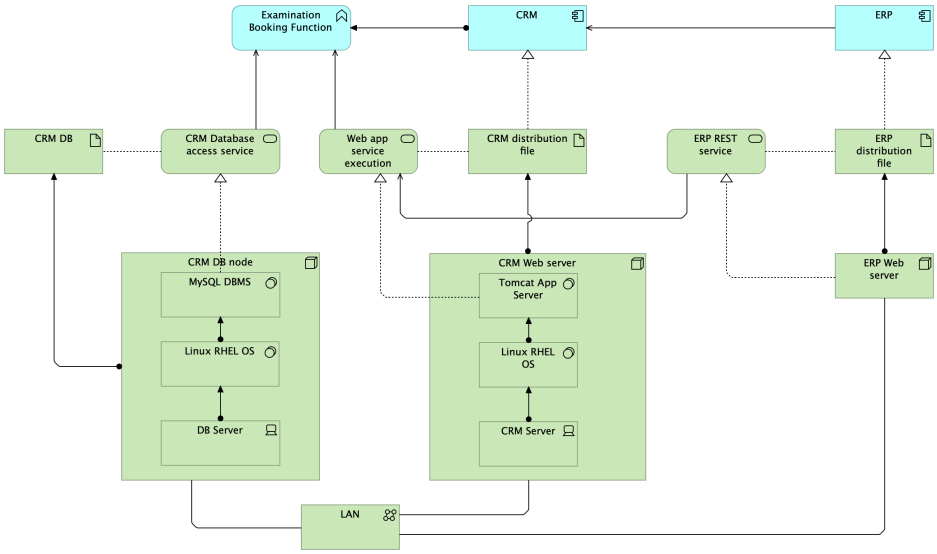

Figure 5.17: Revision of Figure 5.5 to support application integration through services