Chapter 4
Enterprise Architecture Patterns and Principles

André Vasconcelos and Pedro Sousa

Abstract This chapter describes Enterprise Architecture solutions to common problems. Principles are organized by architecture domain and quality attributes addressed. Firstly, sixteen cross-domain principles are described; next, three business layer principles are introduced, followed by five information principles, four application principles, and seven technological principles. Section 4.8 describes IT patterns, including multi-layer IT architectures and IT architectures for high availability. Finally, IT integration patterns are discussed. At the end of the chapter, exercises are proposed.

4.1 Introduction

According to TOGAF [1], principles are general rules and guidelines. In the context of Enterprise Architecture, the definition of principles is expected to support enterprises in fulfilling their mission.

More specifically, Enterprise Architecture principles support organizations in the process of defining an enterprise architecture that fulfils organizational strategic goals, from values through actions and results [2].

Each architectural principle may address one or several EA layers; for instance, an EA principle may be focused on the business layer (e.g., regarding business process or organizational aspects), or a principle might be relevant to different EA layers. In a similar way, a principle may have a direct impact in an architectural or system quality (security) or in several qualities (efficiency, maintainability, and portability).

4.1.1 Principles Description

In order to describe the EA principles, we will address the following topics (based on [1–4]):

P. Sousa, A. Vasconcelos, *Enterprise Architecture and Cartography*,
The Enterprise Engineering Series, https://doi.org/10.1007/978-3-030-96264-7_4

- Name. The name of the principle is expected to be easy to remember and without ambiguities. The name selected should make clear what the principle is.
- Architecture domains. The Enterprise architecture layers where the principle is applicable may include business, information, application, and technology layers.
- Quality attributes. The quality characteristics that the principles address (such as security, performance, or usability) are also an important characteristic of the principle. ISO 9126 [5] and [3] provide 32 quality attributes clustered into six main characteristics—see Fig. 4.1.
- Explanation. For each principle, we provide a short justification on the reasons that support it. The principle explanation ensures that the Enterprise Architect applying it understands its rationale and the principle intentions in order to ensure the principle accurate interpretation.
- Implications. The principle implications, including the derived requirements, are also presented, including its business application or technological impacts.
- Example. For each principle, a brief example of its application, as well as an example that does not comply with the principle, are presented in order to better support its comprehension.

4.1.2 Principles Summary

Tables 4.1 and 4.2 summarize the principles next described, regarding the architecture domains and the quality attributes impacted.

Regarding the architecture domains, most principles are applicable at multiple layers. Maintainability and efficiency qualities are addressed in more than half of the principles described, followed by portability and alignment qualities (addressed in 20% of the principles).

4.2 Cross-Layer Principles

4.2.1 Components Are Centralized

Components are centralized principle is relevant for business, information, application, and technology layers. It is concerned about efficiency and maintainability qualities.

The rationale for the principle is supported in the fact that components in one location are easier to manage (since all efforts are performed in one location). Additionally, consolidation and standardization are easier in central components. Finally, economies of scale are applicable to central components (that are tougher in decentralized environments).

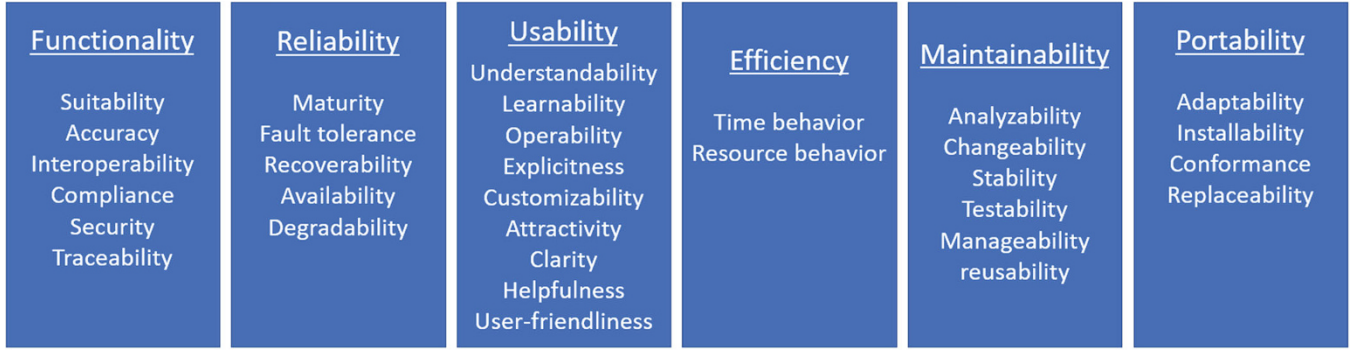


Fig. 4.1 Quality attributes

Table 4.1 Principles summary (part I)

Principle	Architecture domain	Quality attributes
Components are centralized	Business, information, application, technology	Efficiency, maintainability
Front-office processes are separated from back-office processes	Business, information, application	Efficiency, maintainability
Channel-specific is separated from channel-independent	Business, information, application	Reliability, efficiency, maintainability, portability
Data is provided by the source	Information, application	Reliability, efficiency
Data is maintained in the source application	Information, application	Reliability, efficiency, maintainability
Data is captured once	Information, application	Usability, efficiency
IT systems communicate through services	Information, application, technology	Efficiency, maintainability, and portability
Business and information architectures are aligned	Business, information	Efficiency, maintainability, alignment
Business and application architectures are aligned	Business, application	Efficiency, maintainability, alignment
Information and application architectures are aligned	Information, application	Efficiency, maintainability, alignment
Required application services are available	Business, application	Functionality, suitability, alignment
Services have different interfaces	Business, application, technology	Interoperability, maintainability
Applications manage information with the same security level	Business, information, and application	Security, reliability
Critical processes are executed in specific systems	Business, application	Security, alignment
Each information entity is managed by a single application	Information, application	Alignment
Primitive and derived data are managed by different IT components	Information, technology	Alignment
Business units are autonomous	Business	Maintainability, portability
Customers have a single point of contact	Business	Usability and efficiency
Management layers are minimized	Business	Reliability, usability, efficiency, maintainability

Table 4.2 Principles summary (part II)

Principle	Architecture domain	Quality attributes
Information management is everybody's business	Information	Efficiency, maintainability
Common vocabulary and data definitions	Information	Efficiency, maintainability
Content and presentation are separated	Information	Usability, maintainability
Data that is exchanged adhere to a canonical data model	Information	Reliability, maintainability
The number of implementations of the same information entity is minimized	Information	Interoperability, maintainability
Common use applications	Application	Efficiency, maintainability
Presentation logic, process logic, and business logic are separated	Application	Maintainability
Business logic and presentation components do not keep the state	Application	Efficiency
Minimize the number of dependencies and applications per service	Application	Maintainability
Technology independence	Technology	Portability, maintainability
Interoperability	Business, information, and application	Portability, efficiency, maintainability
IT systems are scalable	Application, technology	Efficiency
IT systems adhere to open standards	Information, application, technology	Maintainability, portability
IT systems are preferably open source	Application, technology	Efficiency, maintainability
All messages are exchanged through the enterprise service bus	Information, application, and technology	Maintainability, portability

The major implication that this principle brings is that components should be centralized, unless business, application, or technological requirements require a decentralized approach [2].

Figure 4.2 presents an application architecture that supports the principle components are centralized.

In this example, the company applications (email, finance, human resources, and intranet) are centralized in the headquarters.

On the other hand, Fig. 4.3 presents another application architecture that has several applications replicated through the company offices (e.g., email, finance, human resource applications)—not supporting this principle.

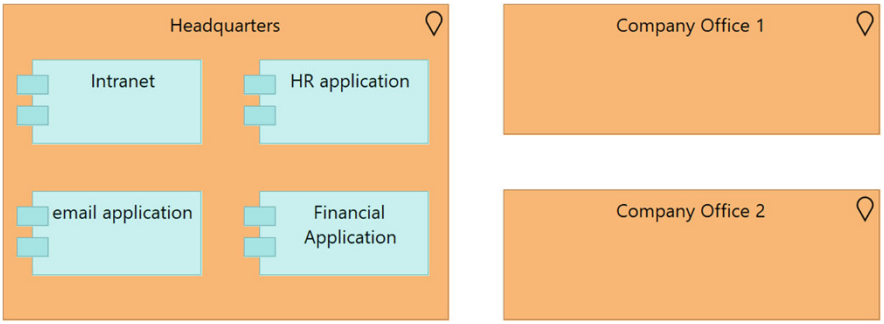


Fig. 4.2 Example of architecture applying the principle components are centralized

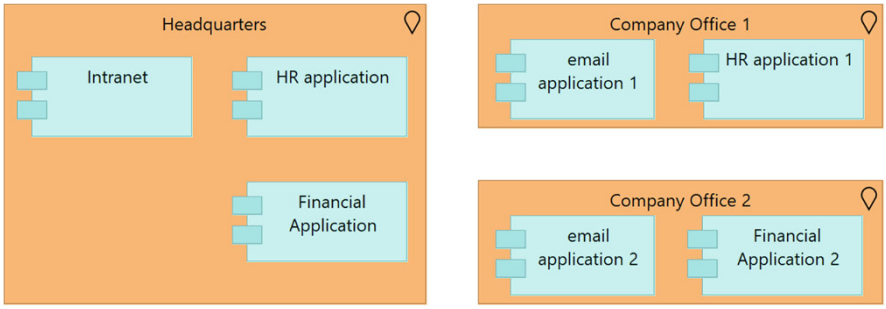


Fig. 4.3 Example of architecture that does not apply the principle components are centralized

4.2.2 Front-Office Processes Are Separated from Back-Office Processes

Front-office processes are separated from back-office processes principle is relevant for business, information, and application layers. This principle addresses the architecture maintainability.

This principle is supported in the fact that the focus of front-office and back-office processes is different. Usually, front-office processes are focused on customer intimacy and back-office processes on operational excellence.

Additionally, the knowledge and skills required for front-office processes (as persuasive speaking skills, empathy, communication, and patience, among others) are different skills and knowledge than back-office processes.

Finally, from an efficiency perspective, separating back-office processes from front-office processes makes easier to reuse back-office processes.

The most significant implications of this principle are at the business architecture, since it is recommended to have a disengagement between front-office and back-office processes, having dedicated processes to the front office and back-office.

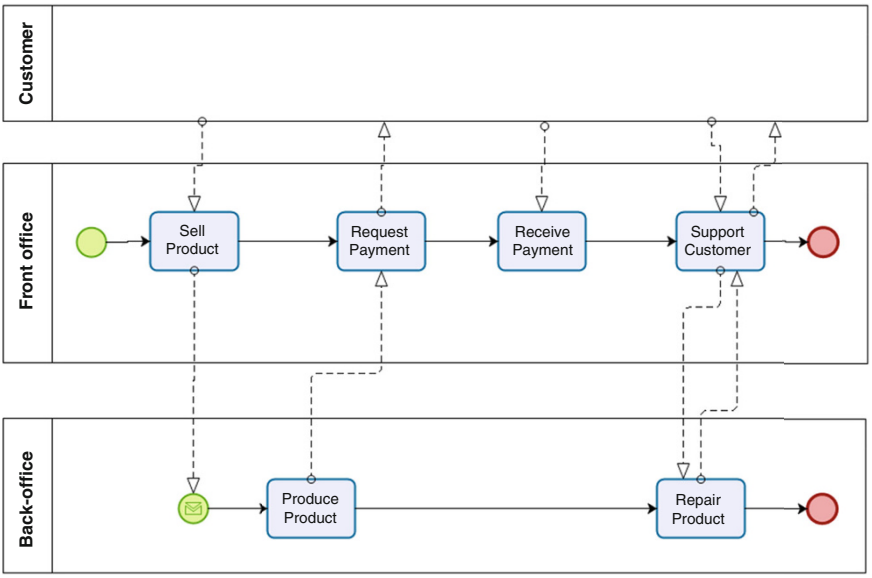


Fig. 4.4 Example of architecture applying the principle front-office processes are separated from back-office processes

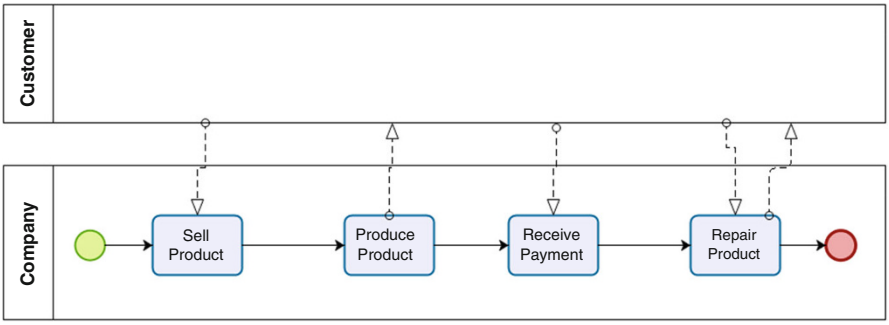


Fig. 4.5 Example of architecture that does not use the principle front-office processes are separated from back-office processes

Consequently, in the application and information architecture, front-office applications shouldn't contain back-office logic or data [2].

Figure 4.4 presents a BPMN diagram of the processes that support the selling, producing, and supporting activities of a company. In this example, there is a full separation of the front-office and the back-office processes.

On the other hand, in Fig. 4.5, back-office and front-office activities are in the same business process (making it difficult to reuse back-office activities or ensuring the right skills to deal with the costumer).

4.2.3 *Channel-Specific Is Separated from Channel-Independent*

Channel-specific is separated from channel-independent principle is relevant for business, information, and application layers. This principle addresses the reliability, efficiency, maintainability, and portability qualities.

This principle is built on top of the assumption that an important part of the business tasks do not depend on the channel used to interact with the customer (telephone, mail, Internet, office). Thus, in order to allow the business to be developed through multiple channels, the data must be managed in channel-independent processes.

According to [2], the implementation of this principle may be achieved by implementing channel-specific activities at the borders of an end-to-end business process and communicating with the other activities in a channel-independent format.

At the application architecture, it is recommended to have dedicated components for channel-specific processing and others that are channel-independent, where the business logic and the data are managed. An interface among channel-specific and channel-independent components must be implemented.

Figure 4.6 presents a view of an online and a face to face selling processes, where the activities that are specific of the channel (Internet or face to face) communicate with channel-independent activities (issue invoice and produce product).

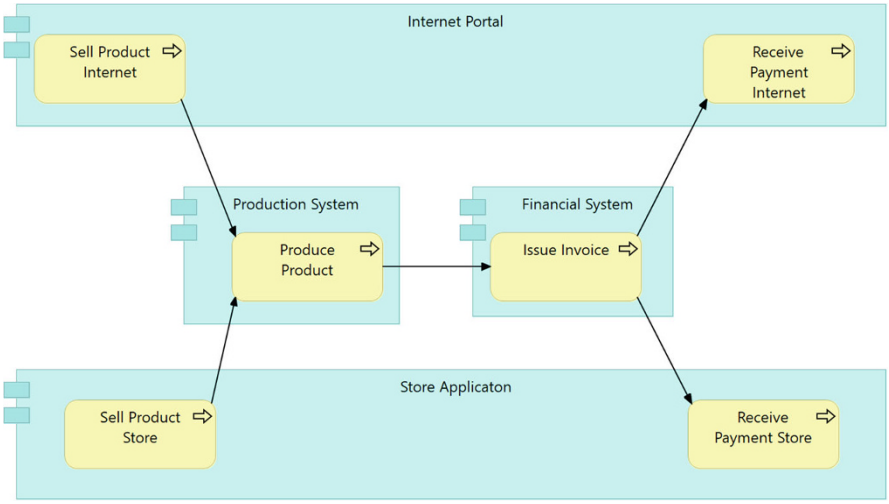


Fig. 4.6 Example of architecture applying the principle channel-specific is separated from channel-independent

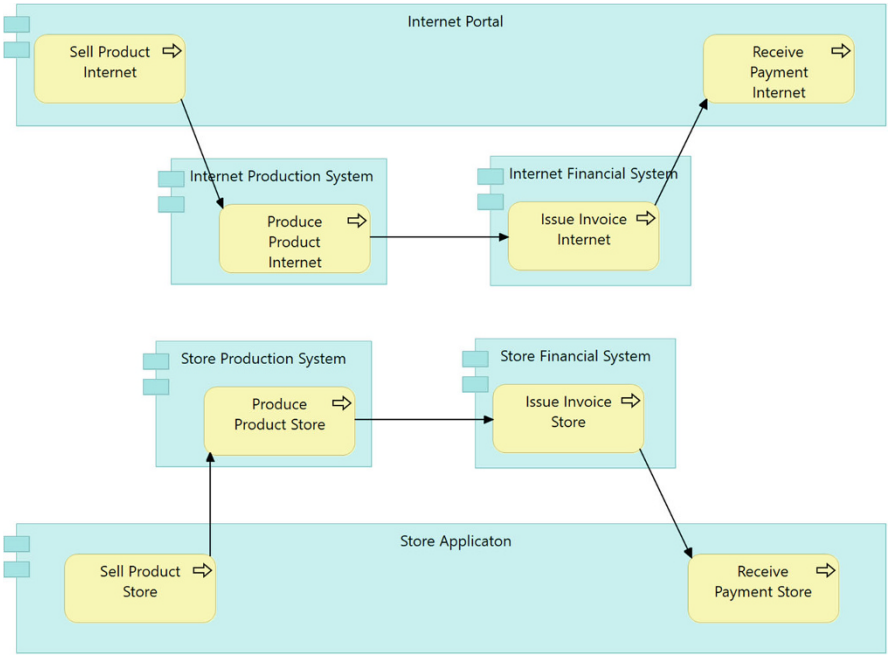


Fig. 4.7 Example of architecture that does not use the principle channel-specific is separated from channel-independent

Figure 4.7 presents an architecture where the channel independent activities and applications are replicated in both channels.

4.2.4 Data Is Provided by the Source

The principle data is provided by the source has impact in the information and application architectures, addressing reliability, performance, and efficiency qualities.

This principle increases efficiency and reliability by removing unnecessary intermediate redirection components, ensuring that the application responsible for managing the data is the one that will also provide it.

Additionally, since the data is provided directly by the source application, without having overhead processing costs or other errors (from other components), the performance and reliability are also expected to increase.

In order to implement this principle, organizations should request customers to insert the data in online forms (avoiding potential intermediary errors), and applications should get the data from the source application [2].

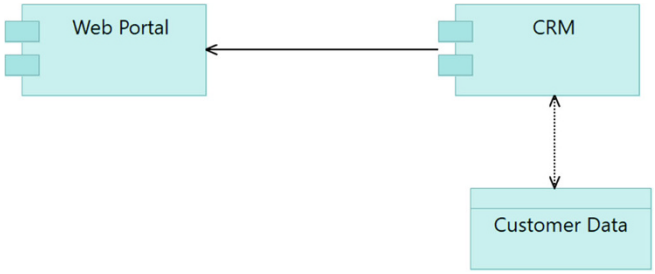


Fig. 4.8 Example of architecture applying the principle data is provided by the source

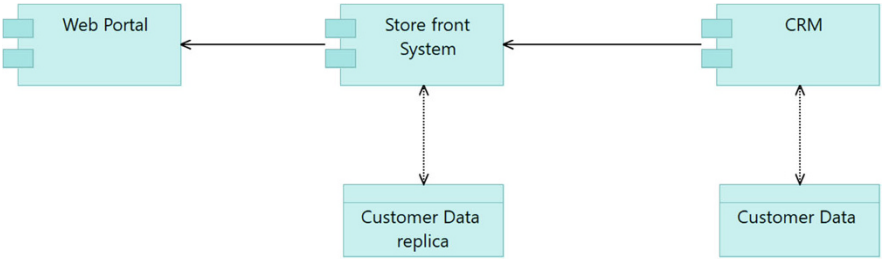


Fig. 4.9 Example of architecture that does not use the principle data is provided by the source

Figure 4.8 presents an application architecture where the customer data is obtained (by the website) directly from the source application (the CRM application).

On the other hand, Fig. 4.9 presents a similar architecture where the data is obtained in an intermediary application (a store front application) that keeps a replica of the customer data.

4.2.5 Data Is Maintained in the Source Application

The data is maintained in the source application principle has several similarities to the data is provided by the source principle. It has impact in the information and application architectures, addressing reliability, efficiency, and maintainability qualities.

Since managing (creating, updating, or deleting) similar data in multiple places introduces inconsistencies and is inefficient, in order to comply with this principle, organizations are expected to [2]:

- Have a clear identification of the source application responsible for managing (create, update, and delete) each data type.
- The data is always obtained from the source application (not from replicas).

- Replicas of data shouldn't be updated (unless synchronization mechanisms are available).
- The data shouldn't be copied before it is finalized.

Figures 4.8 and 4.9 present two application architectures where this principle is followed and not followed, respectively.

4.2.6 Data Is Captured Once

The principle data is captured once has impact in the information and application architectures, addressing usability and efficiency qualities.

This principle is supported by the fact that requesting similar data more than once is inefficient.

In order to comply with this principle, applications should first verify if the data is already available (through services exposed to access it). When the data is already available, it should be used in pre-filling forms [2].

Figure 4.10 presents an application architecture where the CRM application exposes services to access customer data. These services are used by the webPortal and the store front applications (not requesting the user to enter information already available).

On the contrary, Fig. 4.11 presents an application architecture that does request similar data, from the user, depending on the system used.

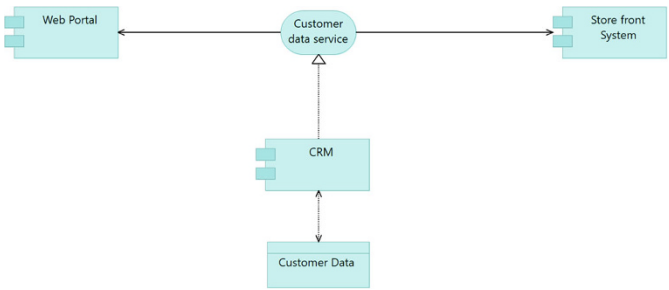


Fig. 4.10 Example of architecture applying the principle data is captured once

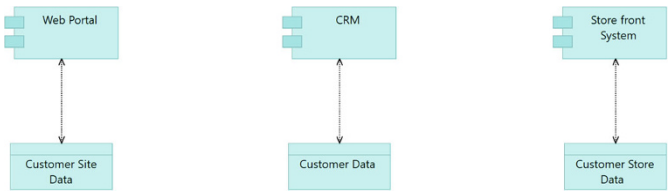


Fig. 4.11 Example of architecture that does not use the principle data is captured once

4.2.7 *Systems Communicate Through Services*

This principle is expected to have an impact in the information, application, and technology layers. Ensuring that systems communicate through services addresses efficiency, maintainability, and portability qualities.

By reusing services, less (new) services are needed, improving efficiency and maintainability. Additionally, reusing services contributes to faster implementation of new applications (that may reuse existing functionalities), reducing new applications implementation effort and duration [2].

In order to implement this principle, organizations must have extra caution in the service definition, in order to ensure services are reusable, hiding implementation details and adopting open standards in its interfaces. It is also recommended to publish services in a service directory [2].

Figure 4.12 presents a CRM application that provides a service that is reusable by three other applications.

In Fig. 4.13, a similar CRM application is integrated with three similar applications, but the services are specific (not reusable).

4.2.8 *Business and Information Architectures Are Aligned*

The business and information architectures are aligned principle, as expected, addresses the business and the application architectures. This principle is concerned about efficiency, maintainability, and alignment.

The focus is on structuring the information necessary to conduct business, both in operations and in management of information. If information and business

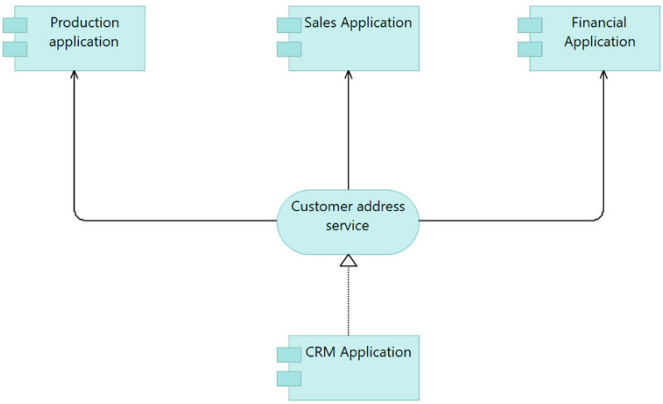


Fig. 4.12 Example of architecture applying the principle systems communicate through services

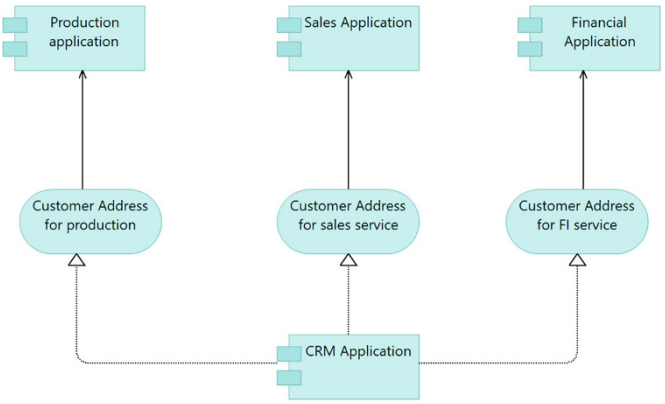


Fig. 4.13 Example of architecture that does not use the principle systems communicate through services

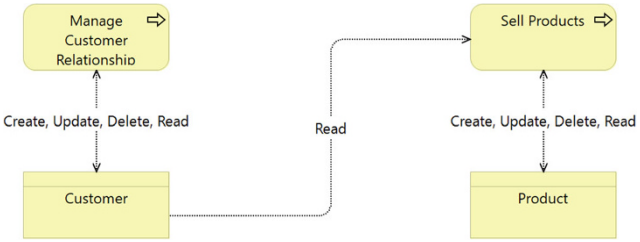


Fig. 4.14 Example of architecture applying the principle business and information architectures are aligned

architectures are aligned, business will have the expected information when needed without wasting unnecessary resources.

In order to implement this principle, organizations should ensure that in its enterprise architecture [4]:

- Information entities contain all information necessary for the activities of processes (automatic or manual).
- All processes that share information entities agree with the concepts behind it.
- The processes that create information entities manage the entire life cycle of those entities.
- All processes create or update at least one information entity.
- Each information entity is read by at least one process.

In Fig. 4.14, each process is responsible for managing (create, update, and delete) its information entity (customer and product). Notice that the sell products process only reads customer information entity (but the create, update, and delete actions are only performed by the manage customer relationship process).

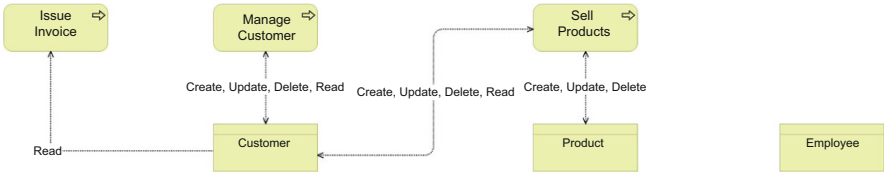


Fig. 4.15 Example of architecture that does not use the principle business and information architectures are aligned

On the other hand, in Fig. 4.15, the customer information entity is managed by two business processes (sell products and manage customer relationship); additionally, the issue invoice process does not create or update any information entity, and the employee information entity is not read by any process.

4.2.9 Business and Application Architectures Are Aligned

This principle addresses the business and application architectures, contributing for efficiency, maintainability, and alignment qualities.

In the alignment between business and applications, the focus is on the automation of the activities of business processes. The larger the alignment, the lower the effort in mechanized operations. It aims to optimize the ratio (operating costs)/(investment) for a given level of service [4].

In order to ensure business and application architectures are aligned [4]:

- All atomic activities of a process are supported by a single system or application.
- The functionalities of the systems are not redundant: support exclusively some activity.
- The characteristics of the activities are in accordance with the features of the systems that support them (e.g., scalability, availability).

In Fig. 4.16, the sell product process is supported by a single application.

In Fig. 4.17, the sell product process is supported by three applications. Assuming that the sell product process is atomic or is performed by the same person, this view presents a misalignment between the business and application architectures.

4.2.10 Information and Application Architectures Are Aligned

This principle is focused on the information and application architectures, addressing efficiency, maintainability, and alignment qualities.

The alignment between information and applications is based on the effectiveness of information systems in business information management. The existence of

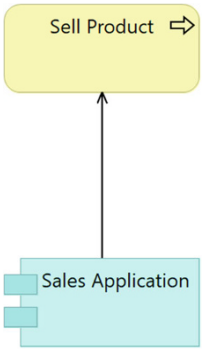


Fig. 4.16 Example of architecture applying the principle business and application architectures are aligned

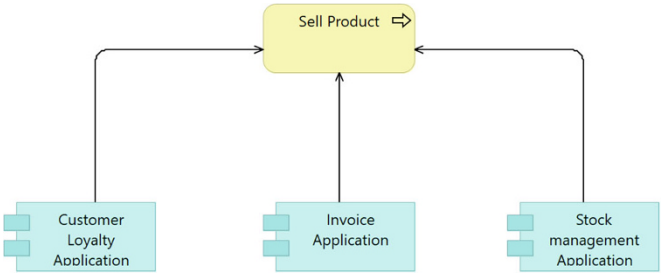


Fig. 4.17 Example of architecture that does not use the principle business and application architectures are aligned

multiple replicas of the same information in different systems is a problem because each replica has structure, syntax, and semantics usually different in different systems, making it difficult to integrate [4].

The following are implications of this principle:

- Each information entity is managed by a single system. Managing means creating and identifying.
- Each attribute of an entity should not be updated by more than one system (different attributes of the same entity may be updated by different systems).
- A system must access information from the system that manages that information, in order to preserve its computational independence.
- Systems must be computationally independent.
- The information characteristics must comply with the characteristics of the system that manages it.
- Distributed transactions should be avoided, and a transaction must involve only one system.

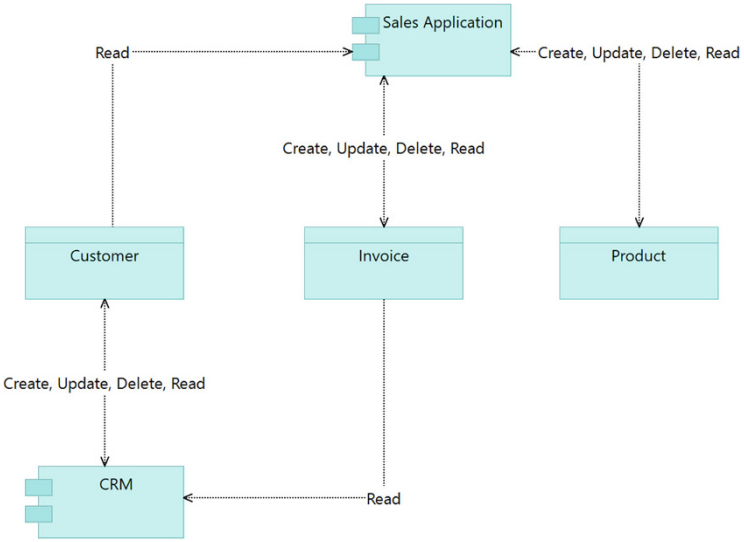


Fig. 4.18 Example of architecture applying the principle information and application architectures are aligned

Figure 4.18 presents three information entities. Each information is managed by one system—product is managed by the Sales application, Invoice is also managed by the Sales application, and Customer is managed by the CRM application.

In Fig. 4.19, the Customer information entity is managed by both applications (Sales and CRM), and the Product information entity is not managed by any application.

4.2.11 Required Application Services Are Available

The required application services are available principle addresses the business and application architectures and the functionality, namely, suitability, and alignment qualities.

In order to ensure that systems functionality is aligned with the business layer, the services required by processes must be supported by application services.

Thus, there should not exist application services required by businesses that are not available in the application architecture [3, 6].

Figure 4.20 presents an online sell process that is supported in two application services (browse products and payment services).

In Fig. 4.21, one application service is missing (Payment service).

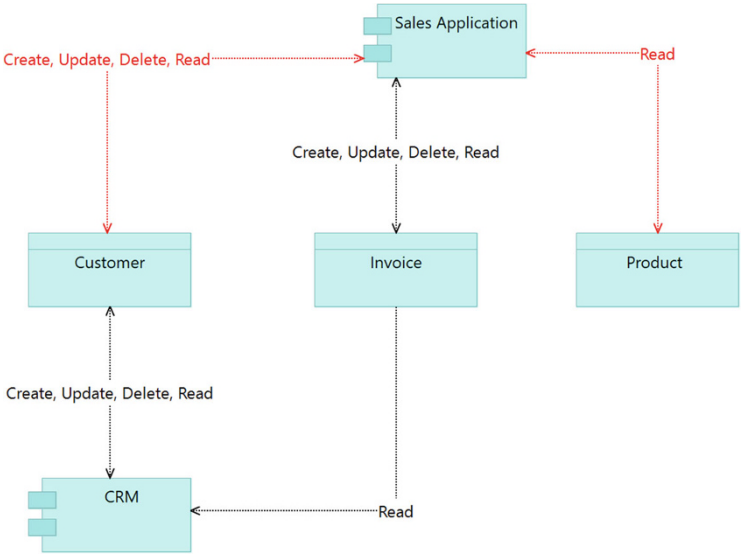
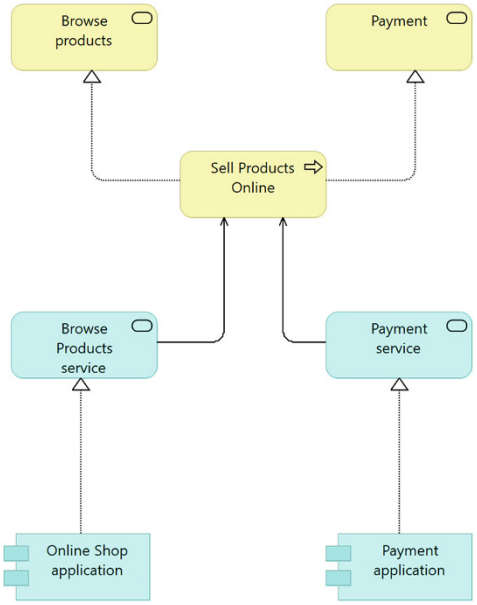


Fig. 4.19 Example of architecture that does not use the principle information and application architectures are aligned

Fig. 4.20 Example of architecture applying the principle required application services are available



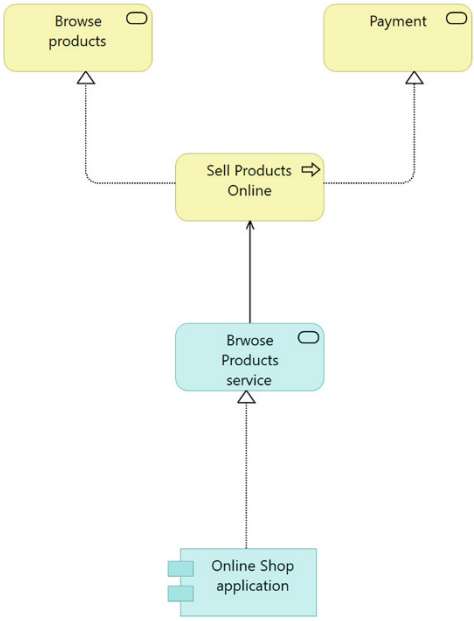


Fig. 4.21 Example of architecture that does not use the principle required application services are available

4.2.12 Services Have Different Interfaces

The services have different interfaces principle should be applied at the business, application, and technology layers of the enterprise architecture. It is concerned about interoperability and maintainability qualities.

According to [7] and [3], the technical interoperability of an architecture increases by providing the same services at different interfaces, including different technologies, and business channels.

In order to support this principle, organizations, whenever possible, should provide services using different interfaces, and no new services should be created because of the need of providing the same service through a new channel or technology [3, 7].

Figure 4.22 presents an invoice application service that is made available in three different interfaces (FTP, webservice, and online form).

In Fig. 4.23, the same service is replicated into three different ones, considering the interface required. This approach increases maintenance and implementation costs and reduces the major benefits of implementing a service-oriented architecture.

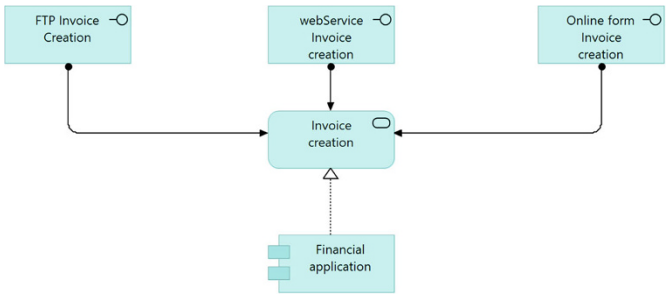


Fig. 4.22 Example of architecture applying the principle services have different interfaces

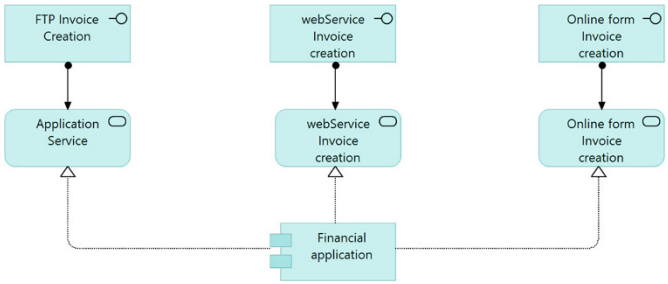


Fig. 4.23 Example of architecture that does not use the principle services have different interfaces

4.2.13 Applications Manage Information with the Same Security Level

The applications manage information with the same security level principle has impact in the information and application architectures. This principle is concerned about security and reliability.

Applications should manage information entities of the same security level, in order not to over- or under-spend resources [3, 4].

In order to implement this principle, information entities with different security requirements should be managed by different systems [3, 4].

In Fig. 4.24, two information entities with different security requirements (customer payment information and a company online catalogue of products) are managed by two different applications (that should be implemented considering the different security requirements of the data managed).

On the other hand, in Fig. 4.25, only one application is managing the two data entities (that have different security requirements)—leading to a mismatch between the application and the information architectures.

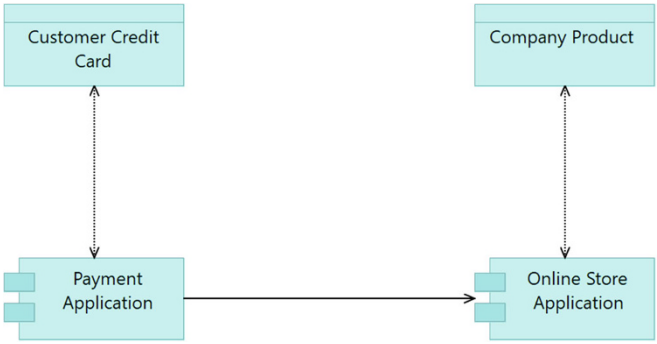


Fig. 4.24 Example of architecture applying the principle applications manage information with the same security level

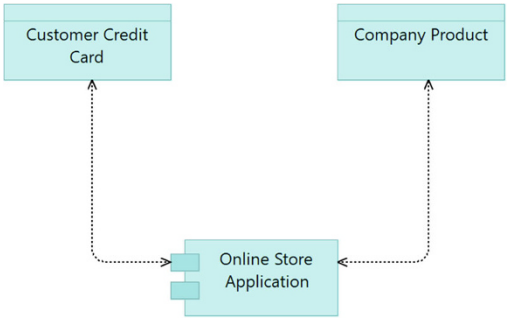


Fig. 4.25 Example of architecture that does not use the principle applications manage information with the same security level

4.2.14 Critical Process Are Executed in Specific Systems

This principle is relevant for the application and business architectures. It is concerned about security and alignment qualities.

As described in [4], the critical business processes should be supported by different applications than non-critical business processes.

Thus, the same application should not manage critical and non-critical business processes [3, 4].

Figure 4.26 presents two business processes. The sales process is considered critical for the company, while the partner management is not (according to the organization business context). Thus, the application architecture has two different applications supporting each process.

In Fig. 4.27, only one application is serving both critical and non-critical business processes.

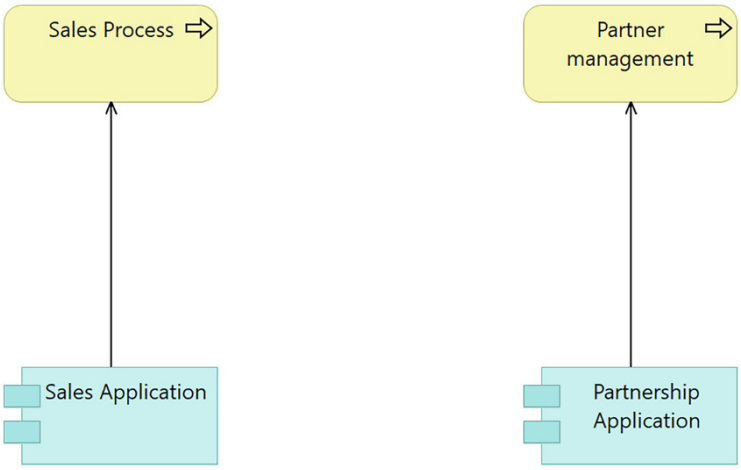


Fig. 4.26 Example of architecture applying the principle critical process are executed in specific systems

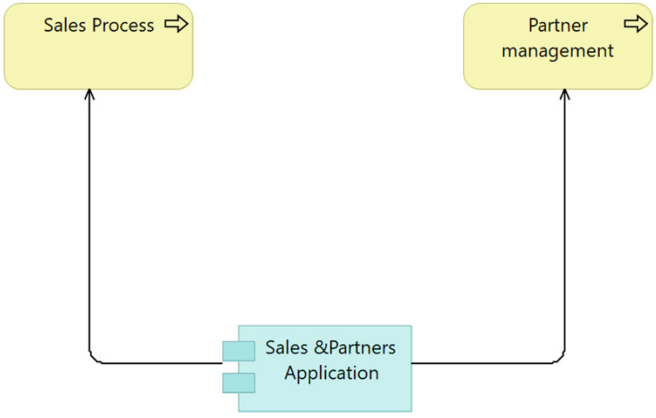


Fig. 4.27 Example of architecture that does not use the principle critical process are executed in specific systems

4.2.15 Each Information Entity Is Managed by a Single Application

This principle has implications for the information and the application architectures and is concerned about alignment quality.

According to [4], each information entity should be managed by a single application. Therefore, the main implications of this principle are [3, 4]:

- Each information entity is created, updated, or deleted by one system.

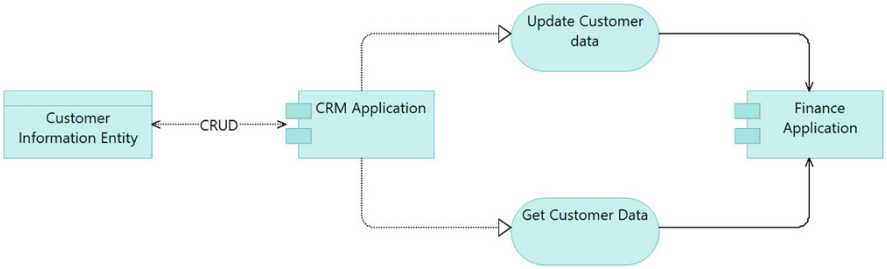
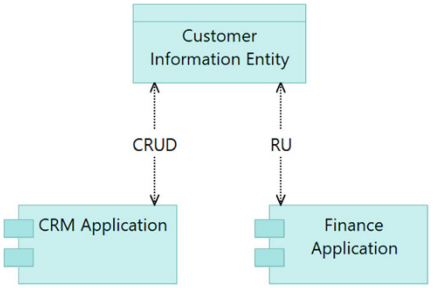


Fig. 4.28 Example of architecture applying the principle each information entity is managed by a single application

Fig. 4.29 Example of architecture that does not use the principle each information entity is managed by a single application



- The application that manages the information entity must provide services on the information entity to other applications.

In Fig. 4.28, the customer information entity is fully managed (created, updated, and deleted) by the CRM application that provides application services to other applications (as the Finance application).

In Fig. 4.29, the Customer information entity is updated by two applications (CRM and finance applications).

4.2.16 Primitive and Derived Data Are Managed by Different IT Components

The primitive and derived data are managed by different IT components is an information-technology principle. This principle is concerned about alignment.

According to Inmon, the primitive and derived data present important differences on performance, accessing patterns, and availability, among other issues [8]. Thus, it is considered a "good architectural practice" to use different technology components to support primitive and derived data.

In order to comply with this principle, organizations should ensure that [3, 4]:

- Derived data is not managed in operational IT components.

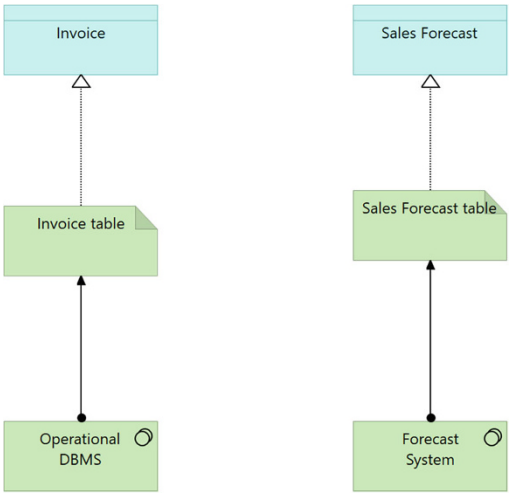


Fig. 4.30 Example of architecture applying the principle primitive and derived data are managed by different IT components

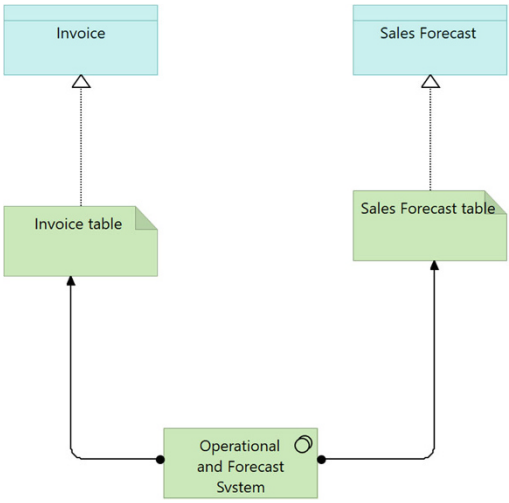


Fig. 4.31 Example of architecture that does not use the principle primitive and derived data are managed by different IT components

- There are separated hardware and software components to manage primitive and derived data.

In Fig. 4.30, the invoice (primitive and operational data) and the sales forecast (derived data) data are managed by different technological components.

In Fig. 4.31, the same system software is used to realize derived and primitive data.

4.3 Business Layer Principles

4.3.1 Business Units Are Autonomous

This principle is concerned about maintainability and portability qualities.

Having autonomous business units ensures a fast adaptation to change, since there are no dependencies to other business units. Additionally, autonomous business units can be easily separated (supporting organizational restructuring) [2].

In order to implement this principle, organizations should ensure that each business unit has a profit and loss center that is used for its evaluation. Additionally, each business unit is responsible for its investments and decisions [2].

4.3.2 Customers Have a Single Point of Contact

Customers have a single point of contact principle addresses usability and efficiency qualities.

According to this principle, it is better for customers to have a single point of contact, instead of contacting several company employees. It is expected that a single point of contact ensures consistence of the information provided. Additionally, from an operational point of view, having dedicated employees to manage customers is expected to increase efficiency and reduce operational activity interruptions [2].

In order to implement this principle, organizations should provide a single access point to customers (e.g., a contact-center, dedicated person, etc.) that detaches the customer from the internal organization. This access point must have enough information to handle customer requests. Only in exceptional situations, the customer interacts directly with the internal organization [2].

In Fig. 4.32, all the interaction between the customer and the company is managed by the Customer Relationship Department.

On the other hand, in Fig. 4.33, the customer interacts directly with three different organization departments.

4.3.3 Management Layers Are Minimized

The management layers are minimized principle is expected to improve business reliability, usability, efficiency, and maintainability qualities.

This principle is supported in the fact that the nonproductive costs are reduced by minimizing management layers. Another important side effect of minimizing management layers is that operational employees tend to take more responsibility for their work [2].

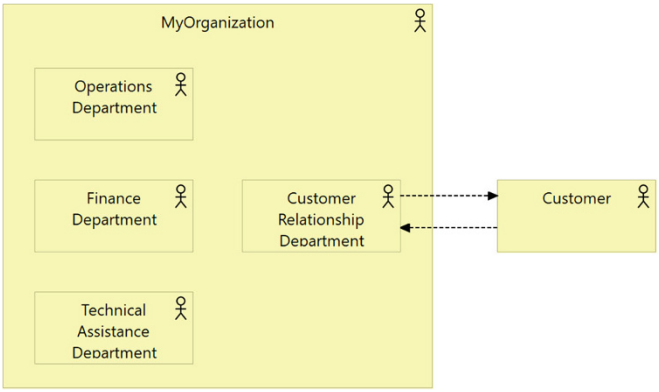


Fig. 4.32 Example of architecture applying the principle customers have a single point of contact

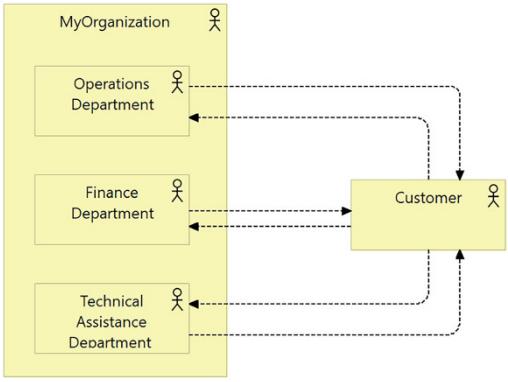


Fig. 4.33 Example of architecture that does not use the principle customers have a single point of contact

From an organizational point of view, this principle imposes a reduction to the minimum of the management layers. In an ideal world, according to this principle, people responsible to perform the actual work would be self-managed in a management layer free organization [2].

Figure 4.34 presents an organization where the management layers are minimized. The production and sales employees are empowered and report directly to the Director.

Figure 4.35 presents an organization with several management layers, where the lower layers report to one or more managers, who report to one or more management layers.

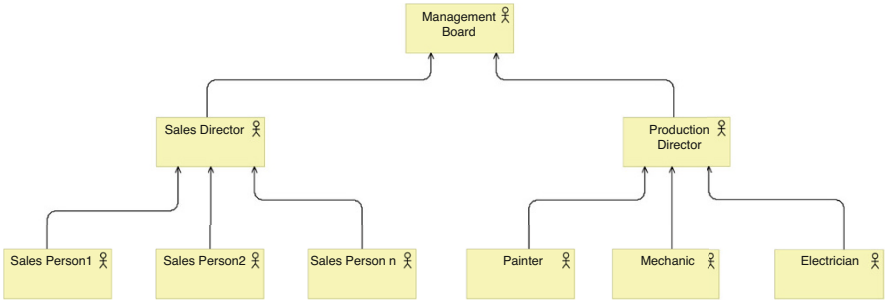


Fig. 4.34 Example of architecture applying the principle management layers are minimized

4.4 Information Layer Principles

4.4.1 Information Management Is Everybody's Business

This principle addresses efficiency and maintainability qualities.

According to the information management is everybody's business principle, information management decision-making is well defined and supports business goals. All organization business units must be involved in information management, working as a team [1].

In order to implement this principle, the internal and external stakeholders of the organization must accept the responsibility of managing the information. The organization must ensure the proper resources to information management [1].

4.4.2 Common Vocabulary and Data Definitions

The common vocabulary and data definitions principle addresses efficiency and maintainability qualities. According to this principle, the enterprise data must be defined and available to everybody in the organization. This data architecture must be used when developing applications [1]. Applications should provide services to exchange data, according to the enterprise common data definitions. From an organizational point of view, data administration responsibilities must be assigned [1].

Figure 4.36 presents an enterprise data architecture, where the data is commonly defined, data management responsibilities are well defined, and applications provide services to access data.

On the other hand, Fig. 4.37 presents an information architecture with data entities dependent on the applications, leading to replicated data and data management conflicts among applications.

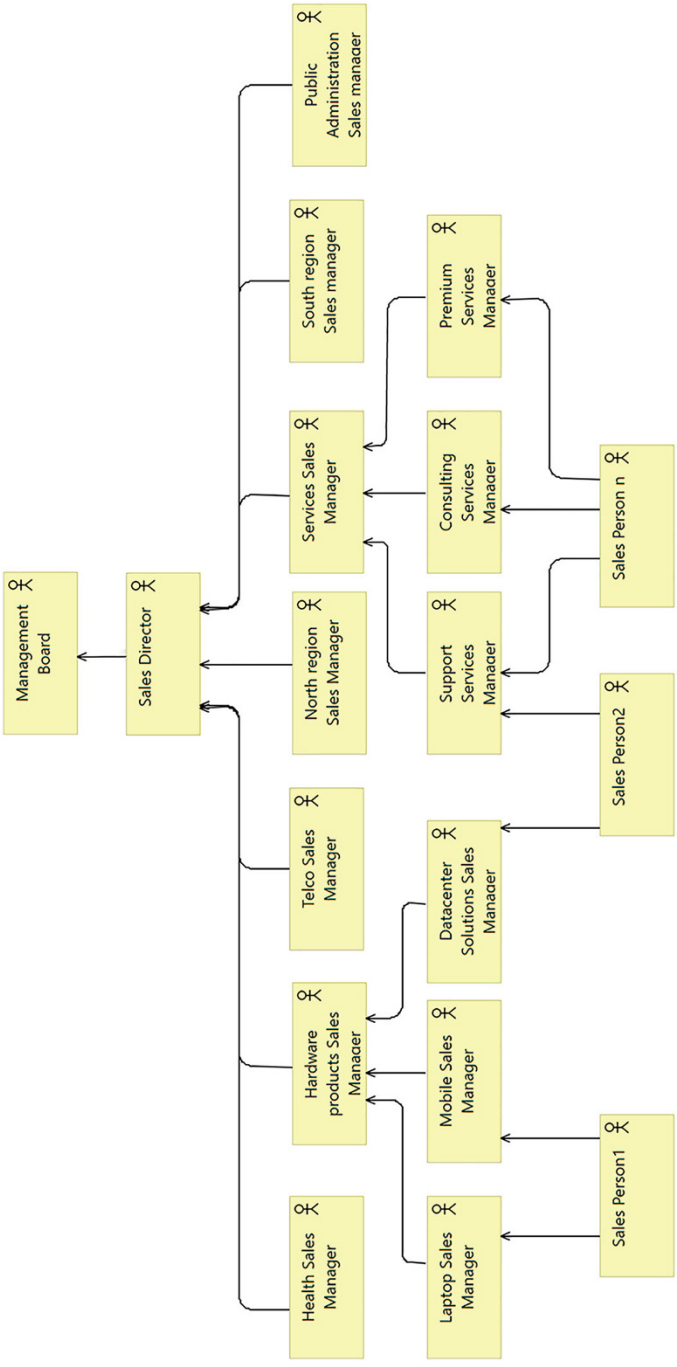


Fig. 4.35 Continued

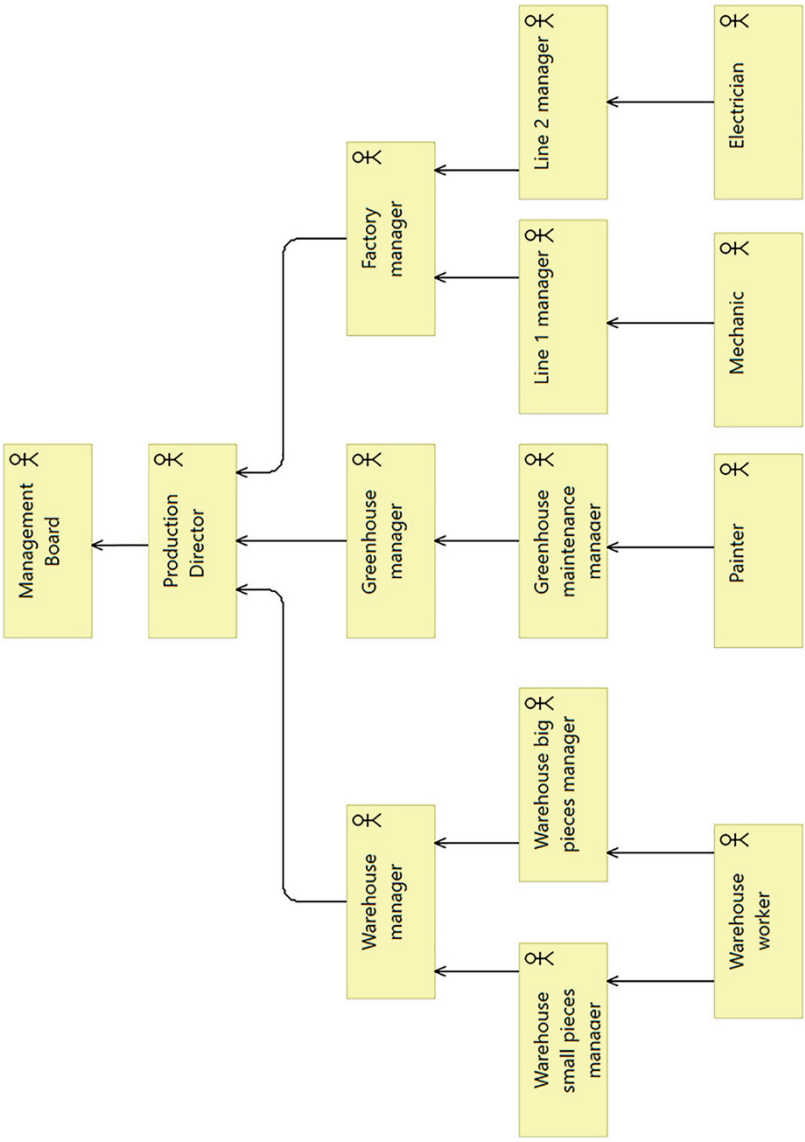


Fig. 4.35 Example of architecture that does not use the principle management layers are minimized

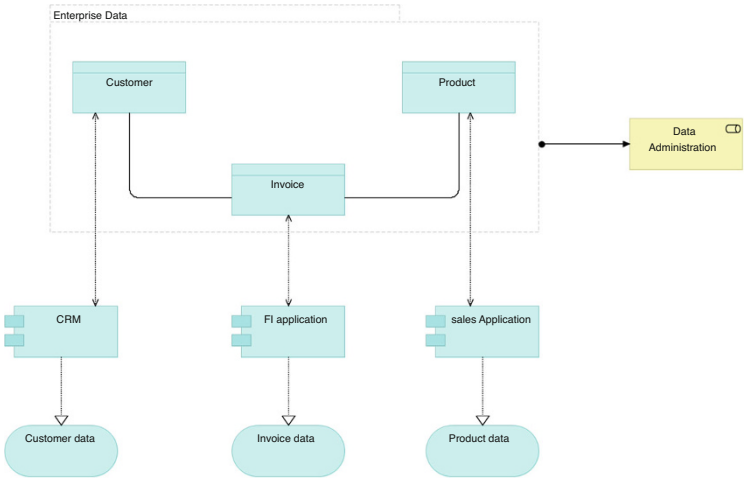


Fig. 4.36 Example of architecture applying the principle common vocabulary and data definitions

4.4.3 Content and Presentation Are Separated

The content and presentation are separated principle main impacts are at the information architecture, ensuring that the content may be reused in multiple channels. By separating content and presentation, each can be managed independently [2].

In order to comply with this principle, organizations should translate data acquired to a format that is independent of the presentation channel. Additionally, there should be specific software components that add to the content the presentation-specific data [2].

Figure 4.38 presents an architecture, where the data of the product information entity is separated between data independent and dependent of the channel (online and physical).

In Fig. 4.39, there is a replication of the product data that is dependent of the channel and the content that is not dependent of the channel.

4.4.4 Data That Is Exchanged Adhere to a Canonical Data Model

According to [2], the adoption of a common definition to the data minimizes the need for translations when applications exchange data, increasing the reliability and the maintainability qualities.

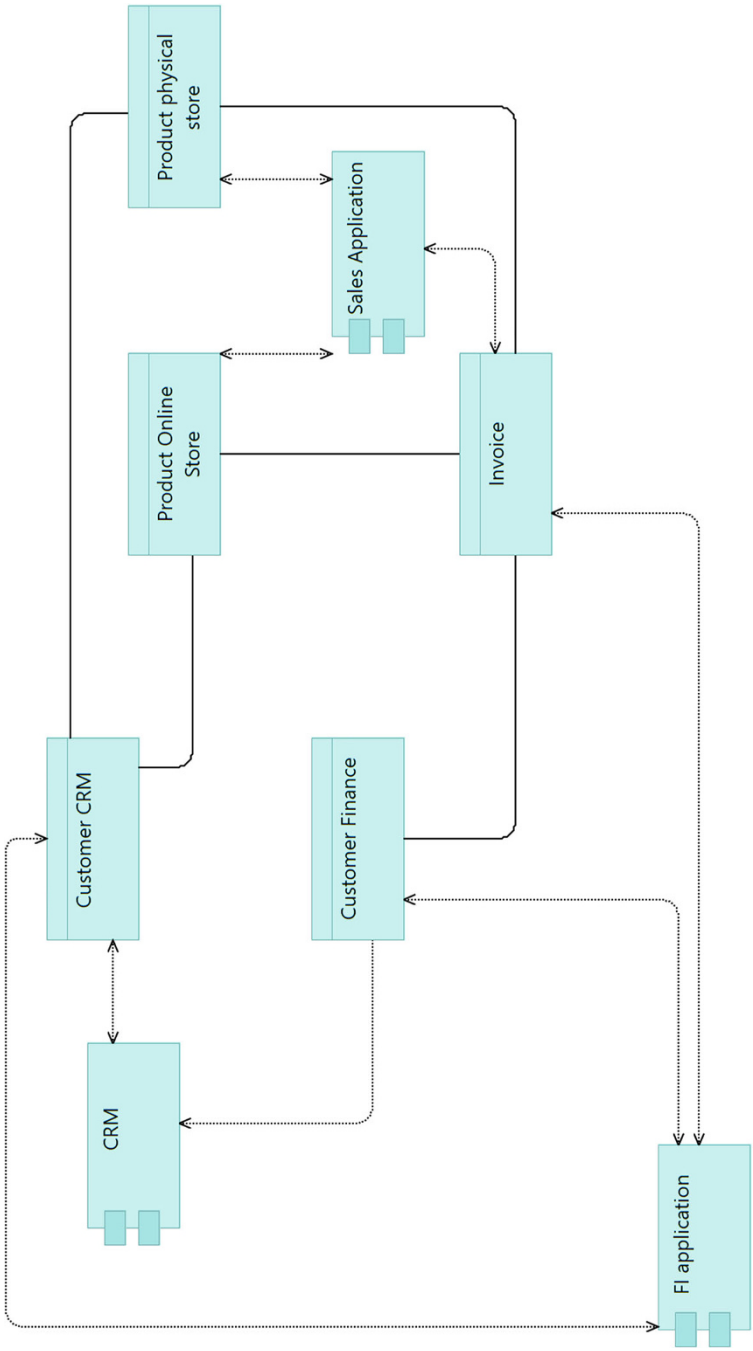


Fig. 4.37 Example of architecture that does not use the principle common vocabulary and data definitions

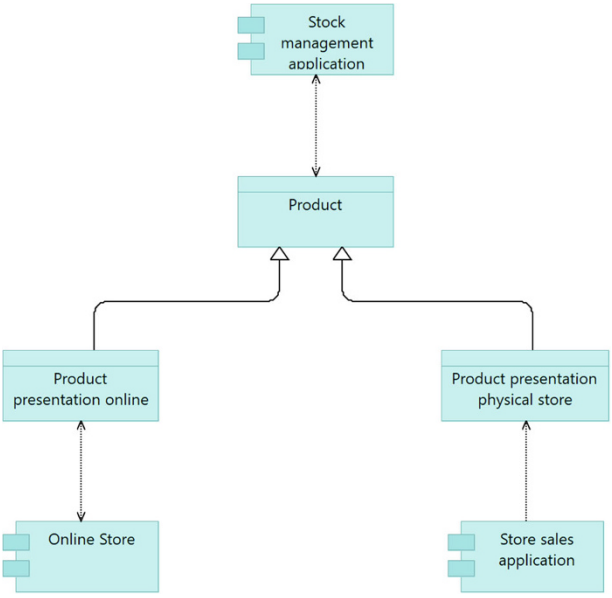


Fig. 4.38 Example of architecture applying the principle content and presentation are separated

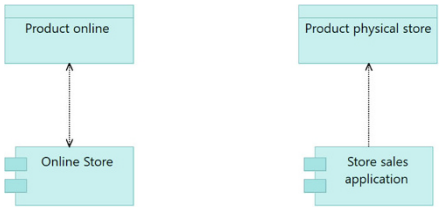


Fig. 4.39 Example of architecture that does not use the principle content and presentation are separated

In order to implement this principle, organizations should manage centrally the canonical data model. The canonical data model shall be used when applications exchange information (either directly by the applications involved in the data exchange or by using a integration software for performing the translation from the application-specific data model to the canonical data model) [2].

In Fig. 4.40, the exchange of the customer data among the three applications is supported in a common definition of the customer data.

In Fig. 4.41, there are different definitions of the customer data (in the sales and in the finance applications).

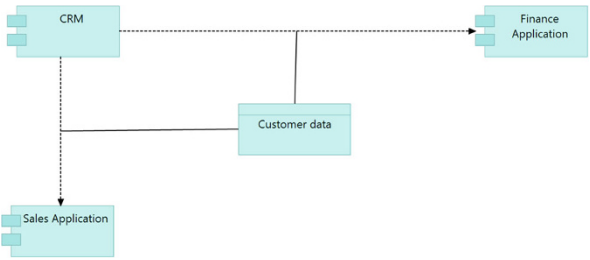


Fig. 4.40 Example of architecture applying the principle data that is exchanged adhere to a canonical data model

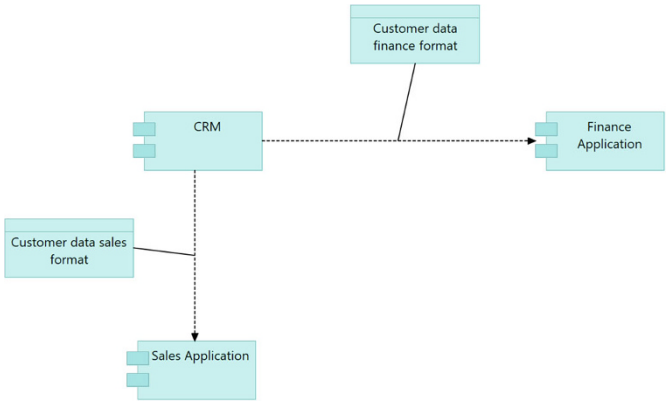


Fig. 4.41 Example of architecture that does not use the principle data that is exchanged adhere to a canonical data model

4.4.5 The Number of Implementations of the Same Information Entity Is Minimized

In order to increase interoperability and maintainability qualities, the number of implementations of the same information entity shall be minimized.

The existence of different implementations of an information entity points to semantic problems for that information entity (e.g., by using different formats or attributes in the implementation of an information entity). Therefore, the realizations of the same information entity in the technology architecture are minimized [3].

Figure 4.42 presents the implementation of the customer and the product information entities in the technology layer. Each data object is realized in a single artifact.

In Fig. 4.43, there are several implementations of each data object—the customer has two different implementations and the product three.

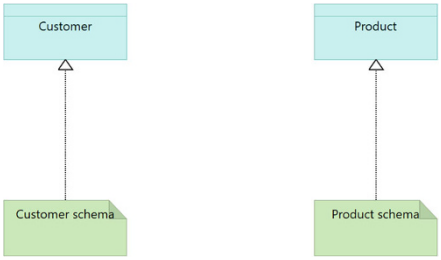


Fig. 4.42 Example of architecture applying the principle data that is exchanged adhere to a canonical data model

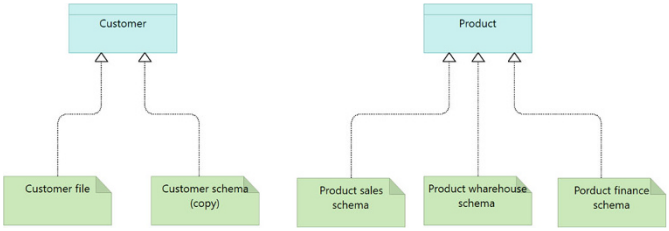


Fig. 4.43 Example of architecture that does not use the principle data that is exchanged adhere to a canonical data model

4.5 Applications Layer Principles

4.5.1 Common Use Applications

The common use of applications principle is concerned about efficiency and maintainability qualities. This principle argues that organizations should avoid the development of multiple similar applications that address a common need. The development of duplicate applications is expensive to implement and maintain and leads to data replication [1].

Thus, business units shall use applications that support the entire enterprise not developing applications for their own needs (for similar enterprise-wide applications) [1].

This principle is a specialization of the components are centralized principle (see Sect. 4.2.1), applied at the application architecture. Figure 4.2 presents an application architecture where the Intranet, email, HR, and Financial applications are used across the organization. In Fig. 4.3, the email, HR, and Financial applications are replicated.

4.5.2 *Presentation Logic, Process Logic, and Business Logic Are Separated*

This principle is concerned about maintainability quality. According to [2], since the presentation, process, and business logic functionalities are different if they are implemented in different software components, it is expected that its reuse will increase.

In order to comply with this principle, application components must have a layered approach separating presentation logic, process logic, and business logic. Additionally, the access to data is only implemented in business logic components [2].

Figure 4.44 presents the implementation of the online store and the face-to-face sales applications using a layered approach (where the process logic and the business logic are shared among the applications).

On the other hand, in Fig. 4.45, a monolithic approach is selected, where each application is implemented in a single software component (replicating common process and business logic functionalities).

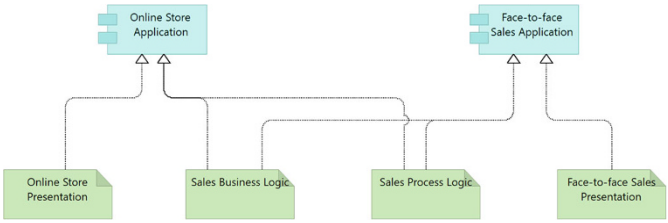


Fig. 4.44 Example of architecture applying the principle presentation logic, process logic, and business logic are separated

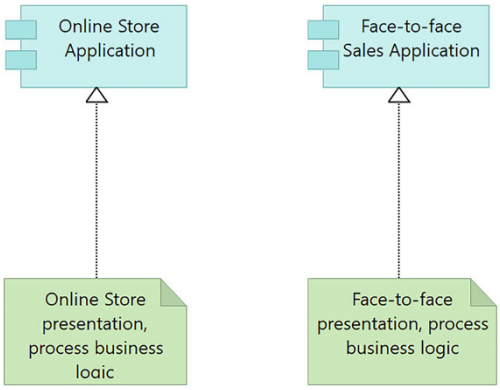


Fig. 4.45 Example of architecture that does not use the principle presentation logic, process logic, and business logic are separated

4.5.3 *Business Logic and Presentation Components Do Not Keep the State*

This principle is concerned about efficiency quality.

The scalability of an application is increased if business and presentation components do not keep the state (since it will be easier for implementing new parallel instances of these components) [9].

The scalability of an application tends to grow if the presentation and the logic application components do not preserve the application state (stateless). The state of the application should be managed by specific data components.

In order to implement this principle, the application architecture should ensure that [3]:

- Data is not recorded at the business or presentation levels.
- The state of the application in record by data components (e.g., Database Management Systems).

4.5.4 *Minimize the Number of Dependencies and Applications per Service*

This principle is concerned about maintainability quality.

According to [10], in the software engineering area, the higher the number of paths in a program, the higher its control flow complexity probably will be. The same occurs in the implementation of application services; they tend to be more complex if they depend on more applications [3]. Thus, in order to reduce complexity, each application service should be realized by the least number of applications [3].

In Fig. 4.46, the update customer address is realized by a single application (CRM Application).

On the other hand, in Fig. 4.47, the same application service is realized in four different applications, increasing its implementation and maintenance complexity.

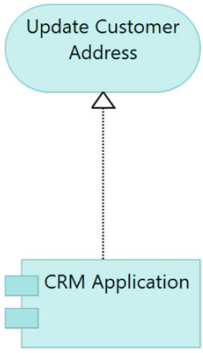


Fig. 4.46 Example of architecture applying the principle minimize the number of dependencies and applications per service

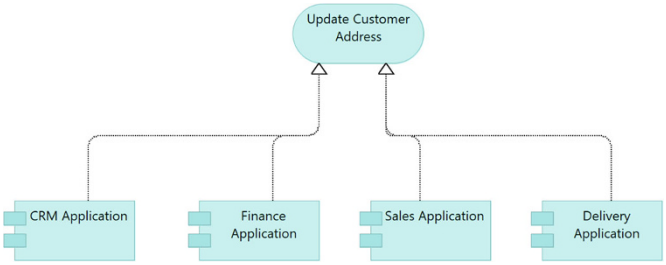


Fig. 4.47 Example of architecture that does not use the principle minimize the number of dependencies and applications per service

4.6 Infrastructure Layer Principles

4.6.1 Technology Independence

This principle argues that applications should not depend on the specific technologies or platforms, following common standards contributing for portability and maintainability qualities.

According to [1], applications that are not technology independent tend not to be developed and operated in a cost-effective way. Thus, the application software used or developed should not be dependent on specific hardware, operating systems, or systems software.

In order to comply with this principle, applications should select standards that support portability, and legacy applications must have Application Program Interfaces (APIs) that enable them to interoperate with the remaining applications. If needed, integration software (middleware, enterprise service bus) may be used to reduce the dependencies on specific technologies [1].

In Fig. 4.48, the SAP application provides a web service interface (independent of the technology).

On the other hand, in Fig. 4.49, the same SAP application only provides SAP Remote Functional Call interface (that is not independent of the SAP technology).

4.6.2 Interoperability

The interoperability principle addresses portability, efficiency, and maintainability qualities.

TOGAF interoperability principle states that software and hardware should conform to defined standards that promote interoperability for data, applications, and technology, in order to maximize return on investment and reduce costs [1]. Thus, it is recommended to comply to standards and to establish internal processes

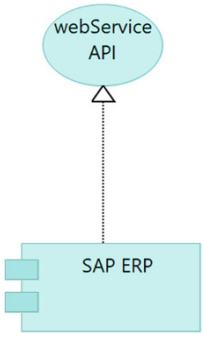


Fig. 4.48 Example of architecture applying the principle technology independence

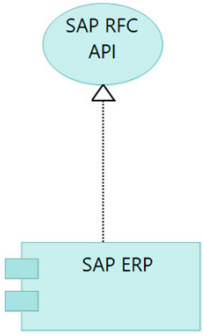


Fig. 4.49 Example of architecture that does not use the principle technology independence

for the definition and revision of the standards accepted in enterprise applications. Technology interoperability is expected to reduce vendor lock-in and increase competition among application vendors.

In order to implement this principle, organizations should use interoperability standards when available [1].

4.6.3 IT Systems Are Scalable

It is important to ensure that IT systems are scalable, since new market opportunities must be supported, even if not anticipated. Since acquiring systems with all the maximum future needs is expensive, considering that the cost of technology tends to be lower over time, it is important to ensure that the technology components are scalable [2].

In order to ensure that IT systems are scalable, IT components must scale horizontally (by adding further nodes performing the same tasks) or vertically (by increasing resources available of existing nodes). Additionally, IT systems should be sized at the current volumes and must be monitored periodically [2].

4.6.4 IT Systems Adhere to Open Standards

IT systems adhere to open standards principle argues that the usage of open standards prevents vendor lock-in and eases the integration of IT systems, increasing maintainability and portability qualities [2].

When selecting standards to adopt, organizations should consider the standard maturity and relevance. Whenever possible, open standards should be selected, and existing proprietary interfaces must be wrapped into open standard interfaces [2].

In Fig. 4.50, the Finance and the HR application follow oAuth open standard to authenticate among both applications.

On the other hand, in Fig. 4.51, the applications use proprietary Interfaces.

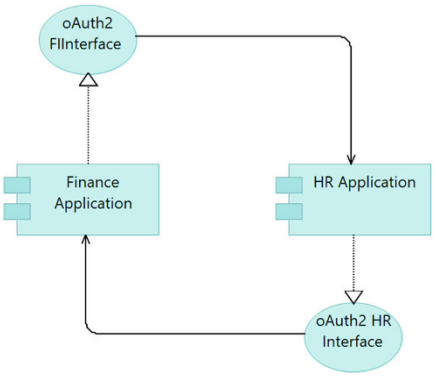


Fig. 4.50 Example of architecture applying the principle IT systems adhere to open standards

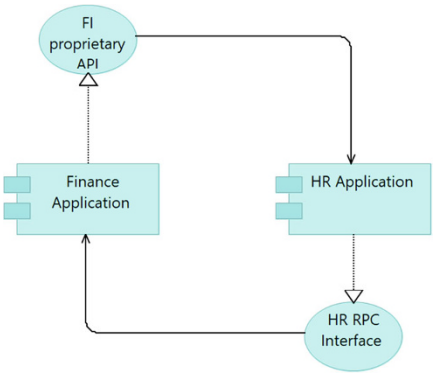


Fig. 4.51 Example of architecture that does not use the principle IT systems adhere to open standards

4.6.5 IT Systems Are Preferably Open Source

According to [2], open-source software prevents vendor lock-in and has lower acquisition costs than commercial software. This principle is expected to have a positive impact in efficiency and maintainability.

To implement this principle, when a functionally equivalent open-source software is available, the open source should be selected rather than the commercial version [2].

4.6.6 All Messages Are Exchanged Through the Enterprise Service Bus

In order to increase maintainability and portability, this principles states that and enterprise service bus (ESB) should be used to exchange messages among systems [2].

Having an ESB ensures that changes in a system won't have impact in other systems, since the ESB hides semantic (e.g., data model) or technology changes (e.g., integration or communication protocols used). Additionally, an ESB may increase the quality of the message exchange since the ESB provides specific tools for its persistence and management. Finally, having an ESB contributes for reuse of services among applications [2].

In order to implement this principle, instead of applications which exchange messages directly to other applications, all messages go through the ESB [2].

In Fig. 4.52, there is an ESB that handles the specificity of the integration with four different applications.

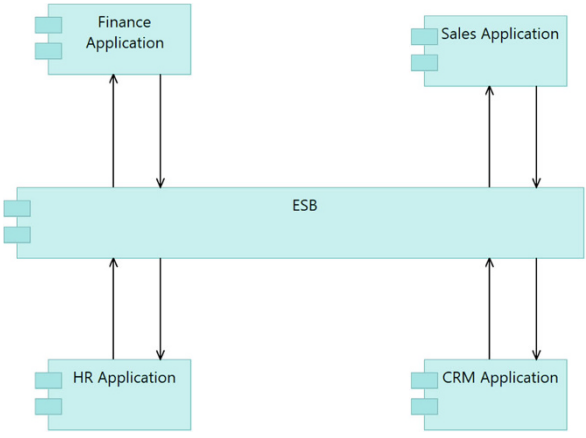


Fig. 4.52 Example of architecture applying the principle all messages are exchanged through the enterprise service bus

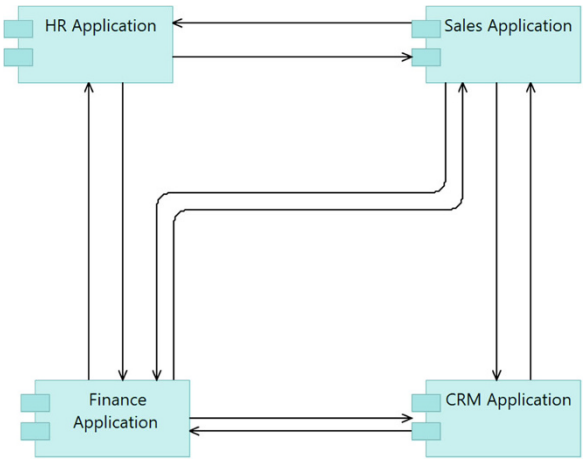


Fig. 4.53 Example of architecture that does not use the principle all messages are exchanged through the enterprise service bus

In Fig. 4.53, the integration is done directly point-to-point among applications, imposing that each application is able to deal with the integration technology and the data model of two or three other applications.

4.6.7 Software Components Are Multi-platform

This principle is concerned about the portability quality. The portability and technical interoperability increase with the number of possible platforms where components are able to operate. In order to implement this principle [3]:

- Components should run in multiple operating systems.
- Components that can run on various software and hardware platforms are preferred.

4.7 IT Architecture Patterns and Practices

4.7.1 IT Architecture Layers Patterns

Almost all applications have three structural components: presentation, logic, and data. The differences begin with the introduction of network sections between the components (see Figs. 4.54 and 4.55).

Regarding distributed presentation, "X Window System," dummy 3270 terminals, and pure HTML web pages (no JavaScript) are examples where calculations for the presentation are mostly done on the server side.

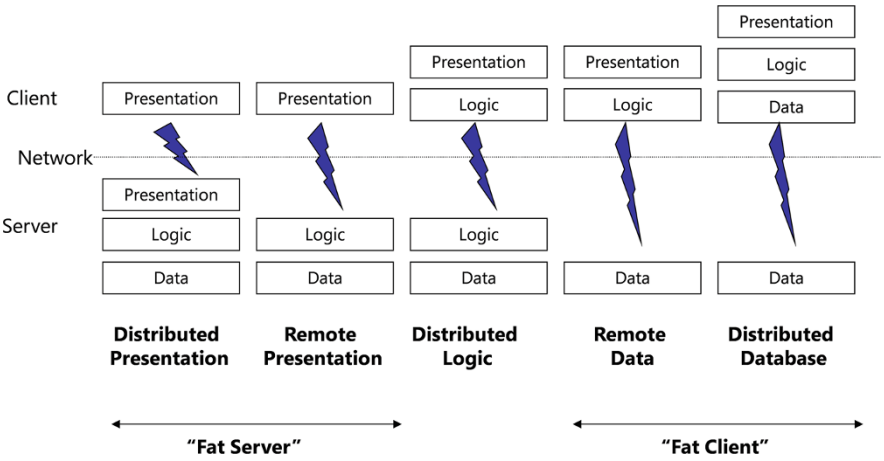


Fig. 4.54 Architectures with one network section's component distribution

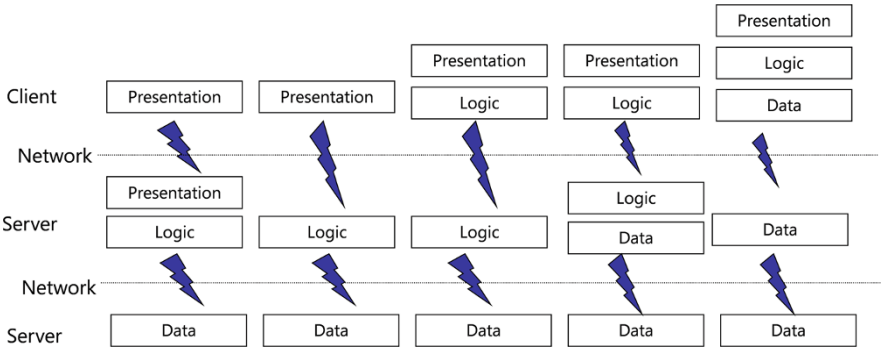


Fig. 4.55 Architectures with two network sections' component distribution

Regarding remote presentation, a web browser with JavaScript is a good example. For distributed logic, a browser running Java applets is an example of this architecture. In remote data architecture, the client makes calls directly to the database. The SQL flows through the network, as well as the answers. In a distributed database environment, part of the data is kept on the client side which is regularly synchronized with the server.

4.7.1.1 Two-Layer Versus Three-Layer Architectures

Three-layer (3L) architectures have, in general, the advantages over two-layer (2L) architectures presented in Table 4.3.

Table 4.3 Two-layer versus three-layer architectures summary

Quality	2 layer	3 layer
Security	–	+
Data encapsulation	–	+
Performance	–	+
Availability	–	+
Reuse	–	+
Ease to develop	+	–
Integration with legacy	–	+
Scalability and flexibility	–	+

Regarding **security**, three-layer architectures have the advantage since:

- 2L—security is done only in terms of data access.
- 3L—security is at methods access level.

Data encapsulation is better in three-layer architectures considering that:

- 2L— tables are traveling on the network (at least the structure of the data in SQL commands).
- 3L—only the methods and the results circulate on the network.

Although one more layer exists in three-layer architectures, **performance** is better since:

- 2L—much unnecessary network traffic is generated.
- 3L—there is no waste of network data; only the methods and results circulate on the network.

Availability is usually high in three-layer architectures since:

- 2L—SQL requests are made directly to a database server (it might reach a maximum number of "open connections").
- 3L—orders are directed to any application server.

Reuse is low in two-layer architectures since:

- 2L—client software must directly deal with the tables (in the DBMS).
- 3L—client software interacts with application servers methods (that hide the DBMS).

Development is easier and faster in 2L, since:

- 2L— low expertise is required.
- 3L—the separation of logic interface and application objects may be difficult in some cases.

Integration with legacy systems is very hard in two-layer architectures since:

- 2L—client software calls tables directly.

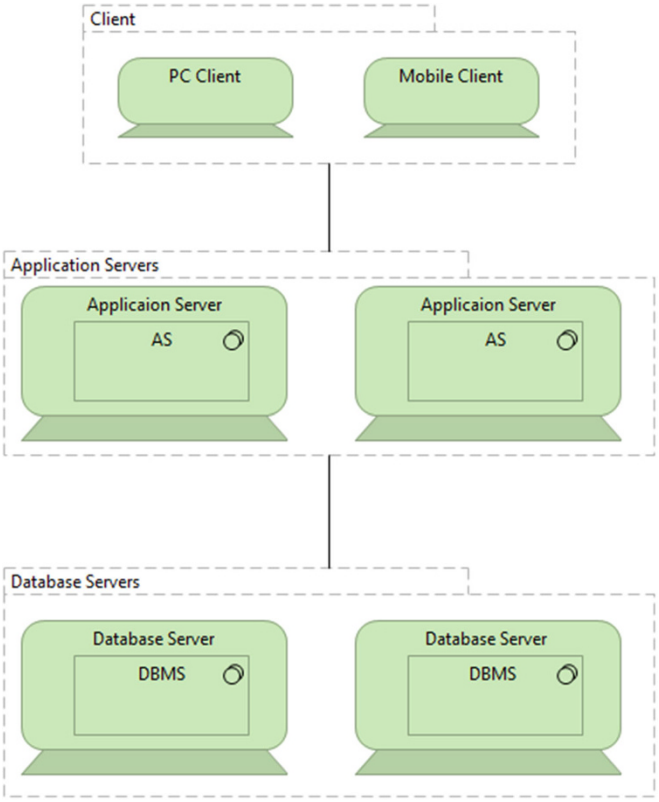


Fig. 4.56 Three-layer architecture

- 3L—Integration with legacy systems can be done by changing the application server.

Two-layer architectures have low **scalability and flexibility in hardware** considering that:

- 2L—Communication problems are raised (e.g., maximum DB connections reached).
- 3L—Scalability and flexibility are higher since clients call a service, not a process. The processes are launched on machines that may easily balance load. The logic is in the components that can be run on any server. There are components that ensure access to the database (ensuring the data integrity) (see Fig. 4.56).

Regarding security, the IT Architecture usually has three distinct security zones—Fig. 4.57.

Firewalls can limit accesses (who and what) between the parts of the network. In addition to security, firewalls are also important to reduce traffic on the network.

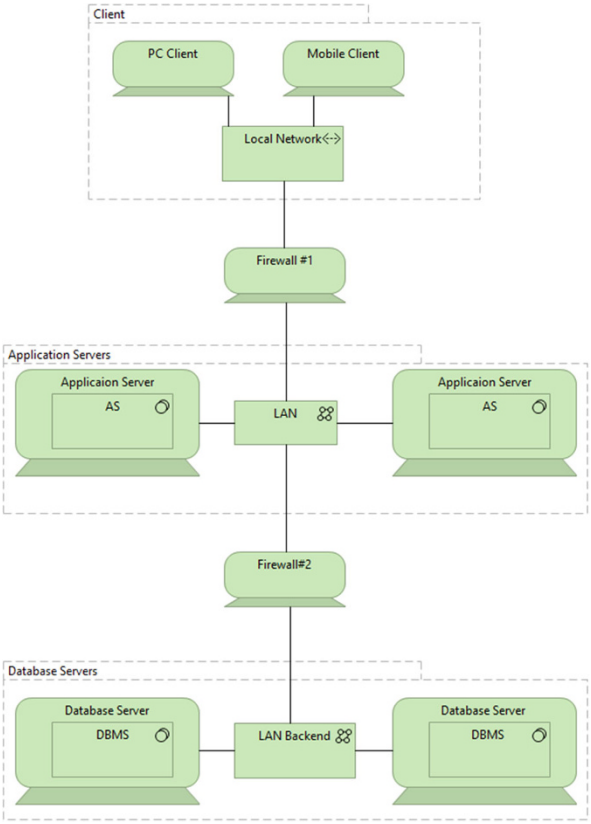


Fig. 4.57 Different security zones

Also notice that the same physical firewall can execute the role of two. For example, in a typical IT architecture for the Web, the same firewall can isolate different network segments—see the example in Fig. 4.58

4.7.2 Architectures for High Availability

Several patterns are available for increasing availability.

A possible approach to increase availability is accomplished using **backup servers** (see Fig. 4.59). This is the cheapest solution, but it has some problems, including:

1. Recovery is complex and slow.
2. A fault detection mechanism must be available: heartbeat which asks if the server is alive.

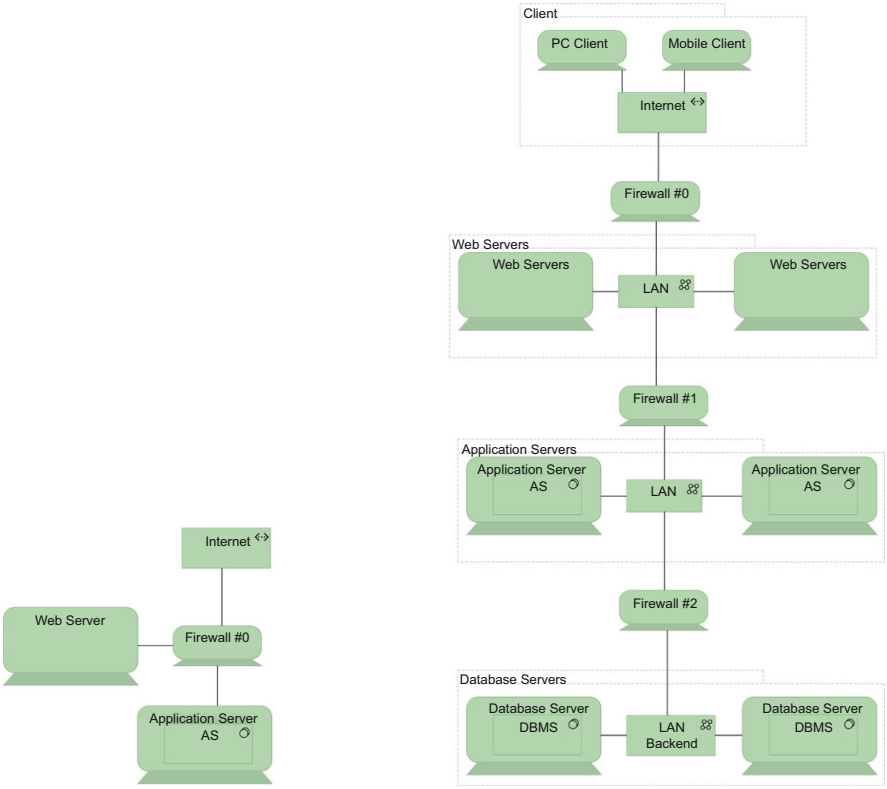


Fig. 4.58 A firewall with different roles

3. Re-processing of messages lost: there are no guarantees, since failure may have been on the message "sent," on the server, or on the server response or the processing of the response by the customer.

Passive Duplication is another approach for high availability (see Fig. 4.59) with the following characteristics:

1. It implies a standby server.
2. It is "transparent" to the user
3. Re-processing of lost messages—Reduces the Log and can reduce the time to replace the Network so that customers do not perceive. It is not used much.

Active-Active duplication is another approach where a "backup server" is used, but operating (see Fig. 4.60). It is "transparent" to the user and the best approach for stateless servers. It might have some loss in performance due to data update. It may be applied to a Database Cluster: both systems write to all disks. Problems might arise with Logs and Locks.

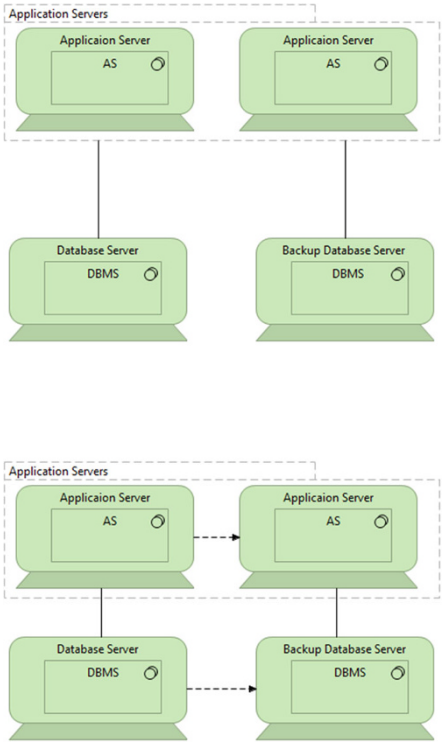


Fig. 4.59 Increasing availability by using backup servers (in the top) and passive duplication (in the bottom)

No architecture solves all problems, including those related to the connection to the outside. It's a business decision how to react when problems are detected.

4.8 IT Integration Patterns

4.8.1 Introduction

Integration was born as a technological possibility to connect multiple machines across networks but has become a way of overcoming the problems posed by the continuous development of business requirements. The scope moved from integration of technologies, for application integration and, more recently, for business integration.

Enterprise Architecture aims to solve similar problems, as Business processes, or significant parts of these processes, are not adequately supported by information systems; inconsistencies, incoherence, and replication exist in operational information; difficulties in the IS response to new business needs; and difficulties in the adoption of new technologies.

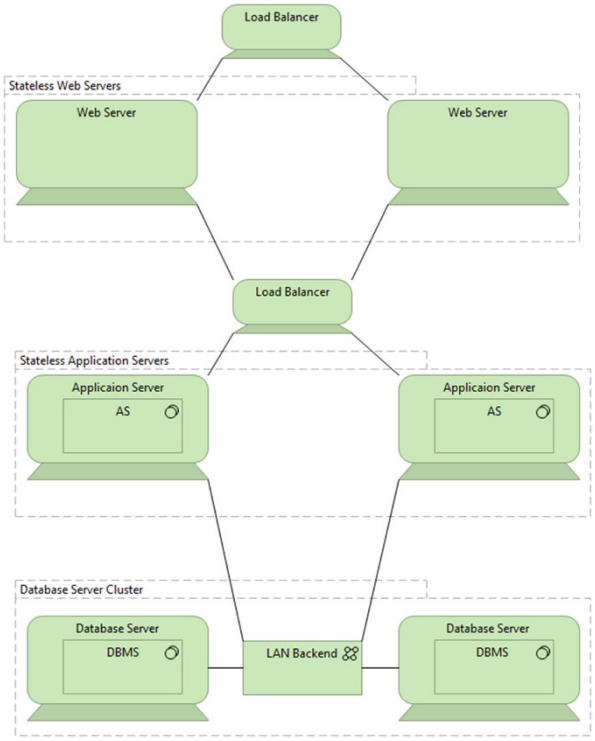


Fig. 4.60 Active-active duplication

We present next the major integration techniques from an architectural point of view.

4.8.2 File Transfer

One of the oldest integration technique is the file transfer. Its main goal is to exchange information between two systems using coding and decoding objects and a file. File Transfer Protocol (FTP) is a two-step technique:

- Encoding: Objects to file;
- Decoding: File to objects.

The main advantage of FTP is that it is a universal approach, since all operating systems and programming languages support the notion of file. However, it has several disadvantages, including:

- The complexity of encoding and decoding increases exponentially with the objects to be transferred.

- It can only be used to exchange objects whose type is relatively simple.
- The performance is limited.
- It implies the duplication of data.

This technique is still widely used despite its drawbacks (e.g., in some legacy financial transactions).

4.8.3 Screen Scraping

Screen scraping, screen harvesting, or form scraping has the main goal of extracting information directly from the user interface to a system in order to be used in another system (Fig. 4.61).

The main advantages of this integration technique are:

- Appropriate to integrate legacy applications where the code cannot be accessed or changed (e.g., COBOL applications on mainframes).
- There is no direct access to the data.
- It is not necessary to change the source application/system.
- Specialized applications exist for this purpose.

The disadvantages of screen scraping are:

- The user interface is not intended to integration.
- The user interface usually does not reflect the data type.
- It is not simple to simulate a (human) user using an application.
- The quality and performance of this approach are generally low, and the complexity is high.

Therefore, screen scraping is only used as a "last resource" technique for encapsulating (closed) legacy systems.

4.8.3.1 Web Scraping or Web Harvesting

Web scraping or web harvesting applies a similar approach of screen scraping to extract information from web pages. It is mostly used by web crawlers. Web pages are built with textual markup languages (HTML, XHTML), usually designed for human consumption. Its design often mixes content with presentation. Due to

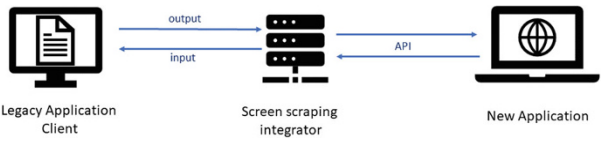


Fig. 4.61 Screen scraping

the widespread use of web scraping techniques, currently several "anti-scraping" techniques also exist.

4.8.4 Remote Procedure Call

Remote Procedure Call (RPC) protocol was created to support distributed programming based on the procedures called by clients that run remotely on the server. It is the simplest way of middleware. It provides the basic infrastructure to transform "procedure calls" into "Remote Procedure Calls" (RPC) in a transparent and uniform way (Fig. 4.62).

Currently, RPC is the "foundation" of almost all other forms of Middleware. "StoreProcedures" is a type of RPC to interact with Databases. Remote Method Invocation (RMI) is identical to RPC but applies to methods of objects (instead of procedures).

The synchronous architectures, as RPC, introduce high cohesion (tight coupling) between the caller and the called. The services and directory names try to minimize this effect, but do not eliminate it completely. This type of architecture presents specific problems in various areas:

- The management level of interaction between client and server is difficult (e.g., fault tolerance, availability, load balancing).
- The level of communication properties and services is hard to ensure (e.g., transactional behavior, compensation, exception handling).

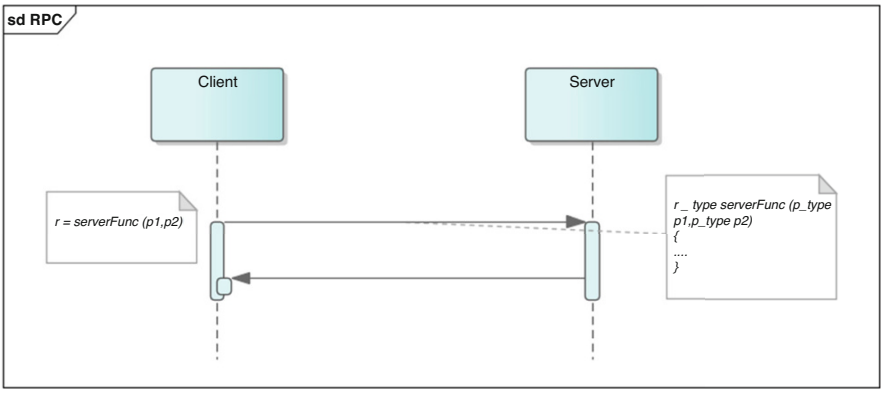


Fig. 4.62 RPC example

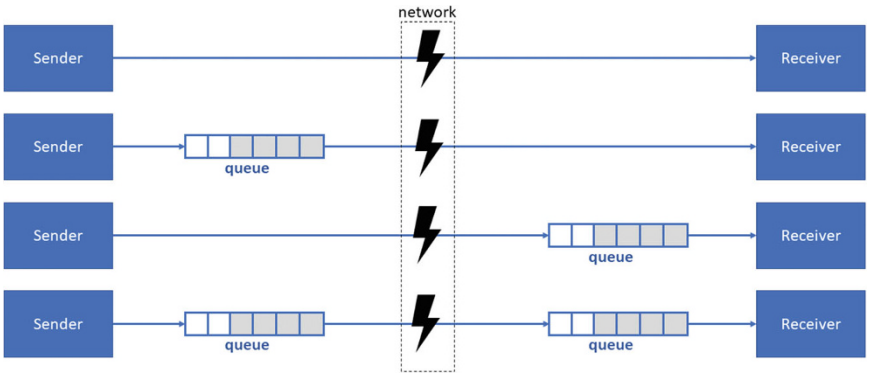


Fig. 4.63 Message queues

4.8.5 Message Queues

Message queues arise considering that the interaction does not always need to be synchronous. It provides access to transactional queues, persistent queues, and a set of primitives for reading and writing to local and remote queues.

Message queues provide a common ground for interoperability based on messages. In this paradigm, clients and servers interact using messages. A message is a set of structured data, characterized by a type, and parameters (set of pairs "name, value").

The interaction model is a publish/subscribe. Considering the possibility of defining routing logic for messages at the broker, there are several possibilities for message-based interactions. The publish/subscribe interaction is the most known and used one. In this pattern, the sender of the message does not send the message to any recipient—it just puts it in the queue. The recipients (interested) are responsible for subscribing a message type. The "queue" then ensures sending a copy of the message to all the subscribers (Fig. 4.63).

In model-driven messaging, client messages are placed in a queue, and when the recipient is ready to process, it invokes a function. Several benefits arise from queue model, including:

- Control when processing messages.
- Increased robustness to failures.
- Better distribution of applications across multiple hardware (for higher performance and availability).
- Message priorities setting.
- Interaction with the message queue system.

The main problems with message queues are:

- Asynchronous communication involves a programming model less intuitive than the RPC (programming events).

- The message queues servers are one more investment, management, and support costs.
- The message queue server is one more component of the architecture of the middleware that needs to be integrated with others.

4.8.6 Message-Oriented Middleware

In message-oriented middleware, the integration is performed using the routing information (messages) among systems. Applications receive and send messages to a message broker.

The messages, once received by the server, can be reformatted, combined, or modified in order to be understood by the target system. It is usually not necessary to modify the systems involved. The message brokers provide adapters for the most common applications.

The main goals of a Message-Oriented Middleware (MOM) integration are:

- Store and forward: message is persisted (saved) and delivered to the recipient (even in case of recipient application failure).
- Broker: all systems interact with a single point.
- Publish and subscribe model: there is a message subscribe queue.
- Assurance that the messages are delivered to the recipient, including two-phase commit protocols.
- Sorting capability.
- Ability to dynamically select the recipient based on the message content.
- Simulating synchronous operation: request/response.
- Confidential support through encryption.
- Sending events (e.g., unavailability of recipient).
- Ability to transform and filter messages as they move between the servers.

4.8.7 Data-Oriented Integration

One of the basic mechanisms for integration is data-oriented integration which includes:

1. Extracting the information from the source repository.
2. Processing or transforming information.
3. Updating the destination repository with the processed data.

The main advantages of data-oriented integration are:

- Simple mechanism to implement.
- There are access mechanisms independent of the technology.
- It does not require rework or application modifications.

The disadvantages of data-oriented integration are:

- It requires technical knowledge about the management systems databases because the operation of access and update may impact the consistency of the information.
- It may require to have knowledge about the inner workings of applications to ensure consistency of information.
- It may involve transformations between data types (possibly incompatible).
- The transformed data is not validated by the application that uses it.
- There is no guarantee of consistency between the replicated information (from repositories).

4.8.7.1 Integration via DBMS

The integration using database management systems (DBMS) defines an API for applications to access information. It converts the API commands in a language the database understands (e.g., SQL). The client sends the command to the DBMS. The DBMS processes the command and sends the result to the client. The client converts the response into a format understandable by the target application.

This interaction model can be seen as a dedicated RPC that interacts with a DBMS. Open Database Connectivity (ODBC) allows access to data in a DBMS, independently of the DBMS technology, the programming languages, and the operating system. It is a standard proposed by Microsoft in 1992. The API is independent of the DBMS. The same API is supported by multiple drivers seamlessly. The ODBC drivers depend on the specific database engines. There are multiple compatible implementations (directly or through bridges), such as Microsoft ODBC, JDBC, iODBC, and IBM i5/OS.

There are also interfaces based on the SQL standard defined in the Java platform to access relational databases. Java Database Connectivity (JDBC) is an API that defines a set of Java classes that allow an application to connect to a database.

ODBC is a mechanism mainly to access (remotely) to a Relational Database. It has been evolving to integrate various types of repositories in a transparent manner:

- relational databases
- non-relational databases
- text files (flat files)
- spreadsheets,
- email

It allows the creation of “Virtual Database,” independent of the shapes of the sources of information.

4.8.8 Application Interface-Oriented Integration

The most commonly used packaged applications usually expose interfaces to access and process information. The trend is that the functionality of these applications is exposed through services rather than proprietary APIs.

Some (few) packages have well-documented interfaces allowing access to information and high-level processes. The main drawback of API integration is that each application defines a different interface. The interfaces are complex (and sometimes poorly documented). Additionally, version evolutions tend not to ensure backward compatibility. As an example, SAP Business Objects (BO) are objects that encapsulate data and processes associated with a business object (e.g., Material, Purchase Order, and Customer). External access to these data and processes is done through specific methods of these objects called Business Application Program Interfaces (BAPI). The BAPI in SAP Web Application Server are implemented as function modules that support the protocol RFC (Remote Function Call) and are described as methods of a SAP BO.

4.8.9 Transactions and Transaction Monitors

Atomicity, consistency, isolation, and durability (ACID) are the four primary attributes ensured by any transaction:

- **Atomicity**: Either all the tasks in a transaction occur or none of them occurs. The transaction must be completed, or else it must be undone (rolled back).
- **Consistency**: Every transaction must preserve the integrity constraints. It cannot leave the data in a contradictory state.
- **Isolation**: Two simultaneous transactions cannot interfere with one another. Intermediate results within a transaction must remain invisible to other transactions.
- **Durability**: Completed transactions cannot be aborted later or their results discarded. They must persist through (for instance) restarts of the DBMS or after crashes.

The properties of a DBMS apply only to transactions within its domain. In principle, the properties are no longer valid when an operation crosses more than one DBMS or when operations access data outside the control of the DBMS (e.g., on the server itself or in other layers of the architecture, including middleware). Therefore, the general rule is that whenever the data is distributed, the properties guaranteed by a DBMS do not apply.

To overcome the problem of the distribution of transactions, The Open Group defined the Distributed Transaction Processing Model (DTPM). This model aims to ensure that a distributed operation (i.e., that crosses multiple DBMS) meets the ACID properties. The model DTPM defines the XA interface. The XA needs

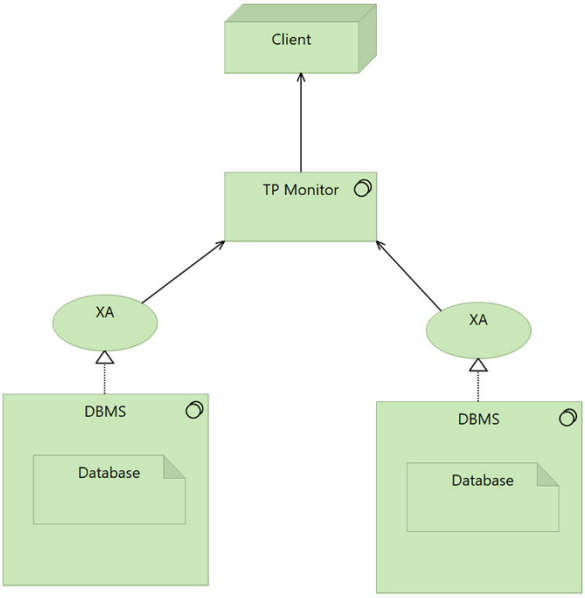


Fig. 4.64 Transactional RPC

to be supported by all nodes participating in a distributed transaction. The XA implementations ensure the atomicity of a distributed transaction supported in a 2 Phase Commit (2PC) transactional protocol.

The limitations of RPC can be resolved by making RPC calls transactional. In practice, this means that they are controlled by a 2PC protocol. As before, an intermediate entity is needed to run 2PC (the client and server could do this themselves, but it is neither practical nor generic enough). This intermediate entity is usually called a **Transaction Manager** (TM) and acts as an intermediary in all interactions between clients, servers, and resource managers. When all the services needed to support RPC, transactional RPC, and additional features are added to the intermediate layer, the result is a TP-Monitor (Fig. 4.64).

A TP-Monitor allows building a common interface to several applications while maintaining or adding transactional properties. A TP-Monitor extends the transactional capabilities of a database beyond the database domain. It provides the mechanisms and tools necessary to build applications in which transactional guarantees are provided. TP-Monitors are, perhaps, the best, oldest, and most complex example of middleware. Some even try to act as distributed operating systems providing file systems, communications, security controls, etc. TP-Monitors have traditionally been associated with the mainframe world. Their functionality, however, has migrated to other environments and has been incorporated into most middleware tools.

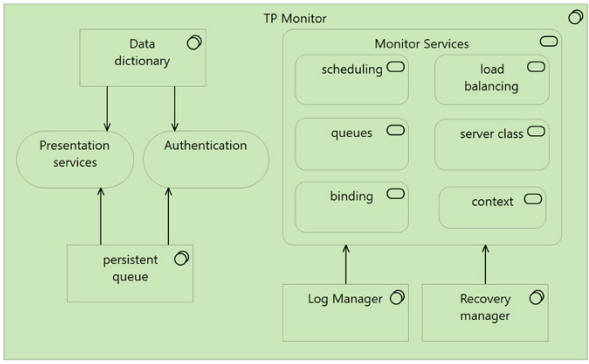


Fig. 4.65 TP-Monitor components

The typical features of a TP Monitor are:

- Features to support RPC (e.g., IDL, nameservers/directory, security, compilers Stubs, etc.).
- Transaction manager with features like logging, recovery, and locking.
- "System monitor"—responsible for the scheduling of threads, prioritization, load balancing, and replication.
- Execution environment for all the applications that use the TP Monitor.
- Specialized components for certain systems or scenarios (e.g., protocols for interaction with mainframes, queues).
- Variety of tools for installation, management, and monitoring of all components.

The main components of a TP Monitor are (Fig. 4.65).:

- Interface—API for programmatic interaction with client applications.
- Program flow—guards and executes programmatic flows possibly written in the proprietary language of the TP Monitor.
- Router—contains the mappings between the physical and logical resources.
- Communications manager—messaging system with delivery guarantees (and rollback) for communication with resources (e.g., databases).
- Wrappers—hides the heterogeneity of different resources.
- Transaction manager—supports the execution of distributed transactions (making use of 2PC protocol).
- Services—offers a comprehensive range of services to ensure high availability, performance, and replication, among others.

4.8.10 Business Process-Oriented Integration

4.8.10.1 Workflow-Oriented Integration

The term "workflow" is often used to designate a formal description of an executable business process. A workflow management system (WfMS) is a software platform

that supports the design, development, implementation, and analysis of workflows. Workflows enable the automation of business processes across the organization. The features and basic functions of a WfMS are:

- Formalization authorization and approval of circuits.
- Digitization of business processes.
- Resource management of the organization.
- Obtaining the operational indicators and KPI.
- Security implementation and management of access control over the processes.
- Support automatic, semi-automatic, and manual activities.

The expected benefits of WfMS are:

- Cleared and structured processes.
- Easy explanation of problems or errors.
- Support for auditing and monitoring process.
- Transparent processes and perception of the path taken.
- Decentralization of work processes.
- Shared vision of the process flows.
- Support the collective work of the organization.
- Distribution of tasks according to the ability of the participants.

The integration using WfMS can be accomplished through two different architectures:

1. The workflow runs "on top" of the existing systems. The workflow provides a common interface, and operations may affect multiple systems. The workflow system keeps and manages the execution context of the process. Workflow systems should also allow to interconnect distributed processes in the organization (Fig. 4.66).

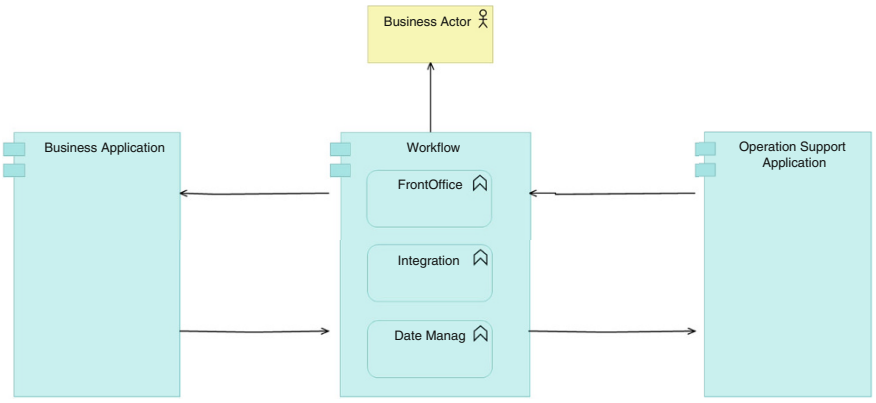


Fig. 4.66 Integration using a "WfMS on the top"

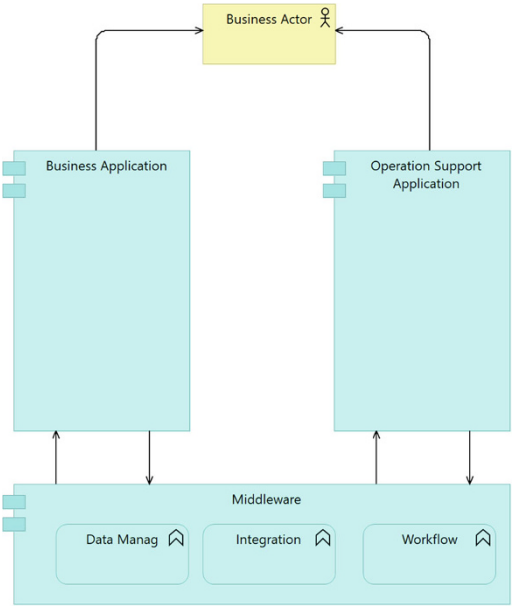


Fig. 4.67 Integration using "WfMS" as a Service Bus

2. When we face a scenario in which the integration of business processes is made only with participants who are systems/applications, the basic concepts of workflow are somewhat compromised. In this scenario (Fig. 4.67), the user interacts directly through existing application, and the WfMS is used to integrate applications (not to interact directly with the end user).

4.8.10.2 Business Process Execution Language

Business Process Execution Language (BPEL) aims to describe the interaction between business partners through sequences of synchronous and asynchronous messaging. It assumes that the interactions are long duration (long running) and have state. BPEL is based on the web services standards stack. It is supported by industry players such as IBM, Microsoft, SAP, and Oracle.

BPEL specifies the XML schema that contains the definitions of the flow of a business process. BPEL aims to generate executable code of the business process. It is an orchestration language (not choreography). Processes arise as service compositions. It defines processes that interact with the entities through web services (using WSDL to describe their contracts).

Business applications place requirements for integration and inter-operation. The current response to these challenges is based on SOA paradigm. Many enterprise applications expose their functionality through Web Services. But developing Web

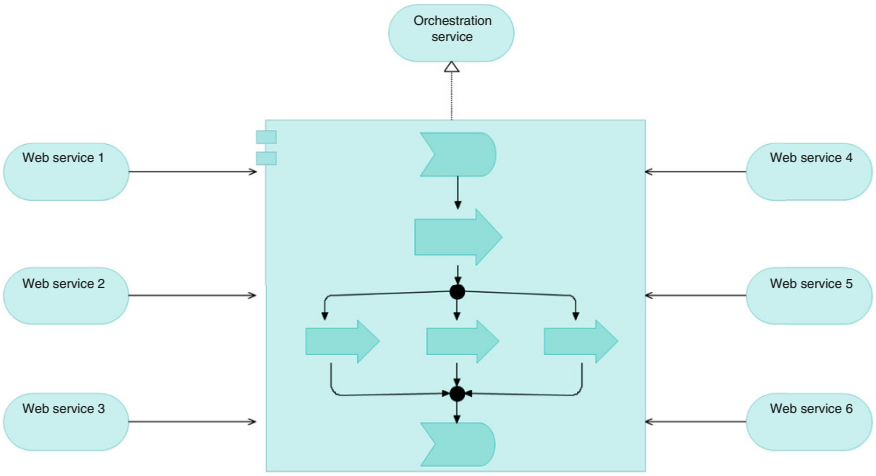


Fig. 4.68 BPEL orchestration

Services and exposing their functionality is not enough. A mechanism is needed to compose and orchestrate these functionalities (Fig. 4.68).

Within the organization, BPEL is used for setting standards for application integration, and for the specification of processes orchestration. Therefore, BPEL is used for the definition of the integration processes between systems that were isolated before. Outside of the organization (i.e., between organizations), BPEL is used for setting standards for inter-organizational communication (e.g., to support B2B), for simplifying integration with business partners, allowing explicit interorganizational processes that were previously implicit. BPEL orchestrates existing web services into a new (higher level) web service.

4.8.10.3 Orchestration vs Choreography

Orchestration is the specification of a flow from the perspective of a single entry point (single endpoint). As presented before, BPEL deals with Orchestration.

Choreography is defined as the exchange of messages, rules, and interactions between two or more entry points (endpoints) of business processes.

In Orchestration, only the central coordinator knows the process and the purpose of the process. A central process (which can also be a web service) takes control of the participants and coordinates the implementation of the different methods (web services) involved in the process. The orchestration is centralized through explicit definitions of operations and ordered calls to the web services involved. Web services involved do not "know" (and do not need to know) that they are involved in the composition of a process and that they are part of a business process at an upper level (Fig. 4.69).

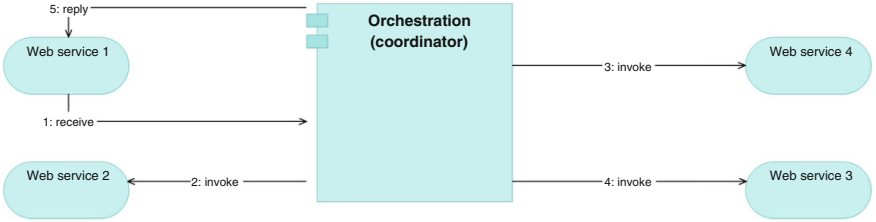


Fig. 4.69 Orchestration integration pattern

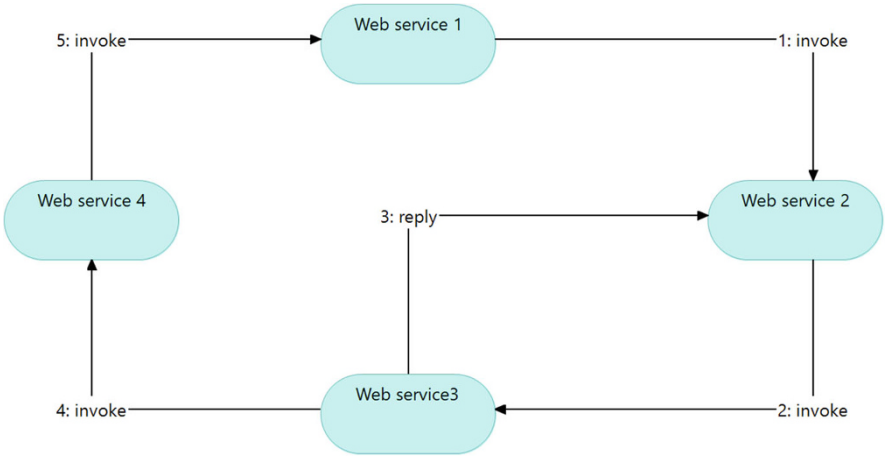


Fig. 4.70 Choreography integration pattern

In Choreography, there is no central coordinator. Choreography is a collaborative effort based on the exchange of messages in a process (usually public). All participants of the choreography have to worry about the process, the operations to be performed, the messages to be exchanged, and in which order messages should be exchanged. Each web service involved in the process knows when to run and meet other web services participants. At a minimum, each service knows the services with which it interacts directly—Fig. 4.70.

4.9 Exercises

Exercise 4.1

As presented in exercise 3.2, Lisbon Institute of Technology (LIT) established the following principles:

- Business units are autonomous.
- Customers have a single point of contact.

- Channel-specific components are separated from channel-independent components.
- Management layers are minimized.
- Common components are centralized (as the HR management process, or the course platforms).
- Messages with all the internal and the external systems are exchanged through LIT Enterprise Service Bus (ESB).

LIT CIO is considering different scenarios for supporting LIT Enterprise strategy and architecture. In one of the scenarios, the architecture described by Fig. 4.71 for supporting the course business process is being analyzed.

1. What architectural principles (described in LIT Strategy) are being violated in the architectural scenario described in Fig. 4.71? Justify your answer.
2. Review the architecture sketch of Fig. 4.71, and present a new one (in ArchiMate) that does not violate LIT architectural principles. Justify your answer.
3. LIT CIO is considering Orchestration and Choreography architectural scenarios. Considering the strategy described, which one would you recommend? Why?

Exercise 4.2

Several organizations in the public administration provide services to citizens that imply the use of the citizen‘s address.

Consider that N organizations intended to provide to citizens a single web interface (in the Internet) for the citizen to use whenever he or she wants to change his or her address; the goal is that the citizen can do “only once” the service and that change of address is propagated to the different organizations. However, each organization has its own systems (System S1,. . . , SN) that use different information technologies and different data models (Entity E1, . . . , EN), and it is not feasible to change the data models of existing information systems.

Therefore, it was decided that each organization will build its own information service (IS1,. . . ,IS10) to update the organization‘s data model and made it accessible to any other public entity, with specific data model for the address.

1. The architectural scenario of implementing a new Global Propagation System (GPS) to propagate an address to a list of information services (ISn) is being discussed. An architectural decision is whether if the GPS service will be implemented using an Orchestration or a Choreography approach. Assume that in any scenario, each organization Information System will only be integrated with a maximum of two other systems. Model in ArchiMate the two different scenarios.
2. What would be your recommendation for each of the architectural decision addressed in the previous questions (Orchestration vs. Choreography approach)? Justify your answer.

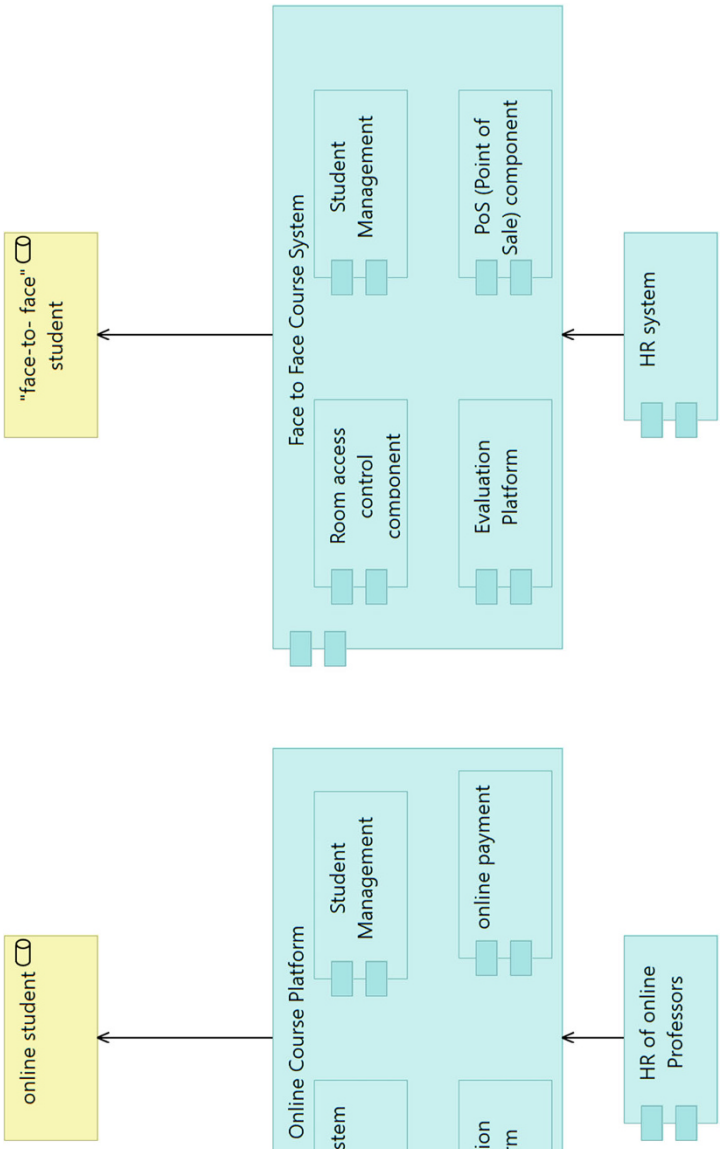


Fig. 4.71 LIT architecture scenario

References

1. The Open Group, *TOGAF Version 9.1,*, vol. 1 (Van Haren Publishing, 2011)
2. E. Proper, D. Greefhorst, Architecture principles—the cornerstones of enterprise architecture. *The Enterprise Engineering Series* (2011)
3. A. Vasconcelos, P. Sousa, J. Tribolet, Enterprise architecture analysis: an information system evaluation approach. Int. J. Enterp. Model. Inform. Syst. Archit. **3**(2), 31–53 (2008)
4. P. Sousa, Enterprise architecture alignment heuristics. Microsoft Enterp. Archit. J. **4** (2004)
5. ISO, *ISO/IEC, ISO 9126—Software Engineering—Product Quality* (2001)
6. R. Maes, D. Rijsenbrij, O. Truijens, H. Goedvolk, Redefining business—IT alignment through a unified framework, white paper, 2000
7. J. Sarkis, R. Sundarraj, Evaluating componentized enterprise information technologies: a multiattribute modeling approach. Inform. Syst. Front. (2003)
8. W. Inmon, Data architecture: the information paradigm (1993)
9. BEA, Scaling EJB applications, 2006
10. T. Mccabe, A complexity measure. IEEE Trans. Softw. Eng. **4**(2), 308–320 (1976)