UNIVERSIDAD
NACIONAL
DE COLOMBIA

A strategy based on model-to-model transformations to evolve service-oriented architectures to microservices architectures

Eduardo Fabio Berrio Charry

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, D. C., Colombia
2021

A strategy based on model-to-model transformations to evolve service-oriented architectures to microservices architectures

Eduardo Fabio Berrio Charry

Thesis presented as a partial requirement for the degree of:
Master in Systems Engineering and Computer Science
"Magíster en Ingeniería - Ingeniería de Sistemas y Computación"

Advised by:
Jeisson Andrés Vergara Vargas, M.Sc.
Henry Roberto Umaña Acosta, M.Sc.

Research Fields:
Software Architecture and Model-Driven Engineering
Research Group:
Colectivo de Investigación en Ingeniería de Software - ColSWE

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, D. C., Colombia
2021

Dedicated to my wife and my children.

"The important thing is not to stop questioning. Curiosity has its own reason for existing."

- Albert Einstein

Acknowledgments

First of all, thanks to the Universidad Nacional de Colombia, it has been a pride and a pleasure to study at this excellent institution and to contribute in some way to the research line of Software Engineering and more specifically to Software Architecture and Model-Driven Engineering. I also want to thank all the professors with whom I interacted during my time at the University, who shared their knowledge and experience, and their teachings have been fundamental not only for the development of this document but for my professional life. I want to thank in a very special way my advisors Jeisson Andrés Vergara Vargas and Henry Roberto Umaña Acosta, who with their guidance, dedication, expertise and excellent willingness contributed to the results obtained; it was a real teamwork.

Thanks to my wife Alba Lucía and my children Sofía and Nicolás for giving me all their love and for supporting me during this stage of my life. Many hours and weekends were invested during my studies at the University and in this research work. Without their patience and motivation it would not have been possible to achieve this goal.

I want to thank all the authors cited in this document, who contributed to give me the basis for this research work. Finally, thanks to the juries Felipe Restrepo Calle and José Joaquin Bocanegra Garcia for sharing their knowledge and experience and for dedicating their valuable time to the evaluation of this research work.

Title in English

A strategy based on model-to-model transformations to evolve service-oriented architectures to microservices architectures

Título en español

Una estrategia basada en transformaciones modelo a modelo para evolucionar arquitecturas orientadas a servicios a arquitecturas de microservicios

Abstract

Microservices architecture has emerged as an architectural style which focuses on the design and development of software systems as a set of small independent services. Although microservices architecture is inspired by the Service-Oriented Architecture style (both are service-based architectures), it presents important differences. Likewise, software architecture must evolve as new architectural styles and software frameworks arise, and the evolution of the software architecture is considered as a central feature of any software system. In this way, this research work presents a proposed approach based on model-to-model transformations to evolve service-oriented architectures to microservices architectures, from an architecture description language called Sarch. To accomplish this, the Sarch language was extended to allow the modeling of the two architectural styles from the component-and-connector view along with the layered and data model views, the decomposition view was included in the Sarch language to allow the description of a system from a functional point of view, and a set of model-to-model transformations was created to support the evolution from service-oriented architectures to microservices architectures.

Keywords: Software Architecture, Architectures Evolution, Service-Oriented Architectures, Microservices Architectures, Architectural Style, Architectural View, Architecture Description Language, Model-Driven Engineering.

Resumen

La arquitectura de microservicios ha surgido como un estilo arquitectónico que se centra en el diseño y desarrollo de sistemas de software como un conjunto de pequeños servicios independientes. Aunque la arquitectura de microservicios está inspirada en el estilo de Arquitectura Orientada a Servicios (ambas son arquitecturas basadas en servicios), presenta diferencias importantes. Asimismo, la arquitectura de software debe evolucionar a medida que surgen nuevos estilos arquitectónicos y marcos de software, y la evolución de la arquitectura de software se considera una característica central de cualquier sistema de software. De esta manera, este trabajo de investigación presenta una propuesta con un enfoque basado en transformaciones modelo a modelo para evolucionar arquitecturas orientadas a servicios a arquitecturas de microservicios, a partir de un lenguaje de descripción de arquitecturas llamado Sarch. Para lograr esto, el lenguaje Sarch se extendió para permitir el modelado de los dos estilos arquitectónicos desde la vista de componentes y conectores junto con las vistas de modelo de datos y en capas, la vista de descomposición se incluyó en el lenguaje Sarch para permitir la descripción de un sistema desde un punto de vista funcional, y se creó un conjunto de transformaciones modelo a modelo para soportar la evolución de arquitecturas orientadas a servicios a arquitecturas de microservicios.

Palabras clave: Arquitectura de Software, Evolución de Arquitecturas, Arquitecturas Orientadas a Servicios, Arquitecturas de Microservicios, Estilo Arquitectónico, Vista Arquitectónica, Lenguaje de Descripción de Arquitecturas, Ingeniería de Software Dirigida por Modelos.

This Master's Thesis was defended on June 28, 2021, and was evaluated by the following juries:

Felipe Restrepo Calle, Ph.D.
Associate Professor
Departamento de Ingeniería de Sistemas e Industrial
Facultad de Ingeniería
Universidad Nacional de Colombia

José Joaquin Bocanegra García, Ph.D.
Postdoctoral Assistant
Departamento de Ingeniería de Sistemas y Computación
Facultad de Ingeniería
Universidad de los Andes

Contents

List of Figures

List of Tables

1 Introduction

Amazon, Netflix, The Guardian, and other companies have evolved their software systems toward the Microservices Architectural (MSA) style [38], which is an approach to develop a software system composed of small services, where each of the services runs in its own process and the services communicate with each other through lightweight mechanisms (usually through interfaces or HTTP-based APIs) [38]. Service-Oriented Architecture (SOA) is an architectural style in which multiple coarse-grained distributed services collaborate to provide some capabilities [19].

MSA derives from SOA [64] and has emerged as a better architecture approach to overcome the drawbacks of SOA [34] by providing some benefits such as the division into small services with low coupling and high cohesion, the application of agile practices for software development, agile practices in testing and deployment of services, continuous deployment of services, decentralized data administration, decentralized government between services, better scalability and resilience, the use of lightweight protocols for inter-service communication, and greater suitability for containers [38], [54], [69], [32], [53], [37], [34]. MSA and SOA are both architectural styles [62], [43], and they are service-based architectures because they place a strong emphasis on services as the primary components [54]. Using these features, it is possible to design flexible, modular software architectures that are easy to evolve over time [20].

From the point of view of software evolution, the software systems architecture must be redesigned and restructured as new software platforms, technologies, architectural styles and frameworks arise [31], [15], [8], [14], [28]. The evolution of software architectures is a central feature of any software system [8], which motivates the idea of being able to evolve a system from one architectural style to another.

Thereby, this research work proposes an approach to evolve service-based architectures, specifically from SOA to MSA. First, an Architecture Description Language (ADL) is presented, which allows the modeling of service-based architectures, specifically SOA and MSA styles. An ADL is a formal language for representing a software system architecture [19]. To achieve this, the capabilities of Sarch language [66], [67] were extended to allow the modeling of software architectures under the specific characteristics of the SOA and MSA styles using component-and-connector (C&C), data model, layered and decomposition architectural views [19]. Second, a set of model-to-model transformations is proposed in order to define an architectural evolution model from SOA to MSA. The ADL and the model-to-model transformations involve the use of Model-Driven Engineering (MDE) techniques. MDE is a paradigm that uses models as the cornerstone throughout the software engineering process, which leverages modeling languages to describe a software system and model transformations to improve its productivity [13].

1.1 Problem Statement

According to the trend of design and implementation of software systems under the MSA style, software architects should have a model that allows them to evolve to this architectural style, from other architectures, particularly from Service-Oriented Architectures. This is valid considering that the MSA architectural style derives from the SOA architectural style, which makes it possible for these two architectural styles to have comparable elements that allow finding equivalences between the two architectural styles.

Based on the results of the literature search, there are no formal models, methods or techniques that allow transformations between architectural elements of Service-Oriented Architectures and Microservices Architectures.

In response to this problem, the present work strives to create an architectural evolution model from SOA to MSA, starting from an abstraction model for each architectural style, and carrying out a model-to-model transformation process that allows this evolution through a Model-Driven Software Engineering approach.

The research question that is answered with the present work is the following: How can a Service-Oriented Architecture evolve to a Microservices Architecture?

1.2 Objectives

1.2.1 General Objective

To create an evolution model from service-oriented architectures to microservices architectures, through a set of model-to-model transformations.

1.2.2 Specific Objectives

- To do two architectural designs for a software system, used as a case study, based on the elements, relations and properties of the SOA and MSA architectural styles.

- To extend the Sarch language, in order to allow the modeling of software architectures under the specific characteristics of the SOA and MSA styles.
- To create two architectural models in Sarch language, from the architectural designs done and the characteristics of SOA and MSA.
- To create a set of model-to-model transformations that define a model of architectural evolution from SOA to MSA.
- To validate the resulting evolution model against the specified case study.

1.3 Contribution

The contributions of this thesis can be summarized as follows:

- An architecture description language for service-based architectures: in order to allow the modeling of service-based architectures, an architecture description language called Sarch is extended, incorporating in the grammar the specific rules and characteristics of each of the SOA and MSA styles.
- An evolution model for service-based architectures: a set of model-to-model transformations is proposed to define an architectural evolution model between service-based architectures, specifically from SOA to MSA, involving Model-Driven Engineering techniques and the use of architectural views.

Figure **1-1** shows an overview of the thesis proposal, including the Evolution Model for Service-Based Architectures and the Architecture Description Language (ADL) for Service-Based Software Architectures with the selected architectural views.

1.4 Outline of the Thesis

This thesis is organized as follows: Chapter 1 introduces the research work developed in the thesis. Chapter 2 provides background information, mainly for service-based architectures, MDE and architecture evolution. Chapter 3 describes the related work. Chapter 4 presents the architecture description language for service-based architectures. In Chapter 5, the architectural evolution model from SOA to MSA is presented and described. Chapter 6 explains the evaluation of the proposed evolution model. Chapter 7 presents the conclusions and future work.

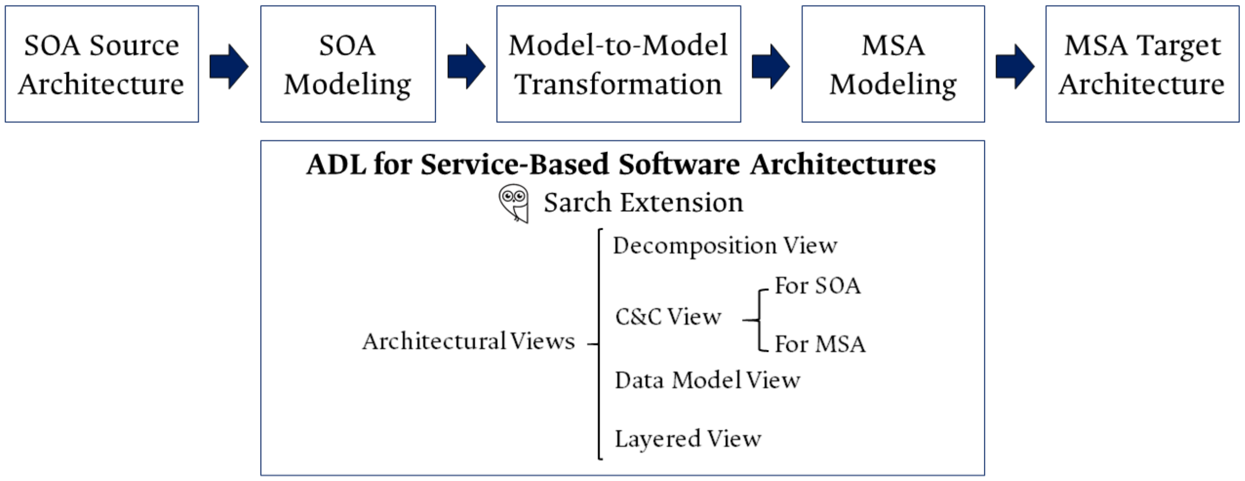


Figure 1-1: Overall graphical representation of the thesis proposal.

2 Background

This chapter gives a general overview around the most important topics that motivate this research work. The topics included in this chapter are Software Architecture, Service-Based Architectures, Model-Driven Engineering, and Architectures Evolution.

2.1 Software Architecture

Software Architecture is defined as a set of structures needed to reason about a software system, the software elements that compose it, the relations between them [57], and their properties [9]. Software architecture is also understood as the set of the main design decisions made on a system [62]. Additionally, the software architecture of a system includes the principles that guide the design of the elements and components that compose it and their evolution [33].

From the interaction of the elements and components that are part of the system, the functionalities and quality attributes emerge (such as security, modifiability, and performance). The larger and more complex the system, the higher the quality attributes, and consequently the software architecture becomes more critical and challenging [19].

Having a software architecture is important for the successful development of a software system. From a technical perspective, software architecture is important for the following reasons [9]:

- Exhibits the quality attributes that must be considered in the software system.
- Contains the most important design decisions that allow to reason about the system and manage changes as system evolves.
- Facilitates communication with key stakeholders.
- Defines a set of constraints that are used in the development of the system, which allows to channel the creativity of the software developers and helps to control the complexity of the system.
- Allows architects and project managers to reason about the cost and duration required to implement the software system.

2.1.1 Architectural Views

The architecture of a software system is complex, and it is difficult to capture all the design decisions at once and represent them in a simple one-dimensional model. There is not a single approach to describe all the aspects of a software architecture [62]. This situation justifies the existence of architectural views to describe and document a software architecture. An architectural view is a representation of a set of system elements and the relationships associated with them [19].

In [19], Clements et al. proposed the Views & Beyond (V&B) Catalog, where views are considered as the fundamental organizing principle for architecture documentation. The V&B Catalog describes three categories of views:

- Module views: allows to document the module structures of a software system.
- Component-and-connector (C&C) views: represents units of execution (called components), and the pathways of interaction and protocols of their interaction (called connectors).
- Allocation views: describes the allocation of software elements (from either a module view or a C&C view) to nonsoftware elements in the environment in which the software is deployed and executed.

The taxonomy of architectural views within the V&B Catalog is shown in Figure **2-1**.

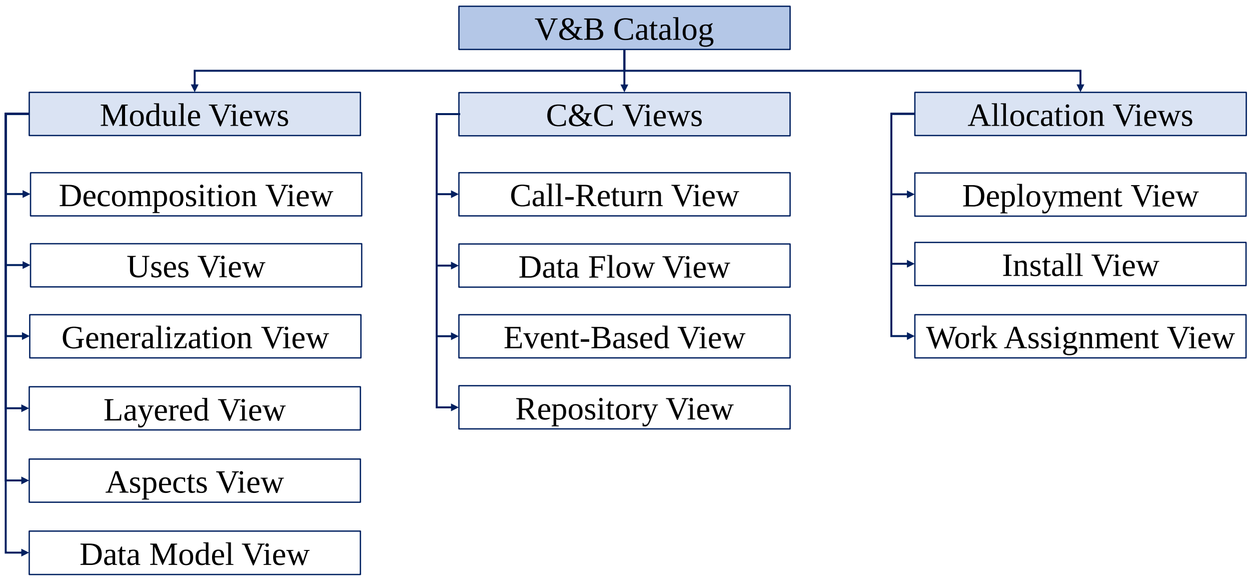


Figure 2-1: Taxonomy of architectural views within the V&B Catalog.

2.1.2 Architectural Styles and Patterns

An architectural style can be defined as a collection of architectural design decisions that are applicable in a given development context, restrict specific architectural decisions to a particular system and context, and promote beneficial qualities in each resulting system [62]. An architectural pattern can be defined as named collection of architectural design decisions that are applicable to a recurring design problem, parameterized to account for different software development contexts in which that problem appears [62].

What is important about an architectural style is that it encapsulates key decisions about architectural elements and emphasizes the constraints of these elements and their relations [52]. The usefulness of an architectural style is that it can be used to restrict architecture and to facilitate cooperation between architects and developers [52]. On the other hand, architectural patterns represent significant reuse of experience and reflect more domain knowledge than architectural styles [62]. Some of the most important architectural styles are Client-Server, Publish-Subscribe (Pub/Sub), Event-Based Architecture, Layered, REpresentational State Transfer (REST), Service-Oriented Architecture (SOA), and Microservices Architecture (MSA) [62], [55]. Some of the most important architectural patterns are State-Logic-Display (Three-Tier), Model-View-Controller (MVC), and Monolithic Architecture [62], [56].

2.1.3 Monolithic Architecture

In the monolithic architecture, all the business functionalities are grouped into a single logical component and built as a single unit [34]. Usually, a monolithic architecture allows the development of different components or modules that are bundled into a single executable unit depending on the programming language or execution platform, where the code is organized in presentation, business logic and data access layers using the layered architectural style (known as layered monolith) [55]. Figure **2-2** shows an example of a monolithic application (or monolith), where all the software modules are packaged into a single unit. The common characteristics of a monolithic application can be summarized as follows [34]:

- Designed, developed, and deployed as a single unit.
- Complex and difficult to maintain and upgrade. This leads to the difficulty of adopting new technologies and frameworks, as all functionalities must be based on homogeneous technologies / frameworks.
- As the monolithic application grows, it may take longer to start up, which adds to the overall cost.

- It is difficult to scale with conflicting resource requirements (some functionalities may require more memory or more CPU).
- One unstable service can cause the entire application to crash.
- It's hard to practice agile development and delivery methodologies, because the application has to be built as a single unit and most of the business capabilities that it offers may not have their own lifecycles.

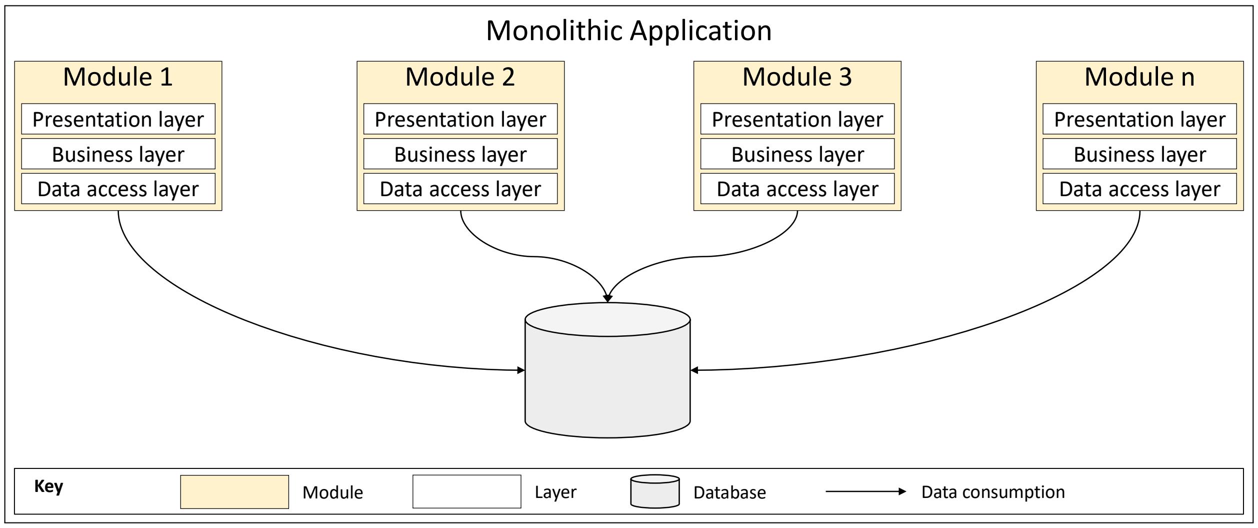


Figure 2-2: Monolithic application example.

2.1.4 Service-Based Architectures

A service is a mechanism that has some business capabilities and provides a well-defined interface to allow access to it [54]. Service-based architectures place a strong emphasis on services as the primary component interfaces used to implement and fulfill functional and non-functional requirements [54].

Both SOA and MSA are considered service-based architectures, which share common characteristics such as the distributed location of their components (distributed architectures) and modularity [54].

2.1.4.1 Service-Oriented Architectures

Service-Oriented Architecture (SOA) is an architectural style in which multiple coarse-grained distributed services (typically known as Web services) collaborate to provide some capabilities. SOA consists of a set of distributed components that provide or consume services. That is, the main components in SOA are service providers and service consumers, which are independent and interoperable [19], [35], [22]. SOA is an abstraction that uses services as basic building blocks to develop distributed systems that are independent of the protocol and the programming language [24].

Another important component in SOA is the Enterprise Service Bus (ESB), which is used to integrate services, data and systems. Consumers use the composite services exposed from the ESB component. Thus, the ESB is used as a centralized bus that connects all these services and systems, where composite services can be built from aggregates of other services. So, the ESB component also contains a significant portion of the business logic of the entire application [34].

Typically, services communicate with each other synchronously or asynchronously. For synchronous communication, SOAP (Simple Object Access Protocol) or REST (REpresentational State Transfer) is used, both being HTTP-based protocols. SOAP is the standard communication protocol in Web services technology, in which service consumers and service providers interact by exchanging request/reply XML messages. In REST, the requests rely on the four basic HTTP verbs (post, get, put, and delete) to indicate the service provider to create, retrieve, update or delete a resource. In asynchronous communication, providers and consumers exchange asynchronous messages, usually through a messaging system [19].

Although SOA tries to approach the challenges of large monolithic applications by segregating the functionalities of monolithic applications into reusable, loosely coupled entities called services, the software systems that provide these services in a SOA ecosystem are still monoliths [34].

2.1.4.2 Microservices Architectures

Microservices Architecture (MSA) is an architectural style used to develop a software system composed of small services, each of which runs in its own process, and communicates with each other through lightweight mechanisms such as HTTP resource API (Application Programming Interface) [38] or a messaging system [34].

Some characteristics of the MSA style are [20]: (i) organization around business capabilities, (ii) automated deployment [69], [16], (iii) intelligence at service access points, and (iv) decentralized control (government) of languages and data [32]. Using these characteristics, it is possible to design flexible, modular architectures that are easy to evolve over time [20].

Key benefits of using the MSA style include the following [20], [51], [6], [26], [34], [56]:

- Heterogeneity of technology: each service can be implemented with different technologies.

- Resilience: since the services are distributed, the system can continue to operate in the event of failure of any of them.
- Scalability: unlike monolithic applications, services can scale computational resources independently in MSA.
- Ease of deployment: changes can be made to specific services and deployed without affecting the rest of the system.
- Decentralized data management: unlike monolithic applications, each microservice may have its own database and database schema which contributes to service independence and loose coupling.
- Business alignment: MSA enables better alignment between software architecture and business compared to monolithic applications.

MSA has emerged as a better architecture approach to overcome the drawbacks of SOA as well as the monolithic application architecture. A monolithic application can be transformed into a microservices architecture by splitting the monolithic application into independent and business functionality oriented services (microservices). Also, the central ESB disappears by breaking its functionalities into each microservice, so that the services take care of the inter-service communication and service composition logic [34]. Because MSA does not include a global mediator like an ESB, it is possible to create coordination services that play the role of orchestrator or mediator, each with responsibility for invoking other microservices to achieve the desired functionality [34], [55], [44].

In MSA, services communicate with each other synchronously or asynchronously. For synchronous communication, HTTP-based REST (REpresentational State Transfer) or gRPC (gRPC Remote Procedure Calls) is used. REST, which is the most common form of microservice communication, uses a navigational scheme to represent objects over a network known as resources, and get (retrieve), put (update), delete, and post (create) are the standard HTTP operations to be performed on the resources. gRPC is an open source remote procedure call system with messages in binary format, and enables the communication between applications built on top of heterogeneous technologies. The asynchronous messaging between microservices is implemented with the use of a lightweight message broker or channel, although it is possible to have a brokerless-based messaging architecture in which services exchange messages directly [34], [56].

MSA uses services as the unit of modularity. A service has an API (Application Programming Interface), which is a boundary of communication with other services and clients [34]. A service's API is a contract between the service and its clients, and consists of operations (which clients can invoke) and events (which are published by the service). An operation has a name, a set of parameters, and a return type. An event has a type and a set of fields and is published to a message channel [56].

Another important component in MSA is the API gateway, which plays the role of a facade and is the entry point into the application from external API clients. It's responsible for request routing, API composition, protocol translation, and other cross-cutting functions (such as authentication, monitoring and rate limiting). An API gateway may provide each client with their own API to meet the requirements of diverse clients (for example, mobile applications or Web applications). Implementing an API gateway with a REST API that supports a diverse set of clients is time consuming; one option is to use a graph-based API language, such as GraphQL, that is designed to support data fetching efficiently for fulfilling queries on existing APIs [56].

Modeling services around a business domain has significant advantages for a microservice architecture, which can be done using Domain-Driven Design (DDD). DDD presents important concepts that aim to better represent a problem domain in software systems, among which Bounded Contexts and Aggregates stand out [44]. A bounded context refers to a cohesive and loosely coupled boundary in an organization, making it ideal for representing a service. A service designed this way is self-contained and only exposes a well-defined interface (API) [43], [54], [34]. An aggregate is a representation of a real domain concept (for example, Order, Customer, Invoice) and may have relationships with other aggregates (for example, a Customer aggregate can be associated with many Orders). A single microservice (and therefore a bounded context) can handle the life cycle and data storage of one or more aggregates [44].

2.1.4.3 Comparisons between SOA and MSA

According to The Open Group [65], which is a consortium that enables the achievement of business goals through information technology standards, MSA is a subset of SOA with additional restrictions about service independence. Both SOA and MSA share the following architectural principles, which facilitate interoperability between the parties:

- Location-independence: there is no preferred location for service consumers or service providers. Services may be located within the system or in systems of different organizations, and, therefore, in different physical locations.
- Implementation-independence: there is no requirement to use specific deployment platforms or technologies that service providers or service consumers must adopt.
- Protocol-independence: SOA or MSA can be built using any available communication protocol, but generally implementations choose a limited set of protocols for the messages that are exchanged.
- Self-contained services: this means that services can be invoked based on the information available in their description, and implies that service consumers are isolated from the

service implementation details.

Table **2-1** presents the differences between MSA and SOA [38], [54], [53], [37], [34].

Table 2-1: Differences between SOA and MSA

SOA	**MSA**
It maximizes the reuse of application services and is based on the concept of sharing as much as possible.	It focuses on decoupling and is based on the concept of sharing as little as possible.
A systematic change requires changing the monolithic application.	A systematic change implies creating a new service.
DevOps and Continuous Delivery are being popular, but they are not an essential part.	It has a strong focus on DevOps and Continuous Delivery.
For communication it usually uses an enterprise service bus (ESB).	For communication it uses simple messaging systems, known as API Layer.
It supports multiple protocols.	It uses lightweight protocols like HTTP and REST.
It uses a common platform to deploy all services.	It can use on-premises servers or cloud platforms.
The use of containers is less popular.	Containers work very well with microservices.
It uses common standards and governance.	The government is decentralized.

2.2 Architectures Evolution

Evolution in software is a process where a software product is continuously updated, maintained, and improved, in order to remain a viable product that evolves over time. As software systems change, it is essential that the software architecture is also updated in such a way that it remains current with these changes [31].

Software evolution capacity is defined as the ability of a software system to adapt to future changes in an easy way. This characteristic of the software is considered fundamental to support strategic decision making, as well as to increase the economic value of software systems. To have long-lived software systems, it is necessary to address their ability to evolve explicitly during their life cycle, including the evolution in the software architecture of these systems [15].

The architecture of software systems must be redesigned and restructured (that is, evolved) as new market opportunities, platforms, technologies and software frameworks emerge, and the evolution of software architectures is seen as a central feature of all software systems [8]. Nowadays, software architects do not have sufficient techniques to plan the evolution of system architectures, considering different solution routes and best practices in specific problem domains [8]. In most cases, the changes are not so simple as to be carried out in one day, but require the elaboration of evolution plans to modify both the architecture and the implementation of the system, through a series of phases or stages until reaching the target system [8].

There are several models and techniques to describe and represent the evolution of software architectures, which are compiled in Table **2-2**.

Table 2-2: Methods for evolution of software architectures

Method	Description
SAEV (Software Architecture Evolution Model) [58]	Model that helps to describe and manage the evolution of software architectures in a homogeneous way at three levels of abstraction: the meta level, the architecture level and the application level. It fills in the shortcomings of ADLs in terms of evolution.
Evolution Paths [8]	Approach to plan and reason about the evolution of software architecture.
Meta-Evolution Style [31]	It proposes a library that is composed of various styles of architectural evolution that correspond to different domains and that model best practices and knowledge in the evolution of software architectures within these specific domains.
SACCS (Software Architecture Change Characterization Scheme) [71]	It allows architects to characterize the causes and effects of changes to software using different criteria, identifying the characteristics of the changes that will have an impact at the software architecture level.
TranSAT [7]	It incorporates an element to the ADL called Architectural Aspect, which allows the description of new concepts and concerns to integrate them into existing architectures.

2.3 Model-Driven Engineering

Model-Driven Engineering (MDE) can be defined as a set of instruments and guidelines for applying the advantages of modeling to software engineering activities. As a methodology, MDE is comprised of concepts, notations, process and rules, and tools. In MDE, the main concepts are

Models and Transformations [13].

MDE conceives models as first-class citizens in software engineering. Models represent realities for a given purpose, and have become crucial in technical fields such as computer science and computer engineering. From the MDE perspective, everything is a model [13]. Models may play two roles in applying abstraction: reduction (where models only focus on the selected aspects of interest) or mapping (where an original individual is taken as a prototype and then generalized to a model). A model conforms to a metamodel, that is, a metamodel is a model that describes models. A metamodel constitutes the definition of a modeling language [13].

Transformations are a set of rules and operations applied to a model in order to produce another model [13]. They are defined at the metamodel level, and then applied at the model level. Through transformations and as part of the MDE process, models are merged, aligned, refactored, refined, or translated. There are two types of model transformations [13]:

- Model-to-Model (M2M) transformations: in this type of transformation, the input and output parameters are models. Usually, the input is one model and the output is one model. However, there may be situations where the input has more than one model or the output corresponds to more than one model.
- Model-to-Text (M2T) transformations: in this type of transformation, the input is a model and the output is a text string. Some outputs of this type of transformation could be documentation, task lists or source code.

Both models and transformations are represented in some notations, which in MDE is done by using modeling languages. Thus, a modeling language is used for specifying models in textual or graphical representation depending on the capabilities of the language [13]. A modeling language contains the structure, terms, notations, syntax, semantics, and rules that are used to express a model [59]. A Domain-Specific Language (DSL) is a kind of modeling language of limited expressiveness focused on a particular domain [25]. An ADL is a domain-specific language for representing a software system architecture [42], [19]

Figure **2-3** summarizes some of the concepts, notations, process and rules, and tools that are present in the MDE methodology [13].

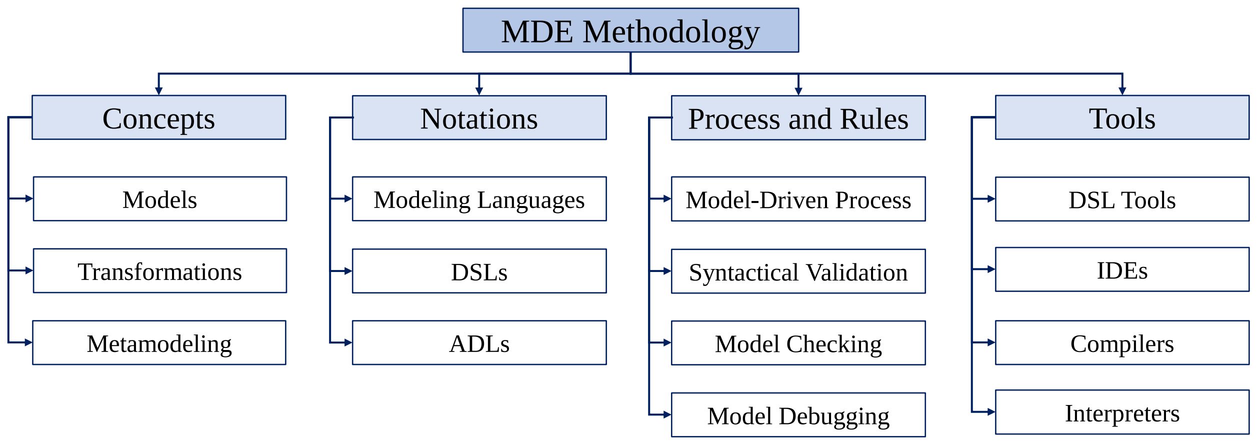


Figure 2-3: Concepts, notations, process and rules, and tools in MDE.

3 Related Work

This chapter presents a brief state of the art related to DSLs and ADLs for describing microservices architectures, DSLs and ADLs for describing service-oriented architectures, and research works on evolution, migration or transformation towards MSA.

3.1 Reviews about Microservices Architectural Style

Di Francesco et al. present a complete state of the art about microservices architectures, where it is indicated that this architectural style is a trend that is being used in the business world [20], [21]. Alshuqayran et al. present a systematic study on microservices architecture and the most relevant architectural challenges, as well as the diagrams, architectural views, methods and models used to represent microservices architectures [5].

On the other hand, there are some reviews about microservices and their use with another disciplines. For instance, there are state of the art in using microservices architectures alongside DevOps [61], [69], [70]. Also, there are reviews of microservices decomposition strategies, highlighting both Model-Driven Engineering and Domain-Driven Design as approaches to support this task [60].

3.2 ADLs for Microservices Architectures

There are several research works that propose languages for the description of MSA. In [63] the authors proposed MicroBuilder, which is a tool used for the specification of a software architecture that follows REST microservice design principles. MicroBuilder comprises MicroDSL and MicroGenerator modules, where the MicroDSL module provides a domain-specific language used for the specification of REST microservice software architectures, which supports textual and graphical concrete syntaxes.

In [30] the authors presented MicroART, an architecture recovery approach for microservice-based systems. As part of that research work, the authors defined a domain-specific language for representing the architectural key aspects of a microservices-based system.

The work in [50], [49] presented $\mu\sigma$ADL, a textual ADL which is an extension of jADL (a formal ADL defined in [48]). $\mu\sigma$ADL was designed in order to provide to software architects and stakeholders a formal way for describing software systems that follow the MSA style. This language follows the C&C view in order to express the architecture of both static and dynamic software systems.

3.3 ADLs for Service-Oriented Architectures

Different research works and specifications were found that support the description of service-oriented architectures. In [36] the authors presented SOADL, an architecture description language for SOA. SOADL specifies the interfaces, behavior, semantics and quality properties of services, provides mechanisms to model and analyze the dynamic and evolving architecture, and supports service composition. In [68] the authors expanded the capabilities of SOADL by proposing SOADL-EH, a service-oriented software architecture description language supporting exception handling.

π-ADL for SOA was proposed in [47] as a service-oriented architecture description language to enable the formal development of dynamic applications. π-ADL for SOA is a member of the π-ADL [46] family.

In [10] PIM4SOA was presented, which is a platform independent metamodel for SOA. The goal of creating the PIM4SOA metamodel was to define a language that could be used to describe SOA at a platform independent level. The PIM4SOA metamodel identifies four aspects where specific concerns can be addressed: information, service, process and quality of service (QoS).

The Service-oriented architecture Modeling Language (SoaML) specification provides a metamodel and a UML profile for the specification and design of services within a service-oriented architecture [45].

3.4 Sarch Language

Sarch is an ADL whose main goal is to allow the design of software architectures [66], [67]. This language is based on a set of architectural views proposed by Clements et al. in the Views & Beyond (V&B) Catalog [19], which allows to cover the necessary elements to support the design of a software system. The included architectural views in Sarch are Data Model, Component-and-Connector (C&C), Layered, and Deployment. Sarch has the capabilities to allow the description of any software architecture in any architectural style.

Also, Sarch has the ability to support both model-to-model and model-to-text transformation processes [67].

3.5 Evolution, Transformation and Migration to MSA

There are previous research works of evolution, migration and transformation towards microservices architectures. Most of these works refer to the evolution from monolithic applications to microservices-based systems. These research works can be classified as follows:

- Migration from monolithic applications in specific case studies or domains: There are research works which document the real migration experience from monolith applications, in specific business domains ([6], [72], [41], [17]) or in more general purpose software systems ([29]).
- Proposal of methods, tools, processes or techniques of evolution from monolithic applications: Some research works propose methods, techniques or tools that allow the evolution or migration from a monolithic application to a microservices architecture ([39], [40], [18], [4], [30], [23], [27]).
- SOA, Microservices and MDD: The research work in [53] presents the differences between SOA and MSA, the state of the art on the application of Model-Driven Development (MDD) to SOA, and the implications of applying MDD to MSA taking into account metamodels, model transformations and modeling languages.

3.6 Gaps in Previous Work

According to the literature reviewed, no previous work was found that specifically proposes a model of evolution or transformation from SOA to MSA based on the formalization of elements, relations and properties. Furthermore, there is a trend for the use of the MSA style within software development and architecture practices [51], which is based on the fact that large companies have evolved their software systems towards microservices architectures [38].

Regarding the reviewed ADLs, the Sarch language allows the modeling of software systems in any architectural style from the conception of architectural views. Table **3-1** summarizes the comparison of the reviewed languages and supports the choice of Sarch to enable modeling of service-based architectures (SOA and MSA). However, the other languages are an important input to identify the elements, relations and properties that are part of each SOA and MSA architectural styles. The main aspects related in this table are: use of more than one architectural view, support for describing software systems in the SOA style, support for describing software systems in the MSA style, support for describing software systems in both SOA and MSA styles, and support of model-to-text or model-to-model transformations.

Table 3-1: Analysis of ADLs: Languages vs. main aspects.

ADL	**Use of more than one architectural view**	**Support for describing software systems in the SOA style**	**Support for describing software systems in the MSA style**	**Support for describing software systems in both SOA and MSA styles**	**Support of model-to-text or model-to-model transformations**
MicroBuilder			X		X
MicroART			X		X
$\mu\sigma$ADL			X		
SOADL-EH		X			
π-ADL		X			
PIM4SOA		X			
SoaML		X			
Sarch	X	X	X	X	X

In response to the problem posed, it is proposed to create an architectural evolution model that starts from an abstraction model for each of the SOA and MSA architectural styles and, using Model-Driven Engineering techniques, performs a model-to-model transformation process that enables this evolution to be directed. Evolving from SOA to MSA is valid considering that the MSA architectural style was born from the SOA architectural style, which makes it possible for these two architectural styles to have comparable elements that allow finding architectural equivalences and optimizations between these two styles.

4 ADL for Service-Based Architectures

Sarch is an ADL that allows the design of software architectures represented through Data Model, Component-and-Connector (C&C), Layered, and Deployment Architectural Views [66], [67]. Sarch has the capabilities to allow the description of any software architecture in any architectural style.

To define an ADL for the design of service-based architectures, specifically for SOA and MSA, it is proposed to extend the Sarch language, incorporating in the grammar the specific rules and characteristics of each of the SOA and MSA styles. This extension was introduced in [11], in which the main focus was on the C&C architectural view. Also, the Decomposition architectural view was introduced in the Sarch language, which allows the description of a software system organized as modules and submodules and shows responsibilities partitioned across them [19].

In this chapter, the proposed ADL for Service-Based Architectures is explained in detail. To achieve this, it is important to highlight three concepts related to the design of the language [13]:

- Grammar: defines all valid sentences and the structure of a language.
- Extended Backus–Naur Form (EBNF): it is a textual representation of the grammar and syntax rules of a computer language.
- Concrete Syntax Tree (CST): it is a tree data structure that represents the grammar of a computer language.

In summary, the Sarch language was extended as follows:

- The C&C view was adapted to restrict the components and connectors depending on the architectural style (SOA or MSA).
- The Decomposition view was introduced to allow the description of a software system from the functional point of view, without considering the specific architectural style.
- The Data Model view was extended to allow a new data model type for associating with the operations in services.

Additionally, the Layered view is taken into account in this research work, therefore it will be described in this chapter. In the next sections, the design of the ADL for service-based architectures, based on Sarch, is described.

4.1 General Architectural Schema

Sarch allows the definition of a software architecture with a terminal called *architecture*, followed by the software system's name, its author, the architectural style, and the five supported architectural views. This general schema of the Sarch grammar is shown in the listing below (as EBNF notation) and in figures **4-1** and **4-2** (as CSTs).

⟨*Architecture*⟩ ::= '`architecture`' '`{`' ⟨*SoftwareSystem*⟩ ⟨*Author*⟩ ⟨*ArchitecturalStyle*⟩ ⟨*Views*⟩ '`}`'

⟨*SoftwareSystem*⟩ ::= '`software_system`' '`:`' ⟨*Id*⟩ '`;`'

⟨*Author*⟩ ::= '`author`' '`:`' ⟨*Id*⟩ '`;`'

⟨*ArchitecturalStyle*⟩ ::= '`architectural_style`' '`:`' ⟨*ArchitecturalStyleType*⟩ '`;`'

⟨*ArchitecturalStyleType*⟩ := '`soa`' | '`msa`' | ⟨*Id*⟩

⟨*Views*⟩ := '`architectural_views`' '`{`' ⟨*DecompositionView*⟩ ⟨*DataModelView*⟩ ⟨*ComponentAndConnectorView*⟩ ⟨*LayeredView*⟩ ⟨*DeploymentView*⟩ '`}`'

⟨*Id*⟩ ::= {a-zA-Z0-9,_}

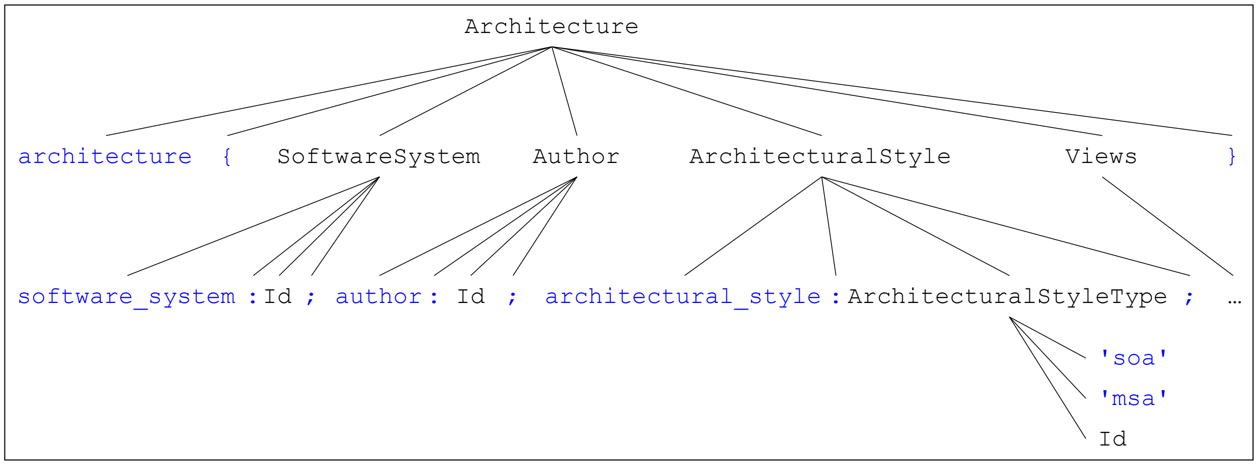


Figure 4-1: CST for Sarch: general schema of the language.

A key feature of the grammar is the *architectural_style*, which triggers the rules that are applicable depending on the specified value as follows:

- If the specified value is *soa*, the language is constrained to the specific rules for a service-oriented architecture.

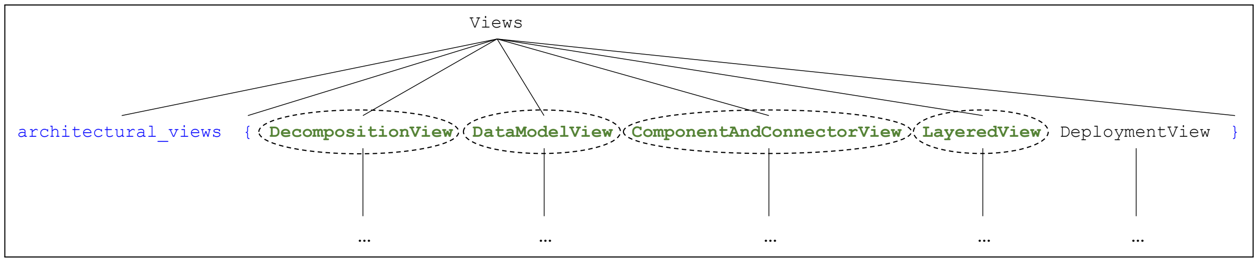


Figure 4-2: CST for the architectural views included in the extended Sarch language.

- If the specified value is *msa*, the language is constrained to the specific rules for a microservices architecture.
- Otherwise, the Sarch language behaves as originally defined and is not constrained to a specific architectural style.

As mentioned previously, the proposed ADL for service-based architectures focuses on the Decomposition, Data Model, C&C, and Layered architectural views in Sarch, which are highlighted in figure **4-2**.

4.2 Decomposition View

This view describes the organization of the software system as modules and submodules and shows how system responsibilities are divided across them [19]. This view is compound of the following elements and relations:

- Elements: **subsystem**, which is a system within a larger system and aggregates modules; **module**, which belongs to a subsystem and encapsulates a set of functionalities; **submodule** which is a module within a larger module; and **functionality**, which specifies what the software system can do. The property of each of these elements is *name*.
- Relations: All decomposition relations are in the form of **is_part_of**; a module is part of a subsystem, a submodule is part of a module, and a functionality is part of a module or a submodule.

The grammar for the decomposition view is shown in the listing below (as EBNF notation) and in figure **4-3** (as a CST).

⟨*DecompositionView*⟩ ::= ‘`decomposition_view`’ ‘`::`’ ⟨*DecompositionElements*⟩ ⟨*DecompositionRelations*⟩ ‘`::`’

⟨*DecompositionElements*⟩ ::= ‘`elements`’ ‘`{`’ ⟨*DecompositionElement*⟩+ ‘`}`’

⟨*DecompositionElement*⟩ := ⟨*Subsystem*⟩ | ⟨*Module*⟩ | ⟨*Submodule*⟩ | ⟨*Functionality*⟩

⟨*Subsystem*⟩ ::= ‘`subsystem`’ ⟨*Id*⟩ ‘`;`’

⟨*Module*⟩ ::= ‘`module`’ ⟨*Id*⟩ ‘`;`’

⟨*Submodule*⟩ ::= ‘`submodule`’ ⟨*Id*⟩ ‘`;`’

⟨*Functionality*⟩ ::= ‘`functionality`’ ⟨*Id*⟩ ‘`;`’

⟨*DecompositionRelations*⟩ ::= ‘`relations`’ ‘`{`’ ⟨*DecompositionRelation*⟩* ‘`}`’

⟨*DecompositionRelation*⟩ := ⟨*IsPartOfA*⟩ | ⟨*IsPartOfB*⟩ | ⟨*IsPartOfC*⟩ | ⟨*IsPartOfD*⟩

⟨*IsPartOfA*⟩ ::= ‘`m:`’ [Module] ‘`is_part_of`’ ‘`ss:`’ [Subsystem] ‘`;`’

⟨*IsPartOfB*⟩ ::= ‘`sm:`’ [Submodule] ‘`is_part_of`’ ‘`m:`’ [Module] ‘`;`’

⟨*IsPartOfC*⟩ ::= ‘`f:`’ [Functionality] ‘`is_part_of`’ ‘`m:`’ [Module] ‘`;`’

⟨*IsPartOfD*⟩ ::= ‘`f:`’ [Functionality] ‘`is_part_of`’ ‘`sm:`’ [Submodule] ‘`;`’

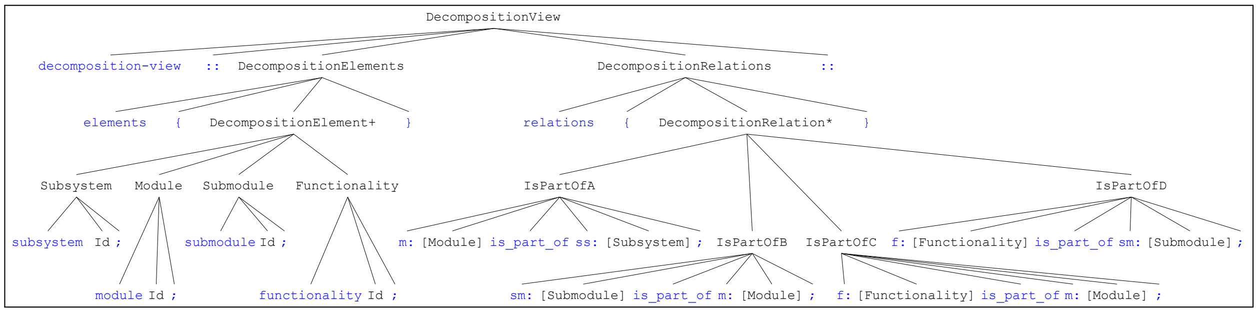


Figure 4-3: CST for the Decomposition view in the extended Sarch language.

4.3 Data Model View

This view describes the static information structure used in the system in terms of data entities and their relationships [19]. This view is compound of the following elements and relations:

- Elements: **data entity**, which is an object that holds information that needs to be stored or represented in a software system. Properties include *name* and data attributes.
- Relations: allow the communication between data entities. The relations can be **generalization/specialization**, logical association (**one-to-one**, **one-to-many**, **many-to-one**, **many-to-many** or **association**), or aggregation (**aggregation** or **composition**).

The grammar of this view allows to define one or more data models for a software system. Each data model has a type (*relational*, *nosql* or *interoperability*), a *name*, and its own data entities and relationships between them. For each data entity a set of attributes can be specified, each one with a *name* and a basic data type (*string*, *byte*, *short*, *int*, *long*, *float*, *double*, *date*, *time*, *datetime*, *bool*, *image*, *video*, *audio*, *file*). Further, it is possible to indicate what modules (defined in the Decomposition view) use which data entities [19].

Also, the grammar allows the definition of a set of operations for the data models of type *interoperability*. Each operation has a *name* and parameters. Each parameter has a *name*, a type (which can be a basic data type or a data entity in the same data model), and a direction (*in* or *out*).

The grammar for the data model view is shown in the listing below (as EBNF notation) and in figure **4-4** (as a CST).

⟨*DataModelView*⟩ ::= ‘`data_model_view`’ ‘`::`’ ⟨*DataModel*⟩+ ‘`::`’

⟨*DataModel*⟩ ::= ⟨*DataModelType*⟩ ‘`data_model`’ ⟨*Id*⟩ ‘`{`’ ⟨*DataModelElements*⟩ ⟨*DataModelRelations*⟩ ⟨*DataModelOperations*⟩ ‘`}`’

⟨*DataModelType*⟩ ::= ‘`relational`’ | ‘`nosql`’ | ‘`interoperability`’

⟨*DataModelElements*⟩ ::= ‘`elements`’ ‘`{`’ ⟨*DataModelElement*⟩+ ‘`}`’

⟨*DataModelElement*⟩ := ⟨*DataEntity*⟩

⟨*DataEntity*⟩ ::= ‘`data_entity`’ ⟨*Id*⟩ ⟨*ModuleRef*⟩? ‘`{`’ ⟨*Attributes*⟩ ‘`}`’

⟨*ModuleRef*⟩ := ‘`(`’ ‘`module`’ [Module] ‘`)`’

⟨*Attributes*⟩ ::= ‘`attributes`’ ‘`{`’ ⟨*Attribute*⟩+ ‘`}`’

⟨*Attribute*⟩ ::= ⟨*DataType*⟩ ⟨*Id*⟩ ‘`;`’

⟨*DataType*⟩ ::= ‘`string`’ | ‘`byte`’ | ‘`short`’ | ‘`int`’ | ‘`long`’ | ‘`float`’ | ‘`double`’ | ‘`date`’ | ‘`time`’ | ‘`datetime`’ | ‘`bool`’ | ‘`image`’ | ‘`video`’ | ‘`audio`’ | ‘`file`’

⟨*DataModelRelations*⟩ ::= ‘`relations`’ ‘`{`’ ⟨*DataModelRelation*⟩* ‘`}`’

⟨*DataModelRelation*⟩ ::= ⟨*RelationType*⟩ ‘`(`’ [DataEntity] ‘`,`’ [DataEntity] ‘`)`’ ‘`;`’

⟨*RelationType*⟩ ::= ⟨*OneToOne*⟩ | ⟨*OneToMany*⟩ | ⟨*ManyToOne*⟩ | ⟨*ManyToMany*⟩ | ⟨*Association*⟩ | ⟨*Aggregation*⟩ | ⟨*Composition*⟩ | ⟨*Generalization*⟩ | ⟨*Specialization*⟩

⟨*OneToOne*⟩ ::= ‘`one_to_one`’

⟨*OneToMany*⟩ ::= ‘`one_to_many`’

⟨*ManyToOne*⟩ ::= ‘`many_to_one`’

⟨*ManyToMany*⟩ ::= ‘`many_to_many`’

⟨*Association*⟩ ::= ‘`association`’

⟨*Aggregation*⟩ ::= ‘`aggregation`’

⟨*Composition*⟩ ::= ‘`composition`’

⟨*Generalization*⟩ ::= ‘`generalization`’

⟨*Specialization*⟩ ::= ‘`specialization`’

⟨*DataModelOperations*⟩ ::= ‘`operations`’ ‘`{`’ ⟨*DataModelOperation*⟩* ‘`}`’

⟨*DataModelOperation*⟩ ::= ‘`operation`’ ⟨*Id*⟩ ‘`{`’ ⟨*Parameters*⟩ ‘`}`’

⟨*Parameters*⟩ ::= ‘`parameters`’ ‘`{`’ ⟨*OperationParameter*⟩* ‘`}`’

⟨*OperationParameter*⟩ ::= ⟨*ParameterType*⟩ ⟨*Array*⟩? ⟨*Id*⟩ ⟨*ParameterDirection*⟩ ‘`;`’

⟨*ParameterType*⟩ ::= ⟨*DataType*⟩ | [DataEntity]

⟨*Array*⟩ ::= ‘`[`’ ‘`]`’

⟨*ParameterDirection*⟩ ::= ‘`in`’ | ‘`out`’

4.4 Component-and-Connector (C&C) View

This view represents elements that have some runtime presence, plus the pathways and protocols of their interaction [19]. In a general form, this view is compound of the following elements and relations:

- Elements: **component**, which represents a processing unit or data store; a component has a *name* and a set of ports through which it interacts with other components (via connectors). And **connector**, which is a pathway of interaction between components; a connector has a *name* and a set of roles that indicate how components can use a connector in interactions.
- Relations: **attachment**, which associates a component port with a connector role to produce a graph of components and connectors.

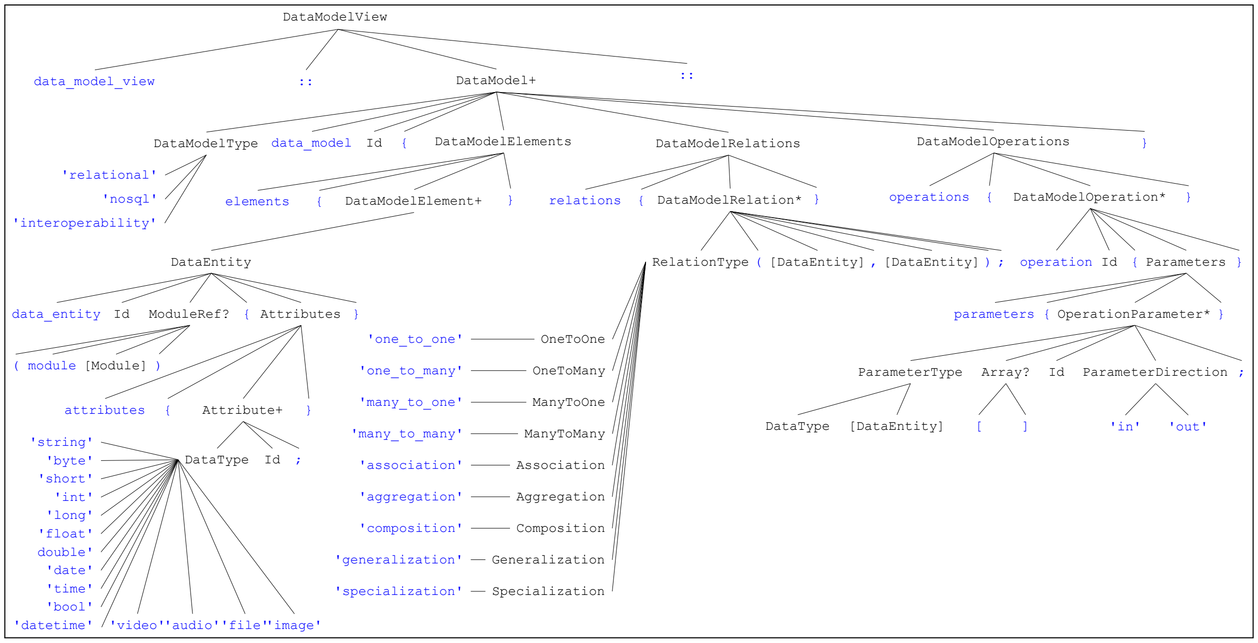

DataModelView
data_model_view
::
DataModel+
::
DataModelType
data_model
Id
{
DataModelElements
DataModelRelations
DataModelOperations
}
'relational'
'nosql'
'interoperability'
elements
{
DataModelElement+
}
relations
{
DataModelRelation*
}
operations
{
DataModelOperation*
}
DataEntity
RelationType ([DataEntity] , [DataEntity]) ;
operation Id { Parameters }
data_entity Id ModuleRef? { Attributes }
parameters { OperationParameter* }
(module [Module])
attributes { Attribute+ }
ParameterType Array? Id ParameterDirection ;
DataType [DataEntity]
[ ]
'in' 'out'
DataType Id ;
'string' 'byte' 'short' 'int' 'long' 'float' double' 'date' 'time' 'bool' 'datetime'
'video' 'audio' 'file' 'image'
'one_to_one' OneToOne
'one_to_many' OneToMany
'many_to_one' ManyToOne
'many_to_many' ManyToMany
'association' Association
'aggregation' Aggregation
'composition' Composition
'generalization' Generalization
'specialization' Specialization

Figure 4-4: CST for the Data Model view in the extended Sarch language.

The general CST for the C&C view is shown in figure **4-5**. In section **elements** of the view, it is possible to define programming languages, orchestration languages (only for SOA style), database systems, and components and connectors that are used in the software system described using Sarch.

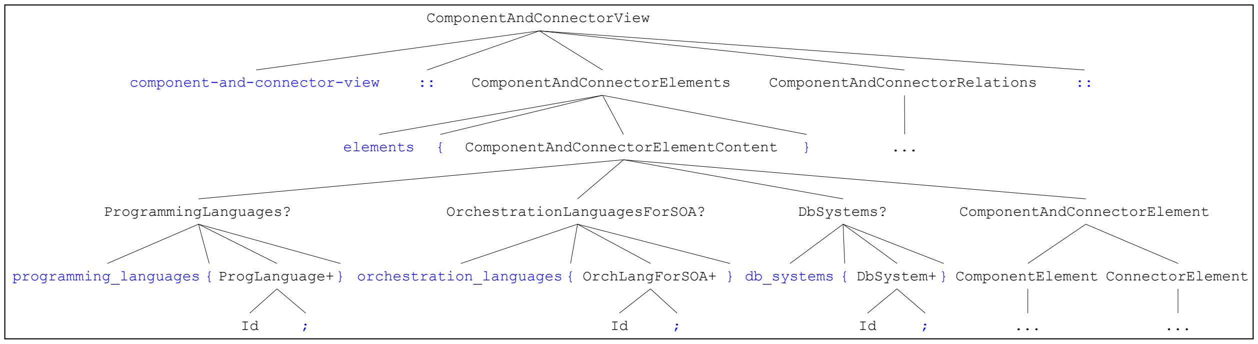

ComponentAndConnectorView
component-and-connector-view
::
ComponentAndConnectorElements
ComponentAndConnectorRelations
::
elements
{
ComponentAndConnectorElementContent
}
...
ProgrammingLanguages?
OrchestrationLanguagesForSOA?
DbSystems?
ComponentAndConnectorElement
programming_languages { ProgLanguage+ }
orchestration_languages { OrchLangForSOA+ }
db_systems { DbSystem+ }
ComponentElement ConnectorElement
Id ;
Id ;
Id ;
...
...

Figure 4-5: CST for the C&C view in the extended Sarch language.

This view presents a set of specific components and connectors that applies to SOA or MSA, depending on the specified architectural style. Table **4-1** summarizes the components and connectors for the C&C view for the MSA and SOA styles, within the extended grammar in Sarch.

Figure **4-6** shows the possible properties for describing a component in Sarch as a CST. Table **4-2** indicates the properties that apply to each component depending on its type for the SOA style, and table **4-3** indicates the properties that apply to each component depending on its type for

Table 4-1: Element summary for C&C view in MSA and SOA styles

Style	**C&C View**	
	Components	**Connectors**
MSA	Microservice	HTTP
	API Gateway	REST
	Web application	gRPC
	Mobile application	GraphQL
	Relational database	Messaging
	NoSQL database	FTP access
	Storage	DB access
SOA	Internal service provider	HTTP
	External service provider	SOAP
	Service consumer	REST
	Enterprise Service Bus (ESB)	Messaging
	Relational database	DB access

the MSA style.

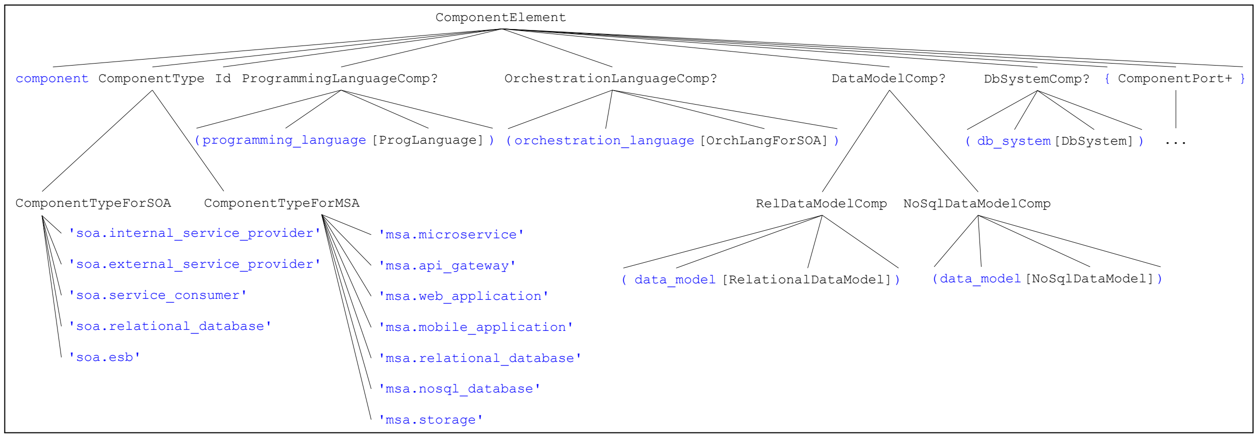


Figure 4-6: CST for the C&C view in the extended Sarch language: Component Element Node.

Each component has ports to indicate the services and accesses required or exposed, which is shown in figure **4-7** as a CST. For each architectural style, there is a predefined set of port types. The port type called *provided_service* has an interoperability data model (defined in the Data Model view) and a coordination type, the latter only for components of type *soa.esb*. The port type called *provided_api* has an interoperability data model. This allows to specify the operations and types of information that each service can provide.

Figure **4-8** shows the possible properties for describing a connector in Sarch as a CST. Each con-

Table 4-2: Properties that apply to each component type in the SOA style

Component type	Programming language	Orchestration language	Relational model	DB system
Internal service provider	X			
External service provider				
Service consumer	X			
ESB		X		
Relational database			X	X

Table 4-3: Properties that apply to each component type in the MSA style

Component type	Programming language	Relational model	NoSQL model	DB system
Microservice	X			
API Gateway				
Web application	X			
Mobile application	X			
Relational database		X		X
NoSQL database			X	X
Storage				

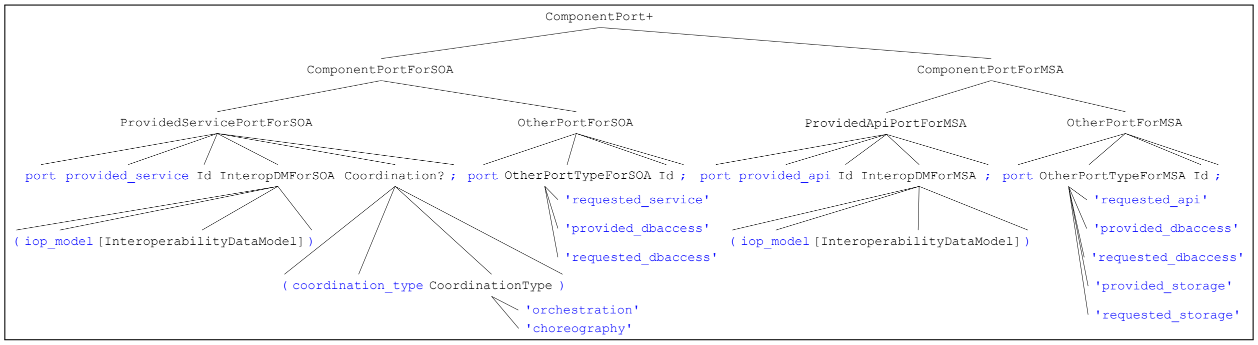

ComponentPort+
ComponentPortForSOA
ComponentPortForMSA
ProvidedServicePortForSOA
OtherPortForSOA
ProvidedApiPortForMSA
OtherPortForMSA
port provided_service Id InteropDMForSOA Coordination? ;
port OtherPortTypeForSOA Id ;
port provided_api Id InteropDMForMSA ;
port OtherPortTypeForMSA Id ;
(iop_model [InteroperabilityDataModel])
'requested_service'
'provided_dbaccess'
'requested_dbaccess'
(iop_model [InteroperabilityDataModel])
'requested_api'
'provided_dbaccess'
'requested_dbaccess'
'provided_storage'
'requested_storage'
(coordination_type CoordinationType)
'orchestration'
'choreography'

Figure 4-7: CST for the C&C view in the extended Sarch language: Component Port Node.

nector has a connector type whose value depends on the specified architectural style, and has two roles called *consumer* and *provider*.

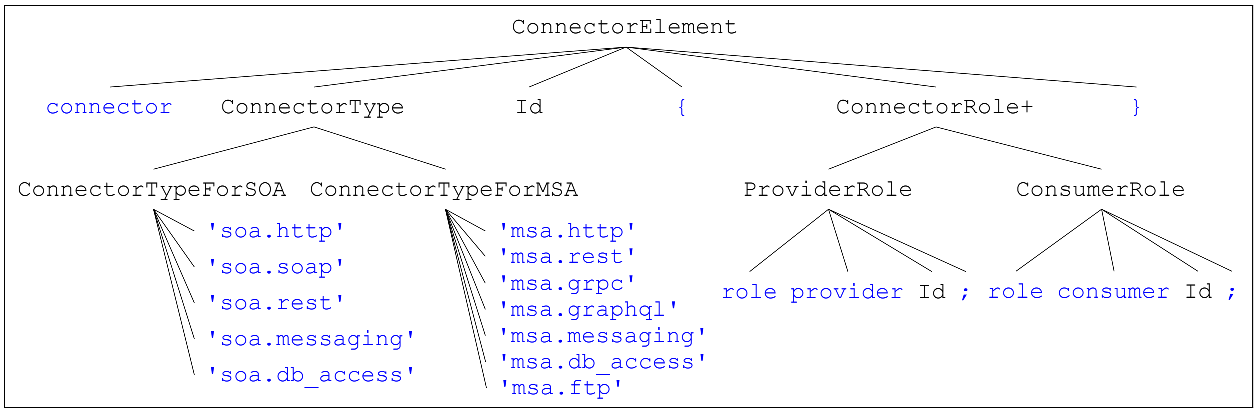


Figure 4-8: CST for the C&C view in the extended Sarch language: Connector Element Node.

The grammar for the supported relations in the C&C view for SOA and MSA styles is shown in Figure **4-9**. Attachments can be made only between compatible component ports and connector roles, which also depends on the component type and the connector type. Table **4-4** contains the attachments allowed for the MSA style and table **4-5** contains the attachments allowed for the SOA style; for simplicity, the connector roles are omitted in these tables because they are always the same (*consumer* attached to the source port and *provider* attached to the target port).

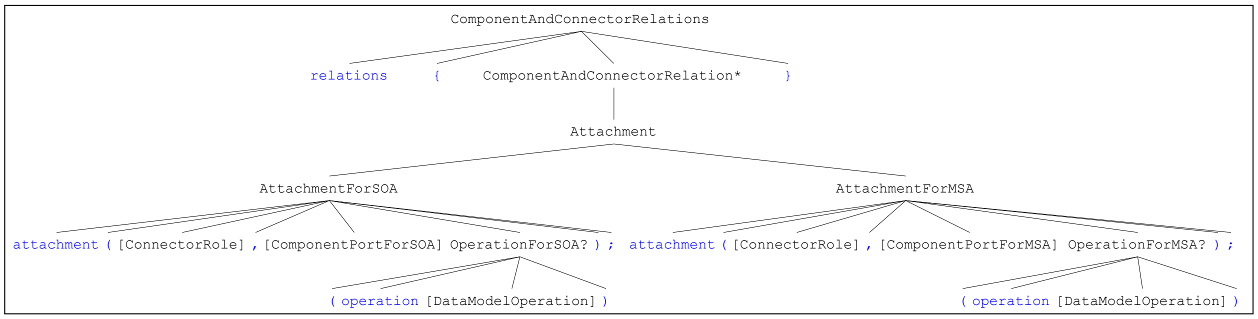

ComponentAndConnectorRelations
relations
{
ComponentAndConnectorRelation*
}
Attachment
AttachmentForSOA
AttachmentForMSA
attachment ([ConnectorRole] , [ComponentPortForSOA] OperationForSOA?) ;
attachment ([ConnectorRole] , [ComponentPortForMSA] OperationForMSA?) ;
(operation [DataModelOperation])
(operation [DataModelOperation])

Figure 4-9: CST for the C&C view in the extended Sarch language: Relations Node.

For both SOA and MSA, it is possible in the attachment to indicate the operation that participates in the relation, which provides a capability for a more granular definition in relations. The specified operation belongs to the interoperability data model associated to the related component port.

The grammar for the C&C view is also shown in the listing below as EBNF notation.

Table 4-4: Valid attachments for the MSA style

Source component	**Source port**	**Connector**	**Target component**	**Target port**
web_application / mobile_-application	requested_api	http / rest / graphql	api_gateway	provided_api
api_gateway / microservice	requested_api	http / rest / grpc / messaging	microservice	provided_api
microservice	requested_-dbaccess	db_access	relational_-database	provided_-dbaccess
microservice	requested_-dbaccess	db_access	nosql_database	provided_-dbaccess
microservice	requested_-storage	ftp	storage	provided_storage

Table 4-5: Valid attachments for the SOA style

Source component	**Source port**	**connector**	**Target component**	**Target port**
service_consumer	requested_-service	http / soap / rest	internal_-service_provider / external_-service_provider / esb	provided_service
service_consumer	requested_-dbaccess	db_access	relational_-database	provided_-dbaccess
esb	requested_-service	http / soap / rest / messaging	internal_-service_provider / external_-service_provider	provided_service
internal_service_-provider	requested_-service	http / soap / rest / messaging	internal_-service_provider	provided_service
internal_service_-provider	requested_-dbaccess	db_access	relational_-database	provided_-dbaccess

⟨*ComponentAndConnectorView*⟩ ::= ‘`component_and_connector_view`’ ‘`::`’ ⟨*ComponentAndConnectorElements*⟩ ⟨*ComponentAndConnectorRelations*⟩ ‘`::`’

⟨*ComponentAndConnectorElements*⟩ ::= ‘`elements`’ ‘`{`’ ⟨*ProgrammingLanguages*⟩? ⟨*OrchestrationLanguagesForSOA*⟩? ⟨*DbSystems*⟩? ⟨*ComponentAndConnectorElement*⟩ ‘`}`’

⟨*ProgrammingLanguages*⟩ := ‘`programming_languages`’ ‘`{`’ ⟨*ProgLanguage*⟩+ ‘`}`’

⟨*ProgLanguage*⟩ := ⟨*Id*⟩ ‘`;`’

⟨*OrchestrationLanguagesForSOA*⟩ := ‘`orchestration_languages`’ ‘`{`’ ⟨*OrchLangForSOA*⟩+ ‘`}`’

⟨*OrchLangForSOA*⟩ := ⟨*Id*⟩ ‘`;`’

⟨*DbSystems*⟩ := ‘`db_systems`’ ‘`{`’ ⟨*DbSystem*⟩+ ‘`}`’

⟨*DbSystem*⟩ := ⟨*Id*⟩ ‘`;`’

⟨*ComponentAndConnectorElement*⟩ := ⟨*ComponentElement*⟩ | ⟨*ConnectorElement*⟩

⟨*ComponentElement*⟩ := ‘`component`’ ⟨*ComponentType*⟩ ⟨*ProgrammingLanguageComp*⟩? ⟨*OrchestrationLanguageComp*⟩? ⟨*DataModelComp*⟩? ⟨*DbSystemComp*⟩? ‘`{`’ ⟨*ComponentPort*⟩+ ‘`}`’

⟨*ComponentType*⟩ := ⟨*ComponentTypeForSOA*⟩ | ⟨*ComponentTypeForMSA*⟩

⟨*ComponentTypeForSOA*⟩ := ‘`soa.internal_service_provider`’ | ‘`soa.external_service_provider`’ | ‘`soa.service_consumer`’ | ‘`soa.relational_database`’ | ‘`soa.esb`’

⟨*ComponentTypeForMSA*⟩ := ‘`msa.microservice`’ | ‘`msa.api_gateway`’ | ‘`msa.web_application`’ | ‘`msa.mobile_application`’ | ‘`msa.relational_database`’ | ‘`msa.nosql_database`’ | ‘`msa.storage`’

⟨*ProgrammingLanguageComp*⟩ := ‘`(`’ ‘`programming_language`’ [ProgLanguage] ‘`)`’

⟨*OrchestrationLanguageComp*⟩ := ‘`(`’ ‘`orchestration_language`’ [OrchLangForSOA] ‘`)`’

⟨*DataModelComp*⟩ := ⟨*RelDataModelComp*⟩ | ⟨*NoSqlDataModelComp*⟩

⟨*RelDataModelComp*⟩ := ‘`(`’ ‘`data_model`’ [RelationalDataModel] ‘`)`’

⟨*NoSqlDataModelComp*⟩ := ‘`(`’ ‘`data_model`’ [NoSqlDataModel] ‘`)`’

⟨*DbSystemComp*⟩ := ‘`(`’ ‘`db_system`’ [DbSystem] ‘`)`’

⟨*ComponentPort*⟩ := ⟨*ComponentPortForSOA*⟩ | ⟨*ComponentPortForMSA*⟩

⟨*ComponentPortForSOA*⟩ := ⟨*ProvidedServicePortForSOA*⟩ | ⟨*OtherPortForSOA*⟩

⟨*ProvidedServicePortForSOA*⟩ := ‘`port`’ ‘`provided_service`’ ⟨*Id*⟩ ⟨*InteropDMForSOA*⟩ ⟨*Coordination*⟩? ‘`;`’

⟨*InteropDMForSOA*⟩ := ‘`(`’ ‘`iop_model`’ [InteroperabilityDataModel] ‘`)`’

⟨*Coordination*⟩ := ‘`(`’ ‘`coordination_type`’ ⟨*CoordinationTypeForSOA*⟩ ‘`)`’

⟨*CoordinationTypeForSOA*⟩ := ‘`orchestration`’ | ‘`choreography`’

⟨*OtherPortForSOA*⟩ := ‘`port`’ ⟨*OtherPortTypeForSOA*⟩ ⟨*Id*⟩ ‘`;`’

⟨*OtherPortTypeForSOA*⟩ := ‘`requested_service`’ | ‘`provided_dbaccess`’ | ‘`requested_dbaccess`’

⟨*ComponentPortForMSA*⟩ := ⟨*ProvidedApiPortForMSA*⟩ | ⟨*OtherPortForMSA*⟩

⟨*ProvidedApiPortForMSA*⟩ := ‘`port`’ ‘`provided_api`’ ⟨*Id*⟩ ⟨*InteropDMForMSA*⟩ ‘`;`’

⟨*InteropDMForMSA*⟩ := ‘`(`’ ‘`iop_model`’ [InteroperabilityDataModel] ‘`)`’

⟨*OtherPortForMSA*⟩ := ‘`port`’ ⟨*OtherPortTypeForMSA*⟩ ⟨*Id*⟩ ‘`;`’

⟨*OtherPortTypeForMSA*⟩ := ‘`requested_api`’ | ‘`provided_dbaccess`’ | ‘`requested_dbaccess`’ | ‘`provided_storage`’ | ‘`requested_storage`’

⟨*ConnectorElement*⟩ := ‘`connector`’ ⟨*ConnectorType*⟩ ⟨*Id*⟩ ‘`{`’ ⟨*ConnectorRole*⟩+ ‘`}`’

⟨*ConnectorType*⟩ := ⟨*ConnectorTypeForSOA*⟩ | ⟨*ConnectorTypeForMSA*⟩

⟨*ConnectorTypeForSOA*⟩ := ‘`soa.http`’ | ‘`soa.soap`’ | ‘`soa.rest`’ | ‘`soa.messaging`’ | ‘`soa.db_access`’

⟨*ConnectorTypeForMSA*⟩ := ‘`msa.http`’ | ‘`msa.rest`’ | ‘`msa.grpc`’ | ‘`msa.graphql`’ | ‘`msa.messaging`’ | ‘`msa.db_access`’ | ‘`msa.ftp`’

⟨*ConnectorRole*⟩ := ⟨*ProviderRole*⟩ | ⟨*ConsumerRole*⟩

⟨*ProviderRole*⟩ := ‘`role`’ ‘`provider`’ ⟨*Id*⟩ ‘`;`’

⟨*ConsumerRole*⟩ := ‘`role`’ ‘`consumer`’ ⟨*Id*⟩ ‘`;`’

⟨*ComponentAndConnectorRelations*⟩ := ‘`relations`’ ‘`{`’ ⟨*ComponentAndConnectorRelation*⟩* ‘`}`’

⟨*ComponentAndConnectorRelation*⟩ := ⟨*Attachment*⟩

⟨*Attachment*⟩ := ⟨*AttachmentForSOA*⟩ | ⟨*AttachmentForMSA*⟩

⟨*AttachmentForSOA*⟩ := ‘`attachment`’ ‘`(`’ [ConnectorRole] ‘`,`’ [ComponentPortForSOA] ⟨*OperationForSOA*⟩? ‘`)`’ ‘`;`’

⟨*OperationForSOA*⟩ := ‘(’ ‘`operation`’ [DataModelOperation] ‘)’

⟨*AttachmentForMSA*⟩ := ‘`attachment`’ ‘(’ [ConnectorRole] ‘`,`’ [ComponentPortForMSA] ⟨*OperationForMSA*⟩? ‘)’ ‘;’

⟨*OperationForMSA*⟩ := ‘(’ ‘`operation`’ [DataModelOperation] ‘)’

4.5 Layered View

This view describes a division of the software into units called layers. Each layer represents a group of modules that offer a cohesive set of services [19]. Also, this view includes the concept of tier that allows grouping components; usually, this grouping is given for components that share the same runtime environment or have the same runtime purpose [19].

This view is compound of the following elements and relations:

- Elements: **layer**, which performs a specific role within an application and provides a set of services to other layers. **layer-segment**, which allows to divide a layer into a finer-grained way; that is, a layer can be composed of layer segments. And **tier**, which allows to group components that share the same runtime environment.

- Relations: **allowed-to-use**, which allows communication between different layers, between different layer segments, or between different tiers. The relation named **allowed-to-use-below** is similar to **allowed-to-use**, but is more specific in the way that one element relates to another element located just below it. The relation named **contains** has three uses; the first is to represent which layer segments belongs to a layer; the second allows to indicate which layers are applicable for specific components, that is, it allows to describe for each component element (defined in the C&C view) which layers it contains; the third allows to specify for each tier which components is contains.

The general CST for the Layered view is shown in figure **4-10**. Figures **4-11**, **4-12** and **4-13** show the details of relations of type **allowed-to-use**, **allowed-to-use-below** and **contains**, respectively.

The grammar for the Layered view is shown in the listing below as EBNF notation.

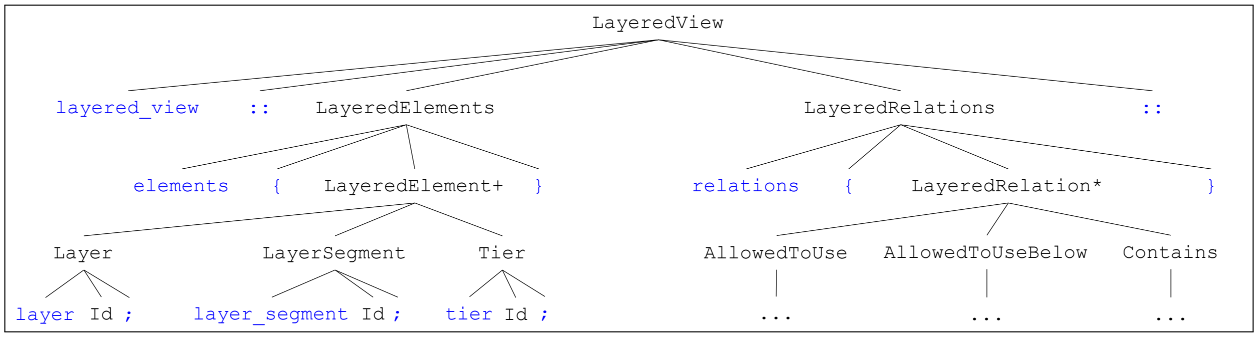


Figure 4-10: CST for the Layered view in the extended Sarch language.

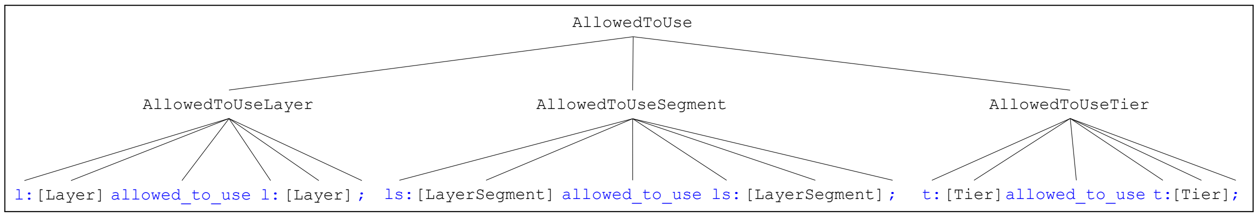

AllowedToUse
AllowedToUseLayer
AllowedToUseSegment
AllowedToUseTier
l: [Layer] allowed_to_use l: [Layer] ;
ls: [LayerSegment] allowed_to_use ls: [LayerSegment] ;
t: [Tier] allowed_to_use t:[Tier];

Figure 4-11: CST for the Layered view in the extended Sarch language: **allowed-to-use** relation.

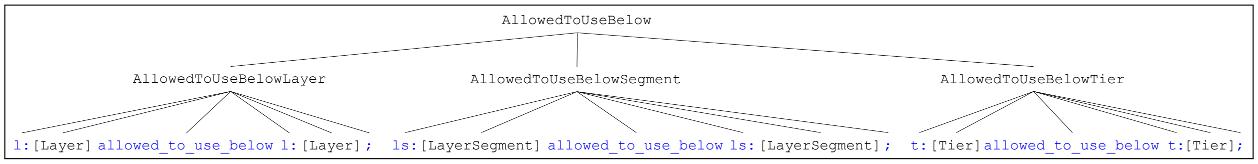

AllowedToUseBelow
AllowedToUseBelowLayer
AllowedToUseBelowSegment
AllowedToUseBelowTier
l: [Layer] allowed_to_use_below l: [Layer] ;
ls: [LayerSegment] allowed_to_use_below ls: [LayerSegment] ;
t: [Tier] allowed_to_use_below t:[Tier];

Figure 4-12: CST for the Layered view in the extended Sarch language: **allowed-to-use-below** relation.

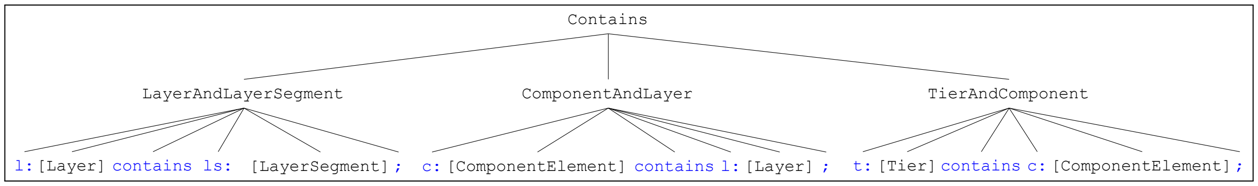

Contains
LayerAndLayerSegment
ComponentAndLayer
TierAndComponent
l: [Layer] contains ls: [LayerSegment] ;
c: [ComponentElement] contains l: [Layer] ;
t: [Tier] contains c: [ComponentElement] ;

Figure 4-13: CST for the Layered view in the extended Sarch language: **contains** relation.

⟨*LayeredView*⟩ ::= ‘`layered_view`’ ‘`::`’ ⟨*LayeredElements*⟩ ⟨*LayeredRelations*⟩ ‘`::`’

⟨*LayeredElements*⟩ ::= ‘`elements`’ ‘`{`’ ⟨*LayeredElement*⟩+ ‘`}`’

⟨*LayeredElement*⟩ := ⟨*Layer*⟩ | ⟨*LayerSegment*⟩ | ⟨*Tier*⟩

⟨*Layer*⟩ ::= ‘`layer`’ ⟨*Id*⟩ ‘`;`’

⟨*LayerSegment*⟩ ::= ‘`layer_segment`’ ⟨*Id*⟩ ‘`;`’

⟨*Tier*⟩ ::= ‘`tier`’ ⟨*Id*⟩ ‘`;`’

⟨*LayeredRelations*⟩ ::= ‘`relations`’ ‘`{`’ ⟨*LayeredRelation*⟩* ‘`}`’

⟨*LayeredRelation*⟩ := ⟨*AllowedToUse*⟩ | ⟨*AllowedToUseBelow*⟩ | ⟨*Contains*⟩

⟨*AllowedToUse*⟩ := ⟨*AllowedToUseLayer*⟩ | ⟨*AllowedToUseSegment*⟩ | ⟨*AllowedToUseTier*⟩

⟨*AllowedToUseLayer*⟩ ::= ‘`l:`’ [Layer] ‘`allowed_to_use`’ ‘`l:`’ [Layer] ‘`;`’

⟨*AllowedToUseSegment*⟩ ::= ‘`ls:`’ [LayerSegment] ‘`allowed_to_use`’ ‘`ls:`’ [LayerSegment] ‘`;`’

⟨*AllowedToUseTier*⟩ ::= ‘`t:`’ [Tier] ‘`allowed_to_use`’ ‘`t:`’ [Tier] ‘`;`’

⟨*AllowedToUseBelow*⟩ := ⟨*AllowedToUseBelowLayer*⟩ | ⟨*AllowedToUseBelowSegment*⟩ | ⟨*AllowedToUseBelowTier*⟩

⟨*AllowedToUseBelowLayer*⟩ ::= ‘`l:`’ [Layer] ‘`allowed_to_use_below`’ ‘`l:`’ [Layer] ‘`;`’

⟨*AllowedToUseBelowSegment*⟩ ::= ‘`ls:`’ [LayerSegment] ‘`allowed_to_use_below`’ ‘`ls:`’ [LayerSegment] ‘`;`’

⟨*AllowedToUseBelowTier*⟩ ::= ‘`t:`’ [Tier] ‘`allowed_to_use_below`’ ‘`t:`’ [Tier] ‘`;`’

⟨*Contains*⟩ := ⟨*LayerAndLayerSegment*⟩ | ⟨*ComponentAndLayer*⟩ | ⟨*TierAndComponent*⟩

⟨*LayerAndLayerSegment*⟩ ::= ‘`l:`’ [Layer] ‘`contains`’ ‘`ls:`’ [LayerSegment] ‘`;`’

⟨*ComponentAndLayer*⟩ ::= ‘`c:`’ [ComponentElement] ‘`contains`’ ‘`l:`’ [Layer] ‘`;`’

⟨*TierAndComponent*⟩ ::= ‘`t:`’ [Tier] ‘`contains`’ ‘`c:`’ [ComponentElement] ‘`;`’

5 Architectural Evolution Model

This chapter presents the architectural evolution model for service-based architectures, specifically from service-oriented architectures to microservices architectures. Also, it presents the implementation of the ADL for service-based architectures and the model-to-model transformations from SOA to MSA in the Sarch-Studio tool.

5.1 Architectural Evolution Model Overview

The proposed evolution model for service-based architectures includes the extended Sarch grammar described previously (the ADL for service-based architectures) and performs a set of model-to-model transformations based on the grammar for SOA and MSA. These model-to-model transformations go from a SOA model to an MSA model, where both models are described using the extended Sarch language.

In [11] an evolution model for service-based architectures was introduced, in which the main focus was on the C&C architectural view for SOA and MSA. However, the present research work proposes an evolution model in greater depth and uses other architectural views of the extended Sarch language.

As illustrated in figure **1-1**, the evolution model for service-based architectures consists of the following steps:

- The architecture of a software system designed under the SOA style is taken as input.
- A functional decomposition of the software system is carried out using the Decomposition view in Sarch.
- The SOA-based system is modeled using Sarch, specifically with the C&C, Data Model, and Layered views.
- A set of model-to-model transformations are applied for the SOA-based system described in Sarch, generating a model that conforms to the grammar defined in Sarch for the MSA style.
- The resulting model would be used as a starting point to represent the software system using the MSA style.

5.2 Model-to-Model Transformations from SOA to MSA

There are key aspects of the Sarch language that are used in the model-to-model transformations from SOA to MSA, which are listed as follows:

- From a functional point of view, a module in the Decomposition view can be seen as a bounded context used in domain-driven design, which groups a set of functionalities from a business domain perspective.
- According to the previous item, and taking into account that it is a good practice to represent a microservice as a bounded context, it can be established that a microservice can contain the functionality of a module defined in the Decomposition view.
- A data entity in the Data Model view can belong to a module in the Decomposition view. Therefore, a data entity can be seen as an aggregate that belongs to a bounded context (that is, a microservice). As stated previously, a microservice can handle the lifecycle of aggregates, which can be represented in its API with operations to create, read, update and delete these domain objects.
- Each service in SOA is represented by a port, which contains a set of cohesive and functionally related operations to meet a business need. For this reason, each port of type *provided_service* of components of type *internal_service_provider* in SOA is a good candidate to become a microservice. Additionally, component ports play an important role in distributed architectures such as SOA and MSA that allow communication between them, and they become a common point between the two architectural styles.

For the proposed evolution model, elements and relations are taken from the Decomposition view, the Data Model view, the C&C view, and the Layered view. Specifically, a transformation function is proposed that receives an element or relation that belongs with the SOA architectural style and an element from the Decomposition view, and returns a list of elements or relations that belong with the MSA architectural style.

Furthermore, the proposed transformation function has the following assumptions regarding the SOA architectural style:

- Service consumers and internal service providers, which are component types in the C&C view, are monolithic applications.
- Each monolithic application is compound of the presentation, business logic, and data access layers.
- A monolithic application in the C&C view is equivalent to a subsystem in the decomposition view.

A high-level transformation function was introduced in [11], with emphasis in the elements and relations described in the C&C view and the Decomposition view. Algorithm 1 provides the definition of the model-to-model transformation function for service-based architectures in a rigorous and formal way, specifically to transform a software system designed in the SOA architectural style into a software system designed under the MSA architectural style, using the Decomposition, Data Model, C&C and Layered views as part of the Sarch grammar.

To achieve this, a formal and rigorous analysis was done of the elements, relations and properties of each architectural style and a formal mapping between the SOA elements and the MSA elements, so that from the concept of component it can be done an architecturally clean transformation. This ensures that the characteristics of the service-based architectures are being used to do a transformation in two different architectural styles but that they start from the concept of services and that it is leveraged in terms of the characteristics and properties of the components.

Additionally, the modules in the decomposition view play a vital role in the proposed transformation process by allowing the generation of a microservice for each module and covering the full functionality of the system. Ports and roles are also important in the transformation process, which allow to connect the components in distributed architectures such as SOA and MSA, and also each port that provides a service in SOA is transformed into a microservice in MSA to meet the needs of interaction between components.

The model-to-model transformation process receives the SOA model described in Sarch, as well as the following input parameters:

- **database_system**, which represents the default database system for the database components that will be generated.
- **database_type**, which represents the default database type (*msa.relational_database* or *msa.nosql_-database*) for the database components that will be generated.
- **ms_programming_language**, which represents the default programming language for the microservice components that will be generated.
- **web_application_name**, which is the name of the web application component that will be generated.
- **web_programming_language**, which represents the default programming language for the web application component that will be generated.

Based on the previous definitions and assumptions, as well as on the specific characteristics of the SOA and MSA architectural styles that can be represented in the ADL for service-based architectures (Sarch), the following are the model-to-model transformations to evolve from a SOA model to a MSA model:

Algorithm 1 SOA to MSA Transformation Algorithm

```
# Overall Declarations
  $e \leftarrow architecturalElement$        $sm \leftarrow submodule$        $MSAComponents \leftarrow C$
  $r \leftarrow architecturalRelation$        $f \leftarrow functionality$        $MSAConnectors \leftarrow D$
  $p \leftarrow architecturalProperty$        $dm \leftarrow dataModel$        $relationalDataModels \leftarrow E$
  $cp \leftarrow component$        $de \leftarrow dataEntity$        $noSQLDataModels \leftarrow F$
  $cn \leftarrow connector$        $architecturalStyle \leftarrow \{cp, cn, dm\}$        $interopDataModels \leftarrow G$
  $ss \leftarrow subsystem$        $SOAComponents \leftarrow A$
  $m \leftarrow module$        $SOAConnectors \leftarrow B$
  $monolith \leftarrow \{internalServiceProvider, serviceConsumer\}$
  $monolith = dataAccess + businessLogic + presentation$
  $SOA \leftarrow \{cp, cn, dm : cp \in A, cn \in B\ dm \in E \vee dm \in G\}$
  $MSA \leftarrow \{cp, cn, dm : cp \in C, cn \in D\ dm \in E \vee dm \in F \vee dm \in G\}$
  $architecturalView \leftarrow \{e, r, p\}$
  $c\&cView \leftarrow \{e, r, p : e = cp \vee e = cn, r = attachment, p = c\&cProperty\}$
  $c\&cViewForSOA \leftarrow \{e, r, p : e = cp \vee e = cn, r = attachment, p = c\&cProperty; cp \in A, cn \in B\}$
  $c\&cViewForMSA \leftarrow \{e, r, p : e = cp \vee e = cn, r = attachment, p = c\&cProperty; cp \in C, cn \in D\}$
  $decompositionView \leftarrow \{e, r, p : e = ss \vee e = m \vee e = sm \vee e = f, r = isPartOf, p = decompositionProperty\}$
  $dataModelView \leftarrow \{e, r, p : e = de, r = dataModelRelation, p = dataModelProperty\}$
# Transformation Function
  $F\colon SOA \rightarrow MSA$
  $F(x, z, w) = y; \{x \in SOA, y \in MSA, x \in c\&cViewForSOA, y \in c\&cViewForMSA, z \in decompositionView\}$
  function F($monolith, subsystem, SOADataModels$)
      $y \leftarrow \varnothing$
      $y.microservices \leftarrow \varnothing$
      $y.attachments \leftarrow \varnothing$
      $y.databases \leftarrow \varnothing$
      $y.datamodels \leftarrow \varnothing$
      $ms \leftarrow \varnothing$
      for $m \in subsystem.module\_set$ do
          $ms \Longleftarrow (monolith.dataAccess, monolith.businessLogic, m)$
          $y.microservices \Longleftarrow y.microservices \cup ms$
          $dm \Longleftarrow (SOADataModels, m)$
          $db \Longleftarrow (database, m)$
          $y.databases \leftarrow y.databases \cup db$
          $y.datamodels \leftarrow y.datamodels \cup dm$
          $y.attachments \leftarrow y.attachments \cup attach(\text{ms}, db)$
      $y.apiGateway \Longleftarrow (monolith.businessLogic, subsystem)$
      $y.attachments \leftarrow y.attachments \cup attach(y.apiGateway, y.microservices)$
      $y.webApplication \Longleftarrow (monolith.presentation, subsystem)$
      $y.attachments \leftarrow y.attachments \cup attach(y.webApplication, y.apiGateway)$
      return $y$
  function F($esb, subsystem$)
      $y \leftarrow \varnothing$
      $y.apiGateway \Longleftarrow esb$
      return $y$
  function F($storage, subsystem$)
      $y \leftarrow \varnothing$
      $y.storage \Longleftarrow storage$
      return $y$
  function F($soapConnector, subsystem$)
      $y \leftarrow \varnothing$
      $y.restConnector \Longleftarrow soapConnector$
      return $y$
  function F($ftpConnector, subsystem$)
      $y \leftarrow \varnothing$
      $y.ftpConnector \Longleftarrow ftpConnector$
      return $y$
```

- Preserve the Decomposition view because it represents the functional decomposition of the software system regardless of the architectural style.
- Generate the Layered view that includes the following:
  - Generate the presentation, business logic and data access layers.
  - Generate a relation of type *allowed_to_use* from the presentation layer to the business logic layer, and from the business logic layer to the data access layer.
  - Generate the presentation, logic and data tiers.
  - Generate a relation of type *allowed_to_use* from the presentation tier to the logic tier, and from the logic tier to the data tier.
- Generate the Data Model view that includes the following:
  - Generate a data model of type *interoperability* for each set of data entities that belong to each module. In each of these generated data models, generate a set of operations (create, read, update and delete) for each data entity to allow managing its lifecycle.
  - Generate a data model of type *relational* or *nosql* for each set of data entities that belong to each module, according to the input parameter **database_type**.
- Generate the C&C view that includes the following:
  - Include the programming languages specified in the input parameters **ms_programming_-language** and **web_programming_language**.
  - Include the database system specified in the input parameter **database_system**.
  - Generate a component of type *msa.microservice* for each module of each subsystem in the decomposition view. Also, generate a port of type *provided_api* for each microservice component with the respective *interoperability* data model generated previously for the module in the data model view.
  - Generate a component of type *msa.relational_database* or *msa.nosql_database* (according to the input parameter **database_type**) for each microservice component generated in the previous step, with the respective *relational* or *nosql* data model generated previously for the module in the data model view.
  - Generate a component of type *msa.web_application* to represent the presentation logic of each monolithic application, with a name equal to that specified in the input parameter **web_application_name**.
  - Generate a component of type *msa.api_gateway* to represent the APIs exposed to the Web application component.
  - Generate a component of type *msa.microservice* for each port of type *provided_service* of each component of type *soa.internal_service_provider* in the SOA model; if there is a

microservice that can supply the functionality of the microservice that needs to be generated (based on the microservice's name), a new microservice is not generated and the existing one is used to supply the functionality. Also, generate a port of type *provided_api* for each microservice component generated in this step with the respective *interoperability* data model (which is generated or complemented by the operations and data entities involved).

- Generate a component of type *msa.relational_database* or *msa.nosql_database* (according to the input parameter **database_type**) for each microservice component generated in the previous step.
- Generate a port of type *requested_dbaccess* for each microservice component generated so far.
- Generate a port of type *provided_dbaccess* for each database component generated so far.
- Generate a connector of type *msa.db_access* for each pair of microservice and database components. This connector has two roles: **src** to connect the microservice component and **tgt** to connect the database component.
- Generate two attachments for each connector generated in the previous step. One attachment is for the relation between the **src** role of the connector and the port of type *requested_dbaccess* of the microservice component, and another attachment is for the relation between the role **tgt** of the connector and the port of type *provided_-dbaccess* of the database component.
- Generate a component of type *msa.microservice* for each port of type *provided_service* and coordination of type *orchestration* of each component of type *soa.esb* in the SOA model. Generate a port of type *provided_api* for each microservice component generated in this step with the respective *interoperability* data model (which is generated with the operations and data entities involved). Also, determine the microservices that are related to each microservice component generated in this step, and generate the respective ports of type *requested_api*, connectors of type *msa.rest* and attachments.
- Generate a port of type *requested_api* in the web application component for each port of type *requested_service* of each component of type *soa.service_consumer* in the SOA model.
- Generate a port of type *provided_api* in the API gateway component for each port that is attached to the ports of each component of type *soa.service_consumer* in the SOA model, with the respective *interoperability* data model (which is generated or complemented by the operations and data entities involved). Also, generate a port of type *requested_api* in the API gateway component for each port of type *provided_api* in this component.

  - Generate a port of type *provided_api* in the API gateway component for each port that is attached to the ports of each component of type *soa.service_consumer* in the SOA model, with the respective *interoperability* data model (which is generated or complemented by the operations and data entities involved).
  - Generate a port of type *requested_api* in the API gateway component for each port of type *provided_api* in this component.
  - Generate the respective connectors of type *msa.rest* and attachments to establish the relations between the API gateway component and the corresponding microservice components.

- In the Layered view, generate the following:
  - A relation of type *contains* for the Web application component and the presentation layer.
  - A relation of type *contains* for each microservice component and business and data access layers.
  - A relation of type *contains* for the presentation tier and the Web application component.
  - A relation of type *contains* for the logic tier and each microservice component.
  - A relation of type *contains* for the logic tier and the API gateway component.
  - A relation of type *contains* for the data tier and each database component.

5.3 Implementation in Sarch-Studio Tool

Sarch-Studio is a tool that allows the definition of software architectures using the grammar rules defined in Sarch. Also, Sarch-Studio offers a model-driven environment to automate the generation of other models through model transformations [67].

Because the Sarch language was extended in the present work to allow the definition of service-based architectures (specifically SOA and MSA), Sarch-Studio was enriched to support the new features of the Sarch language. Likewise, the set of model-to-model transformations that allow the evolution of a SOA model to an MSA model was implemented in Sarch-Studio. A representation of the tool is shown in figure **5-1**, which summarizes the components and modules that are part of the tool [67].

The technologies on which the Sarch-Studio tool is built are the following:

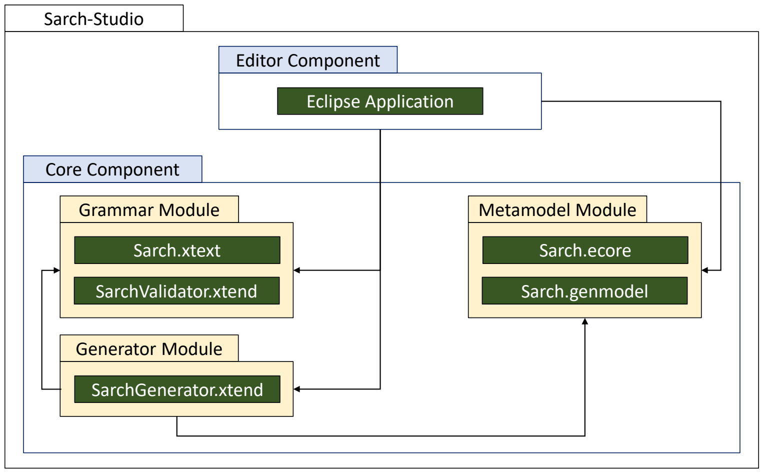


Figure 5-1: Components and modules in Sarch-Studio.

- Xtext: is a framework for developing programming languages and DSLs. Xtext has a powerful set of features for defining the grammar language, including a parser, a linker, a type checker, a compiler and a textual editor for Eclipse [3].
- Eclipse Modeling Framework (EMF): it is a modeling framework and code generation facility for building tools and other applications based on a structured data model. Xtext uses EMF models as the in-memory representation of any parsed text files, that is, the grammar defined in Xtext [1].
- Xtend: it is an expressive dialect of Java programming language that is mainly used for code generation [2] [12].

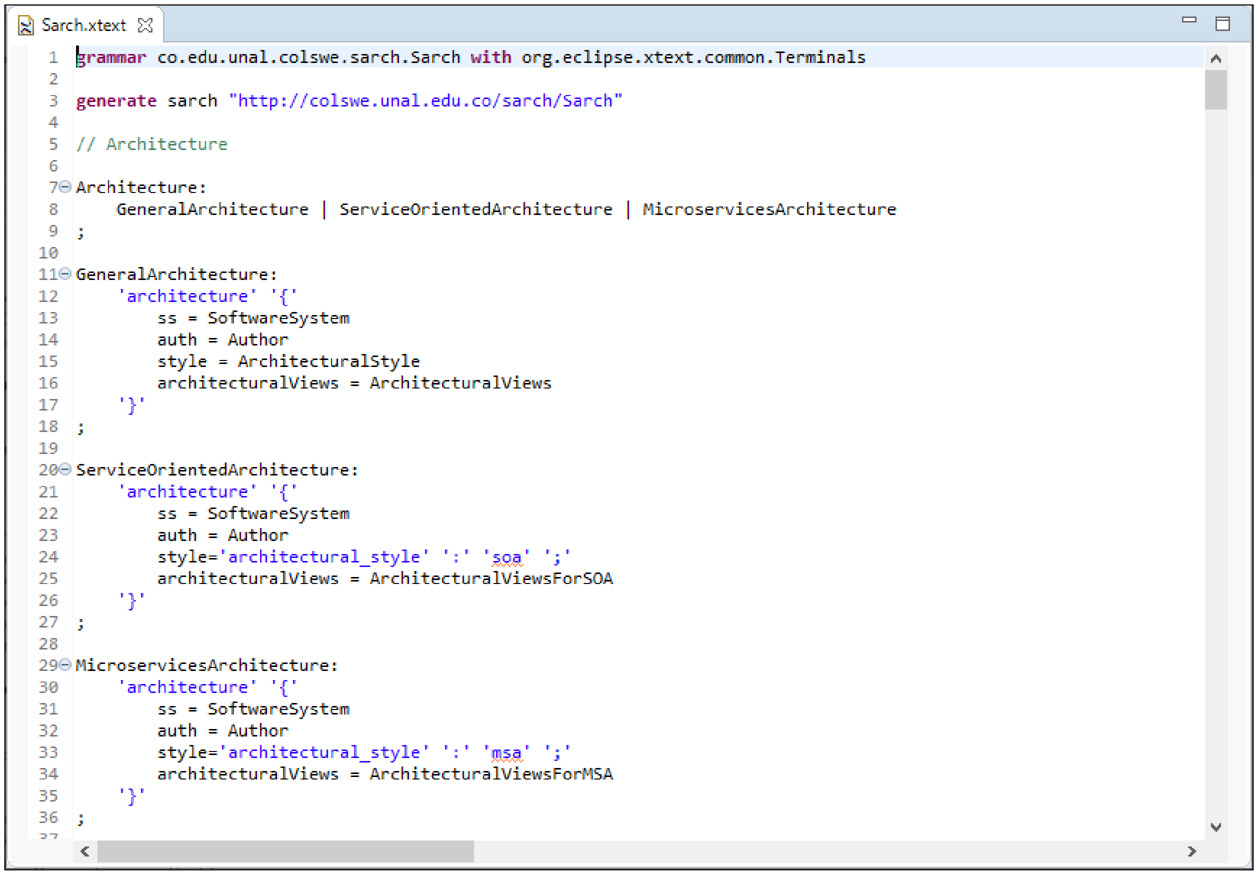
Sarch-Studio tool has two components:

- Core Component: this component integrates the grammar design, the metamodel, and the model transformations. These responsibilities are divided into three modules, which are described below along with the modifications and extensions made in this research work:
  - Grammar Module: this module, which is implemented using Xtext, contains the Sarch grammar. This module was updated with the grammar rules that was described in chapter 4 to allow the definition of an ADL for the design of service-based architectures, specifically for the SOA and MSA architectural styles. Figure **5-2** shows an overview of the Sarch grammar represented in Xtext, and figure **5-3** shows a sample of the validation rules implemented through the custom validation mechanism provided by Xtext in order to restrict the properties that apply to each of the elements and relations according to the architectural views and the architectural styles.
  - Generator Module: this module, which is implemented using Xtend, contains the model transformations rules implemented for model-driven processes. This module

was extended to include the transformations described in section 5.2, that is, the model-to-model transformations to evolve a SOA model to a MSA model. Figure **5-4** shows a sample of the transformations implemented in this module.

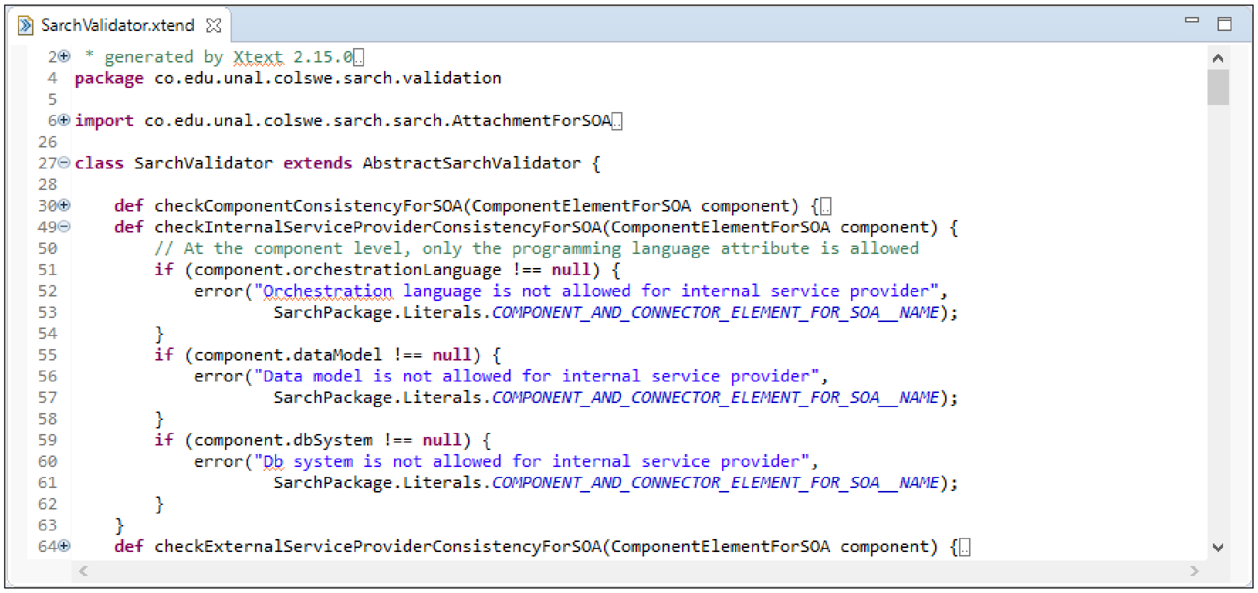
  – Metamodel Module: this module, which is implemented using EMF, contains the metamodel of Sarch grammar in terms of objects. This object representation of the Sarch language is used in both the Sarch validation of the grammar module and the model-to-model transformations of the generator module.

- Editor Component: this component is an Eclipse application that allows the description of software architectures using the Sarch language. Based on the grammar and validations implemented in the grammar module, the editor activates the elements, relations and properties depending on the architectural style that is specified. The editor has a set of usability features like syntax coloring, autocompletion and validation of grammar elements. Figure **5-5** shows a screenshot of the editor component.



```
grammar co.edu.unal.colswe.sarch.Sarch with org.eclipse.xtext.common.Terminals

generate sarch "http://colswe.unal.edu.co/sarch/Sarch"

// Architecture

Architecture:
    GeneralArchitecture | ServiceOrientedArchitecture | MicroservicesArchitecture
;

GeneralArchitecture:
    'architecture' '{'
        ss = SoftwareSystem
        auth = Author
        style = ArchitecturalStyle
        architecturalViews = ArchitecturalViews
    '}'
;

ServiceOrientedArchitecture:
    'architecture' '{'
        ss = SoftwareSystem
        auth = Author
        style='architectural_style' ':' 'soa' ';'
        architecturalViews = ArchitecturalViewsForSOA
    '}'
;

MicroservicesArchitecture:
    'architecture' '{'
        ss = SoftwareSystem
        auth = Author
        style='architectural_style' ':' 'msa' ';'
        architecturalViews = ArchitecturalViewsForMSA
    '}'
;
```

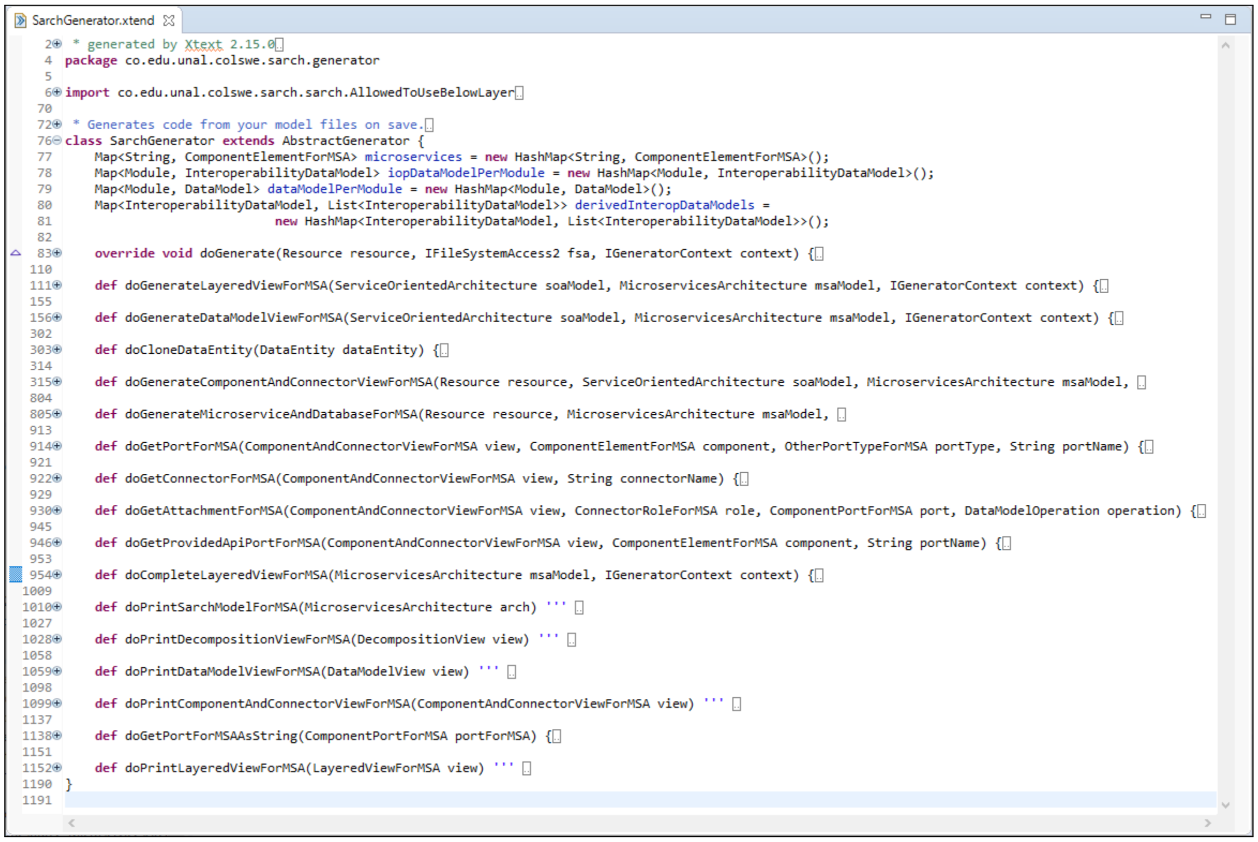
Figure 5-2: Sarch grammar overview in Xtext in Grammar Module.



SarchValidator.xtend

```
 * generated by Xtext 2.15.0
package co.edu.unal.colswe.sarch.validation

import co.edu.unal.colswe.sarch.sarch.AttachmentForSOA

class SarchValidator extends AbstractSarchValidator {

    def checkComponentConsistencyForSOA(ComponentElementForSOA component) {
    def checkInternalServiceProviderConsistencyForSOA(ComponentElementForSOA component) {
        // At the component level, only the programming language attribute is allowed
        if (component.orchestrationLanguage !== null) {
            error("Orchestration language is not allowed for internal service provider",
                    SarchPackage.Literals.COMPONENT_AND_CONNECTOR_ELEMENT_FOR_SOA__NAME);
        }
        if (component.dataModel !== null) {
            error("Data model is not allowed for internal service provider",
                    SarchPackage.Literals.COMPONENT_AND_CONNECTOR_ELEMENT_FOR_SOA__NAME);
        }
        if (component.dbSystem !== null) {
            error("Db system is not allowed for internal service provider",
                    SarchPackage.Literals.COMPONENT_AND_CONNECTOR_ELEMENT_FOR_SOA__NAME);
        }
    }
    def checkExternalServiceProviderConsistencyForSOA(ComponentElementForSOA component) {
```

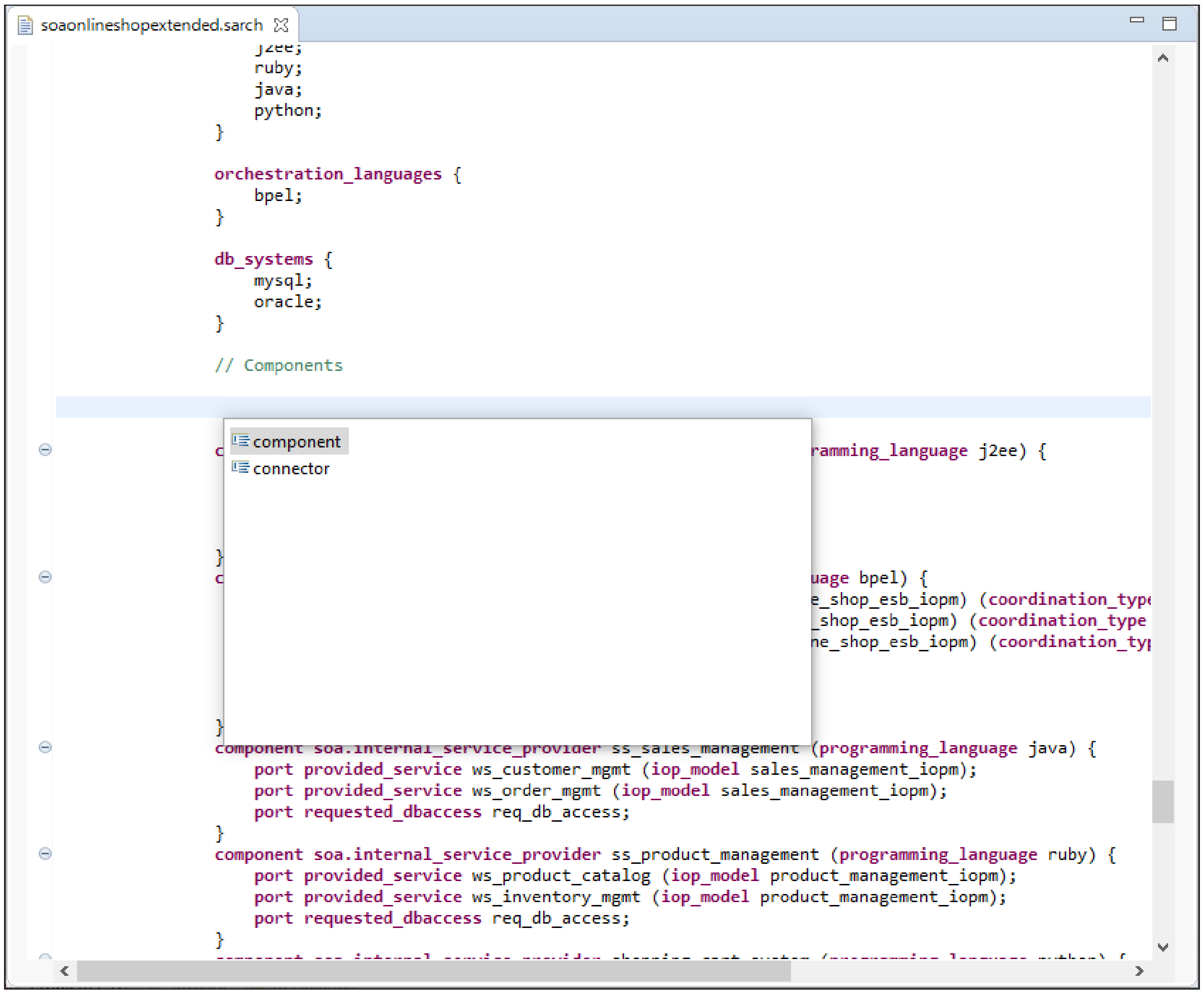
Figure 5-3: Sample of Sarch validations in Grammar Module.



SarchGenerator.xtend

```
 * generated by Xtext 2.15.0
package co.edu.unal.colswe.sarch.generator

import co.edu.unal.colswe.sarch.sarch.AllowedToUseBelowLayer

 * Generates code from your model files on save.
class SarchGenerator extends AbstractGenerator {
    Map<String, ComponentElementForMSA> microservices = new HashMap<String, ComponentElementForMSA>();
    Map<Module, InteroperabilityDataModel> iopDataModelPerModule = new HashMap<Module, InteroperabilityDataModel>();
    Map<Module, DataModel> dataModelPerModule = new HashMap<Module, DataModel>();
    Map<InteroperabilityDataModel, List<InteroperabilityDataModel>> derivedInteropDataModels =
                            new HashMap<InteroperabilityDataModel, List<InteroperabilityDataModel>>();

    override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {

    def doGenerateLayeredViewForMSA(ServiceOrientedArchitecture soaModel, MicroservicesArchitecture msaModel, IGeneratorContext context) {

    def doGenerateDataModelViewForMSA(ServiceOrientedArchitecture soaModel, MicroservicesArchitecture msaModel, IGeneratorContext context) {

    def doCloneDataEntity(DataEntity dataEntity) {

    def doGenerateComponentAndConnectorViewForMSA(Resource resource, ServiceOrientedArchitecture soaModel, MicroservicesArchitecture msaModel,

    def doGenerateMicroserviceAndDatabaseForMSA(Resource resource, MicroservicesArchitecture msaModel,

    def doGetPortForMSA(ComponentAndConnectorViewForMSA view, ComponentElementForMSA component, OtherPortTypeForMSA portType, String portName) {

    def doGetConnectorForMSA(ComponentAndConnectorViewForMSA view, String connectorName) {

    def doGetAttachmentForMSA(ComponentAndConnectorViewForMSA view, ConnectorRoleForMSA role, ComponentPortForMSA port, DataModelOperation operation) {

    def doGetProvidedApiPortForMSA(ComponentAndConnectorViewForMSA view, ComponentElementForMSA component, String portName) {

    def doCompleteLayeredViewForMSA(MicroservicesArchitecture msaModel, IGeneratorContext context) {

    def doPrintSarchModelForMSA(MicroservicesArchitecture arch) '''

    def doPrintDecompositionViewForMSA(DecompositionView view) '''

    def doPrintDataModelViewForMSA(DataModelView view) '''

    def doPrintComponentAndConnectorViewForMSA(ComponentAndConnectorViewForMSA view) '''

    def doGetPortForMSAAsString(ComponentPortForMSA portForMSA) {

    def doPrintLayeredViewForMSA(LayeredViewForMSA view) '''
}
```

Figure 5-4: Sample of Sarch transformations in Generator Module.



```
        j2ee;
        ruby;
        java;
        python;
    }

    orchestration_languages {
        bpel;
    }

    db_systems {
        mysql;
        oracle;
    }

    // Components


    c                                                 ramming_language j2ee) {



    }
    c                                                 uage bpel) {
                                                      e_shop_esb_iopm) (coordination_type
                                                      _shop_esb_iopm) (coordination_type
                                                      ne_shop_esb_iopm) (coordination_typ



    }
    component soa.internal_service_provider ss_sales_management (programming_language java) {
        port provided_service ws_customer_mgmt (iop_model sales_management_iopm);
        port provided_service ws_order_mgmt (iop_model sales_management_iopm);
        port requested_dbaccess req_db_access;
    }
    component soa.internal_service_provider ss_product_management (programming_language ruby) {
        port provided_service ws_product_catalog (iop_model product_management_iopm);
        port provided_service ws_inventory_mgmt (iop_model product_management_iopm);
        port requested_dbaccess req_db_access;
    }
    component soa.internal_service_provider shopping_cart_system (programming_language python) {
```

Figure 5-5: Sample of Editor Component.

6 Evaluation

This chapter presents the evaluation of the proposed strategy based on model-to-model transformations to evolve service-oriented architectures to microservices architectures. In order to achieve this, a case study is presented.

The case study consists of an online shop application, which is a well-known and common software system in different business and engineering fields. Through this online shop application a customer authenticates, searches for products, adds or removes products from the shopping cart, and finally places the purchase order. The purchase order involves checking the available inventory, saving the purchase order, managing the inventory, destroying the current shopping cart for the order, and returning the order information to the customer. Figure **6-1** shows the context diagram for the case study, with the Online Shop Subsystem under consideration and its boundaries with the Sales Management Subsystem, the Customer Management Subsystem, the Sales Management Subsystem, and the Shopping Cart Database.

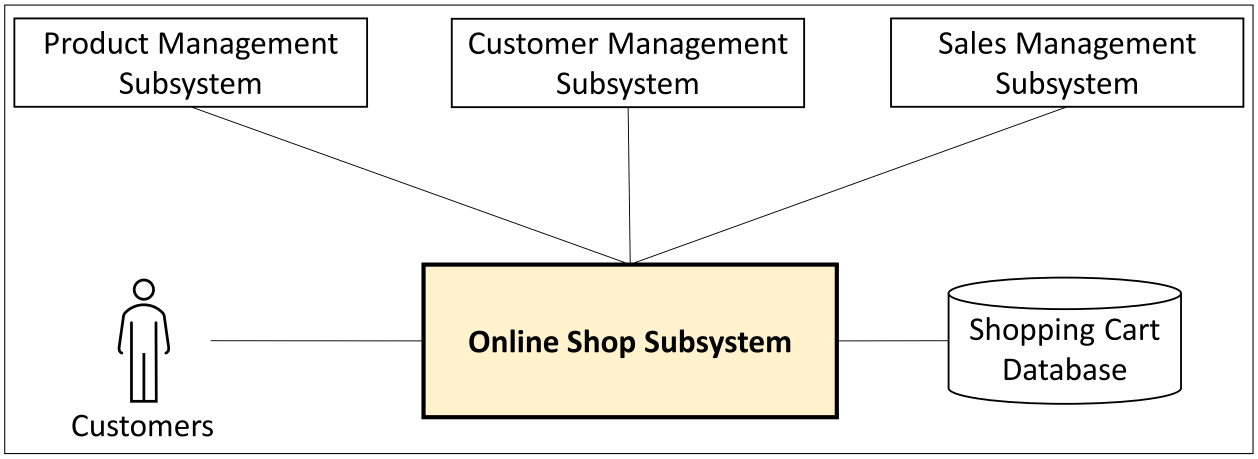

Product Management Subsystem
Customer Management Subsystem
Sales Management Subsystem
Online Shop Subsystem
Shopping Cart Database
Customers

Figure 6-1: Context diagram for the case study.

To perform the evaluation, the steps described in section 5.1 will be followed to evolve a service-oriented architecture to a microservices architecture applied to the selected case study.

6.1 SOA-Based System as Input

To do this, a reference implementation was created for a simplified online shop application. Based on this reference implementation, the case study is described by the Decomposition, Data Model, C&C and Layered views. Figure **6-2** shows the Decomposition view with the subsystems, modules and submodules that are part of the system. Figures **6-3**, **6-4** and **6-5** show the data models that are part of the Data Model view for the case study designed in the SOA style. Figure **6-6** shows the C&C view for the case study designed in the SOA style. Figure **6-7** shows the Layered view for the case study designed in the SOA style.

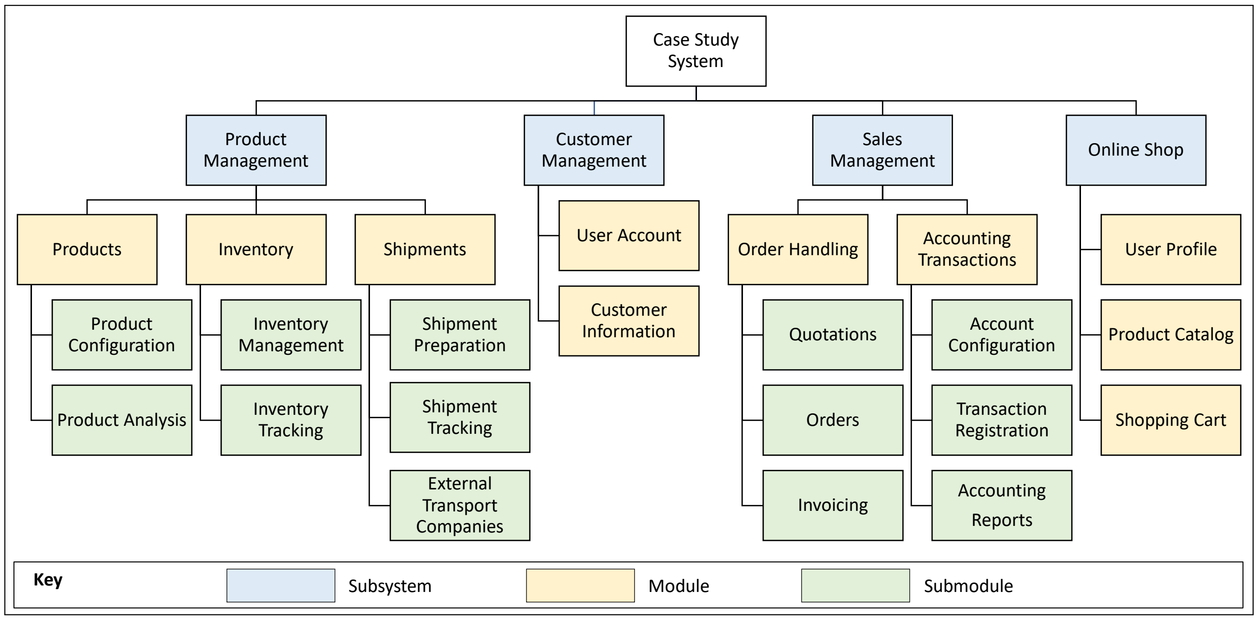


Figure 6-2: Decomposition view for the case study.

Also, the same software system was designed in the MSA style using the C&C view, which is shown in figure **6-8**. This representation will be used later to validate the MSA model resulting from the model-to-model transformations.

6.2 Functional Decomposition in Sarch

In this step, the Sarch-Studio tool is used to describe the case study using the Decomposition view. Figure **6-9** shows the Decomposition view for the case study, where the subsystems, modules and submodules indicated in the first step have been included, plus the functionalities that belong to each submodule or module.

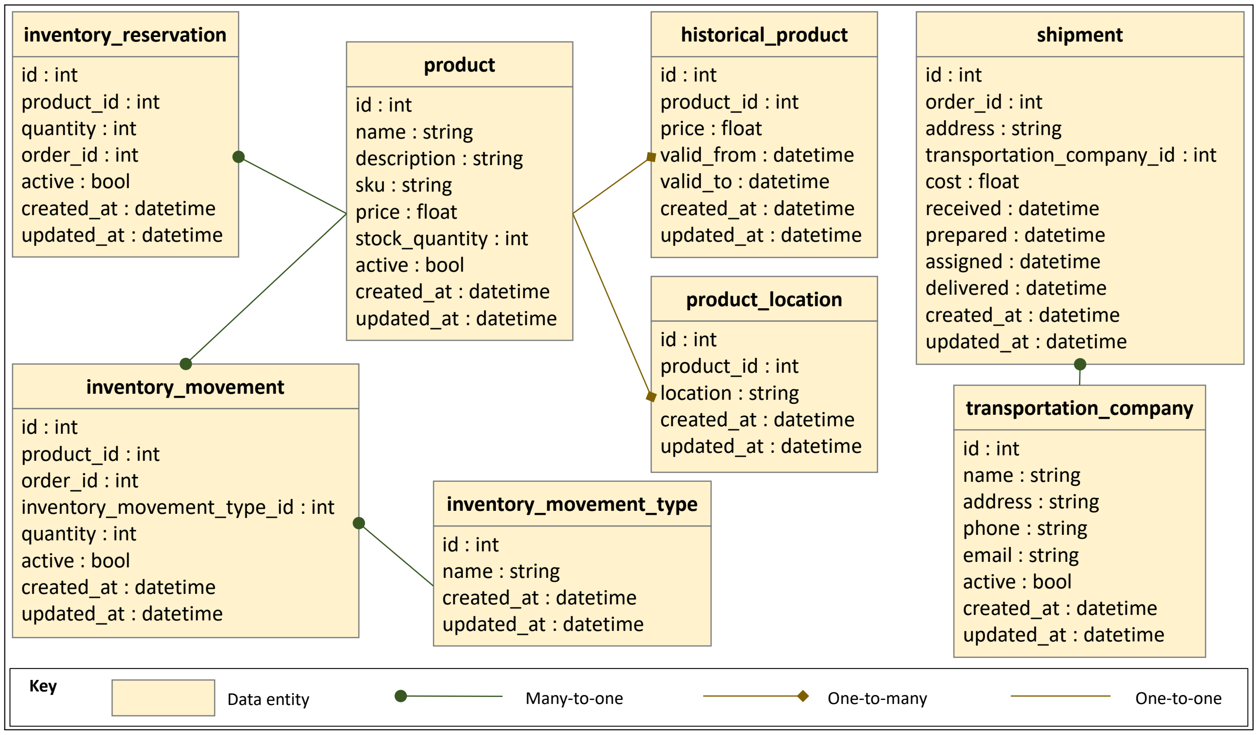


Figure 6-3: Data Model view for the case study designed in the SOA style - product management data model.

6.3 SOA-Based System Modeled in Sarch

In this step, the SOA-based system for the case study is modeled using the Sarch-Studio tool, specifically with the Data Model, C&C, and Layered views.

The most important aspects for modeling the Data Model view for the case study are described below:

- This view has three data models of type *relational*, one for each database component. For each data entity of the Data Model view, it is indicated to which module in the Decomposition view it belongs. Figure **6-10** shows the relational data models included in this view.

- This view has four data models of type *interoperability*, one for each internal service provider (three for the case study) and another for the ESB component, which allow defining the data exposed by the services as well as the operations they provide. Figure **6-11** shows the interoperability data models included in this view.

The most important aspects for modeling the C&C view for the case study are described below:

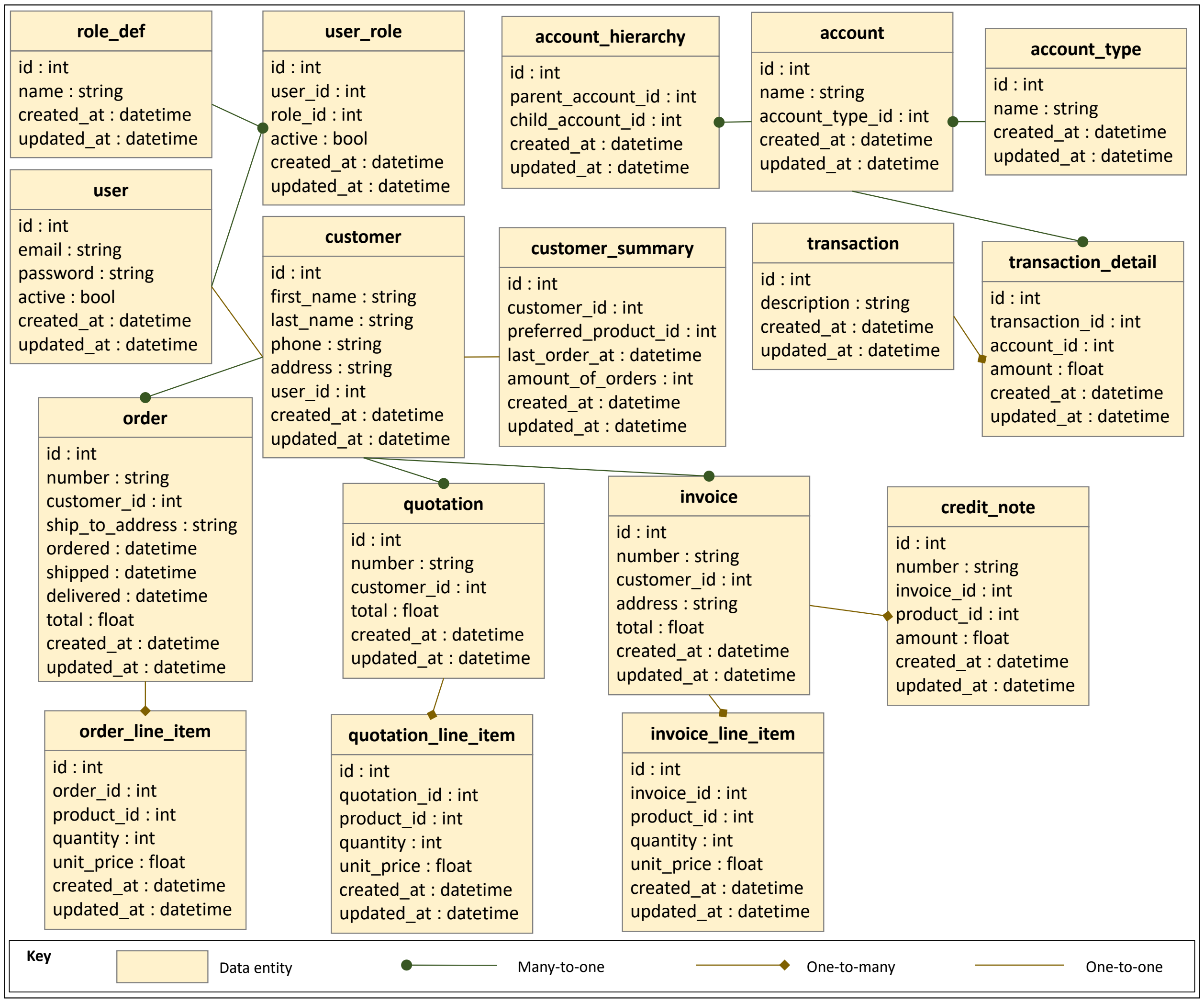


Figure 6-4: Data Model view for the case study designed in the SOA style - sales management data model.

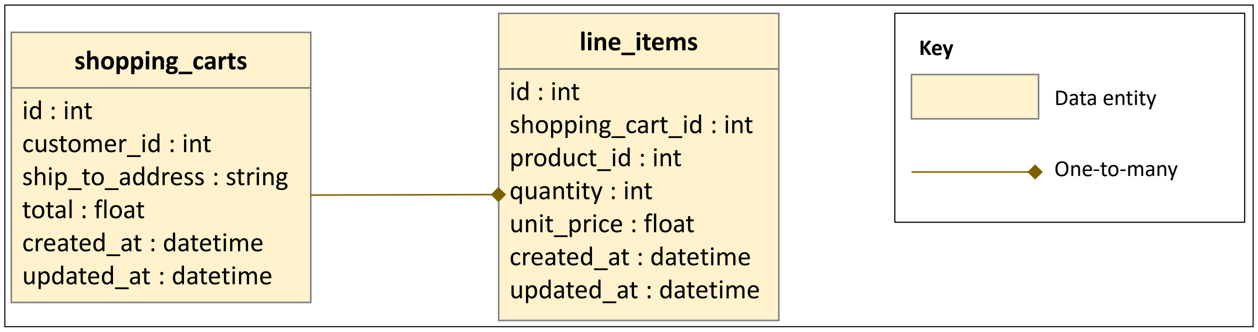


Figure 6-5: Data Model view for the case study designed in the SOA style - shopping cart data model.

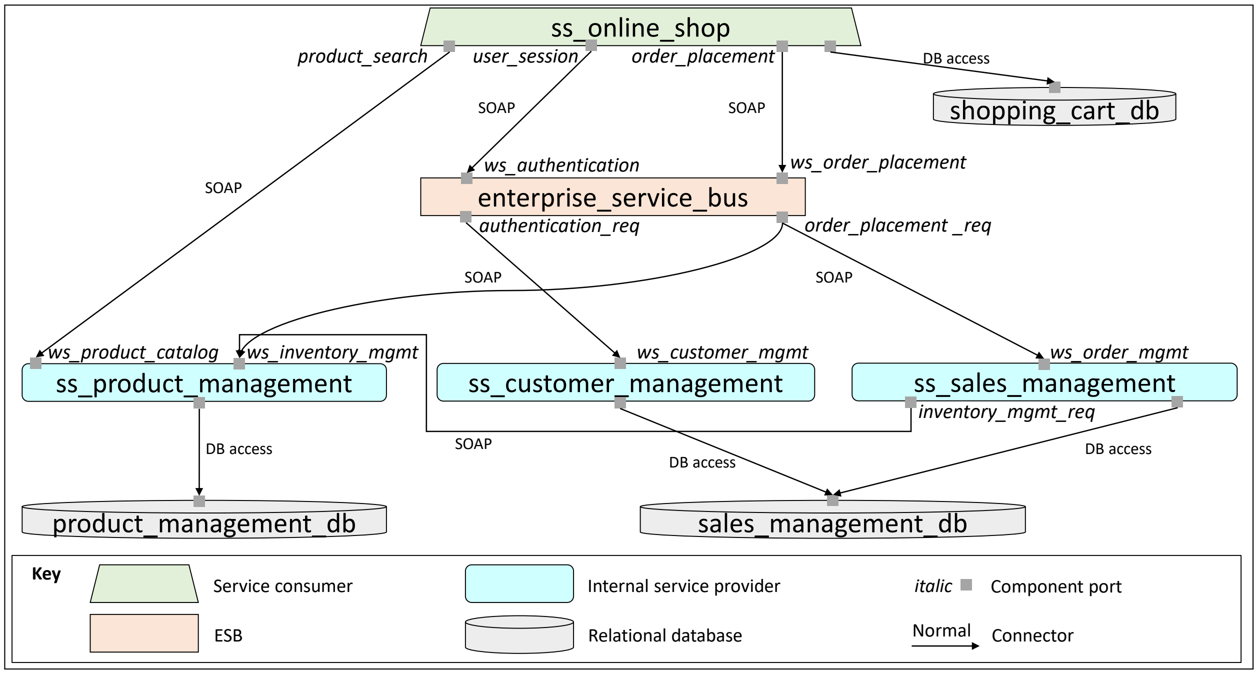


Figure 6-6: C&C view for the case study designed in the SOA style.

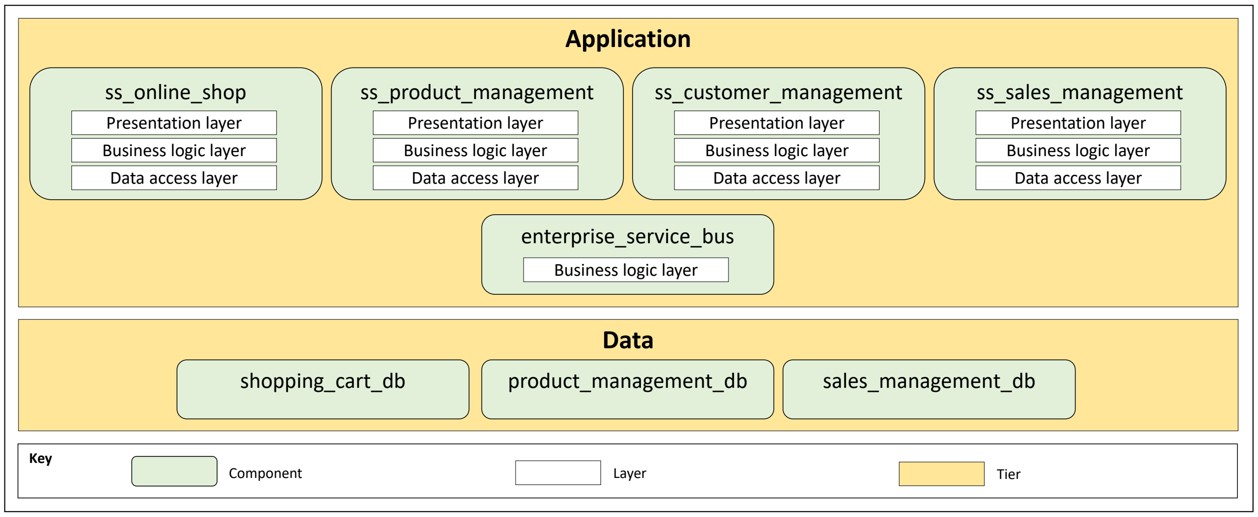


Figure 6-7: Layered view for the case study designed in the SOA style.

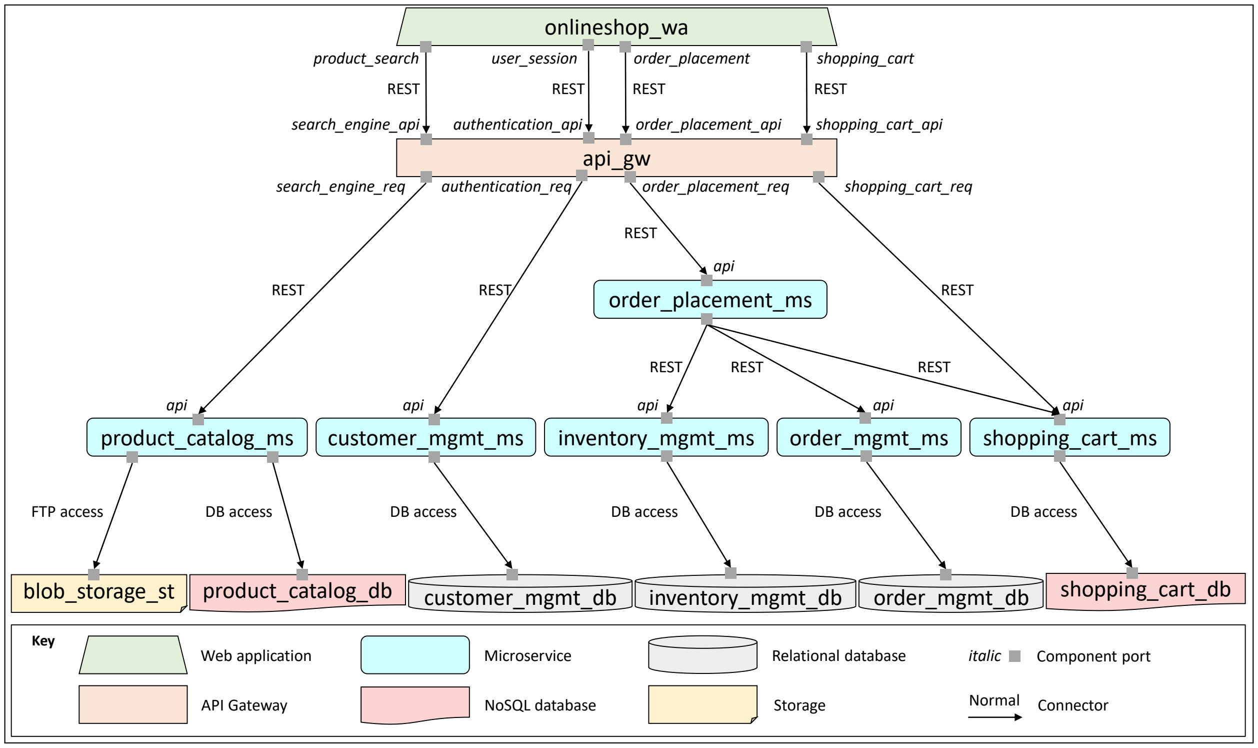


Figure 6-8: C&C view for the case study designed in the MSA style.

- For the port named **ws_order placement** of the enterprise service bus, the **coordination_-type** property was indicated with a value equal to *orchestration*, considering that placing an order in the system involves several operations with business logic.
- Each port of type *provided_service* has the respective interoperability data model defined in the Data Model view.

Figures **6-12**, **6-13** and **6-14** show the components, connectors and relations included in the C&C view, respectively.

Figure **6-15** shows the Layered view for the case study, which includes the following:

- The presentation, business logic and data access layers.
- The application and data tiers.
- The relation of type *allowed_to_use* between the presentation layer and the business logic layer and between the business logic layer and the data access layer.
- The relation of type *allowed_to_use* between the application tier and the data tier.
- A relation of type *contains* between each service consumer (ss_online_shop) and each layer.

Grammar in Sarch

```
DecompositionView:
  'decomposition_view' '::’
    'elements' '{‘
      (decompositionElement += DecompositionElement)+
    '}’
    'relations' '{‘
      (decompositionRelation += DecompositionRelation)*
    '}’
  '::'
;

DecompositionElement:
  Subsystem | Module | Submodule | Functionality
;

DecompositionRelation:
  IsPartOfA | IsPartOfB | IsPartOfC | IsPartOfD
;

IsPartOfA:
  'm:' module=[Module] 'is_part_of' 'ss:' subsystem=[Subsystem] ';'
;

IsPartOfB:
  'sm:' submodule=[Submodule] 'is_part_of' 'm:' module=[Module] ';'
;

IsPartOfC:
  'f:' functionality=[Functionality] 'is_part_of' 'm:' module=[Module] ';'
;

IsPartOfD:
  'f:' functionality=[Functionality] 'is_part_of' 'sm:'
submodule=[Submodule] ';'
;
```

Decomposition View

```
decomposition_view ::
  elements {
    subsystem product_management;
    module products;
    submodule product_configuration;
    submodule product_analysis;
    module inventory;
    submodule inventory_management;
    submodule inventory_tracking;
    module shipments;
    submodule shipment_preparation;
    submodule shipment_tracking;
    submodule external_transport_companies;
    subsystem customer_management;
    module user_account;
    module customer_information;
    subsystem sales_management;
    module order_handling;
    submodule quotations;
    submodule orders;
    submodule invoicing;
    module accounting_transactions;
    submodule account_configuration;
    submodule transaction_registration;
    submodule accounting_reports;
    subsystem online_shop;
    module user_profile;
    module product_catalog;
    module shopping_cart;
  }
  relations {
    m: products is_part_of ss: product_management;
    sm: product_configuration is_part_of m: products;
    sm: product_analysis is_part_of m: products;
    m: inventory is_part_of ss: product_management;
    sm: inventory_management is_part_of m: inventory;
    sm: inventory_tracking is_part_of m: inventory;
    m: shipments is_part_of ss: product_management;
    sm: shipment_preparation is_part_of m: shipments;
    sm: shipment_tracking is_part_of m: shipments;
    sm: external_transport_companies is_part_of m: shipments;
    m: user_account is_part_of ss: customer_management;
    m: customer_information is_part_of ss: customer_management;
    m: order_handling is_part_of ss: sales_management;
    sm: quotations is_part_of m: order_handling;
    sm: orders is_part_of m: order_handling;
    sm: invoicing is_part_of m: order_handling;
    m: accounting_transactions is_part_of ss: sales_management;
    sm: account_configuration is_part_of m: accounting_transactions;
    sm: transaction_registration is_part_of m: accounting_transactions;
    sm: accounting_reports is_part_of m: accounting_transactions;
    m: user_profile is_part_of ss: online_shop;
    m: product_catalog is_part_of ss: online_shop;
    m: shopping_cart is_part_of ss: online_shop;
  }
::
```

Figure 6-9: Decomposition view for the case study using Sarch.

Grammar in Sarch

```
DataModelView:
  'data_model_view' '::’
    (dataModel += DataModel)+
  '::'
;

DataModel:
  RelationalDataModel | NoSqlDataModel | InteroperabilityDataModel
;

RelationalDataModel:
  dataModelType='relational' 'data_model' name=ID '{‘
    'elements' '{‘
      (dataModelElement += DataModelElement)+
    '}’
    ('relations' '{‘
      (dataModelRelation += DataModelRelation)*
    '}’)?
  '}'
;

NoSqlDataModel:
  dataModelType='nosql' 'data_model' name=ID '{‘
    'elements' '{‘
      (dataModelElement += DataModelElement)+
    '}’
    ('relations' '{‘
      (dataModelRelation += DataModelRelation)*
    '}’)?
  '}'
;

DataModelElement:
  DataEntity
;

DataEntity:
  'data_entity' name=ID ('(' 'module' module=[Module] ')')? '{‘
   'attributes' '{‘
      (attribute += Attribute)+
    '}’
  '}'
;

Attribute:
  dataType=DataType name=ID ';'
;

DataModelRelation:
  relationType=RelationType '(' sourceDataEntity=[DataEntity] ',’
      targetDataEntity=[DataEntity] ')' ';'
;

RelationType:
  OneToOne | OneToMany | ManyToOne | ManyToMany | Association
      | Aggregation | Composition | Generalization | Specialization
;
```

Relational Data Models

```
data_model_view ::
  relational data_model product_management_dm {
    elements {
      data_entity product (module products) { ... }
      data_entity inventory_reservation (module inventory) { ... }
      data_entity historical_product (module products) { ... }
      data_entity product_location (module products) { ... }
      data_entity inventory_movement_type (module inventory) { ... }
      data_entity inventory_movement (module inventory) { ... }
      data_entity transportation_company (module shipments) { ... }
      data_entity shipment (module shipments) { ... }
    }
    relations {
      one_to_many (product, historical_product);
      one_to_many (product, product_location);
      many_to_one (inventory_reservation, product);
      many_to_one (inventory_movement, product);
      many_to_one (inventory_movement, inventory_movement_type);
      many_to_one (shipment, transportation_company);
    }
  }
  relational data_model sales_management_dm {
    elements {
      data_entity role_def (module user_account) { ... }
      data_entity user (module user_account) { ... }
      data_entity user_role (module user_account) { ... }
      data_entity customer (module customer_information) { ... }
      data_entity customer_summary (module customer_information) { ... }
      data_entity quotation (module order_handling) { ... }
      data_entity quotation_line_item (module order_handling) { ... }
      data_entity order (module order_handling) { ... }
      data_entity order_line_item (module order_handling) { ... }
      data_entity invoice (module order_handling) { ... }
      data_entity invoice_line_item (module order_handling) { ... }
      data_entity credit_note (module order_handling) { ... }
      data_entity account_type (module accounting_transactions) { ... }
      data_entity account (module accounting_transactions) { ... }
      data_entity transaction (module accounting_transactions) { ... }
      data_entity transaction_detail (module accounting_transactions) { ... }
      data_entity account_hierarchy (module accounting_transactions) { ... }
    }
    relations {
      many_to_one (user_role, role_def);
      many_to_one (user_role, user);
      one_to_one (customer, user);
      one_to_one (customer_summary, customer);
      many_to_one (quotation, customer);
      one_to_many (quotation, quotation_line_item);
      many_to_one (quotation_line_item, product);
      many_to_one (order, customer);
      one_to_many (order, shipment);
      one_to_many (order, inventory_reservation);
      one_to_many (order, inventory_movement);
      one_to_many (order, order_line_item);
      many_to_one (order_line_item, product);
      many_to_one (invoice, customer);
      one_to_many (invoice, invoice_line_item);
      many_to_one (invoice_line_item, product);
      one_to_many (invoice, credit_note);
      many_to_one (credit_note, product);
      many_to_one (account, account_type);
      many_to_one (account_hierarchy, account);
      one_to_many (transaction, transaction_detail);
      many_to_one (transaction_detail, account);
    }
  }
  relational data_model shopping_cart_dm {
    elements {
      data_entity shopping_carts (module shopping_cart) { ... }
      data_entity line_items (module shopping_cart) { ... }
    }
    relations {
      one_to_many (shopping_carts, line_items);
    }
  }
```

Figure 6-10: Data Model view for the case study using Sarch in the SOA style - relational data models.

Grammar in Sarch

```
DataModelView:
  'data_model_view' '::’
    (dataModel += DataModel)+
  '::'
;

DataModel:
  RelationalDataModel | NoSqlDataModel | InteroperabilityDataModel
;

InteroperabilityDataModel:
  dataModelType='interoperability' 'data_model' name=ID '{‘
    'elements' '{‘
      (dataModelElement += DataModelElement)*
    '}’
    ('relations' '{‘
      (dataModelRelation += DataModelRelation)*
    '}’)?
    ('operations' '{‘
      (dataModelOperation += DataModelOperation)*
    '}’)?
  '}’
;

DataModelOperation:
  'operation' name=ID '{‘
    'parameters' '{‘
      (parameter += OperationParameter)*
    '}’
  '}'
;

OperationParameter:
  dataType=ParameterType (array=Array)? name=ID
direction=ParameterDirection ';'
;

ParameterType:
  basic=DataType | kind=[DataEntity]
;

Array:
  '[' ']'
;

enum ParameterDirection:
  IN='in' | OUT='out'
;
```

Interoperability Data Models

```
data_model_view ::
  interoperability data_model product_management_iopm {
    elements {
      data_entity product_summary {
        attributes {
          int id;
          string name;
          string description;
          double price;
        }
      }
      data_entity item { ... }
    }
    operations {
      operation search_products {
        parameters {
          string filter in;
          double min_price in;
          double max_price in;
          product_summary[] result out;
        }
      }
      operation check_inventory { ... }
      operation manage_inventory { ... }
      operation update_inventory { ... }
    }
  }
  interoperability data_model customer_management_iopm {
    elements {
      data_entity customer { ... }
    }
    operations {
      operation authenticate { ... }
      operation get_customer { ... }
    }
  }
 interoperability data_model sales_management_iopm {
    elements {
      data_entity item { ... }
      data_entity order { ... }
    }
    operations {
      operation register_order { ... }
      operation get_order { ... }
      operation get_customer_orders { ... }
      operation get_order_items { ... }
    }
  }
  interoperability data_model online_shop_esb_iopm {
    elements {
      data_entity customer { ... }
      data_entity item { ... }
    }
    operations {
      operation authenticate { ... }
      operation make_order { ... }
    }
  }
::
```

Figure 6-11: Data Model view for the case study using Sarch in the SOA style - interoperability data models.

Grammar in Sarch

```
ComponentAndConnectorViewForSOA:
'component_and_connector_view' '::’
  'elements' '{‘
    ('programming_languages' '{‘
      (programmingLanguage += ProgrammingLanguage)+
    '}’)?
    ('orchestration_languages' '{‘
      (orchestrationLanguage += OrchestrationLanguage)+
    '}’)?
    ('db_systems' '{‘
      (dbSystem += DbSystem)+
    '}’)?
    (componentAndConnectorElement += ComponentAndConnectorElementForSOA)+
  '}’
  'relations' '{‘
    (componentAndConnectorRelation += ComponentAndConnectorRelationForSOA)*
  '}'
'::'
;
OrchestrationLanguage:
  name=ID ';'
;
ComponentAndConnectorElementForSOA:
  ComponentElementForSOA | ConnectorElementForSOA
;

ComponentElementForSOA:
  'component' componentType=ComponentTypeForSOA name=ID ('('
'programming_language' programmingLanguage=[ProgrammingLanguage] ')')? ('('
'orchestration_language' orchestrationLanguage=[OrchestrationLanguage] ')')?
('(' 'data_model' dataModel=([RelationalDataModel]) ')')? ('(' 'db_system'
dbSystem=[DbSystem] ')')? '{'
(port += ComponentPortForSOA)+
'}'
;
enum ComponentTypeForSOA:
  INTERNAL_SERVICE_PROVIDER='soa.internal_service_provider’ |
  EXTERNAL_SERVICE_PROVIDER='soa.external_service_provider’ |
  SERVICE_CONSUMER='soa.service_consumer’ |
  RELATIONAL_DATABASE='soa.relational_database’ |
  ENTERPRISE_SERVICE_BUS='soa.esb'
;

ComponentPortForSOA:
  ProvidedServicePortForSOA | RequestedServicePortForSOA |
  ProvidedDBAccessPortForSOA | RequestedDBAccessPortForSOA
;

ProvidedServicePortForSOA:
  'port' 'provided_service' name=ID ('(' 'iop_model’
      iopModel=[InteroperabilityDataModel] ')') ('(' 'coordination_type’
      coordinationType=CoordinationTypeForSOA ')')? ';'
;
enum CoordinationTypeForSOA:
  NONE='none' | ORCHESTRATION='orchestration' | CHOREOGRAPHY='choreography'
;

RequestedServicePortForSOA:
  'port' 'requested_service' name=ID ';'
;
ProvidedDBAccessPortForSOA:
  'port' 'provided_dbaccess' name=ID ';'
;
RequestedDBAccessPortForSOA:
  'port' 'requested_dbaccess' name=ID ';'
;
```

Component Elements

```
component_and_connector_view ::
  elements {
    programming_languages {
      j2ee;
    }
    orchestration_languages {
      bpel;
    }
    db_systems {
      mysql;
      oracle;
    }
    component soa.service_consumer ss_online_shop
            (programming_language j2ee) {
      port requested_service user_session;
      port requested_service product_search;
      port requested_service order_placement;
      port requested_dbaccess req_db_access;
    }
    component soa.esb enterprise_service_bus
            (orchestration_language bpel) {
      port provided_service ws_authentication
            (iop_model online_shop_esb_iopm)
            (coordination_type none);
      port provided_service ws_order_placement
            (iop_model online_shop_esb_iopm)
            (coordination_type orchestration);
      port requested_service authentication_req;
      port requested_service order_placement_req;
    }
    component soa.internal_service_provider ss_product_management
            (programming_language j2ee) {
      port provided_service ws_product_catalog
            (iop_model product_management_iopm);
      port provided_service ws_inventory_mgmt
            (iop_model product_management_iopm);
      port requested_dbaccess req_db_access;
    }
    component soa.internal_service_provider ss_customer_management
            (programming_language j2ee) {
      port provided_service ws_customer_mgmt
            (iop_model customer_management_iopm);
      port requested_dbaccess req_db_access;
    }
    component soa.internal_service_provider ss_sales_management
            (programming_language j2ee) {
      port provided_service ws_order_mgmt
            (iop_model sales_management_iopm);
      port requested_service inventory_mgmt_req;
      port requested_dbaccess req_db_access;
    }
    component soa.relational_database shopping_cart_db
            (data_model shopping_cart_dm) (db_system mysql) {
      port provided_dbaccess provided_access;
    }
    component soa.relational_database product_management_db
            (data_model product_management_dm) (db_system oracle) {
      port provided_dbaccess provided_access;
    }
    component soa.relational_database sales_management_db
            (data_model sales_management_dm) (db_system oracle) {
      port provided_dbaccess provided_access;
    }
```

Figure 6-12: C&C view for the case study using Sarch in the SOA style - components.

Grammar in Sarch

```
ComponentAndConnectorViewForSOA:
'component_and_connector_view' '::’
  'elements' '{‘
    ('programming_languages' '{‘
      (programmingLanguage += ProgrammingLanguage)+
    '}’)?
    ('orchestration_languages' '{‘
      (orchestrationLanguage += OrchestrationLanguage)+
    '}’)?
    ('db_systems' '{‘
      (dbSystem += DbSystem)+
    '}’)?
    (componentAndConnectorElement +=
ComponentAndConnectorElementForSOA)+
  '}’
  'relations' '{‘
    (componentAndConnectorRelation +=
ComponentAndConnectorRelationForSOA)*
  '}'
'::'
;
ComponentAndConnectorElementForSOA:
  ComponentElementForSOA | ConnectorElementForSOA
;

ConnectorElementForSOA:
  'connector' connectorType=ConnectorTypeForSOA name=ID '{‘
     (role += ConnectorRoleForSOA)+
'}'
;

enum ConnectorTypeForSOA:
  HTTP='soa.http' | SOAP='soa.soap' | REST='soa.rest’ |
  MESSAGING='soa.messaging' | DB_ACCESS='soa.db_access'
;

ConnectorRoleForSOA:
  ProviderRoleForSOA | ConsumerRoleForSOA
;

ProviderRoleForSOA:
  'role' 'provider' name=ID ';'
;

ConsumerRoleForSOA:
  'role' 'consumer' name=ID ';'
;
```

Connector Elements

```
component_and_connector_view ::
  elements {
    connector soa.soap ols_usersession_soap {
      role consumer source;
      role provider target_service;
    }
    connector soa.soap ols_productsearch_soap {
      role consumer source;
      role provider target_service;
    }
    connector soa.soap ols_orderplacement_soap {
      role consumer source;
      role provider target_service;
    }
    connector soa.soap esb_authentication_soap {
      role consumer source;
      role provider target_service;
    }
    connector soa.soap esb_orderplacement_soap {
      role consumer source;
      role provider target_service;
    }
    connector soa.soap salesmgmt_inventorymgmt_soap {
      role consumer source;
      role provider target_service;
    }
    connector soa.db_access ols_ss_db {
      role consumer source;
      role provider target_db;
    }
    connector soa.db_access product_management_ss_db {
      role consumer source;
      role provider target_db;
    }
    connector soa.db_access customer_management_ss_db {
      role consumer source;
      role provider target_db;
    }
    connector soa.db_access sales_management_ss_db {
      role consumer source;
      role provider target_db;
    }
  }
```

Figure 6-13: C&C view for the case study using Sarch in the SOA style - connectors.

Grammar in Sarch

```
ComponentAndConnectorViewForSOA:
'component_and_connector_view' '::’
  'elements' '{‘
    ('programming_languages' '{‘
      (programmingLanguage += ProgrammingLanguage)+
    '}’)?
    ('orchestration_languages' '{‘
      (orchestrationLanguage += OrchestrationLanguage)+
    '}’)?
    ('db_systems' '{‘
      (dbSystem += DbSystem)+
    '}’)?
    (componentAndConnectorElement +=
            ComponentAndConnectorElementForSOA)+
  '}’
  'relations' '{‘
    (componentAndConnectorRelation +=
            ComponentAndConnectorRelationForSOA)*
  '}'
'::’
;

ComponentAndConnectorRelationForSOA:
  AttachmentForSOA
;

AttachmentForSOA:
  'attachment' '(' role=[ConnectorRoleForSOA | QualifiedName]
      ',’ port=[ComponentPortForSOA | QualifiedName] ('(‘
      'operation’ operation=[DataModelOperation |
      QualifiedName] ')')? ')' ';'
;
```

Attachment Relations

```
component_and_connector_view ::
  relations {
    // Services consumed from online application, provided by the ESB
    attachment (ols_usersession_soap.source, ss_online_shop.user_session);
    attachment (ols_usersession_soap.target_service,
      enterprise_service_bus.ws_authentication
      (operation online_shop_esb_iopm.authenticate));
    attachment (ols_orderplacement_soap.source,
      ss_online_shop.order_placement);
    attachment (ols_orderplacement_soap.target_service,
      enterprise_service_bus.ws_order_placement
      (operation online_shop_esb_iopm.make_order));
    // Services consumed from online application, provided by the
    // ss_product_management
    attachment (ols_productsearch_soap.source,
      ss_online_shop.product_search);
    attachment (ols_productsearch_soap.target_service,
      ss_product_management.ws_product_catalog
      (operation product_management_iopm.search_products));
    // Services consumed from the ESB
    attachment (esb_authentication_soap.source,
      enterprise_service_bus.authentication_req
      (operation online_shop_esb_iopm.authenticate));
    attachment (esb_authentication_soap.target_service,
      ss_customer_management.ws_customer_mgmt
      (operation customer_management_iopm.authenticate));
    // These attachments represent the relationships required to place an
    // order, from the ESB to the appropiate service providers (subsystems)
    attachment (esb_orderplacement_soap.source,
      enterprise_service_bus.order_placement_req
      (operation online_shop_esb_iopm.make_order));
    attachment (esb_orderplacement_soap.target_service,
      ss_product_management.ws_inventory_mgmt
      (operation product_management_iopm.check_inventory));
    attachment (esb_orderplacement_soap.target_service,
      ss_sales_management.ws_order_mgmt
      (operation sales_management_iopm.register_order));
    attachment (esb_orderplacement_soap.target_service,
      ss_product_management.ws_inventory_mgmt
      (operation product_management_iopm.manage_inventory));
    attachment (esb_orderplacement_soap.target_service,
      ss_sales_management.ws_order_mgmt
      (operation sales_management_iopm.get_order));
    // Services consumed from ss_sales_management, provided by
    // ss_product_management
    attachment (salesmgmt_inventorymgmt_soap.source,
      ss_sales_management.inventory_mgmt_req);
    attachment (salesmgmt_inventorymgmt_soap.target_service,
      ss_product_management.ws_inventory_mgmt
      (operation product_management_iopm.update_inventory));
    // Database access from subsystems
    attachment (ols_ss_db.source, ss_online_shop.req_db_access);
    attachment (ols_ss_db.target_db, shopping_cart_db.provided_access);
    attachment (product_management_ss_db.source,
      ss_product_management.req_db_access);
    attachment (product_management_ss_db.target_db,
      product_management_db.provided_access);
    attachment (customer_management_ss_db.source,
      ss_customer_management.req_db_access);
    attachment (customer_management_ss_db.target_db,
      sales_management_db.provided_access);
    attachment (sales_management_ss_db.source,
      ss_sales_management.req_db_access);
    attachment (sales_management_ss_db.target_db,
      sales_management_db.provided_access);
    }
::
```

Figure 6-14: C&C view for the case study using Sarch in the SOA style - relations

- A relation of type *contains* between each internal service provider (ss_product_management, ss_customer_management and ss_sales_management) and each layer.
- A relation of type *contains* between the application tier and each service consumer (ss_-online_shop).
- A relation of type *contains* between the application tier and each internal service provider (ss_product_management, ss_customer_management and ss_sales_management).
- A relation of type *contains* between the application tier and the enterprise service bus.
- A relation of type *contains* between the data tier and each database (shopping_cart_db, product_management_db and sales_management_db).

6.4 Model-to-Model Transformations

In this step, the model-model transformations were applied to the SOA model in order to generate a new model that follows the specific characteristics of the MSA style. This section shows the different architectural views generated in Sarch for the new model in the MSA style.

The model-to-model transformation process receives as input the SOA model defined in Sarch as well as the following parameters:

- **database_system**: mongodb
- **database_type**: *msa.nosql_database*
- **ms_programming_language**: java
- **web_application_name**: onlineshop_wa
- **web_programming_language**: javascript

The model-to-model transformation process produces a MSA model as output, which is described using the Decomposition, Data Model, C&C and Layered views in the Sarch language. When the generated MSA model is opened in the editor module of Sarch-Studio, the model conforms to the grammar rules defined in Sarch for the MSA architectural style.

As described in section 5.2, the Decomposition view is preserved from the original SOA model, so this view is not shown here.

Grammar in Sarch

```
LayeredViewForSOA:
  'layered_view' '::’
    'elements' '{‘
      (layeredElement += LayeredElement)+
    '}’
    'relations' '{‘
      (layeredRelation += LayeredRelationForSOA)*
    '}’
  '::’
;

LayeredElement:
  Layer | Tier | LayerSegment
;

LayeredRelationForSOA:
  AllowedToUse | AllowedToUseBelow | ContainsAForSOA
;

AllowedToUse:
  AllowedToUseTier | AllowedToUseLayer | AllowedToUseSegment
;

AllowedToUseTier:
  't:' sourceTier=[Tier] 'allowed_to_use' 't:' targetTier=[Tier] ';'
;

AllowedToUseLayer:
  'l:' sourceLayer=[Layer] 'allowed_to_use' 'l:' targetLayer=[Layer] ';'
;

AllowedToUseSegment:
  'ls:' sourceLayerSegment=[LayerSegment] 'allowed_to_use' 'ls:’
      targetLayerSegment=[LayerSegment] ';'
;

AllowedToUseBelow:
  AllowedToUseBelowTier | AllowedToUseBelowLayer | AllowedToUseBelowSegment
;

AllowedToUseBelowTier:
  't:' sourceTier=[Tier] 'allowed_to_use_below' 't:' targetTier=[Tier] ';'
;

AllowedToUseBelowLayer:
  'l:' sourceLayer=[Layer] 'allowed_to_use_below' 'l:' targetLayer=[Layer] ';'
;

AllowedToUseBelowSegment:
  'ls:' sourceLayerSegment=[LayerSegment] 'allowed_to_use_below' 'ls:’
      targetLayerSegment=[LayerSegment] ';'
;

ContainsAForSOA:
  TierAndComponentForSOA | ComponentAndLayerForSOA | LayerAndLayerSegment
;

TierAndComponentForSOA:
  't:' tier=[Tier] 'contains' 'c:' componentElement=[ComponentElementForSOA]
';'
;

ComponentAndLayerForSOA:
  'c:' componentElement=[ComponentElementForSOA] 'contains' 'l:’
      layer=[Layer] ';'
;
```

Layered View

```
layered_view ::
  elements {
    layer presentation;
    layer business_logic;
    layer data_access;
    tier application;
    tier data;
  }
  relations {
    l:presentation allowed_to_use l:business_logic;
    l:business_logic allowed_to_use l:data_access;
    t: application allowed_to_use t: data;
    c:ss_online_shop contains l:presentation;
    c:ss_online_shop contains l:business_logic;
    c:ss_online_shop contains l:data_access;
    c:ss_product_management contains l:presentation;
    c:ss_product_management contains l:business_logic;
    c:ss_product_management contains l:data_access;
    c:ss_customer_management contains l:presentation;
    c:ss_customer_management contains l:business_logic;
    c:ss_customer_management contains l:data_access;
    c:ss_sales_management contains l:presentation;
    c:ss_sales_management contains l:business_logic;
    c:ss_sales_management contains l:data_access;
    c:enterprise_service_bus contains l:business_logic;
    t: application contains c: ss_online_shop;
    t: application contains c: ss_product_management;
    t: application contains c: ss_customer_management;
    t: application contains c: ss_sales_management;
    t: application contains c: enterprise_service_bus;
    t: data contains c: shopping_cart_db;
    t: data contains c: product_management_db;
    t: data contains c: sales_management_db;
  }
::
```

Figure 6-15: Layered view for the case study using Sarch in the SOA style.

Regarding the Data Model view, the relational data models were disaggregated depending on the modules to which the data entities belong. Additionally, the interoperability data models were divided according to the operations and data entities associated with each generated microservice and API gateway. Figure **6-16** shows a sample of the No SQL data models generated in this view for the MSA model, and Figure **6-17** shows a sample of the interoperability data models generated in this view for the MSA model.

```
data_model_view ::
  nosql data_model products_dm {
    elements {
      data_entity product {
        attributes {
          int id;
          string name;
          string description;
          string sku;
          float price;
          int stock_quantity;
          bool active;
          datetime created_at;
          datetime updated_at;
        }
      }
      data_entity historical_product { ... }
      data_entity product_location { ... }
    }
    relations {
      one_to_many (product, historical_product);
      one_to_many (product, product_location);
    }
  }
  nosql data_model customer_information_dm {
    elements {
      data_entity customer { ... }
      data_entity customer_summary { ... }
    }
    relations {
      one_to_one (customer_summary, customer);
    }
  }
  nosql data_model user_account_dm {
    elements {
      data_entity role_def { ... }
      data_entity user { ... }
      data_entity user_role { ... }
    }
    relations {
      many_to_one (user_role, role_def);
      many_to_one (user_role, user);
    }
  }
  nosql data_model shopping_cart_dm {
    elements {
      data_entity shopping_carts { ... }
      data_entity line_items { ... }
    }
    relations {
      one_to_many (shopping_carts, line_items);
    }
  }
```

```
  nosql data_model accounting_transactions_dm {
    elements {
      data_entity account_type { ... }
      data_entity account { ... }
      data_entity transaction { ... }
      data_entity transaction_detail { ... }
      data_entity account_hierarchy { ... }
    }
    relations {
      many_to_one (account, account_type);
      many_to_one (account_hierarchy, account);
      one_to_many (transaction, transaction_detail);
      many_to_one (transaction_detail, account);
    }
  }
  nosql data_model order_handling_dm {
    elements {
      data_entity quotation { ... }
      data_entity quotation_line_item { ... }
      data_entity order { ... }
      data_entity order_line_item { ... }
      data_entity invoice { ... }
      data_entity invoice_line_item { ... }
      data_entity credit_note { ... }
    }
    relations {
      one_to_many (quotation, quotation_line_item);
      one_to_many (order, order_line_item);
      one_to_many (invoice, invoice_line_item);
      one_to_many (invoice, credit_note);
    }
  }
  nosql data_model inventory_dm {
    elements {
      data_entity inventory_reservation { ... }
      data_entity inventory_movement_type { ... }
      data_entity inventory_movement { ... }
    }
    relations {
      many_to_one (inventory_movement, inventory_movement_type);
    }
  }
  nosql data_model shipments_dm {
    elements {
      data_entity transportation_company { ... }
      data_entity shipment { ... }
    }
    relations {
      many_to_one (shipment, transportation_company);
    }
  }
::
```

Figure 6-16: Data Model view generated for the case study using Sarch in the MSA style - No SQL data models.

Regarding the C&C view, a microservice and a database were generated for each module in the Decomposition view, with the respective interoperability and No SQL data models. Also, a Web application and an API gateway were generated. Figure **6-18** shows a sample of the elements

```
data_model_view ::
  interoperability data_model products_iopm {
    elements {
      data_entity product {
        attributes {
          int id;
          string name;
          string description;
          string sku;
          float price;
          ...
        }
      }
      data_entity historical_product { ... }
      data_entity product_location { ... }
      data_entity product_summary { ... }
    }
    relations {
      one_to_many (product, historical_product);
      one_to_many (product, product_location);
    }
    operations {
      operation createProduct { ... }
      operation readProduct { ... }
      operation updateProduct { ... }
      operation deleteProduct { ... }
      ...
      operation search_products { ... }
  }
  interoperability data_model customer_information_iopm {
    elements {
      data_entity customer { ... }
      data_entity customer_summary { ... }
    }
    relations { ... }
    operations {
      ...
      operation authenticate { ... }
    }
  }
  interoperability data_model user_account_iopm {
    elements {
      data_entity role_def { ... }
      data_entity user { ... }
      data_entity user_role { ... }
    }
    relations { ... }
    operations { ... }
  }
  interoperability data_model shopping_cart_iopm {
    elements {
      data_entity shopping_carts { ... }
      data_entity line_items { ... }
    }
    relations { ... }
    operations { ... }
  }
  interoperability data_model inventory_iopm {
    elements {
      data_entity inventory_reservation { ... }
      data_entity inventory_movement_type { ... }
      data_entity inventory_movement { ... }
      data_entity item { ... }
    }
    relations { ... }
    operations {
      ...
      operation check_inventory { ... }
      operation manage_inventory { ... }
      operation update_inventory { ... }
    }
  }
  interoperability data_model order_handling_iopm {
    elements {
      data_entity quotation { ... }
      data_entity quotation_line_item { ... }
      data_entity order { ... }
      data_entity order_line_item { ... }
      data_entity invoice { ... }
      data_entity invoice_line_item { ... }
      data_entity credit_note { ... }
      data_entity item { ... }
    }
    relations { ... }
    operations {
      ...
      operation register_order {}
      operation get_order { ... }
    }
  }
  interoperability data_model shipments_iopm {
    elements {
      data_entity transportation_company { ... }
      data_entity shipment { ... }
    }
    relations { ... }
    operations { ... }
  }
  interoperability data_model accounting_transactions_iopm {
    elements {
      data_entity account_type { ... }
      data_entity account { ... }
      data_entity transaction { ... }
      data_entity transaction_detail { ... }
      data_entity account_hierarchy { ... }
    }
    relations { ... }
    operations { ... }
  }
  interoperability data_model api_gw_iopm {
    elements {
      data_entity customer { ... }
      data_entity product_summary { ... }
      data_entity item { ... }
    }
    relations { ... }
    operations {
      operation authenticate { ... }
      operation search_products { ... }
      operation make_order { ... }
    }
  }
  interoperability data_model order_placement_iopm {
    elements {
      data_entity item { ... }
    }
    relations {}
    operations {
      operation make_order { ... }
    }
  }
::
```

Figure 6-17: Data Model view generated for the case study using Sarch in the MSA style - interoperability data models.

(components and connectors) generated in this view for the MSA model, and Figure **6-19** shows a sample of the relations (attachments) generated in this view for the MSA model.

```
component_and_connector_view ::
  elements {
    programming_languages {
      java;
      javascript;
    }
    db_systems {
      mongodb;
    }
    component msa.microservice products_ms (programming_language java) {
      port provided_api api (iop_model products_iopm);
      port requested_dbaccess dbreq;
    }
    component msa.nosql_database products_db (nosql_data_model products_dm)
            (db_system mongodb) {
      port provided_dbaccess dbprov;
    }
    connector msa.db_access products_ms_db {
      role consumer src;
      role provider tgt;
    }
    component msa.microservice inventory_ms (programming_language java) {
      port provided_api api (iop_model inventory_iopm);
      port requested_dbaccess dbreq;
    }
    component msa.nosql_database inventory_db (nosql_data_model
            inventory_dm) (db_system mongodb) { ... }
    connector msa.db_access inventory_ms_db { ... }
    component msa.microservice shipments_ms (programming_language java) {
      port provided_api api (iop_model shipments_iopm);
      port requested_dbaccess dbreq;
    }
    component msa.nosql_database shipments_db (nosql_data_model shipments_dm)
            (db_system mongodb) { ... }
    connector msa.db_access shipments_ms_db { ... }
    component msa.microservice user_account_ms (programming_language java) {
      port provided_api api;
      port requested_dbaccess dbreq;
    }
    component msa.nosql_database user_account_db (db_system mongodb) { ... }
    connector msa.db_access user_account_ms_db { ... }
    component msa.microservice customer_information_ms (programming_language
            java) {
      port provided_api api (iop_model customer_information_iopm);
      port requested_dbaccess dbreq;
    }
    component msa.nosql_database customer_information_db (nosql_data_model
            customer_information_dm) (db_system mongodb) { ... }
    connector msa.db_access customer_information_ms_db { ... }
    component msa.microservice order_handling_ms (programming_language java) {
      port provided_api api (iop_model order_handling_iopm);
      port requested_dbaccess dbreq;
      port requested_api apireq;
    }
    connector msa.rest order_handling_inventory_mgmt_rest {
      role consumer src;
      role provider tgt;
    }
    component msa.nosql_database order_handling_db (nosql_data_model
            order_handling_dm) (db_system mongodb) { ... }
    connector msa.db_access order_handling_ms_db { ... }
    component msa.microservice accounting_transactions_ms (programming_language
            java) {
      port provided_api api (iop_model accounting_transactions_iopm);
      port requested_dbaccess dbreq;
    }
    component msa.nosql_database accounting_transactions_db (nosql_data_model
        accounting_transactions_dm) (db_system mongodb) { ... }
    connector msa.db_access accounting_transactions_ms_db { ... }
    component msa.microservice shopping_cart_ms (programming_language java) {
      port provided_api api (iop_model shopping_cart_iopm);
      port requested_dbaccess dbreq;
    }
    component msa.nosql_database shopping_cart_db (nosql_data_model
              shopping_cart_dm) (db_system mongodb) {
      port provided_dbaccess dbprov;
    }
    connector msa.db_access shopping_cart_ms_db { ... }
    component msa.web_application onlineshop_wa (programming_language
              javascript) {
      port requested_api user_session;
      port requested_api product_search;
      port requested_api order_placement;
      port requested_api shopping_cart;
    }
    component msa.api_gateway api_gw {
      port provided_api authentication_api (iop_model api_gw_iopm);
      port requested_api authentication_req;
      port provided_api product_catalog_api (iop_model api_gw_iopm);
      port requested_api product_catalog_req;
      port provided_api order_placement_api (iop_model api_gw_iopm);
      port requested_api order_placement_req;
      port provided_api shopping_cart_api (iop_model api_gw_iopm);
      port requested_api shopping_cart_req;
      port provided_api accounting_transactions_api (iop_model api_gw_iopm);
      port requested_api accounting_transactions_req;
      port provided_api user_account_api (iop_model api_gw_iopm);
      port requested_api user_account_req;
      port provided_api shipments_api (iop_model api_gw_iopm);
      port requested_api shipments_req;
    }
    component msa.microservice order_placement_ms (programming_language java) {
      port provided_api api (iop_model order_placement_iopm);
      port requested_api apireq;
    }
    connector msa.rest order_placement_inventory_mgmt_rest {
      role consumer src;
      role provider tgt;
    }
    connector msa.rest order_placement_order_mgmt_rest {
      role consumer src;
      role provider tgt;
    }
    connector msa.rest wa_ag_authentication_rest {
      role consumer src;
      role provider tgt;
    }
    connector msa.rest ag_customer_information_ms_rest {
      role consumer src;
      role provider tgt;
    }
    connector msa.rest wa_ag_product_catalog_rest { ... }
    connector msa.rest ag_products_ms_rest  { ... }
    connector msa.rest wa_ag_order_placement_rest { ... }
    connector msa.rest ag_order_placement_ms_rest { ... }
    connector msa.rest wa_ag_shopping_cart_rest { ... }
    connector msa.rest ag_shopping_cart_ms_rest { ... }
    connector msa.rest ag_accounting_transactions_ms_rest { ... }
    connector msa.rest ag_user_account_ms_rest { ... }
    connector msa.rest ag_shipments_ms_rest { ... }
  }
```

Figure 6-18: C&C view generated for the case study using Sarch in the MSA style - elements.

Figure **6-20** shows the Layered view generated for the MSA model, according to the transformation rules defined.

```
relations {
  attachment (products_ms_db.src, products_ms.dbreq);
  attachment (products_ms_db.tgt, products_db.dbprov);
  attachment (inventory_ms_db.src, inventory_ms.dbreq);
  attachment (inventory_ms_db.tgt, inventory_db.dbprov);
  attachment (shipments_ms_db.src, shipments_ms.dbreq);
  attachment (shipments_ms_db.tgt, shipments_db.dbprov);
  attachment (user_account_ms_db.src, user_account_ms.dbreq);
  attachment (user_account_ms_db.tgt, user_account_db.dbprov);
  attachment (customer_information_ms_db.src, customer_information_ms.dbreq);
  attachment (customer_information_ms_db.tgt, customer_information_db.dbprov);
  attachment (order_handling_ms_db.src, order_handling_ms.dbreq);
  attachment (order_handling_ms_db.tgt, order_handling_db.dbprov);
  attachment (order_handling_inventory_mgmt_rest.src, order_handling_ms.apireq);
  attachment (order_handling_inventory_mgmt_rest.tgt, inventory_ms.api (operation inventory_iopm.update_inventory));
  attachment (accounting_transactions_ms_db.src, accounting_transactions_ms.dbreq);
  attachment (accounting_transactions_ms_db.tgt, accounting_transactions_db.dbprov);
  attachment (shopping_cart_ms_db.src, shopping_cart_ms.dbreq);
  attachment (shopping_cart_ms_db.tgt, shopping_cart_db.dbprov);
  attachment (order_placement_inventory_mgmt_rest.src, order_placement_ms.apireq);
  attachment (order_placement_inventory_mgmt_rest.tgt, inventory_ms.api (operation inventory_iopm.check_inventory));
  attachment (order_placement_order_mgmt_rest.src, order_placement_ms.apireq);
  attachment (order_placement_order_mgmt_rest.tgt, order_handling_ms.api (operation order_handling_iopm.register_order));
  attachment (order_placement_inventory_mgmt_rest.tgt, inventory_ms.api (operation inventory_iopm.manage_inventory));
  attachment (order_placement_order_mgmt_rest.tgt, order_handling_ms.api (operation order_handling_iopm.get_order));
  attachment (wa_ag_authentication_rest.src, onlineshop_wa.user_session);
  attachment (wa_ag_authentication_rest.tgt, api_gw.authentication_api (operation api_gw_iopm.authenticate));
  attachment (ag_customer_information_ms_rest.src, api_gw.authentication_req);
  attachment (ag_customer_information_ms_rest.tgt, customer_information_ms.customer_information_api (operation
      customer_information_iopm.authenticate));
  attachment (wa_ag_product_catalog_rest.src, onlineshop_wa.product_search);
  attachment (wa_ag_product_catalog_rest.tgt, api_gw.product_catalog_api (operation api_gw_iopm.search_products));
  attachment (ag_products_ms_rest.src, api_gw.product_catalog_req);
  attachment (ag_products_ms_rest.tgt, products_ms.products_api (operation products_iopm.search_products));
  attachment (wa_ag_order_placement_rest.src, onlineshop_wa.order_placement);
  attachment (wa_ag_order_placement_rest.tgt, api_gw.order_placement_api (operation api_gw_iopm.make_order));
  attachment (ag_order_placement_ms_rest.src, api_gw.order_placement_req);
  attachment (ag_order_placement_ms_rest.tgt, order_placement_ms.order_placement_api (operation order_placement_iopm.make_order));
  attachment (wa_ag_shopping_cart_rest.src, onlineshop_wa.shopping_cart);
  attachment (wa_ag_shopping_cart_rest.tgt, api_gw.shopping_cart_api);
  attachment (ag_shopping_cart_ms_rest.src, api_gw.shopping_cart_req);
  attachment (ag_shopping_cart_ms_rest.tgt, shopping_cart_ms.shopping_cart_api);
  attachment (ag_accounting_transactions_ms_rest.src, api_gw.accounting_transactions_req);
  attachment (ag_accounting_transactions_ms_rest.tgt, accounting_transactions_ms.accounting_transactions_api);
  attachment (ag_user_account_ms_rest.src, api_gw.user_account_req);
  attachment (ag_user_account_ms_rest.tgt, user_account_ms.user_account_api);
  attachment (ag_shipments_ms_rest.src, api_gw.shipments_req);
  attachment (ag_shipments_ms_rest.tgt, shipments_ms.shipments_api);
}
::
```

Figure 6-19: C&C view generated for the case study using Sarch in the MSA style - relations.

```
layered_view ::
  elements {
    layer presentation;
    layer business_logic;
    layer data_access;
    tier presentation;
    tier logic;
    tier data;
  }
  relations {
    l: presentation allowed_to_use l: business_logic;
    l: business_logic allowed_to_use l: data_access;
    t: presentation allowed_to_use t: logic;
    t: logic allowed_to_use t: data;
    c: onlineshop_wa contains l: presentation;
    c: api_gw contains l: business_logic;
    c: products_ms contains l: business_logic;
    c: products_ms contains l: data_access;
    c: inventory_ms contains l: business_logic;
    c: inventory_ms contains l: data_access;
    c: shipments_ms contains l: business_logic;
    c: shipments_ms contains l: data_access;
    c: user_account_ms contains l: business_logic;
    c: user_account_ms contains l: data_access;
    c: customer_information_ms contains l: business_logic;
    c: customer_information_ms contains l: data_access;
    c: order_handling_ms contains l: business_logic;
    c: order_handling_ms contains l: data_access;
    c: accounting_transactions_ms contains l:business_logic;
    c: accounting_transactions_ms contains l:data_access;
    c: shopping_cart_ms contains l: business_logic;
    c: shopping_cart_ms contains l: data_access;
    c: order_placement_ms contains l: business_logic;
    c: order_placement_ms contains l: data_access;
    t: presentation contains c: onlineshop_wa;
    t: logic contains c: products_ms;
    t: logic contains c: inventory_ms;
    t: logic contains c: shipments_ms;
    t: logic contains c: user_account_ms;
    t: logic contains c: customer_information_ms;
    t: logic contains c: order_handling_ms;
    t: logic contains c: accounting_transactions_ms;
    t: logic contains c: shopping_cart_ms;
    t: logic contains c: order_placement_ms;
    t: logic contains c: api_gw;
    t: data contains c: products_db;
    t: data contains c: inventory_db;
    t: data contains c: shipments_db;
    t: data contains c: user_account_db;
    t: data contains c: customer_information_db;
    t: data contains c: order_handling_db;
    t: data contains c: accounting_transactions_db;
    t: data contains c: shopping_cart_db;
  }
::
```

Figure 6-20: Layered view generated for the case study using Sarch in the MSA style.

6.5 Generated Model for a MSA-Based System

Based on the MSA model resulting from applying the model-to-model transformations from the SOA model, in this step the system is graphically designed under the MSA architectural style. Figure **6-21** shows the Data Model view with the resulting No SQL data models. Figure **6-22** shows the C&C view with the resulting components, connectors, and relations; for simplicity, this diagram does not show the details of the associated interoperability data models and operations. Figure **6-23** shows the Layered view.

Additional refinements may be performed from this starting point with respect to the architectural design done for the case study in the MSA style illustrated in figure **6-8**:

- Change the type of some databases to *relational*.
- Change the type of some connectors according to specific communication requirements.
- Use the component of type *msa.storage* to fulfill specific object storage needs.
- Create relations between order_placement_ms and shopping_cart_ms to complement order placement requirements, which involves defining the necessary connectors and attachments as well as creating the operations in the respective interoperability data model.
- Create specific operations in the respective interoperability data models associated with the provided API ports for shipments_ms, accounting_transactions_ms and user_account_ms.

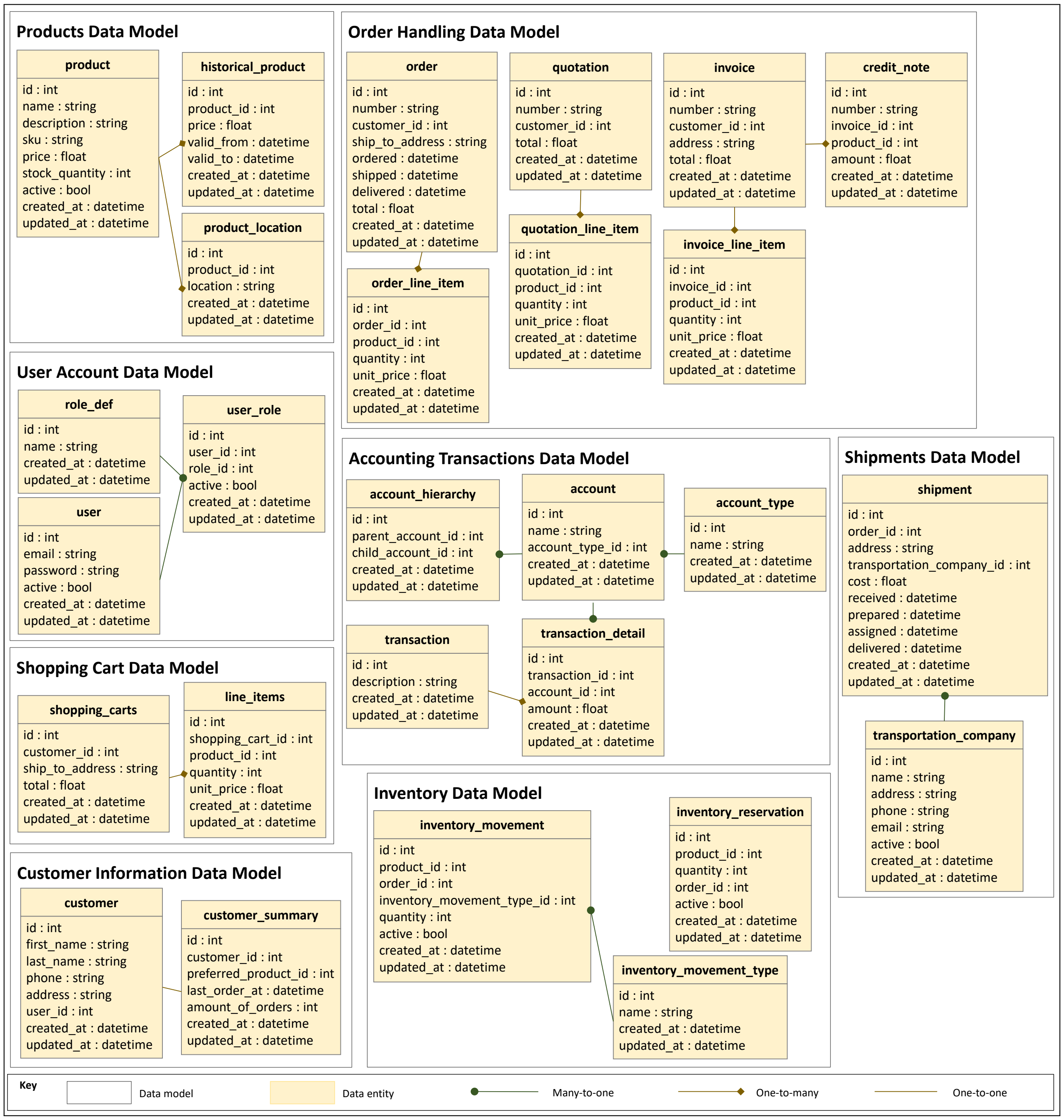


Figure 6-21: Graphical representation of the resulting Data Model view for the MSA model.

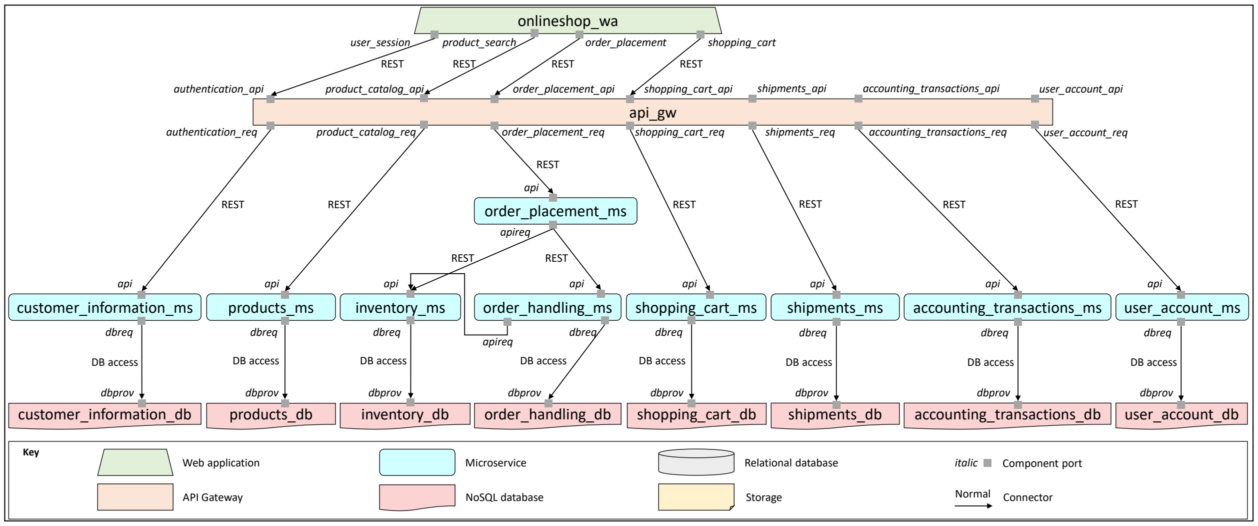


Figure 6-22: Graphical representation of the resulting C&C view for the MSA model.

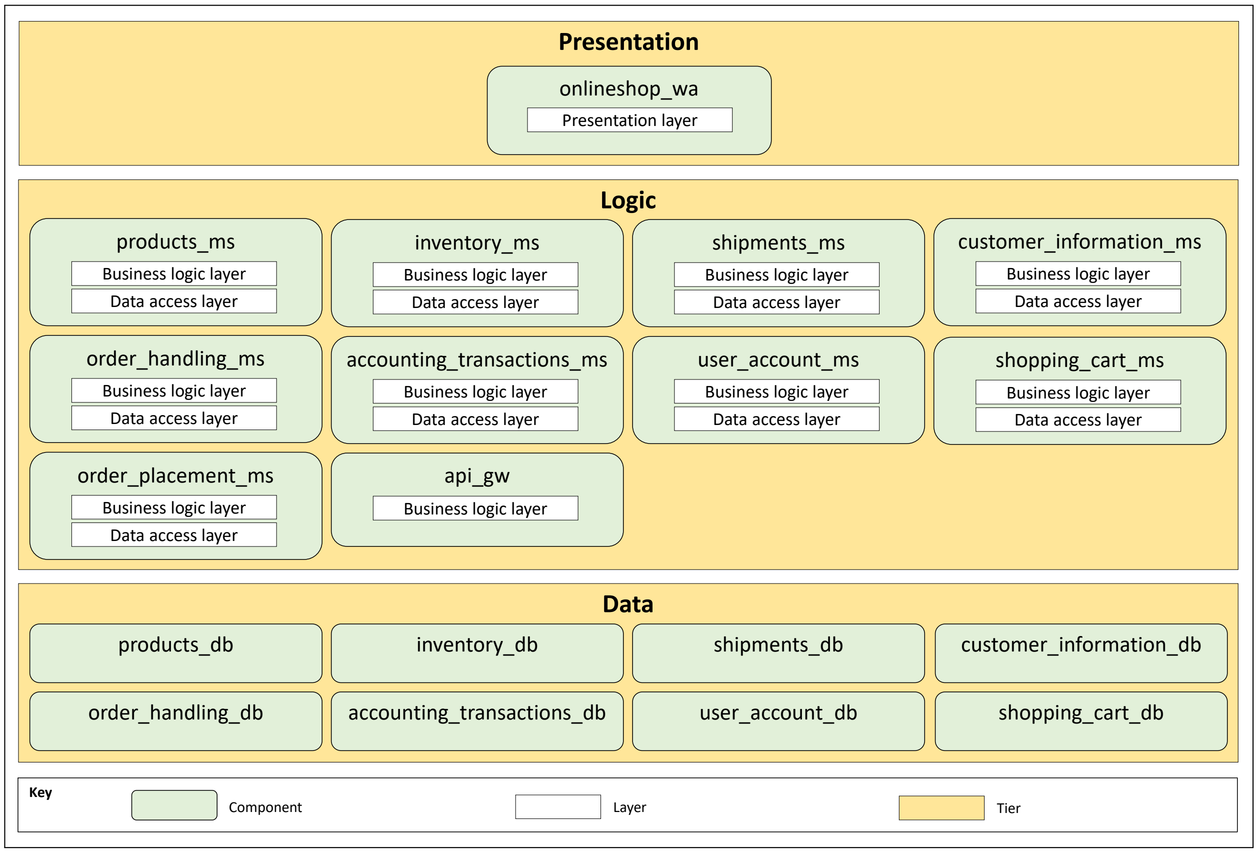


Figure 6-23: Graphical representation of the resulting Layered view for the MSA model.

6.6 Evolution Model Validation

As shown through the case study, the evolution model generated a valid architectural representation in Sarch of an MSA-based system from the architecture of a SOA-based system, which complies with the elements, relations and properties of the MSA architectural style.

Next, the evolution model will be validated based on the architecture of a generic system, which is described as follows:

1. Figure **6-24** shows the SOA source architecture of a generic system, which is represented using the decomposition view and the C&C view.
2. The architecture of the source system is modeled using Sarch, which is shown in figures **6-25**, **6-26**, and **6-27**, using the decomposition and C&C views.
3. The transformation rules are applied to generate an MSA model that is valid from the point of view of the architectural style. Some of these model-to-model transformations are shown in figure **6-28**.
4. Figures **6-29** and **6-30** show the elements and relations that are part of the C&C view, for the MSA model generated after applying the transformation rules. The generated MSA model is represented in the Sarch language and complies with the grammar defined for the architectural style.
5. Finally, the system is graphically represented under the MSA architectural style, based on the MSA model resulting from applying the model-to-model transformations from the SOA model, which is shown in figure **6-31** and represents the MSA target architecture in the evolution model.

For the transformation process, there is a formal and rigorous mapping between the elements, the relations and the properties of each architectural style. This allows to have a more closed target set for the MSA architectural style where microservices architectures can be generated from service-oriented architectures, taking into account that the result in MSA formally complies with all the characteristics from the point of view of the architectural style. This ensures that the transformation model works for any service-oriented architecture and that it provides the foundation for an MSA-based system. Additionally, the model generated in MSA supplies the complete functionality of the system by generating a microservice for each of the subsystem modules present in the Decomposition view, as well as a microservice for each service provided in the SOA model based on the component ports.

It is important to bear in mind that the result of the transformation process is a base architecture for MSA, but that this base architecture will allow or require the addition of new architectural features to meet all the characteristics and needs of the system.

The proposed evolution model is based on two fields of Software Engineering, on the one hand there are aspects related to Software Architecture, and on the other hand MDE is being used as a paradigm for the automation of the model-to-model transformation. The core of the validation is given from the point of view of the internal process that is being done at the language level to be able to convert an input into an output, where the input is the formal elements of the SOA architectural style and the output is the formal elements MSA architectural style.

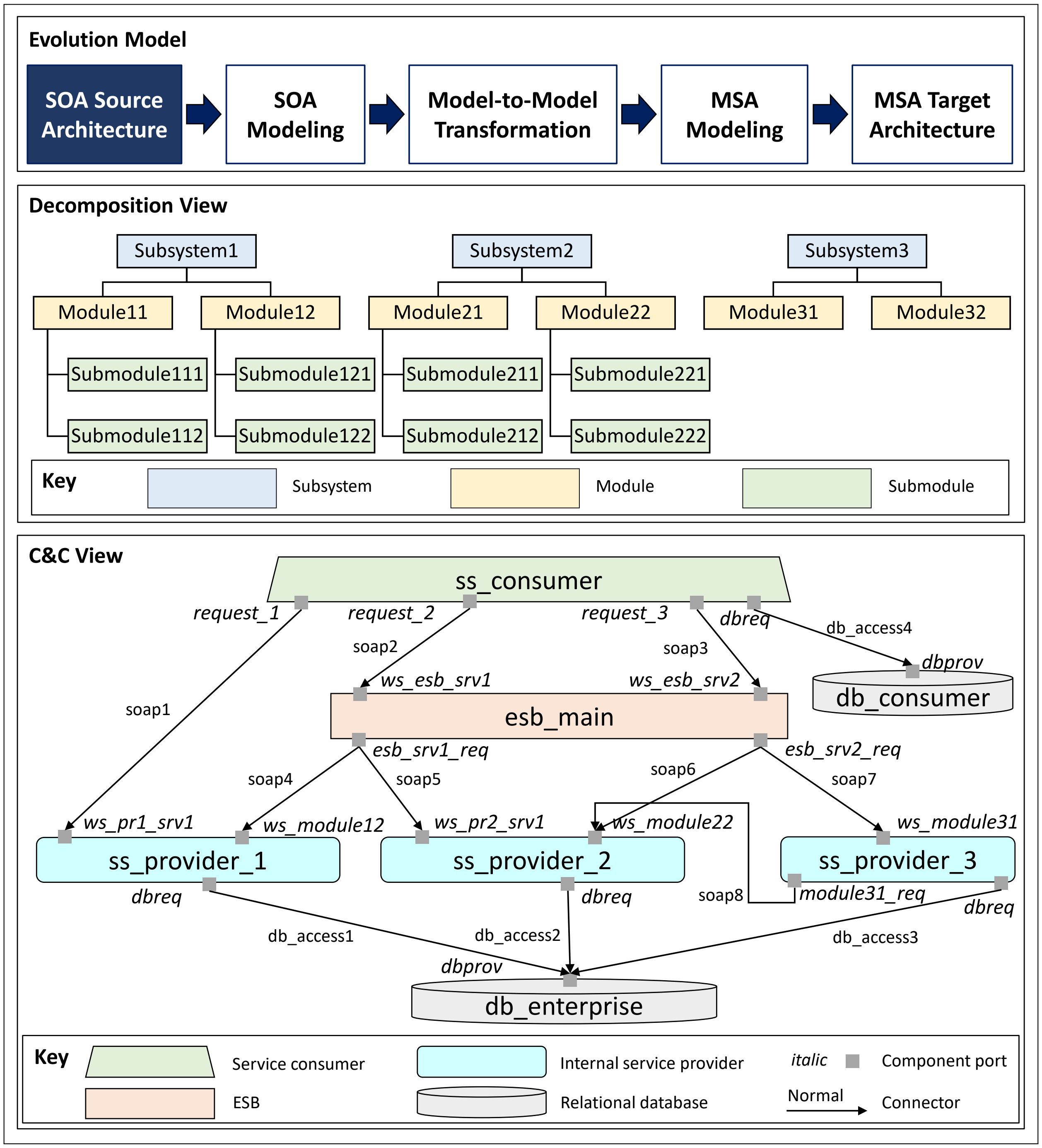
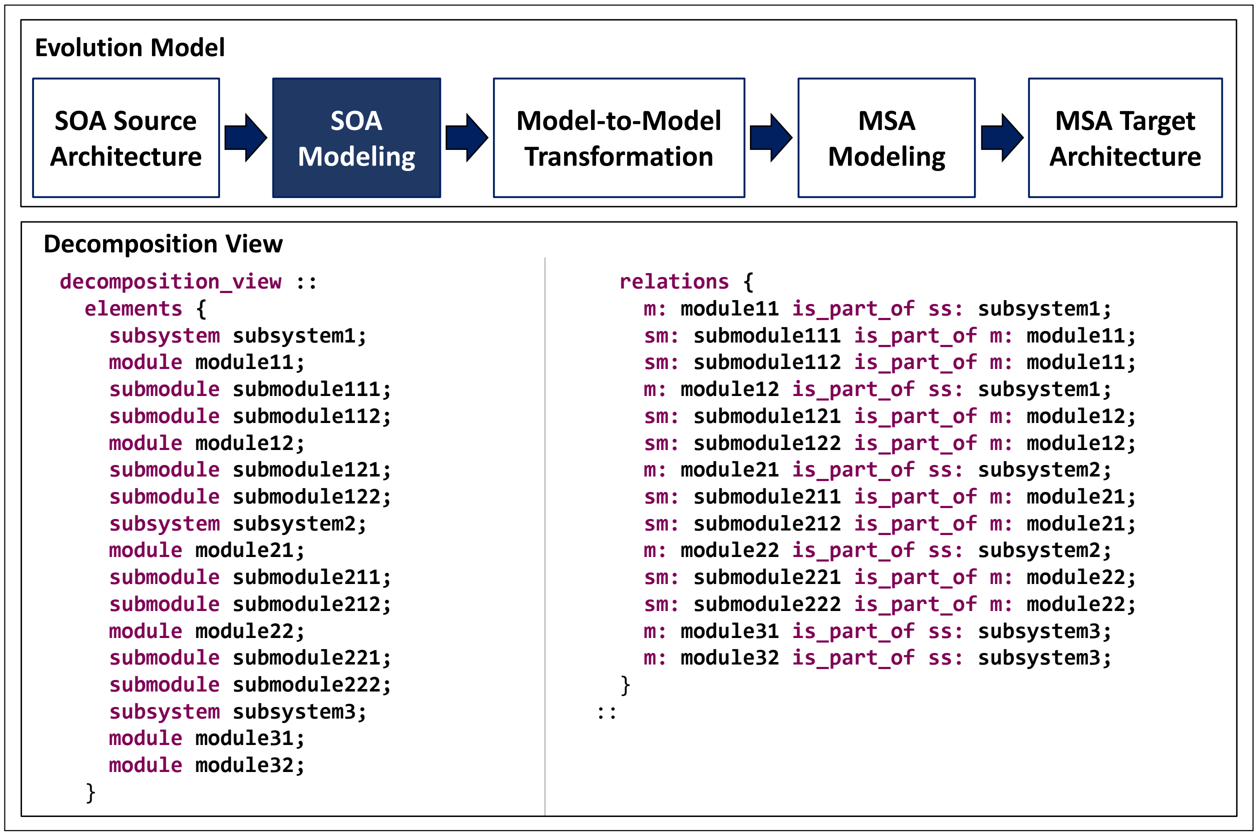
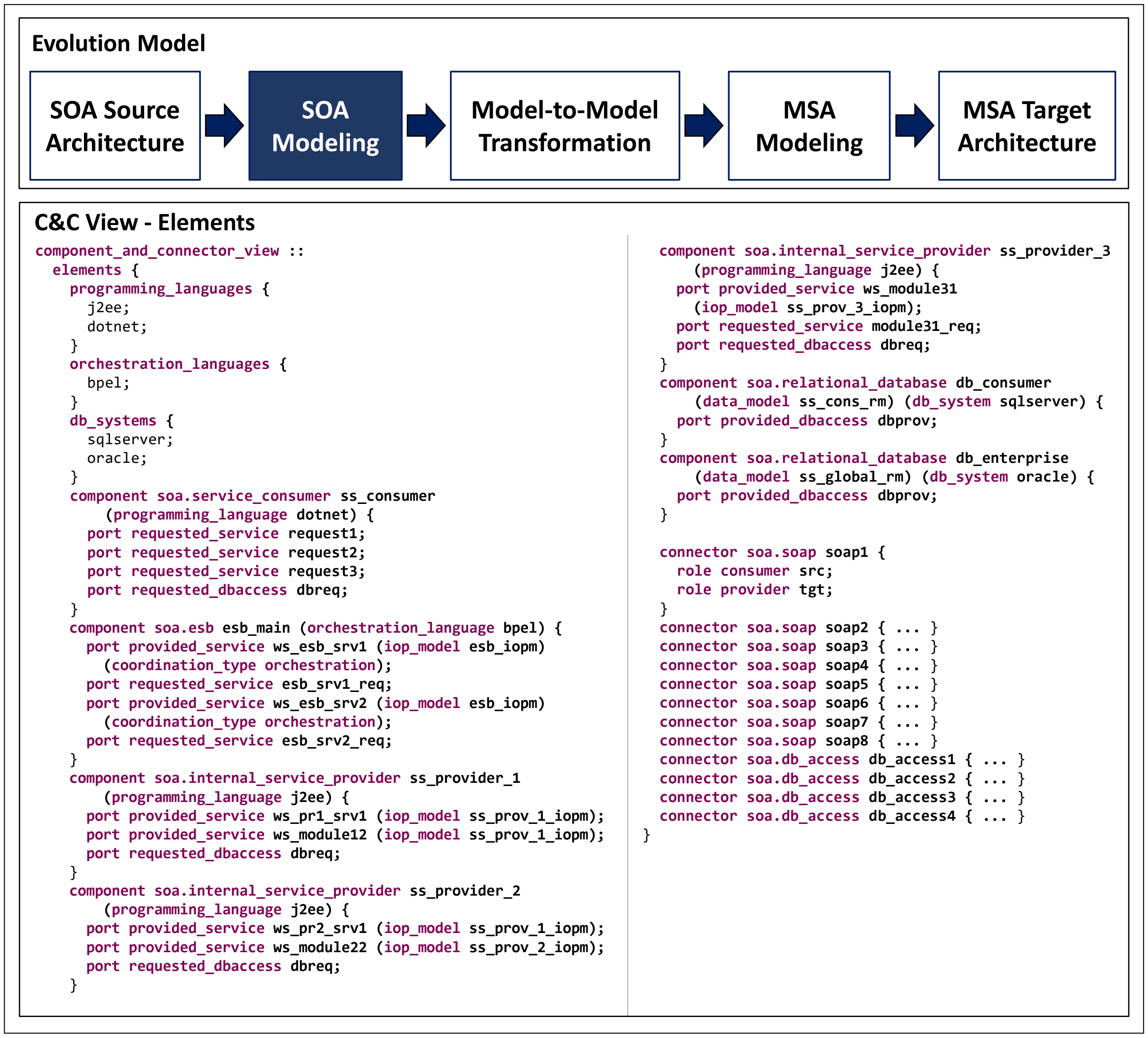


Figure 6-24: Evolution model validation - SOA source architecture.



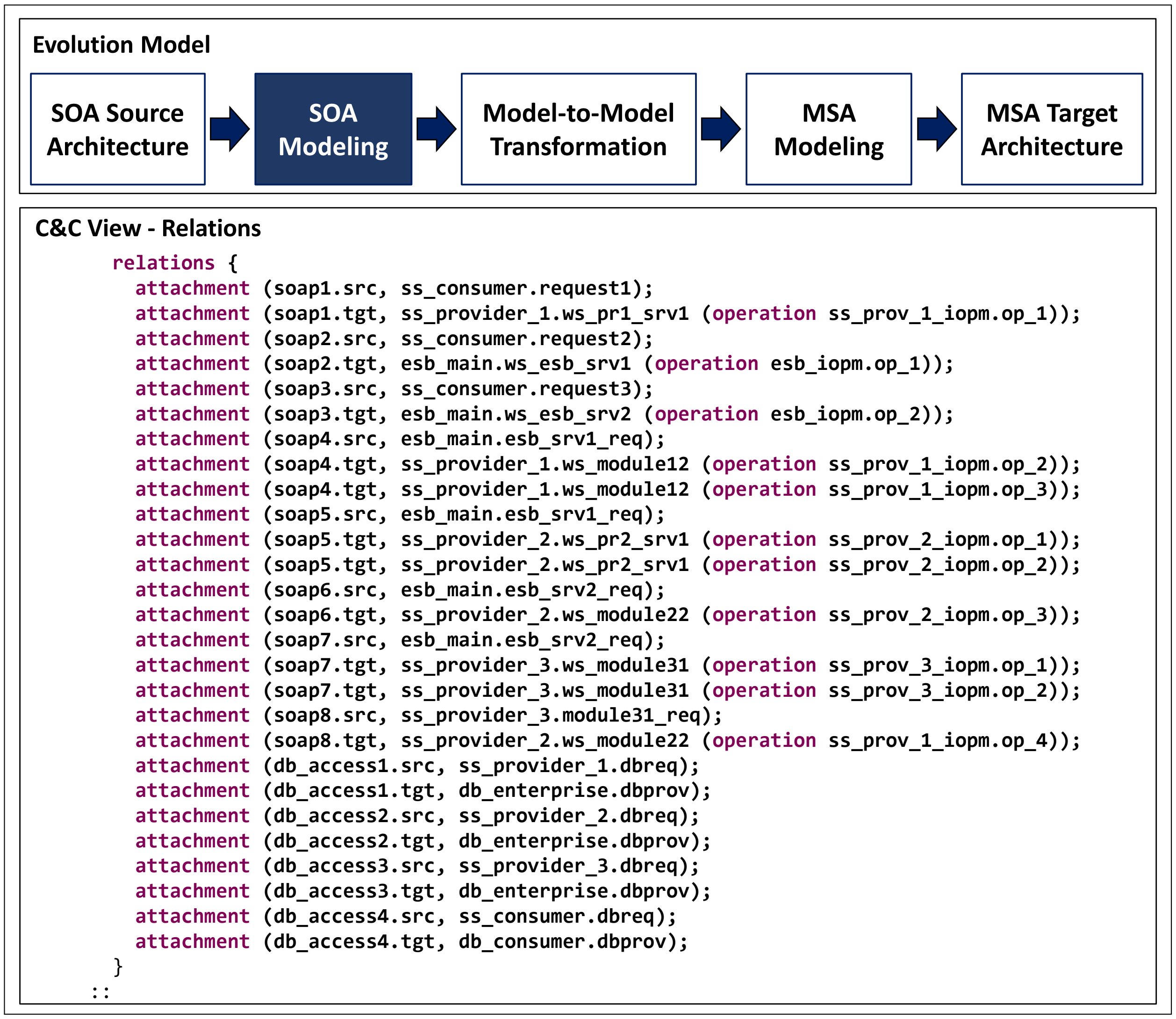
```
decomposition_view ::
  elements {
    subsystem subsystem1;
    module module11;
    submodule submodule111;
    submodule submodule112;
    module module12;
    submodule submodule121;
    submodule submodule122;
    subsystem subsystem2;
    module module21;
    submodule submodule211;
    submodule submodule212;
    module module22;
    submodule submodule221;
    submodule submodule222;
    subsystem subsystem3;
    module module31;
    module module32;
  }
  relations {
    m: module11 is_part_of ss: subsystem1;
    sm: submodule111 is_part_of m: module11;
    sm: submodule112 is_part_of m: module11;
    m: module12 is_part_of ss: subsystem1;
    sm: submodule121 is_part_of m: module12;
    sm: submodule122 is_part_of m: module12;
    m: module21 is_part_of ss: subsystem2;
    sm: submodule211 is_part_of m: module21;
    sm: submodule212 is_part_of m: module21;
    m: module22 is_part_of ss: subsystem2;
    sm: submodule221 is_part_of m: module22;
    sm: submodule222 is_part_of m: module22;
    m: module31 is_part_of ss: subsystem3;
    m: module32 is_part_of ss: subsystem3;
  }
::
```

Figure 6-25: Evolution model validation - SOA Modeling - part I.



```
component_and_connector_view ::
  elements {
    programming_languages {
      j2ee;
      dotnet;
    }
    orchestration_languages {
      bpel;
    }
    db_systems {
      sqlserver;
      oracle;
    }
    component soa.service_consumer ss_consumer
        (programming_language dotnet) {
      port requested_service request1;
      port requested_service request2;
      port requested_service request3;
      port requested_dbaccess dbreq;
    }
    component soa.esb esb_main (orchestration_language bpel) {
      port provided_service ws_esb_srv1 (iop_model esb_iopm)
        (coordination_type orchestration);
      port requested_service esb_srv1_req;
      port provided_service ws_esb_srv2 (iop_model esb_iopm)
        (coordination_type orchestration);
      port requested_service esb_srv2_req;
    }
    component soa.internal_service_provider ss_provider_1
        (programming_language j2ee) {
      port provided_service ws_pr1_srv1 (iop_model ss_prov_1_iopm);
      port provided_service ws_module12 (iop_model ss_prov_1_iopm);
      port requested_dbaccess dbreq;
    }
    component soa.internal_service_provider ss_provider_2
        (programming_language j2ee) {
      port provided_service ws_pr2_srv1 (iop_model ss_prov_1_iopm);
      port provided_service ws_module22 (iop_model ss_prov_2_iopm);
      port requested_dbaccess dbreq;
    }
    component soa.internal_service_provider ss_provider_3
        (programming_language j2ee) {
      port provided_service ws_module31
        (iop_model ss_prov_3_iopm);
      port requested_service module31_req;
      port requested_dbaccess dbreq;
    }
    component soa.relational_database db_consumer
        (data_model ss_cons_rm) (db_system sqlserver) {
      port provided_dbaccess dbprov;
    }
    component soa.relational_database db_enterprise
        (data_model ss_global_rm) (db_system oracle) {
      port provided_dbaccess dbprov;
    }

    connector soa.soap soap1 {
      role consumer src;
      role provider tgt;
    }
    connector soa.soap soap2 { ... }
    connector soa.soap soap3 { ... }
    connector soa.soap soap4 { ... }
    connector soa.soap soap5 { ... }
    connector soa.soap soap6 { ... }
    connector soa.soap soap7 { ... }
    connector soa.soap soap8 { ... }
    connector soa.db_access db_access1 { ... }
    connector soa.db_access db_access2 { ... }
    connector soa.db_access db_access3 { ... }
    connector soa.db_access db_access4 { ... }
  }
```

Figure 6-26: Evolution model validation - SOA Modeling - part II.



```
      relations {
        attachment (soap1.src, ss_consumer.request1);
        attachment (soap1.tgt, ss_provider_1.ws_pr1_srv1 (operation ss_prov_1_iopm.op_1));
        attachment (soap2.src, ss_consumer.request2);
        attachment (soap2.tgt, esb_main.ws_esb_srv1 (operation esb_iopm.op_1));
        attachment (soap3.src, ss_consumer.request3);
        attachment (soap3.tgt, esb_main.ws_esb_srv2 (operation esb_iopm.op_2));
        attachment (soap4.src, esb_main.esb_srv1_req);
        attachment (soap4.tgt, ss_provider_1.ws_module12 (operation ss_prov_1_iopm.op_2));
        attachment (soap4.tgt, ss_provider_1.ws_module12 (operation ss_prov_1_iopm.op_3));
        attachment (soap5.src, esb_main.esb_srv1_req);
        attachment (soap5.tgt, ss_provider_2.ws_pr2_srv1 (operation ss_prov_2_iopm.op_1));
        attachment (soap5.tgt, ss_provider_2.ws_pr2_srv1 (operation ss_prov_2_iopm.op_2));
        attachment (soap6.src, esb_main.esb_srv2_req);
        attachment (soap6.tgt, ss_provider_2.ws_module22 (operation ss_prov_2_iopm.op_3));
        attachment (soap7.src, esb_main.esb_srv2_req);
        attachment (soap7.tgt, ss_provider_3.ws_module31 (operation ss_prov_3_iopm.op_1));
        attachment (soap7.tgt, ss_provider_3.ws_module31 (operation ss_prov_3_iopm.op_2));
        attachment (soap8.src, ss_provider_3.module31_req);
        attachment (soap8.tgt, ss_provider_2.ws_module22 (operation ss_prov_1_iopm.op_4));
        attachment (db_access1.src, ss_provider_1.dbreq);
        attachment (db_access1.tgt, db_enterprise.dbprov);
        attachment (db_access2.src, ss_provider_2.dbreq);
        attachment (db_access2.tgt, db_enterprise.dbprov);
        attachment (db_access3.src, ss_provider_3.dbreq);
        attachment (db_access3.tgt, db_enterprise.dbprov);
        attachment (db_access4.src, ss_consumer.dbreq);
        attachment (db_access4.tgt, db_consumer.dbprov);
      }
    ::
```

Figure 6-27: Evolution model validation - SOA Modeling - part III.

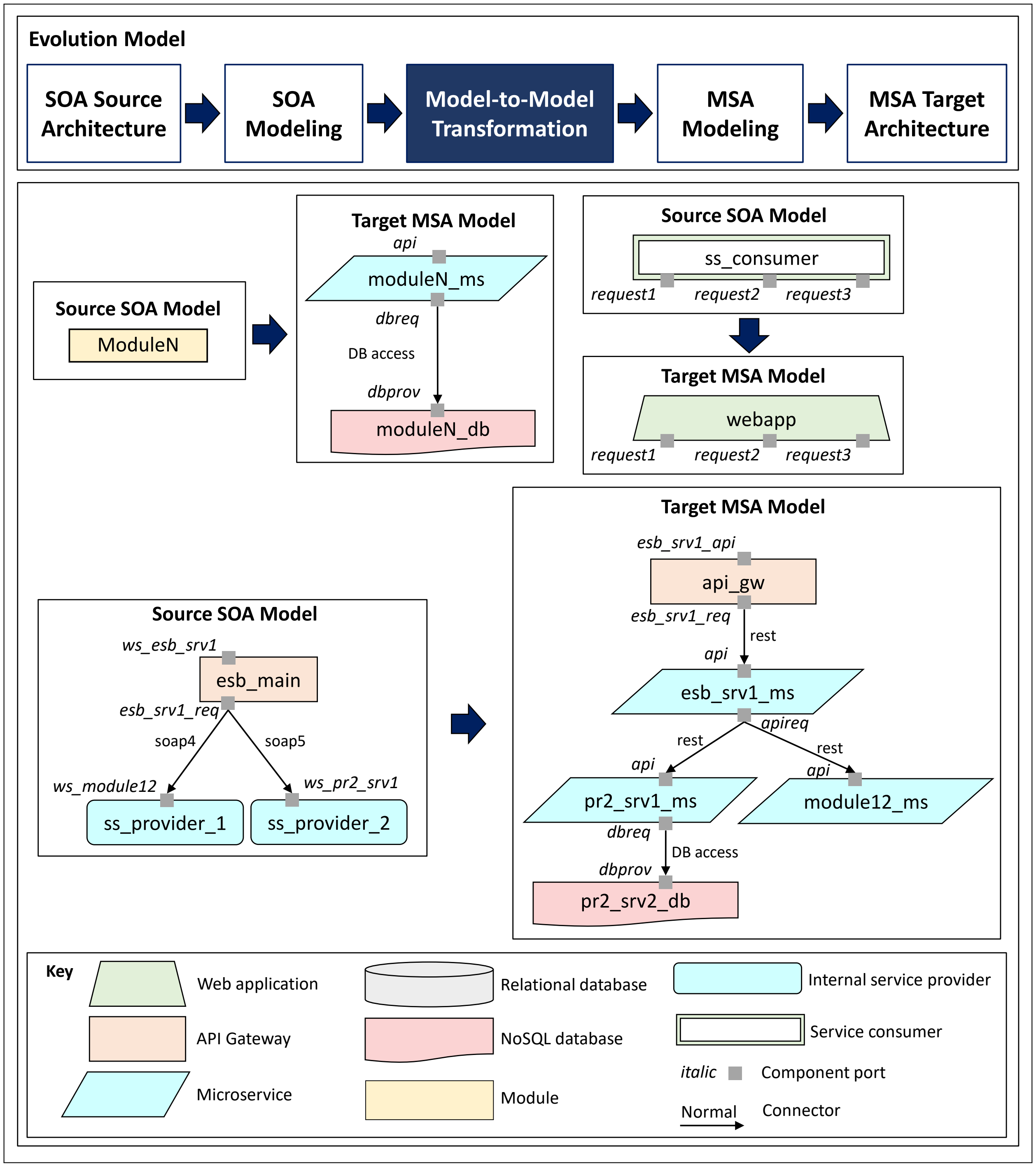
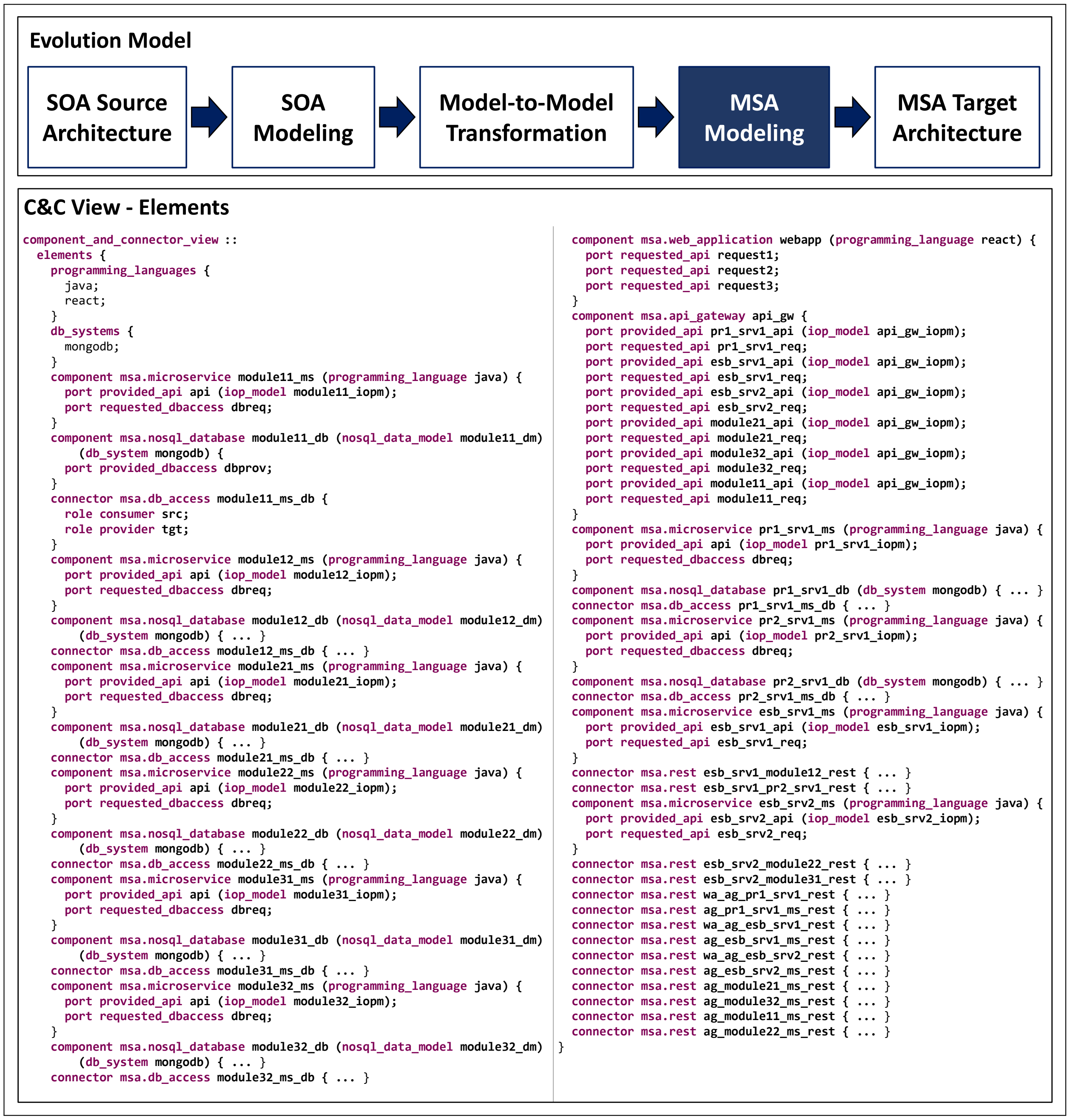
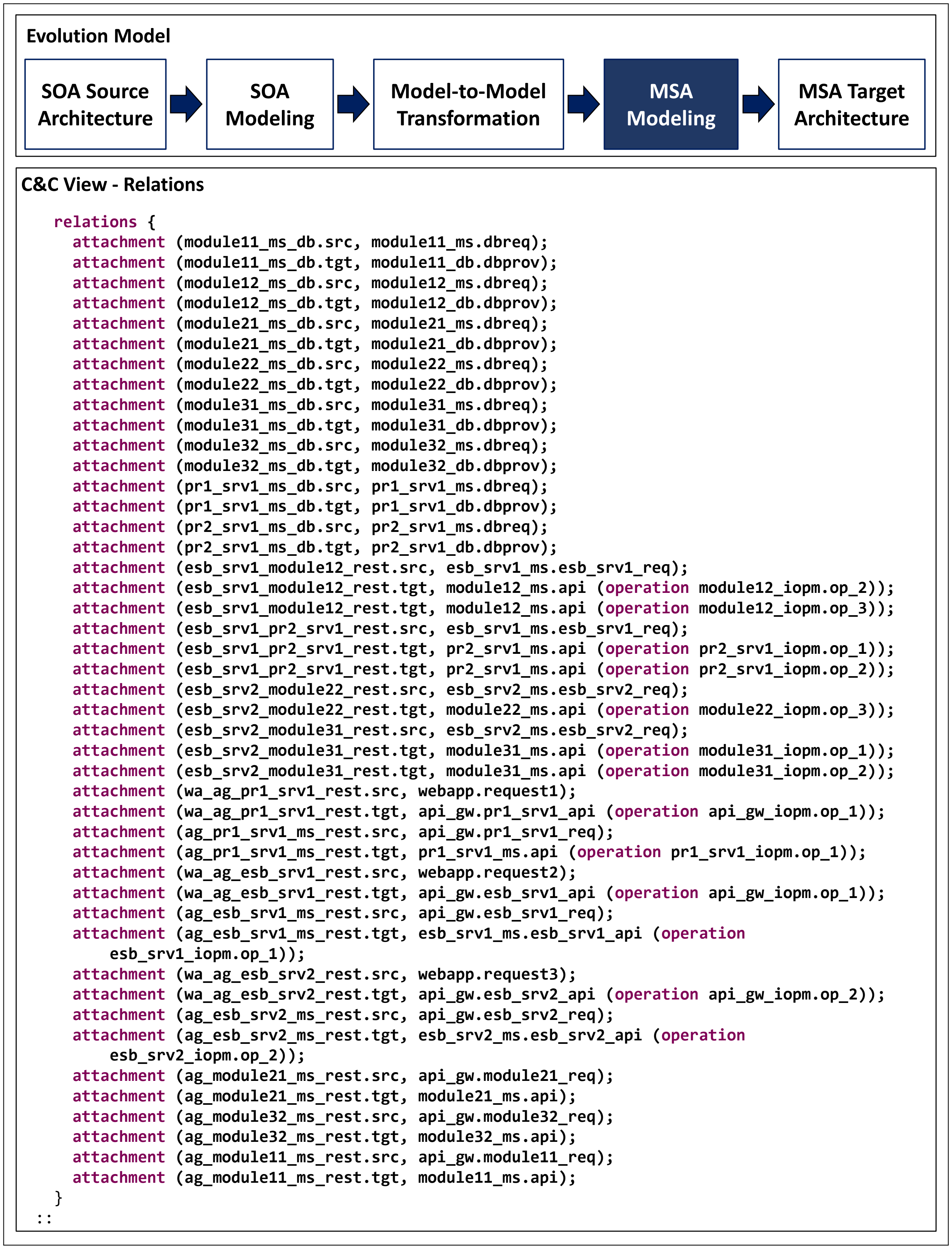


Figure 6-28: Evolution model validation - model-to-model transformations.



```
component_and_connector_view ::
  elements {
    programming_languages {
      java;
      react;
    }
    db_systems {
      mongodb;
    }
    component msa.microservice module11_ms (programming_language java) {
      port provided_api api (iop_model module11_iopm);
      port requested_dbaccess dbreq;
    }
    component msa.nosql_database module11_db (nosql_data_model module11_dm)
        (db_system mongodb) {
      port provided_dbaccess dbprov;
    }
    connector msa.db_access module11_ms_db {
      role consumer src;
      role provider tgt;
    }
    component msa.microservice module12_ms (programming_language java) {
      port provided_api api (iop_model module12_iopm);
      port requested_dbaccess dbreq;
    }
    component msa.nosql_database module12_db (nosql_data_model module12_dm)
        (db_system mongodb) { ... }
    connector msa.db_access module12_ms_db { ... }
    component msa.microservice module21_ms (programming_language java) {
      port provided_api api (iop_model module21_iopm);
      port requested_dbaccess dbreq;
    }
    component msa.nosql_database module21_db (nosql_data_model module21_dm)
        (db_system mongodb) { ... }
    connector msa.db_access module21_ms_db { ... }
    component msa.microservice module22_ms (programming_language java) {
      port provided_api api (iop_model module22_iopm);
      port requested_dbaccess dbreq;
    }
    component msa.nosql_database module22_db (nosql_data_model module22_dm)
        (db_system mongodb) { ... }
    connector msa.db_access module22_ms_db { ... }
    component msa.microservice module31_ms (programming_language java) {
      port provided_api api (iop_model module31_iopm);
      port requested_dbaccess dbreq;
    }
    component msa.nosql_database module31_db (nosql_data_model module31_dm)
        (db_system mongodb) { ... }
    connector msa.db_access module31_ms_db { ... }
    component msa.microservice module32_ms (programming_language java) {
      port provided_api api (iop_model module32_iopm);
      port requested_dbaccess dbreq;
    }
    component msa.nosql_database module32_db (nosql_data_model module32_dm)
        (db_system mongodb) { ... }
    connector msa.db_access module32_ms_db { ... }
    component msa.web_application webapp (programming_language react) {
      port requested_api request1;
      port requested_api request2;
      port requested_api request3;
    }
    component msa.api_gateway api_gw {
      port provided_api pr1_srv1_api (iop_model api_gw_iopm);
      port requested_api pr1_srv1_req;
      port provided_api esb_srv1_api (iop_model api_gw_iopm);
      port requested_api esb_srv1_req;
      port provided_api esb_srv2_api (iop_model api_gw_iopm);
      port requested_api esb_srv2_req;
      port provided_api module21_api (iop_model api_gw_iopm);
      port requested_api module21_req;
      port provided_api module32_api (iop_model api_gw_iopm);
      port requested_api module32_req;
      port provided_api module11_api (iop_model api_gw_iopm);
      port requested_api module11_req;
    }
    component msa.microservice pr1_srv1_ms (programming_language java) {
      port provided_api api (iop_model pr1_srv1_iopm);
      port requested_dbaccess dbreq;
    }
    component msa.nosql_database pr1_srv1_db (db_system mongodb) { ... }
    connector msa.db_access pr1_srv1_ms_db { ... }
    component msa.microservice pr2_srv1_ms (programming_language java) {
      port provided_api api (iop_model pr2_srv1_iopm);
      port requested_dbaccess dbreq;
    }
    component msa.nosql_database pr2_srv1_db (db_system mongodb) { ... }
    connector msa.db_access pr2_srv1_ms_db { ... }
    component msa.microservice esb_srv1_ms (programming_language java) {
      port provided_api esb_srv1_api (iop_model esb_srv1_iopm);
      port requested_api esb_srv1_req;
    }
    connector msa.rest esb_srv1_module12_rest { ... }
    connector msa.rest esb_srv1_pr2_srv1_rest { ... }
    component msa.microservice esb_srv2_ms (programming_language java) {
      port provided_api esb_srv2_api (iop_model esb_srv2_iopm);
      port requested_api esb_srv2_req;
    }
    connector msa.rest esb_srv2_module22_rest { ... }
    connector msa.rest esb_srv2_module31_rest { ... }
    connector msa.rest wa_ag_pr1_srv1_rest { ... }
    connector msa.rest ag_pr1_srv1_ms_rest { ... }
    connector msa.rest wa_ag_esb_srv1_rest { ... }
    connector msa.rest ag_esb_srv1_ms_rest { ... }
    connector msa.rest wa_ag_esb_srv2_rest { ... }
    connector msa.rest ag_esb_srv2_ms_rest { ... }
    connector msa.rest ag_module21_ms_rest { ... }
    connector msa.rest ag_module32_ms_rest { ... }
    connector msa.rest ag_module11_ms_rest { ... }
    connector msa.rest ag_module22_ms_rest { ... }
  }
```

Figure 6-29: Evolution model validation - MSA Modeling - part I.



```
    relations {
      attachment (module11_ms_db.src, module11_ms.dbreq);
      attachment (module11_ms_db.tgt, module11_db.dbprov);
      attachment (module12_ms_db.src, module12_ms.dbreq);
      attachment (module12_ms_db.tgt, module12_db.dbprov);
      attachment (module21_ms_db.src, module21_ms.dbreq);
      attachment (module21_ms_db.tgt, module21_db.dbprov);
      attachment (module22_ms_db.src, module22_ms.dbreq);
      attachment (module22_ms_db.tgt, module22_db.dbprov);
      attachment (module31_ms_db.src, module31_ms.dbreq);
      attachment (module31_ms_db.tgt, module31_db.dbprov);
      attachment (module32_ms_db.src, module32_ms.dbreq);
      attachment (module32_ms_db.tgt, module32_db.dbprov);
      attachment (pr1_srv1_ms_db.src, pr1_srv1_ms.dbreq);
      attachment (pr1_srv1_ms_db.tgt, pr1_srv1_db.dbprov);
      attachment (pr2_srv1_ms_db.src, pr2_srv1_ms.dbreq);
      attachment (pr2_srv1_ms_db.tgt, pr2_srv1_db.dbprov);
      attachment (esb_srv1_module12_rest.src, esb_srv1_ms.esb_srv1_req);
      attachment (esb_srv1_module12_rest.tgt, module12_ms.api (operation module12_iopm.op_2));
      attachment (esb_srv1_module12_rest.tgt, module12_ms.api (operation module12_iopm.op_3));
      attachment (esb_srv1_pr2_srv1_rest.src, esb_srv1_ms.esb_srv1_req);
      attachment (esb_srv1_pr2_srv1_rest.tgt, pr2_srv1_ms.api (operation pr2_srv1_iopm.op_1));
      attachment (esb_srv1_pr2_srv1_rest.tgt, pr2_srv1_ms.api (operation pr2_srv1_iopm.op_2));
      attachment (esb_srv2_module22_rest.src, esb_srv2_ms.esb_srv2_req);
      attachment (esb_srv2_module22_rest.tgt, module22_ms.api (operation module22_iopm.op_3));
      attachment (esb_srv2_module31_rest.src, esb_srv2_ms.esb_srv2_req);
      attachment (esb_srv2_module31_rest.tgt, module31_ms.api (operation module31_iopm.op_1));
      attachment (esb_srv2_module31_rest.tgt, module31_ms.api (operation module31_iopm.op_2));
      attachment (wa_ag_pr1_srv1_rest.src, webapp.request1);
      attachment (wa_ag_pr1_srv1_rest.tgt, api_gw.pr1_srv1_api (operation api_gw_iopm.op_1));
      attachment (ag_pr1_srv1_ms_rest.src, api_gw.pr1_srv1_req);
      attachment (ag_pr1_srv1_ms_rest.tgt, pr1_srv1_ms.api (operation pr1_srv1_iopm.op_1));
      attachment (wa_ag_esb_srv1_rest.src, webapp.request2);
      attachment (wa_ag_esb_srv1_rest.tgt, api_gw.esb_srv1_api (operation api_gw_iopm.op_1));
      attachment (ag_esb_srv1_ms_rest.src, api_gw.esb_srv1_req);
      attachment (ag_esb_srv1_ms_rest.tgt, esb_srv1_ms.esb_srv1_api (operation
          esb_srv1_iopm.op_1));
      attachment (wa_ag_esb_srv2_rest.src, webapp.request3);
      attachment (wa_ag_esb_srv2_rest.tgt, api_gw.esb_srv2_api (operation api_gw_iopm.op_2));
      attachment (ag_esb_srv2_ms_rest.src, api_gw.esb_srv2_req);
      attachment (ag_esb_srv2_ms_rest.tgt, esb_srv2_ms.esb_srv2_api (operation
          esb_srv2_iopm.op_2));
      attachment (ag_module21_ms_rest.src, api_gw.module21_req);
      attachment (ag_module21_ms_rest.tgt, module21_ms.api);
      attachment (ag_module32_ms_rest.src, api_gw.module32_req);
      attachment (ag_module32_ms_rest.tgt, module32_ms.api);
      attachment (ag_module11_ms_rest.src, api_gw.module11_req);
      attachment (ag_module11_ms_rest.tgt, module11_ms.api);
    }
  ::
```

Figure 6-30: Evolution model validation - MSA Modeling - part II.

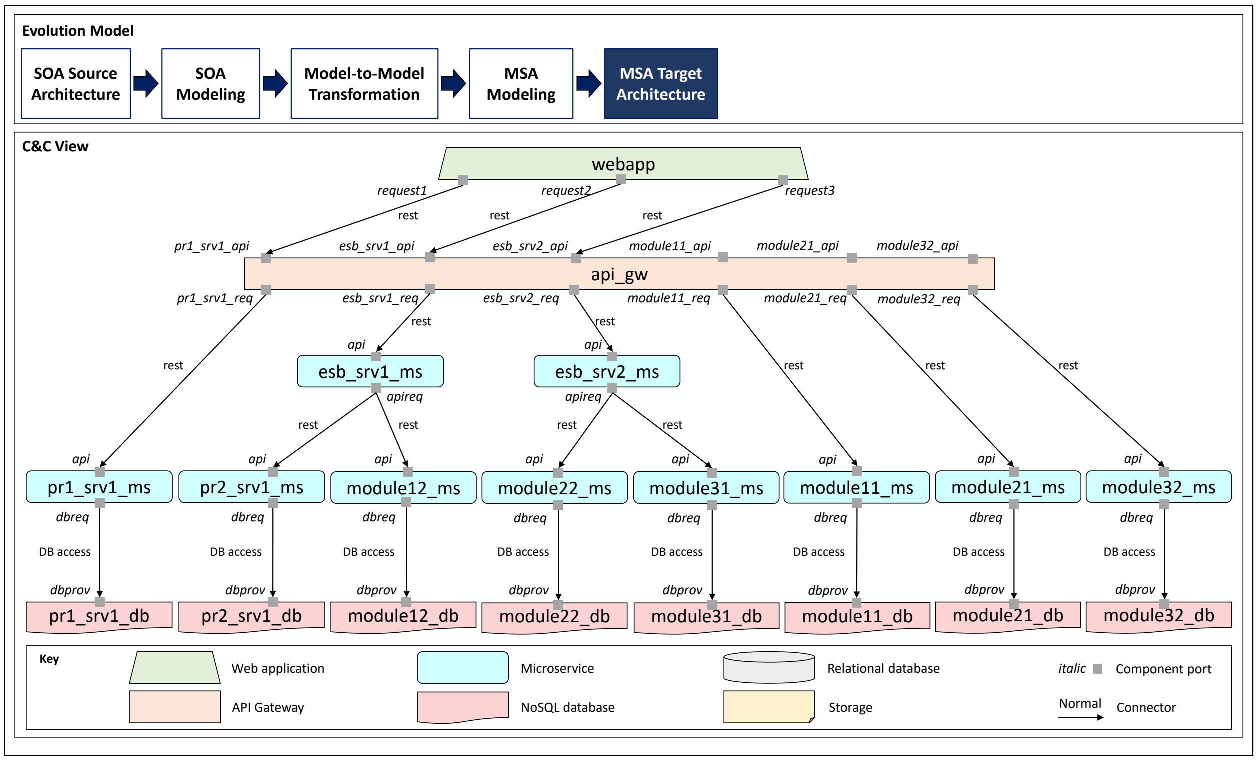


Figure 6-31: Evolution model validation - MSA target architecture.

7 Conclusions and Future Work

7.1 Conclusions

In this work, a new strategy has been proposed to evolve service-based architectures, specifically from service-oriented architectures to microservices architectures. First, a rigorous and formal analysis of the elements, relations and properties of each SOA and MSA architectural style was done, and the Decomposition, Data Model, C&C and Layered views were used to represent software system architectures for SOA and MSA. As a result, the Sarch language was complemented as an ADL for service-based architectures, specifically incorporating the grammar rules necessary to represent software architectures in the SOA and MSA styles. On the other hand, a formal mapping was done between the elements, relations and properties of the SOA and MSA styles, which allowed defining a set of model-to-model transformations that receives an SOA model as input and generates an MSA model as a result. Thanks to this formal process, it is possible to generate a microservices architecture from any service-oriented architecture, which guarantees the effectiveness of the evolution model proposed from the elements, relations and properties of each architectural style.

7.2 Future Work

As future work, it is proposed to include the use of quality attributes in the model-to-model transformation process that the architecture of the target software system must meet. On the other hand, include transformation processes that allow to generate architectural models represented in Sarch from source code. Additionally, continue the extension of the Sarch language to allow the description of software architectures with architectural styles in a specific way, as well as to incorporate other architectural views. Finally, extend the Sarch-Studio tool to allow the graphical representation of each of the architectural views that are part of Sarch.

7.3 Academic Production

The academic production derived from this research, so far, includes:

- Conference paper entitled "A Component-Based Evolution Model for Service-Based Software Architectures" in the *2020 11th IEEE International Conference on Software Engineering and*

Service Science (ICSESS 2020). https://doi.org/10.1109/ICSESS49938.2020.9237747. In this paper, a first version of the evolution model was proposed with main emphasis on component-and-connector and decomposition architectural views.

Bibliography

[1] *Eclipse Modeling Framework*. `https://www.eclipse.org/modeling/emf/`

[2] *Xtend*. `https://www.eclipse.org/xtend/`

[3] *Xtext*. `https://www.eclipse.org/Xtext/`

[4] ACEVEDO, Cesar Augusto J. ; GOMEZ Y JORGE, Juan P. ; PATINO, Ivan R.: Methodology to transform a monolithic software into a microservice architecture. In: *2017 6th International Conference on Software Process Improvement (CIMPS)*, IEEE, oct 2017. – ISBN 978-1-5386-3230-7, 1–6

[5] ALSHUQAYRAN, Nuha ; ALI, Nour ; EVANS, Roger: A Systematic Mapping Study in Microservice Architecture. In: *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, IEEE, nov 2016. – ISBN 978-1-5090-4781-9, 44–51

[6] BALALAIE, Armin ; HEYDARNOORI, Abbas ; JAMSHIDI, Pooyan: Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. In: *IEEE Software* 33 (2016), may, Nr. 3, 42–52. `http://dx.doi.org/10.1109/MS.2016.64`. – DOI 10.1109/MS.2016.64

[7] BARAIS, Olivier ; LE MEUR, Anne F. ; DUCHIEN, Laurence ; LAWALL, Julia: Software Architecture Evolution. Version: 2008. `http://dx.doi.org/10.1007/978-3-540-76440-3_10`. In: *Software Evolution*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2008. – DOI 10.1007/978-3-540-76440-3_10, 233–262

[8] BARNES, Jeffrey M. ; GARLAN, David ; SCHMERL, Bradley: Evolution styles: foundations and models for software architecture evolution. In: *Software & Systems Modeling* 13 (2014), may, Nr. 2, S. 649–678. `http://dx.doi.org/10.1007/s10270-012-0301-9`. – DOI 10.1007/s10270-012-0301-9

[9] BASS, Len. ; CLEMENTS, Paul ; KAZMAN, Rick.: *Software architecture in practice*. Addison-Wesley, 2013. – 589 S. – ISBN 9780321815736

[10] BENGURIA, Gorka ; LARRUCEA, Xabier ; ELVESÆTER, Brian ; NEPLE, Tor ; BEARDSMORE, Anthony ; FRIESS, Michael: A Platform Independent Model for Service Oriented Architectures. In: *Enterprise Interoperability* (2007), S. 23–32. `http://dx.doi.org/10.1007/978-1-84628-714-5_3`. – DOI 10.1007/978-1-84628-714-5_3

[11] Berrio-Charry, Eduardo ; Vergara-Vargas, Jeisson ; Umana-Acosta, Henry: A Component-Based Evolution Model for Service-Based Software Architectures. In: *2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS)*, IEEE, oct 2020. – ISBN 978–1–7281–6578–3, 111–115

[12] Bettini, Lorenzo: *Implementing Domain-Specific Languages with Xtext and Xtend*. 2nd Editio. Packt Publishing, 2016. – ISBN 978–1–78646–496–5

[13] Brambilla, Marco ; Cabot, Jordi ; Wimmer, Manuel: *Model-Driven Software Engineering in Practice*. Second. 2017. http://dx.doi.org/10.2200/S00751ED2V01Y201701SWE004. http://dx.doi.org/10.2200/S00751ED2V01Y201701SWE004. – ISBN 9781627059886

[14] Breivold, Hongyu P. ; Crnkovic, Ivica ; Eriksson, Peter J.: Analyzing Software Evolvability. In: *2008 32nd Annual IEEE International Computer Software and Applications Conference*, IEEE, 2008. – ISBN 978–0–7695–3262–2, 327–330

[15] Breivold, Hongyu P. ; Crnkovic, Ivica ; Larsson, Magnus: A systematic review of software architecture evolution research. In: *Information and Software Technology* 54 (2012), jan, Nr. 1, 16–40. http://dx.doi.org/10.1016/j.infsof.2011.06.002. – DOI 10.1016/j.infsof.2011.06.002

[16] Britto, Ricardo ; Smite, Darja ; Damm, Lars-Ola: Software Architects in Large-Scale Distributed Projects: An Ericsson Case Study. In: *IEEE Software* 33 (2016), nov, Nr. 6, 48–55. http://dx.doi.org/10.1109/MS.2016.146. – DOI 10.1109/MS.2016.146. – ISSN 0740–7459

[17] Bucchiarone, Antonio ; Dragoni, Nicola ; Dustdar, Schahram ; Larsen, Stephan T. ; Mazzara, Manuel: From Monolithic to Microservices: An Experience Report from the Banking Domain. In: *IEEE Software* 35 (2018), may, Nr. 3, 50–55. http://dx.doi.org/10.1109/MS.2018.2141026. – DOI 10.1109/MS.2018.2141026. – ISSN 0740–7459

[18] Chen, Rui ; Li, Shanshan ; Li, Zheng: From Monolith to Microservices: A Dataflow-Driven Approach. In: *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, IEEE, dec 2017. – ISBN 978–1–5386–3681–7, 466–475

[19] Clements, P. ; Bachmann, F. ; Bass, L. ; Garlan, D. ; Ivers, J. ; Little, R. ; Nord, R. ; Stafford, J.: *Documenting software architectures: views and beyond*. Addison-Wesley, 2011. – 582 S. http://dx.doi.org/10.1109/icse.2003.1201264. http://dx.doi.org/10.1109/icse.2003.1201264. – ISBN 0321552687

[20] Di Francesco, P. ; Malavolta, I. ; Lago, P.: Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. In: *Proceedings - 2017 IEEE International Conference on Software Architecture, ICSA 2017*, 2017. – ISBN 9781509057290, S. 21–30

[21] Di Francesco, Paolo ; Lago, Patricia ; Malavolta, Ivano: Architecting with microservices: A systematic mapping study. In: *Journal of Systems and Software* 150 (2019), apr, S. 77–97. http://dx.doi.org/10.1016/j.jss.2019.01.001. – DOI 10.1016/j.jss.2019.01.001. – ISSN 01641212

[22] Erl, Thomas.: *Service-oriented architecture : concepts, technology, and design*. Prentice Hall Professional Technical Reference, 2005. – 760 S. https://www.oreilly.com/library/view/service-oriented-architecture-concepts/0131858580/. – ISBN 0131858580

[23] Fan, Chen-Yuan ; Ma, Shang-Pin: Migrating Monolithic Mobile Application to Microservice Architecture: An Experiment Report. In: *2017 IEEE International Conference on AI & Mobile Services (AIMS)*, IEEE, jun 2017. – ISBN 978–1–5386–1999–5, 109–112

[24] Fernando, Erick ; Touriano, Derist ; Rico: Impact of Service-Oriented Architecture adoption in information system. In: *2015 2nd International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*, IEEE, oct 2015. – ISBN 978–1–4799–9861–6, 52–55

[25] Fowler, Martin ; Parsons, Rebecca: *Domain-Specific Languages*. Addison-Wesley Professional, 2010. – ISBN 978–0321712943

[26] Francesco, Paolo D.: Architecting Microservices. In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, IEEE, apr 2017. – ISBN 978–1–5090–4793–2, 224–229

[27] Furda, Andrei ; Fidge, Colin ; Zimmermann, Olaf ; Kelly, Wayne ; Barros, Alistair: Migrating Enterprise Legacy Source Code to Microservices: On Multitenancy, Statefulness, and Data Consistency. In: *IEEE Software* 35 (2018), may, Nr. 3, 63–72. http://dx.doi.org/10.1109/MS.2017.440134612. – DOI 10.1109/MS.2017.440134612. – ISSN 0740–7459

[28] Godfrey, Michael W. ; German, Daniel M.: The past, present, and future of software evolution. In: *2008 Frontiers of Software Maintenance*, IEEE, sep 2008. – ISBN 978–1–4244–2654–6, 129–138

[29] Gouigoux, Jean-Philippe ; Tamzalit, Dalila: From Monolith to Microservices: Lessons Learned on an Industrial Migration to a Web Oriented Architecture. In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, IEEE, apr 2017. – ISBN 978–1–5090–4793–2, 62–65

[30] Granchelli, Giona ; Cardarelli, Mario ; Francesco, Paolo D. ; Malavolta, Ivano ; Iovino, Ludovico ; Salle, Amleto D.: Towards recovering the software architecture of microservice-based systems. In: *Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings*, Institute of Electrical and Electronics Engineers Inc., jun 2017. – ISBN 9781509047932, S. 46–53

[31] Hassan, Adel ; Oussalah, Mourad: Meta-Evolution Style for Software Architecture Evolution. Version: 2016. http://dx.doi.org/10.1007/978-3-662-49192-8_39. 2016. – DOI 10.1007/978-3-662-49192-8_39, S. 478–489

[32] Hassan, Sara ; Ali, Nour ; Bahsoon, Rami: Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity. In: *2017 IEEE International Conference on Software Architecture (ICSA)*, IEEE, apr 2017. – ISBN 978-1-5090-5729-0, S. 1–10

[33] IEEE Computer Society. Software Engineering Standards Committee. ; Institute of Electrical and Electronics Engineers. ; IEEE-SA Standards Board.: *IEEE recommended practice for architectural description of software-intensive systems*. Institute of Electrical and Electronics Engineers, 2000. – 23 S. https://ieeexplore-ieee-org.ezproxy.unal.edu.co/document/875998. – ISBN 0738125180

[34] Indrasiri, Kasun ; Siriwardena, Prabath: *Microservices for the Enterprise*. 1st Editio. Berkeley, CA : Apress, 2018. http://dx.doi.org/10.1007/978-1-4842-3858-5. http://dx.doi.org/10.1007/978-1-4842-3858-5. – ISBN 978-1-4842-3857-8

[35] Jammes, F. ; Smit, H.: Service-Oriented Paradigms in Industrial Automation. In: *IEEE Transactions on Industrial Informatics* 1 (2005), feb, Nr. 1, 62–70. http://dx.doi.org/10.1109/TII.2005.844419. – DOI 10.1109/TII.2005.844419. – ISSN 1551-3203

[36] Jia, Xiangyang ; Ying, Shi ; Zhang, Tao ; Cao, Honghua ; Xie, Dan: A new architecture description language for service-oriented architecture. In: *Proceedings of the 6th International Conference on Grid and Cooperative Computing, GCC 2007*, 2007. – ISBN 0769528716, S. 96–103

[37] Larrucea, Xabier ; Santamaria, Izaskun ; Colomo-Palacios, Ricardo ; Ebert, Christof: Microservices. In: *IEEE Software* 35 (2018), may, Nr. 3, 96–100. http://dx.doi.org/10.1109/MS.2018.2141030. – DOI 10.1109/MS.2018.2141030. – ISSN 0740-7459

[38] Lewis, James ; Fowler, Martin: *Microservices*. https://martinfowler.com/articles/microservices.html. Version: 2014

[39] Lin, Jyhjong ; Lin, Lendy C. ; Huang, Shiche: Migrating web applications to clouds with microservice architectures. In: *2016 International Conference on Applied System Innovation (ICASI)*, IEEE, may 2016. – ISBN 978-1-4673-9888-6, 1–4

[40] Mazlami, G. ; Cito, J. ; Leitner, P.: Extraction of Microservices from Monolithic Software Architectures. In: *Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017*, 2017. – ISBN 9781538607527, S. 524–531

[41] Mazzara, Manuel ; Dragoni, Nicola ; Bucchiarone, Antonio ; Giaretta, Alberto ; Larsen, Stephan T. ; Dustdar, Schahram: Microservices: Migration of a Mission Critical System. In: *IEEE Transactions on Services Computing* (2018), 1–1. http://dx.doi.org/10.1109/TSC.2018.2889087. – DOI 10.1109/TSC.2018.2889087. – ISSN 1939-1374

[42] MEDVIDOVIC, N. ; TAYLOR, R.N.: A classification and comparison framework for software architecture description languages. In: *IEEE Transactions on Software Engineering* 26 (2000), Nr. 1, 70–93. `http://dx.doi.org/10.1109/32.825767`. – DOI 10.1109/32.825767

[43] NEWMAN, Sam: *Building microservices : designing fine-grained systems*. 2015. – ISBN 9781491950357

[44] NEWMAN, Sam ; O'REILLY MEDIA (Hrsg.): *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. 1st Editio. 2019. – 272 S. – ISBN 978-1492047841

[45] OMG: Service oriented architecture Modeling Language (SoaML) Specification. In: *Language* (2012), Nr. March, S. 1–144

[46] OQUENDO, Flavio: π-ADL: An Architecture Description Language based on the Higher-Order Typed π-Calculus for Specifying Dynamic and Mobile Software Architectures. In: *ACM SIGSOFT Software Engineering Notes* 29 (2004), Nr. 3, S. 1. `http://dx.doi.org/10.1145/986710.986728`. – DOI 10.1145/986710.986728. – ISSN 01635948

[47] OQUENDO, Flavio: Formal approach for the development of business processes in terms of Service-Oriented Architectures using π-ADL. In: *Proceedings of the 4th IEEE International Symposium on Service-Oriented System Engineering* (2008), Nr. i, S. 154–159. `http://dx.doi.org/10.1109/SOSE.2008.38`. – DOI 10.1109/SOSE.2008.38. ISBN 9780769534992

[48] PAPAPOSTOLU, Anastasios ; BIROV, Dimitar: Structured component and connector communication. In: *ACM International Conference Proceeding Series* Part F1309 (2017). `http://dx.doi.org/10.1145/3136273.3136291`. – DOI 10.1145/3136273.3136291. ISBN 9781450352857

[49] PAPAPOSTOLU, Tasos: $\mu\sigma$ADL: An Architecture Description Language for MicroServices. In: TAIAR R., COLSON S., CHOPLIN A., Ahram T. (Hrsg.): *1st International Conference on Human Interaction and Emerging Technologies, IHIET 2019*, Springer Verlag, 2020. – ISBN 978-303025628-9, S. 885–889

[50] PAPAPOSTOLU, Tasos ; BIROV, Dimitar: Towards a Methodology for Designing Microservice Architectures Using $\mu\sigma$ADL. In: *Lecture Notes in Business Information Processing* Bd. 319, Springer Verlag, 2018. – ISBN 9783319942131, S. 421–431

[51] PAUTASSO, Cesare ; ZIMMERMANN, Olaf ; AMUNDSEN, Mike ; LEWIS, James ; JOSUTTIS, Nicolai: Microservices in Practice, Part 1: Reality Check and Service Design. In: *IEEE Software* 34 (2017), jan, Nr. 1, 91–98. `http://dx.doi.org/10.1109/MS.2017.24`. – DOI 10.1109/MS.2017.24. – ISSN 0740–7459

[52] PERRY, Dewayne E. ; WOLF, Alexander L.: Foundations for the study of software architecture. In: *ACM SIGSOFT Software Engineering Notes* 17 (1992), oct, Nr. 4, 40–52. `http://dx.doi.org/10.1145/141874.141884`. – DOI 10.1145/141874.141884

[53] RADEMACHER, Florian ; SACHWEH, Sabine ; ZUNDORF, Albert: Differences between Model-Driven Development of Service-Oriented and Microservice Architecture. In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, IEEE, apr 2017. – ISBN 978–1–5090–4793–2, 38–45

[54] RICHARDS, Mark.: *Microservices vs. Service-Oriented Architecture*. First Edit. O'Reilly Media, Inc, 2016. – 57 S. – ISBN 9781491975657

[55] RICHARDS, Mark ; FORD, Neal ; MEDIA, O'Reilly (Hrsg.): *Fundamentals of Software Architecture: An Engineering Approach*. 1st Editio. 2020. – 432 S. – ISBN 978–1492043454

[56] RICHARDSON, Chris: *Microservices Patterns*. Manning, 2019. – ISBN 9781617294549

[57] ROZANSKI, Nick. ; WOODS, Eoin.: *Software systems architecture : working with stakeholders using viewpoints and perspectives*. Addison-Wesley, 2005. – 546 S. – ISBN 0321112296

[58] SADOU, N. ; TAMZALIT, D. ; OUSSALAH, M.: A unified Approach for Software Architecture Evolution at different abstraction levels. In: *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, IEEE. – ISBN 0–7695–2349–8, 65–70

[59] SCHALLES, Christian: *Usability evaluation of modeling languages: An empirical research study*, Diss., 2013. http://dx.doi.org/10.1007/978-3-658-00051-6. – DOI 10.1007/978-3-658-00051-6. – 1–181 S

[60] SCHMIDT, Roger A. ; THIRY, Marcello: Microservices identification strategies : A review focused on Model-Driven Engineering and Domain Driven Design approaches. In: *Iberian Conference on Information Systems and Technologies, CISTI* Bd. 2020-June, IEEE Computer Society, jun 2020. – ISBN 9789895465903

[61] TAIBI, Davide ; LENARDUZZI, Valentina ; PAHL, Claus: Continuous architecting with microservices and DevOps: A systematic mapping study. In: *Communications in Computer and Information Science* Bd. 1073, Springer Verlag, 2019. – ISBN 9783030291921, S. 126–151

[62] TAYLOR, Richard N. ; MEDVIDOVIÇ, Nenad. ; DASHOFY, Eric M. (Eric M.: *Software architecture : foundations, theory, and practice*. Wiley, 2010. – 712 S. – ISBN 0470167742

[63] TERZIÇ, Branko ; DIMITRIESKI, Vladimir ; KORDIÇ, Slavica ; MILOSAVLJEVIÇ, Gordana ; LUKOVIÇ, Ivan: Development and evaluation of MicroBuilder: a Model-Driven tool for the specification of REST Microservice Software Architectures. In: *Enterprise Information Systems* 12 (2018), oct, Nr. 8-9, S. 1034–1057. http://dx.doi.org/10.1080/17517575.2018.1460766. – DOI 10.1080/17517575.2018.1460766. – ISSN 17517583

[64] THE OPEN GROUP: *Microservices Architecture – SOA and MSA*. http://www.opengroup.org/soa/source-book/msawp/p3.htm

[65] The Open Group: *Microservices Architecture – SOA and MSA*. http://www.opengroup.org/soa/source-book/msawp/p3.htm

[66] Vergara-Vargas, Jeisson ; Umana-Acosta, Henry: A model-driven deployment approach for scaling distributed software architectures on a cloud computing platform. In: *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS* Bd. 2017-Novem, IEEE Computer Society, apr 2018. – ISBN 9781538645703, S. 99–103

[67] Vergara-Vargas, Jeisson A.: *A model-driven deployment approach for applying the performance and scalability perspective from a set of software architecture styles*, Universidad Nacional de Colombia, Diss., 2017. http://bdigital.unal.edu.co/61128/

[68] Wang, Quanyu ; Ying, Shi ; Jia, Xiangyang ; Lv, Guobin ; Shuai, Yun: SOADL-EH: Service-oriented architecture description language supporting exception handling. In: *Advanced Materials Research* 433-440 (2012), S. 3500–3509. http://dx.doi.org/10.4028/www.scientific.net/AMR.433-440.3500. – DOI 10.4028/www.scientific.net/AMR.433-440.3500. – ISBN 9783037853191

[69] Waseem, Muhammad ; Liang, Peng: Microservices Architecture in DevOps. In: *2017 24th Asia-Pacific Software Engineering Conference Workshops (APSECW)*, IEEE, dec 2017. – ISBN 978-1-5386-2649-8, S. 13–14

[70] Waseem, Muhammad ; Liang, Peng ; Shahin, Mojtaba: A Systematic Mapping Study on Microservices Architecture in DevOps. In: *Journal of Systems and Software* 170 (2020), dec. http://dx.doi.org/10.1016/j.jss.2020.110798. – DOI 10.1016/j.jss.2020.110798. – ISSN 01641212

[71] Williams, Byron J. ; Carver, Jeffrey C.: Characterizing software architecture changes: A systematic review. In: *Information and Software Technology* 52 (2010), jan, Nr. 1, 31–51. http://dx.doi.org/10.1016/j.infsof.2009.07.002. – DOI 10.1016/j.infsof.2009.07.002

[72] Yugopuspito, Pujianto ; Panduwinata, Frans ; Sutrisno, Sutrisno: Microservices architecture: Case on the migration of reservation-based parking system. In: *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, IEEE, oct 2017. – ISBN 978-1-5090-3944-9, 1827–1831