



UNIVERSIDAD
NACIONAL
DE COLOMBIA

A model-driven deployment approach for applying the performance and scalability perspective from a set of software architecture styles

Jeisson Andrés Vergara Vargas

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, D. C., Colombia
2017

A model-driven deployment approach for applying the performance and scalability perspective from a set of software architecture styles

Jeisson Andrés Vergara Vargas

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:
M.Sc., Henry Roberto Umaña Acosta

Research Field:
Software 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
2017

Dedicated to my grandma.

“There is no certainty in sciences where one of the mathematical sciences cannot be applied, or which are not in relation with these mathematics.”

- Leonardo da Vinci

Acknowledgments

First of all, I want to express my deepest thanks to my dear university, the Universidad Nacional de Colombia. I am very proud to belong to this university and much more to contribute to the development of the software engineering, even if it is a minimal contribution. Likewise, my most sincere thanks to all the professors who were part of my life at the university, providing me an excellent education. Special thanks to my academic father and advisor, professor Henry Roberto Umaña Acosta, for his teaching, support and guidance during the last years.

Thanks to my girlfriend Daniela for her huge love and for being my motivation to finish this research work. In the same way, I want to thank my mom for her guidance in my life and for her motivation to develop my master program. Likewise, thanks to my uncle, my aunt and my grandpa for their support and understanding. Special thanks to Saray for her accompaniment and for transmitting me her happiness. In addition, thanks to Ana, Eder and Daniel for their invaluable friendship and motivation.

Thanks to all the authors cited in this document, without their work, it would have been really hard to develop this research work. Finally, I want to thank the juries, professors Felipe Restrepo Calle and Kelly Garcés Pernet, for sharing their knowledge and some of their time in the evaluation of this research work.

Title in English

A model-driven deployment approach for applying the performance and scalability perspective from a set of software architecture styles.

Título en español

Un enfoque de despliegue dirigido por modelos para aplicar la perspectiva de rendimiento y escalabilidad a partir de un conjunto de estilos arquitectónicos de software.

Abstract

Software architecture aims to satisfy software requirements from different points of view. This is represented by models, which are the reference to understand the structure and behavior of the software. Nevertheless, one of the great challenges of software engineering is to ensure that design, implementation and deployment of the software are consistent. In the same way, another challenge is to ensure that a system improves its performance when it is in a scenario of receiving many requests per unit of time. In this manner, this research work presents a proposed model-driven deployment approach which from software architecture models, automates the deployment of software systems on a cloud computing platform by means of the application of scalability tactic, specifically horizontal scaling. In addition, this work includes a traditional model-driven development process which automates the implementation of the software system to be deployed. Likewise, Sarch is designed and proposed, a domain-specific language based on the specification of a set of architectural styles and their representation as architectural views. Finally, a tool called Sarch-Studio is built, which allows writing in Sarch language and performs automatic development and deployment processes.

Keywords: Software Architecture, Performance, Scalability, Horizontal Scaling, Cloud Computing, Model-Driven Deployment, Model-Driven Development, Domain-Specific Language, Architectural Style, Architectural View.

Resumen

La arquitectura de software pretende satisfacer los requisitos de software a partir de diferentes puntos de vista. Esta es representada por medio de modelos, los cuales son la referencia para comprender la estructura y comportamiento de el software. Sin embargo, uno de los grandes retos de la ingeniería de software es asegurar que el diseño, la implementación y el despliegue del software sean consistentes. De la misma forma, otro reto es lograr que un sistema de software mejore su rendimiento cuando este se encuentra en un escenario de recepción de muchas solicitudes por unidad de tiempo. De esta manera, este trabajo de investigación presenta un enfoque de despliegue dirigido por modelos, que a partir de modelos de arquitectura de software, automatiza el despliegue de sistemas de software en una plataforma de computación en la nube, por medio de la aplicación de tácticas de escalabilidad, específicamente de la táctica de escalamiento horizontal. Además, este trabajo incluye un proceso tradicional de desarrollo dirigido por modelos, el cual automatiza la implementación de los sistemas de software a ser desplegados. Así mismo, se diseña y se propone Sarch, un lenguaje de dominio específico basado en la especificación de un conjunto de estilos arquitectónicos y su representación como vistas arquitectónicas. Finalmente, se implementa una herramienta llamada Sarch-Studio, que permite escribir en lenguaje Sarch y es la encargada de realizar los procesos automáticos de desarrollo y despliegue.

Palabras clave: Arquitectura de Software, Rendimiento, Escalabilidad, Escalamiento Horizontal, Computación en la Nube, Despliegue Dirigido por Modelos, Desarrollo Dirigido por Modelos, Lenguaje de Dominio Específico, Estilo Arquitectónico, Vista Arquitectónica.

Contents

Acknowledgments	vii
Abstract	x
List of Figures	xiv
List of Tables	1
1. Introduction	2
1.1. Problem Statement	3
1.2. Objectives	3
1.2.1. General Objective	3
1.2.2. Specific Objectives	3
1.3. Contribution	4
1.4. Outline of the Thesis	4
2. Background	6
2.1. Software Architecture	6
2.1.1. Architectural Styles	7
2.1.2. Architectural Views	7
2.1.3. Performance and Scalability Perspective	9
2.1.4. Distributed Architectures	10
2.2. Domain-Specific Languages	10
2.3. Model-Driven Engineering	11
2.4. Cloud Computing	14
3. Related Work	15
3.1. Architecture Description Languages	15
3.2. DSLs for Documenting Software Architectures	16
3.3. Support the Software Development Process	17
3.4. Support the Software Deployment Process	17
3.5. Deployment in a Cloud Computing Platform	18
4. Model-Driven Deployment	19
4.1. Definition	19

4.2. Methodology	20
5. Sarch Language	22
5.1. Architectural Schema	23
5.2. Data Model View	23
5.3. Layered View	25
5.4. Component-and-Connector (C&C) View	27
5.5. Deployment View	28
6. Sarch-Studio Tool	32
6.1. Sarch-Studio Architecture	32
6.1.1. Associated Technologies	33
6.1.2. Components	33
6.2. Target Software Architecture	35
6.2.1. Associated Technologies	35
6.2.2. Software Architecture Description	37
6.3. Model-Driven * Processes	38
6.3.1. Automation of Development	38
6.3.2. Automation of Deployment	39
7. Evaluation	42
7.1. Conceptual Analysis	42
7.2. Practical Analysis	43
8. Conclusions and Future Work	46
8.1. Conclusions	46
8.2. Future Work	46
A. Appendix: Sarch Language Metamodel	47
B. Appendix: Views of the Case Study	48
C. Appendix: Case Study designed in Sarch Language	50
D. Appendix: Implementation and Generation of Case Study	52
Bibliography	56

List of Figures

1-1. Graphical representation of thesis proposal.	5
2-1. Software Architecture context.	6
2-2. Relationship between styles and views.	7
2-3. 4+1 View Model.	8
2-4. Classification of styles/views in the Views & Beyond (V&B) Catalog.	9
2-5. Microservices Architecture (MSA).	11
2-6. Domain-Specific Languages (DSLs) context.	11
2-7. MDE process: relationship between metamodels, models, modeling languages and transformations.	12
2-8. Model-Driven Engineering (MDE) context.	13
2-9. Cloud Delivery and Cloud Deployment models.	14
3-1. Architecture Description Languages context.	15
3-2. Documenting software architectures.	16
3-3. Supporting the software development process.	17
3-4. Supporting the software deployment process.	18
3-5. Deploying software systems in the cloud.	18
4-1. MDDep in the context of MDE.	19
4-2. MDDep methodology.	20
5-1. Sarch language.	22
5-2. CST for the architectural schema in Sarch language.	23
5-3. CST for the Data Model view in Sarch language.	25
5-4. CST for the Layered view in Sarch language.	26
5-5. CST for the Component-and-Connector (C&C) view in Sarch language.	28
5-6. CST for the Deployment view in Sarch language (part A).	30
5-7. CST for the Deployment view in Sarch language (part B).	31
6-1. Layered architecture of Sarch-Studio.	32
6-2. Xtext, Eclipse Modeling Framework (EMF), and Xtend.	33
6-3. Definition of Sarch grammar in the Xtext environment.	34
6-4. Definition of Sarch transformations in the Xtend environment.	34

6-5.	Editor component of Sarch-Studio.	35
6-6.	MySQL, Java, Maven, GlassFish, Docker, Rancher, HAProxy, Amazon Web Services (AWS), and JMeter.	36
6-7.	Target architecture from the point of view of the C&C view in a semiformal representation	38
6-8.	Final interaction between Sarch-Studio and the Cloud Platform.	41
7-1.	Rancher server environment with 4-nodes infrastructure on the cloud.	43
7-2.	Performance curves with knee for each scenario (requests vs. response time).	45
7-3.	Performance curves for each scenario (requests vs. throughput).	45
A-1.	Sarch Language Metamodel.	47
B-1.	Graphic representation of the Data Model view.	48
B-2.	Graphic representation of the Layered view.	48
B-3.	Graphic representation of the C&C view.	49
B-4.	Graphic representation of the Deployment view.	49
C-1.	Software architecture wrote in Sarch (i).	50
C-2.	Software architecture wrote in Sarch (ii).	51
D-1.	Code structure of the project.	52
D-2.	Docker structure for database, microservice, web application and load balancer.	53
D-3.	Model and Resource classes of microservice.	54
D-4.	Web application visualization (Index and Create Post).	55

List of Tables

7-1. Conceptual analysis of Sarch: approaches vs. main aspects.	42
7-2. Stress test results.	44

1. Introduction

Taylor et al. in [53] describe the *Software Architecture (SA)* as the set of principal design decisions made about a software system. In general, software architecture is the design of the highest abstract level of the structure of a software system. An architecture is selected and designed based on functional and nonfunctional requirements, and many challenges of designing an architecture are related to the ever increasing complexity of the software. However, the way of building software architectures is not unique, all of design decisions possibilities establish a common objective that allows to define a specialization of elements, relations and properties at a high level. These specializations are called *Architectural Styles*.

On the other hand, one of the most important quality attributes that must be guaranteed by the software architecture is the *Scalability*, i.e., the ability of a software system to handle the increased workload, which may be due to an increase in the number of requests or transactions [49]. In this way, improving the performance of the system. Thereby, the key premise in this research work is that a set of styles can be considered as the basis of a language adjusted to a particular domain, i.e., a *Domain-Specific Language (DSL)* focused on software architecture modeling. In this context, we propose Sarch, a DSL aimed to the design of software architectures from the definition of the *Data Model*, *Component-and-Connector (C&C)*, *Layered* and *Deployment Architectural Views*, representations of a set of architectural styles presented by Clements et al. in the *Views and Beyond (V&B)* catalog [19].

Sarch is proposed from two approaches: support the software development and the software deployment. The first approach is done through a *Model-Driven Development (MDD)* paradigm, which improve the software implementation process based on models supported by powerful tools and processes [15]. In the same way, the second approach is done through a *Model-Driven Deployment (MDDep)* approach, which is a proposed paradigm that uses models, as in MDD, but using them to automate the software deployment. Hence, Sarch is used as the metamodel of MDDep and MDD processes, where a Sarch instance represents a software architecture model. This model allows to generate the source code of a Java web-based software system with a distributed software architecture, an architecture composes of a set of autonomous components that work together. Likewise, the model allows to deploy the system on a cloud computing platform from the definition of a *Horizontal Scaling* schema for each one of the components that are part of the architecture. This method allows to improve the performance of the system and support an increase in the number of concurrent users supported [49].

1.1. Problem Statement

This work focuses its research question on two important aspects. First, the way as a software engineering process can be more consistent is exist an harmonic relationship between software design, implementation and deployment. And second, the way as a software system can improve its performance and scalability when it is in a scenario of receiving many requests per unit of time. In the two cases, software architecture plays a main role because it defines the most important design decision at a high level: supporting good design practices, basis for an adequate software implementation, providing the way how a software system should be deployed after satisfying the functional requirements, and finally, applying architectural tactics to guarantees the fulfillment of some nonfunctional requirements. In this way, there are two related research questions that are proposed to solve the problems described: 1) How can consistency be ensured between designing, implementing and deploying a software when there are nonfunctional requirements as performance and scalability that may affect the quality of the software?, 2) How can automatic deployments be generated from the formal definition of a software architecture design?

1.2. Objectives

1.2.1. General Objective

Propose a model-driven deployment approach for applying the performance and scalability perspective from a software architecture designed with the data model, component-and-connector, layered and deployment styles.

1.2.2. Specific Objectives

- Design a set of domain-specific languages and meta-models for the architectural styles of data model, component-and-connector, layered and deployment.
- Implement a tool that supports the design of software architectures for web applications from the data model, component-and-connector, layered and deployment styles.
- Generate the functional architecture of a web application, in a defined programming language, from an architectural design in the data model and component-and-connector styles, and a predefined architectural design in the layered style.
- Generate the deployment architecture for a cloud computing platform from an architectural design in the deployment style.

- Perform stress tests on a set of generated deployment architectures, in order to demonstrate the benefits of using the scaling architectural tactic to optimize the performance and scalability of a web application.

1.3. Contribution

The contributions of this thesis can be defined as follows:

- *A new Model-Driven Engineering (MDE) approach*: we recognize the advantages of MDE for automate important aspects into the software engineering field, specially, the code generation with the model-driven development approach. In this way, a *Model-Driven Deployment (MD-Dep)* approach is proposed. It automates software deployments from a formal software architecture specification.
- *A domain-specific language focused on the software architecture modeling*: in order to allow the design of software architecture models, Sarch is proposed, a domain-specific language that allows the design of distributed software architectures. Its design is based on a set of architectural styles and its representation as architectural views. Given its domain, Sarch can be considered as a new Architecture Description Language (ADL).
- *A tool for modeling software architectures*: a tool called Sarch-Studio is proposed. It supports writing code in Sarch language, automatic code generation and automatic deployment of the software systems designed.

An overview of the thesis proposal is shown in Figure 1-1. It includes the general process of definition, transformation and automation.

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 software architecture and model-driven engineering. Chapter 3 gives the related work. Chapter 4 introduces the model-driven deployment approach. In Chapter 5, Sarch language is presented and described. Chapter 6 presents the Sarch-Studio tool that we have developed for modeling software architectures through Sarch language. In Chapter 7, an evaluation of Sarch language and Sarch-Studio is explained. Chapter 8 presents the conclusions and future work.

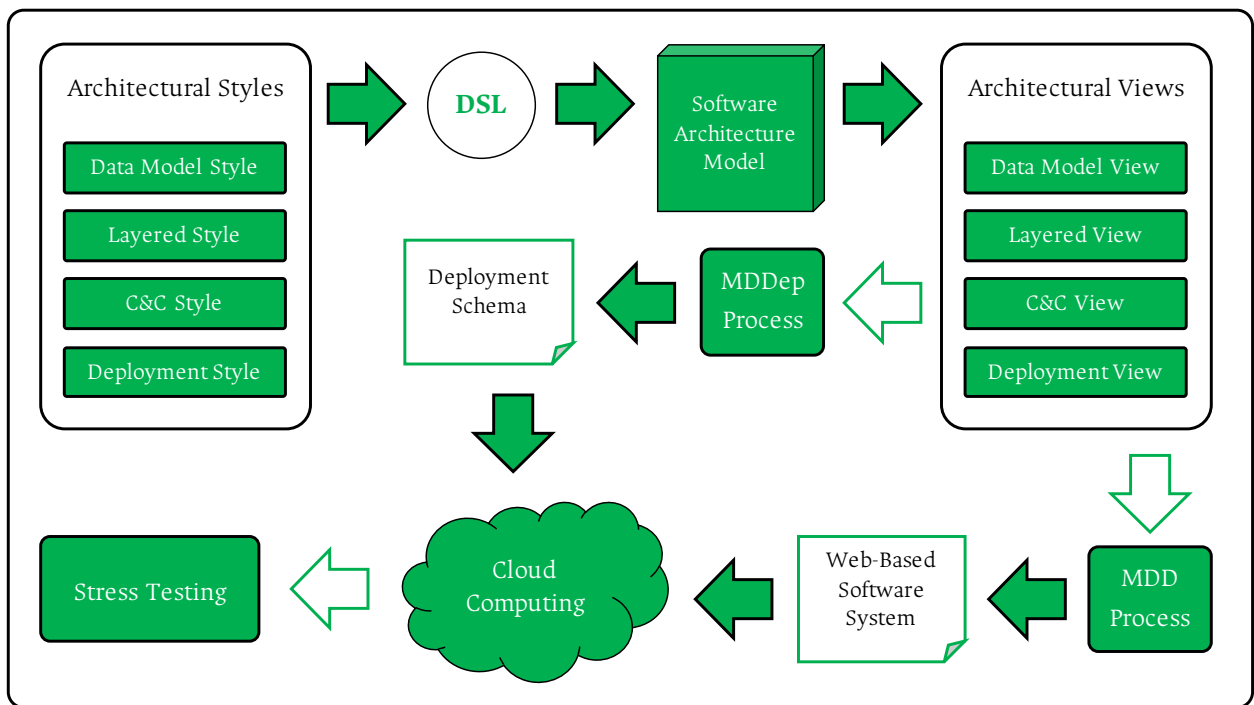


Figure 1-1.: Graphical representation of thesis proposal.

2. Background

The main goal of this chapter is to give a general overview around the most important topics that motivate this research work. This chapter is organized as follows: Software Architecture, Domain-Specific Languages (DSLs), Model-Driven Engineering (MDE), and Cloud Computing.

2.1. Software Architecture

Software Engineering defines a set of tasks that allows the application of common engineering activities on the process of software creation. Thus, we can describe six activities as the most important tasks in the software engineering field: software requirements, software design, software development, software deployment, software testing and software maintenance [52].

In this way, *Software Design* is the activity of specifying programs and sub-systems, as well as their constituent parts and their operation in order to meet software product specifications [30]. The software design includes two important parts: the *architectural design*, which defines the functional and nonfunctional aspects of a specific software system, and the *detailed design* that specifies the aspects of each one of the elements defined in the architectural design. This research work considers only the architectural design, in order to show the importance of this design as the highest abstraction level of a software. As well as its representation as models to assure its understanding by the different stakeholders of whole software lifecycle.

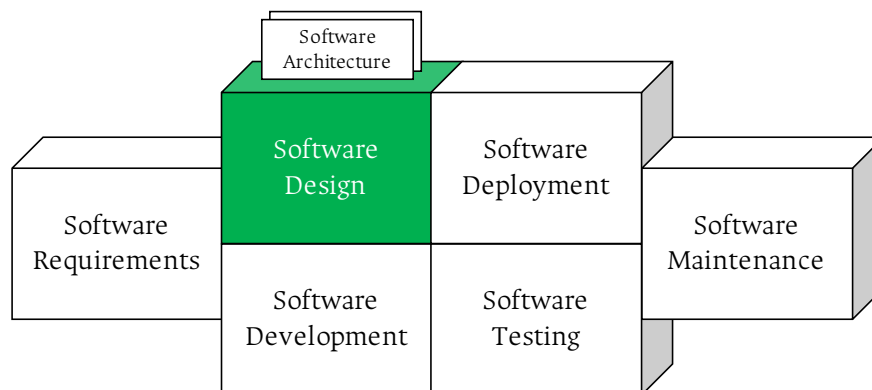


Figure 2-1.: Software Architecture context.

The discipline that is responsible of the architectural design, is the *Software Architecture (SA)*. Taylor et al. in [53] describe the software architecture as the set of principal design decisions made about a software system. Likewise, Rozanski et al. in [49] describe the software architecture as the structure of structures of the system, which comprise software elements, relations among them, and properties of both. In general, software architecture is the design of the highest abstract level of the structure of a software system.

2.1.1. Architectural Styles

A software architecture is selected and designed based on functional and nonfunctional requirements, and many challenges of designing a software architecture are related to the ever increasing complexity of the software. However, the way of building software architectures is not unique, all design decisions possibilities establish a common objective that allows to define a specialization of elements, relations and properties of software systems. Each one of these specializations is known as an *Architectural Style* [19]. Hence, these styles abstract design solutions at a high level.

Thus, an architectural style can be defined as a named collection of architectural design decisions with a set of configuration rules, a vocabulary of design elements, and a semantic interpretation [53]. Among the most popular architectural styles, we can find: Client-Server, Peer-to-Peer (P2P), Layered, Pipe and Filter, Publish-Subscribe (Pub/Sub), Event-Based Architecture, Object-Oriented Architecture, Component-Based Architecture, Service-Oriented Architecture (SOA), Agent-Oriented Architecture, REpresentational State Transfer (REST), and Microservices Architecture (MSA).

2.1.2. Architectural Views

An architectural view (software architecture view) is a representation of a set of system elements and the relationships associated with them [19]. Thus, when an architectural style is applied to a software system, an architectural view is obtained.

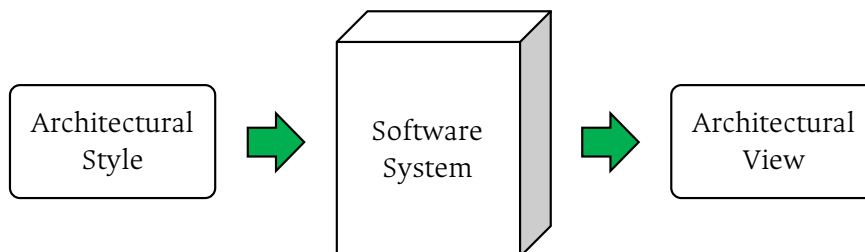


Figure 2-2.: Relationship between styles and views.

There are many proposals about how a software system is represented through a set of views and these views are commonly represented through diagrams. Among the most popular catalogs of architectural views, we can find:

- **4+1 View Model (Kruchten's 4+1)**: this model was proposed in 1995 by Kruchten [39]. It consists of four main views: Logical, Process, Development and Physical view, and an complementary view: Scenarios. Each view describes the concerns of the various stakeholders of the architecture: end-user, developers, systems engineers, project managers, etc.

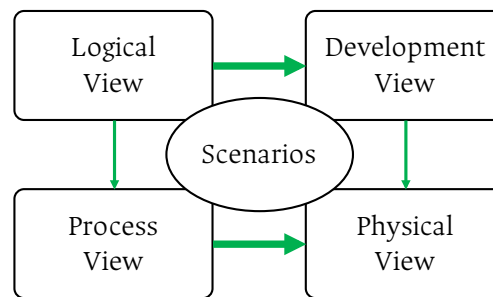


Figure 2-3.: 4+1 View Model.

- **ISO/IEC/IEEE 42010 (Systems and Software Engineering - Architecture Description)**: this International Standard specifies the manner in which architecture descriptions of systems are organized and expressed [36]. It specifies architecture viewpoints, architecture frameworks and architecture description languages.
- **The Viewpoint Catalog**: this catalog was proposed by Rozanski & Woods in [49], and it describes six different viewpoints: Functional, Information, Concurrency, Development, Deployment, and Operational. This propose focuses on the description of stakeholders, concerns and models related to each viewpoint.
- **Views & Beyond (V&B) Catalog**: this catalog was proposed by Clements et al. in [19], and it is designed from the concept of architectural style, specifically, from its definition as a package of design decisions. The V&B Catalog describes three categories of styles. The *Module Styles* that introduce a set of modules (implementation unit) of a software system, and a set of rules that allow the way how the modules can be combined. The *Component-and-Connector (C&C) Styles* which expresses the runtime behavior of a software system. And finally, the *Allocation Styles* that describe the way how a set of software elements (presented in the C&C) can be allocated on a hardware environment. In this way, the V&B Catalog uses the concept of architectural view, i.e., the result of apply an architectural style on a software system. Thus, this catalog offers a set of architectural views: *Module Views*, *Component-and-Connector (C&C) Views*, and *Allocation Views*. In addition, for each set of views,

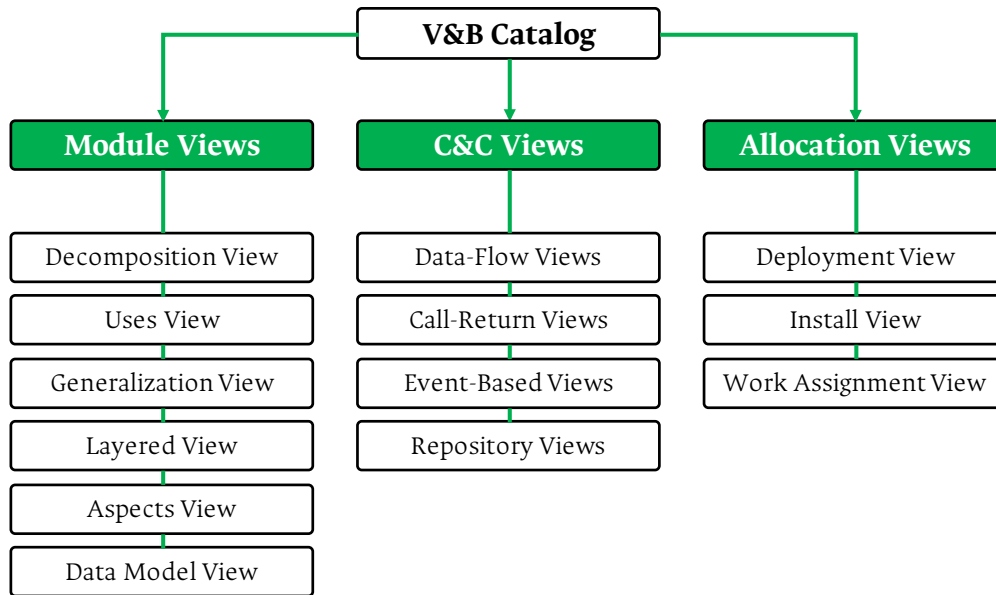


Figure 2-4.: Classification of styles/views in the Views & Beyond (V&B) Catalog.

a set of elements, relations and properties are described in detail. The classification of styles/views in the V&B Catalog is shown in Figure 2-4.

2.1.3. Performance and Scalability Perspective

The most representative nonfunctional software requirements are the *Software Quality Properties (Attributes)*, i.e., a set of factors which can be affect the software design and the runtime behavior of a software system. Hence, an *Architectural Perspective* defines a collection of activities and tactics that are used to ensure that a software system exhibits a set of related quality properties [49]. Among the most important quality properties we can mention: Performance, Scalability, Availability, Resilience, Interoperability, Security, and Usability.

The *Performance and Scalability Perspective* defines a set of architectural tactics related to the treatment of two important concepts:

- **Response Time:** the time it takes to complete an interaction with a software system [49].
- **Throughput:** the amount of workload that a software system is capable of handling in a unit time period [49].

Thus, the main objective of this perspective is to handle the environment of a software system in order to increase its *Performance*: decrease response times and increase throughput. This is done through the *Scalability*, the ability of a software system to handle this increased workload

[49], and there are two common architectural tactics that can be applied: *Vertical Scaling*, i.e., increase the power of existing system by adding more powerful hardware, and *Horizontal Scaling*, i.e., add more nodes to server of the software system.

2.1.4. Distributed Architectures

From the definition of architectural styles, these present a set elements that allows the conception of a software system to be built. However, the way as this build is done depends of the functional and nonfunctional requirements.

One of the most common nonfunctional requirements is to handle aspects related to quality attributes, mainly related to the perspectives of scalability and high availability, where it is necessary to reduce the size of software elements in order to handle each one in a more complete way, i.e., guaranteeing its functionality, performance and availability separately. This helps to reduce possible points of failure but increase the complexity of the interaction between elements. In this way, software architectures are composed of a set of elements that work collaboratively with each other and make the architecture have a distributed focus.

Two important architectural styles that have a distributed approach are described below:

- **REpresentational State Transfer (REST)**: it is a client-server-based architectural style that is structured around a small set of create, read, update, delete operations which are known in REST as POST, GET, PUT, DELETE respectively [14] [54]. In addition, REST is based on a single addressing scheme (Uniform Resource Identifier (URI)), an schema that allows to identify resources in the web. It is important to highlight that REST uses HTTP (Hypertext Transfer Protocol) as its communication protocol. Thus, RES is considered as the architectural style of the web and it has a distributed focus because the web is distributed by definition: a set of resources distributed around the world.
- **Microservices Architecture (MSA)**: it is a service-based architecture, an architecture composed of a set of elements, called *microservices*. A microservice is a loosely coupled services, which implement business capabilities. On the other hand, microservices are connected through REST and a a system composed of multiple, collaborating services, it is possible to use different technologies inside each one [45]. In addition, it is a good practice that each microservice has its own database. this makes the data models quite small.

2.2. Domain-Specific Languages

In the context of computer languages, there are two types of languages: the *General-Purpose Languages (GPLs)* and the *Domain-Specific Languages (DSLs)*. The first that can be applied to any domain

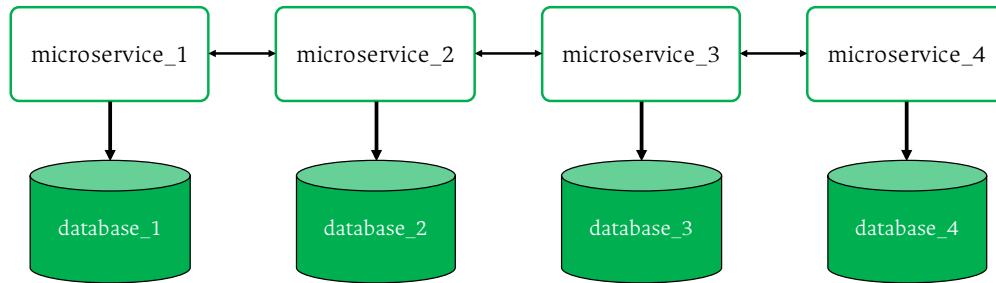


Figure 2-5.: Microservices Architecture (MSA).

for programming purposes, and the second, that are defined for a specific context or domain. In this manner, a Domain-Specific Language (DSL) is a computer language of limited expressiveness where it is adjusted to a particular domain and provide notations and abstractions close to it [29], [20], [25]. Both GPLs and DSLs can be of different type, mainly *Programming Languages* and *Modeling Languages*. A general classification of computer languages as the context of DSLs is shown in Figure 2-6.

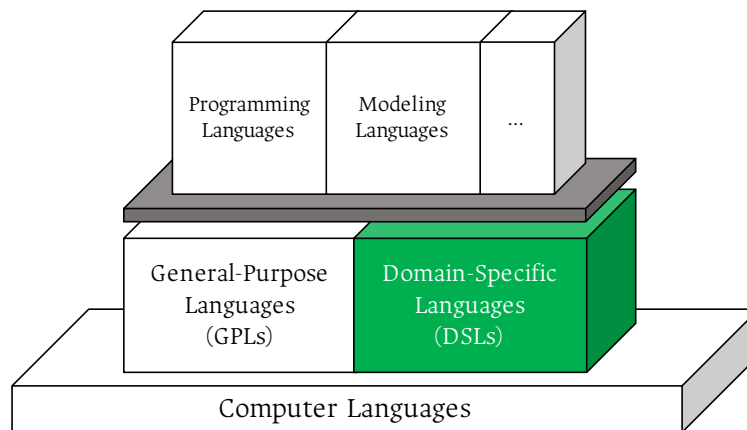


Figure 2-6.: Domain-Specific Languages (DSLs) context.

C, C++, Java, Python, Ruby and Go (programming languages), and UML (Unified Modeling Language) are some examples of GPLs. On the other hand, R for statistical operations and processes, SQL for relational databases operations, XML for data transport, MatLab for mathematics, and HTML for the development of web pages, are examples of DSLs.

2.3. Model-Driven Engineering

Model-Driven Software Engineering (MDSE) or simply *Model-Driven Engineering (MDE)* is defined as a methodology for applying the advantages of modeling to software engineering activities [15]. Thus, MDE comprises concepts, notations, process, rules and tools, important features that allow the modeling process. Nevertheless, the most important concepts related to MDE are *Models*

and *Transformations*, and the key promise of both concepts is that with a set of models and a set of transformations applied on them, it is possible to create software in an automatic or semiautomatic way [15].

Models provide abstractions of a physical system that help software engineers to reason about a software system by ignoring extraneous details while focusing on the relevant ones. All forms of engineering rely on models as essential to understanding complex real-world systems. Likewise, a related concept is *Metamodel* which describes concepts that can be used for modeling the models, i.e., models can be defined as instances of metamodels [50]. On the other hand, *transformations* define the MDE aspect that allow the definition of mappings between different models and they are defined at the metamodel level, and then applied at the model level [15]. A model transformation is performed between a source and a target model [50]. There are two types of models transformations: Model-to-Model (M2M) and Model-to-Text (M2T) transformations [15], in the M2M transformations both input and output parameters are models, while in the M2T transformations the input is a model and the output is a text string. These transformations techniques are based on two phases [15]: defining a mapping between elements of a model to elements to another one; and automating the generation of the actual transformation rules through a system that receives as input the two model definitions and the mapping between them and produces the transformations.

Finally, a third concept that is important highlight is the *Domain*, because it is the starting point of a MDE process and it describes a bounded field of interest or knowledge [50], generally, the domain specifies the space of the problem where a MDE process is applied. Hence, in a MDE process, the domain is specified through a DSL. In this case a *Domain-Specific Modeling Language (DSML)* which defines the basis of the metamodels, and the language instances are considered as the models. Figure 2 shows an overall vision of a MDE process and the relationship between the models, metamodels, modeling languages and transformations.

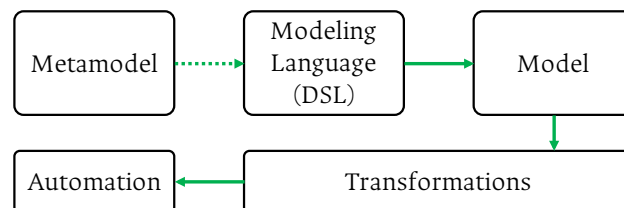


Figure 2-7.: MDE process: relationship between metamodels, models, modeling languages and transformations.

A MDE process can be considered as a part of *Model-Based Engineering (MBE)* that is a process in which software models play an important role although they are not necessarily the key artifacts of the development [15]. This is the main difference between the word "driven" and "based".

In addition, MDE defines three main approaches [15]:

- **Model-Driven Development (MDD):** Model-Driven Software Development (MDS), or simply Model-Driven Development (MDD) is a development paradigm that uses models as the primary artifact of the development process. MDD promises to improve the software construction process based on a software-driven models supported by powerful tools and process. In addition, *Model-driven Architecture (MDA)* is a particular MDD proposal of the Object Management Group (OMG) ¹.
- **Model-Driven Interoperability (MDI):** it aim at defining bridges to achieve interoperability between two or more software systems by applying model-driven techniques. It is important to mention that interoperability is a quality property defined as the ability of two or more systems (or software components) to exchange information [14].
- **Model-Driven Reverse Engineering (MDRE):** process of obtaining useful higher-level representations of software systems already built.

The MDE context is shown in Figure 2-8. The representation presents the relationships between the different approaches described above: MBE, MDE, MDD, MDA, MDI, and MDRE.

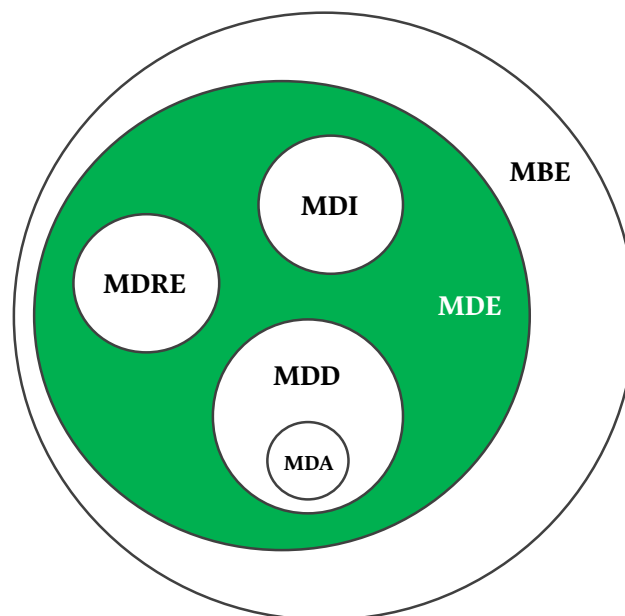


Figure 2-8.: Model-Driven Engineering (MDE) context.

¹See: <http://www.omg.org/>

2.4. Cloud Computing

Cloud Computing is defined by Gartner as a style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service using Internet technologies ². Thus, cloud computing has two important roles: *cloud provider* which provides cloud-based IT resources, and *cloud consumer* which has a formal contract or arrangement with a cloud provider to use IT resources made available by the cloud provider [27].

Likewise, cloud computing has two important models [27] that define the main characteristics of cloud services (show 2-9).

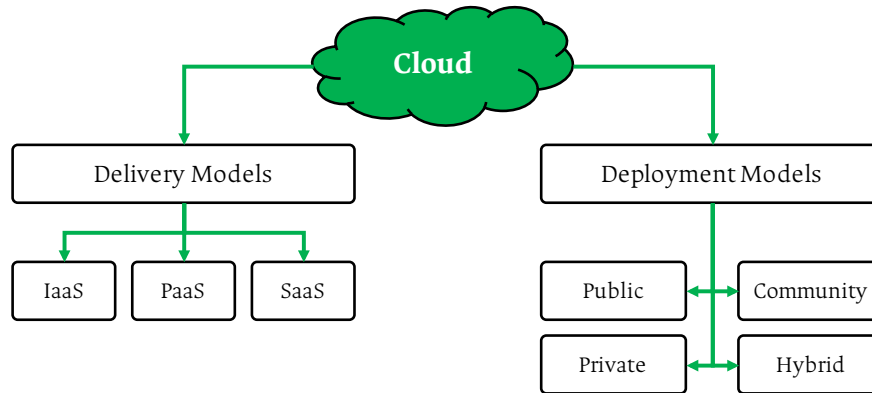


Figure 2-9.: Cloud Delivery and Cloud Deployment models.

- **Cloud Delivery Models:** it represents a specific, pre-packaged combination of IT resources offered by a cloud provider. There are three common delivery models which are described as follows [27]. *Infrastructure-as-a-Service (IaaS)* that represents an environment comprised of infrastructure resources. *Platform-as-a-Service (PaaS)* that represents an environment comprised of already deployed and configured resources to software creation. And *Software-as-a-Service (SaaS)* that represents a set of reusable software services which have a determined functionalities.
- **Cloud Deployment Models:** it represents a specific type of cloud environment, primarily distinguished by ownership, size, and access [27]. Thus, there are four common deployment models: public, private, community and hybrid clouds.

²See: <http://www.gartner.com/>

3. Related Work

This chapter provides a brief but comprehensive state of the art of the approximations related with the proposal of this research work. Hence, a set of five different approaches has been identified and they are described below. In general terms, these approaches have been classified from the scope of previous research works related with DSLs for software architecture and the automation of software development and deployment processes from a software architecture design.

3.1. Architecture Description Languages

The first approximation is the definition of an *Architecture Description Language (ADL)*. An ADL is a domain-specific language designed to model a software system [44]. The ADLs are formal languages that can be used to represent the architecture of a software-intensive system [18]. The main advantage is that an ADL describes the software and the hardware of a software system, covering different types of features, components, and processes such as threads, data, sub-programs, processors, devices and memory. There are different specifications of ADLs but in general terms, the ADLs are born with the objective of describe software systems from the conception of *components* and *connectors*. Independently of the ADL, it will describe the way as the components behave and the way they communicate with each other through connectors.

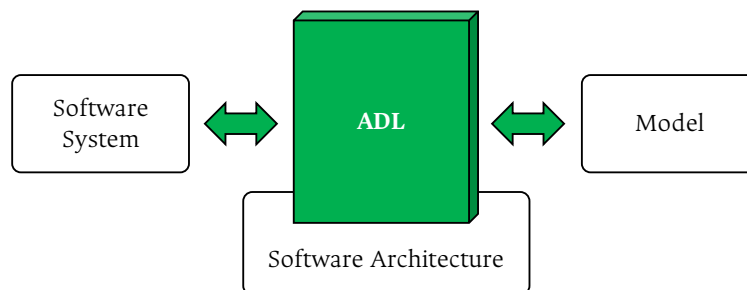


Figure 3-1.: Architecture Description Languages context.

Among the most relevant contributions related with ADLs, it is important to mention: ACME [31] and ACME++ [24], ADLs focused on the evaluation of nonfunctional attributes in traditional architectural styles like Client-Server and Peer-to-Peer. On the other hand, there are some approximations related with software systems based on distributed architectures like Component-

Based Architecture [57], [48] and Service-Oriented Architecture (SOA) [37], these ADLs are focused on the description of systems that have a collection of components/services and how they satisfy the functional and nonfunctional requirements by means of their constant interaction. It is worth mentioning AADL, an ADL that provides a means for the formal specification of the hardware and software architecture of embedded computer systems and system of systems [28].

Finally, there are some works focused on the design of software architectures from proposed DSLs, however these DSLs can be considered as ADLs due to its domain. E.g., Zdun with a DSL toolkit for taking decisions around a set of candidate architectures [56]. And Gilson et al. who propose a DSL for designing architectures from the conception of requirement interactions [33].

3.2. DSLs for Documenting Software Architectures

A second approximation is the definition of some architectural styles as Domain-Specific Languages, specifically the styles defined by Clements et al. in the Views and Beyond catalog [19]. This approach is used to generate automatic software architecture documentation. The work includes two relevant parts: the definition of the Domain-Specific Languages [23] and the build of tools that allows modeling Architectural Views [22] to define the functionality of automatic architecture document generation. Moreover, as a remarkable work, Demirli in [21] presents each one of the Module, Allocation and Component-and-Connector styles [19], [23] as a meta-model, which shows a MDD approach with the implementation of a tool called SAVE-Bench [22].

On the other hand, another work is related to the relationship that exists between the software architecture and the developers and others stakeholders with respect to the development process. In this way, Yazdanshenas et al. propose a DSL to allow developers to communicate properly through the abstraction of the development viewpoint of the Viewpoint Catalog [55] [49]. Its purpose is the communication between different stakeholders as engineers, testers, architects and developers.

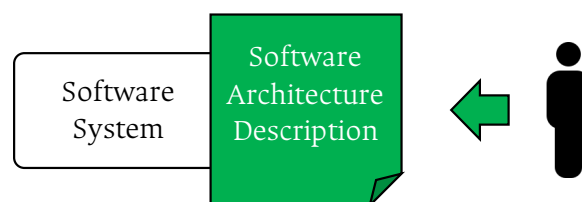


Figure 3-2.: Documenting software architectures.

3.3. Support the Software Development Process

Many works have been proposed in this field aimed at generating source code automatically; a remarkable one is WebDSL, a DSL designed to support the implementation of dynamic web applications from some features as a rich Data Model [10], the definition of user interface elements and actions that defines the behavior of the application. On the other hand, there are few works that use a DSL based on the specification of software architecture styles. One of these works is proposed by Cavalcante et al. [17], who introduce the implementation of a MDD process in which, from the π -ADL [47] architectural description, it is possible the semiautomatic source code generation in Go programming language. Likewise, Aldrich et al. [13] propose ArchJava, a Java extension that unifies Software Architecture with implementation, ensuring that the implementation conforms to architectural constraints; in this case, it is important to note that the extension is made based on the Component-and-Connector style [19].

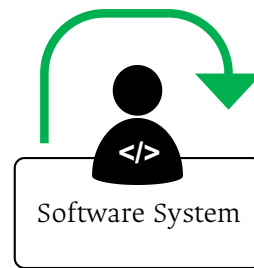


Figure 3-3.: Supporting the software development process.

3.4. Support the Software Deployment Process

It is very important to highlight some works to support the deployment process take into account the basis of model-driven engineering, i.e., automate the processes. As a remarkable work, Lascu et al. present an automatic deployment approach of component-based applications [41]. Specifically, connecting a large number of heterogeneous software components through the use of complex algorithms designed and formal model of components with the objective of computing the sequence of actions that permits the deployment of software system with a desired configurations [42]. In addition, Khalgui et al. present a similar approach but from an heuristic based method [38]. Likewise, Lan et al. propose an important work to automate the deployment process of large-scale systems [40]. In the first step, a tool visualizes the target software architecture in order to help deployers to understand the structure of the software system. In the second step, the tool can automatically generate the deployment information from the architecture description. And as the third step, the tool can monitor the hardware resource consumptions and deploy the subsystems onto multiple machines simultaneously.

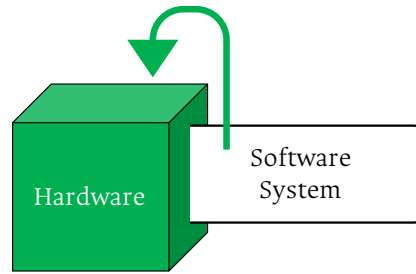


Figure 3-4.: Supporting the software deployment process.

On the other hand, Nicolas et al. propose an approach to the automatic deployment of embedded systems from an UML specification [46]. Hoenisch et al. and Lee et al. present an approach for the software auto-scaling using containers [35], [43], a way to package software in order to run isolated. More details of this technology is presented in the next chapters. In addition, Gas-sara et al. propose a formal method based on a formal language called BRS (Bigraphical Reactive System) in order to guarantee the correctness of the deployment architecture, also with an auto-scaling focus and component-based software systems [32]. Finally, Gonzalez-Herrera et al. [34] present a work that proposes a framework called *Kevoore*. This framework is an open-source component platform that allows the design of distributed software systems from the definition of their components, in order to have a dynamic adaption in a set of nodes with multiple execution platforms.

3.5. Deployment in a Cloud Computing Platform

Finally, there are some works related with the automatic deployment but in a cloud platform, e.g., a set of frameworks for the automatic deployment of component-based applications in different kind of cloud providers [16], [26]. The main objective of these works is to automate the way as a software system can make use of different resources of the cloud, specially, resource associated with the delivery models. Likewise, this seeks to integrate compute resources for higher efficiency on the software systems execution.

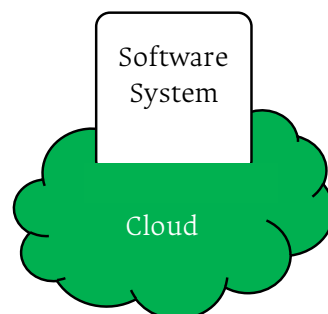


Figure 3-5.: Deploying software systems in the cloud.

4. Model-Driven Deployment

4.1. Definition

Taking into account the advantages of Model-Driven Engineering (MDE), a new approach is proposed in this research work: the concept of *Model-Driven Deployment (MDDep)*. MDDep in the context of MDE is shown in Figure 4-1.

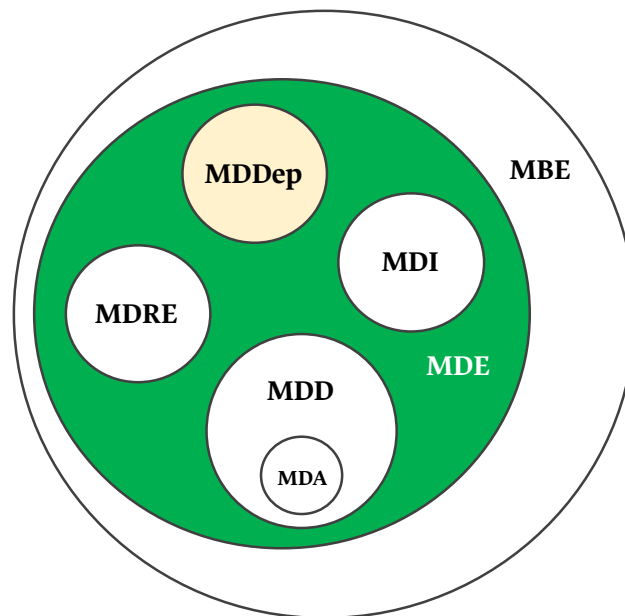


Figure 4-1.: MDDep in the context of MDE.

In this manner, Model-Driven Deployment (MDDep) is defined as a way of software deployment automation. It consists of abstracting at a high level, a model containing all the elements necessary to carry out the deployment of a software system in particular. From this model, we perform a set of transformations associated with a valid deploy scheme and after the last transformation, the software system is deployed according to the initial nonfunctional requirements.

The model that abstracts the process must be based on high-level architectural specifications that allow to differentiate and to relate aspects of both software and hardware. It is important to emphasize that this approach allows a more specific treatment of quality attributes of the software that recurrently depend on the harmony between the software and the elements of

the environment in which it is executed. Therefore, MDDep can be applied to architectural perspectives such as performance, scalability and high availability.

4.2. Methodology

In order to apply this approach, it is necessary to define some steps that allow the automatic deployment of software systems from a specified model. However, given the scope of this research work, the methodology below presents an approach applied to the *Performance and Scalability Perspective*.

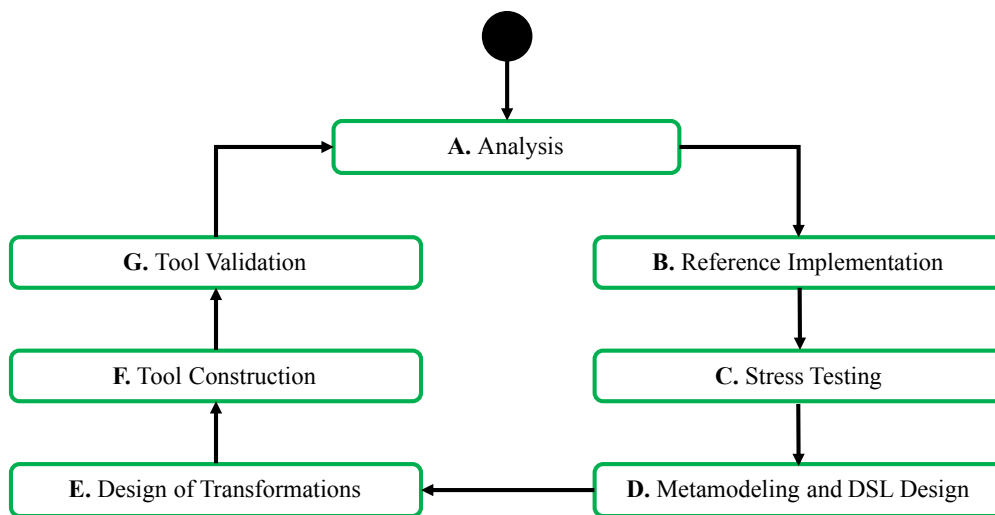


Figure 4-2.: MDDep methodology.

- A. **Analysis:** it consists of identifying the first software requirements associated with the patterns, styles and technologies used in the deployment of a particular software system in a particular infrastructure. It is important the identification of the elements, relations and properties exposed in a specific domain. For this research work, the domain is the set of architectural styles and architectural views.
- B. **Reference Implementation:** based on the results of the previous phase, a reference implementation of a software system deployed on a defined platform is performed manually. The reference implementation must be completely in line with a set of defined requirements.
- C. **Stress Testing:** stress tests are performed on a defined infrastructure, with the objective of obtaining the response times for each of the functional modules that are part of the software system. Subsequently, the same procedure is performed, but making changes in the deployment architecture, by means of the architectural tactic of horizontal scaling.

-
- D. **Metamodeling and DSL Design:** based on the above phases, this phase consists of the design of a set of DSLs and metamodels. Both the set of DSLs designed and the set of metamodels designed will be in charge of defining the MDDep process. In this research work, a DSL and a metamodel are built, based on the software architecture specifications.
 - E. **Design of Transformations:** this phase is in charge of mapping elements between the selected platforms and the formal definition of the selected model, in order to allow the automatic generation of a deployment scheme in the respective platform. This is done from models built thanks to the specification of the DSLs and metamodels of the previous phase.
 - F. **Tool Construction:** defining the DSLs, the metamodels and the respective transformations, a tool is built to support the creation of instances of the implemented DSLs, through a graphic editor that has validation, suggestion and autocomplete characteristics of code fragments. For this research work, the tool is the medium in which software architectures are created, using models that represent architectural views in the defined styles.
 - G. **Tool Validation:** in this phase and by means of the built tool, an instance of the defined model is designed, in order to automatically generate the deploying schemes. For this research work, the software architecture is designed for the software system associated with the reference implementation phase (phase B). Designed the architecture, validates the tool, from the automatic generation of the deployment scheme for the defined platform. The main objective is to ensure that both the deployment schema made in the reference implementation and the automatically generated deployment schema are the same. Finally, stress tests are performed again in order to validate the behavior of the software system in the infrastructure used by the tool.

5. Sarch Language

Sarch is a DSL proposed as part of this research, whose objective is to allow the design of software architectures. This design was based on a set of architectural styles proposed by Clements et al. in the (V&B) Catalog [19]. This styles cover the necessary elements to support the implementation and deployment of a software system based on distributed architectures.



Figure 5-1.: Sarch language.

The selected styles: Data Model, Layered, Component-and-Connector (C&C), and Deployment styles, are the basis for the definition of the Sarch structure and its design reflects the respective views.

On the other hand, in order to describe the design of Sarch language, it is important to highlight three relevant concepts related to the grammar design and the syntactic analysis of computer languages.

- **Grammar:** description of the structure of the elements presented in a language through the use of defined rules. This rules are the object of study of the *Syntax*.
- **Extended Backus–Naur Form (EBNF):** it is used to describe the syntax of computer languages. In particular, it is a notation for describing *context-free grammars*, i.e., grammars where some production rule is of the form $A \rightarrow \alpha$, where A is a nonterminal symbol and α is a string composed by a set of terminals and/or nonterminals.
- **Concrete Syntax Tree (CST):** also called *Parse Tree*, it is a tree data structure that represents the grammar (syntactic structure) of a computer language.

Thus, the complete Sarch design is presented in the following subsections as EBNF and CST representations.

5.1. Architectural Schema

This section of the grammar defines the general schema of a software architecture designed in Sarch. In this way, Sarch allows to start the architectural design with a terminal called *architecture*, followed by the name of the respective software system, and finally, with the definition of the four related architectural views. The general schema of the Sarch grammar is shown below (as EBNF representation) and in Figure 5-2 (as a CST).

$\langle \text{Architecture} \rangle ::= \text{'architecture' } \langle \text{Identifier} \rangle \text{'{' } \langle \text{Views} \rangle \text{'}'}$

$\langle \text{Views} \rangle ::= \langle \text{DataModelView} \rangle \langle \text{LayeredView} \rangle \langle \text{ComponentAndConnectorView} \rangle \langle \text{DeploymentView} \rangle$

$\langle \text{Identifier} \rangle ::= \{a-zA-Z0-9, _ \}$

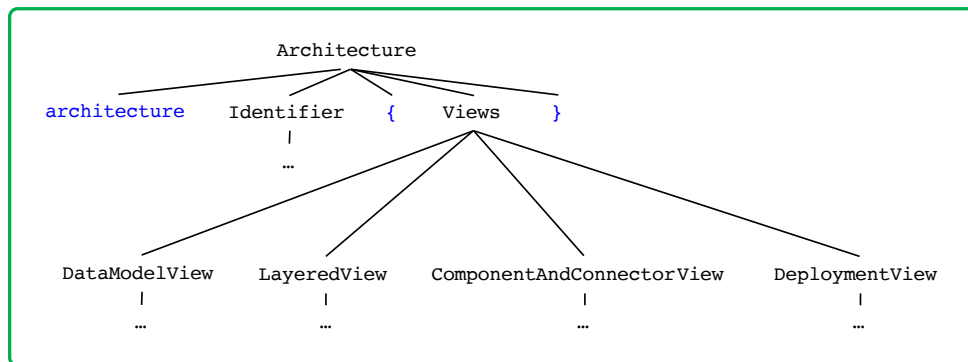


Figure 5-2.: CST for the architectural schema in Sarch language.

5.2. Data Model View

This view describes the static information structure of a software system. From the definition of architectural style, elements and relations of this view are described below:

- *Elements*: **data entity**, which is an object that holds the information that needs to be stored or somehow represented in a software system [19]. Its properties include mainly, *name* and *data attributes*.
- *Relations*: the data model view has a set of relations that allows the communication between data entities. Among the best known are **one-to-one**, **one-to-many** and **many-to-many** relations.

Based on the above, Sarch defines the Data Model view as one of its views. The grammar of this view represents the ability of define a set of data models, each one with a set of data entities, and a set of relations between them.

This view has a set of features related to the common data model specifications. For each data model defined, it is possible to define a set of attributes. Each attribute is represented by an associated *name* and a defined *type* (*String*, *int*, *float*, *double*, and *date*). On the other hand, this view allows the definition of a set of operations which are based on the traditional CRUD operations: *create*, *read* (*getAll* and *getById*), *update* and *delete*. The definition of these operations indicate that it can be applied over the attributes defined previously. The details of the view are shown below (as EBNF representation) and in Figure 5-3 (as a CST).

```

<DataModelView> ::= 'data-model-view' '::' <DataModel>+ '::'
<DataModel> ::= 'data-model' <Identifier> '{' <DataModelElements> <DataModelRelations> '}'
<DataModelElements> ::= 'elements' '{' <DataModelElement>+ '}'
<DataModelElement> ::= <DataEntity>
<DataEntity> ::= 'data-entity' <Identifier> '{' <Attributes> <Operations> '}'
<Attributes> ::= 'attributes' '{' <Attribute>+ '}'
<Attribute> ::= <DataType> <Identifier> ';
<DataType> ::= 'String' | 'int' | 'float' | 'double' | 'date'
<Operations> ::= 'operations' '{' <Operation>+ '}'
<Operation> ::= <OperationType> <Identifier> ';
<OperationType> ::= <GetAll> | <GetById> | <Create> | <Update> | <Delete>
<GetAll> ::= 'getAll'
<GetById> ::= 'getById'
<Create> ::= 'create'
<Update> ::= 'update'
<Delete> ::= 'delete'
<DataModelRelations> ::= 'relations' '{' <DataModelRelation>* '}'

```

$\langle \text{DataModelRelation} \rangle ::= \langle \text{RelationType} \rangle ' (' [\text{DataEntity}] ', ' [\text{DataEntity}] ') ' ;'$

$\langle \text{RelationType} \rangle ::= \langle \text{OneToOne} \rangle \mid \langle \text{OneToMany} \rangle$

$\langle \text{OneToOne} \rangle ::= \text{'one-to-one'}$

$\langle \text{OneToMany} \rangle ::= \text{'one-to-many'}$

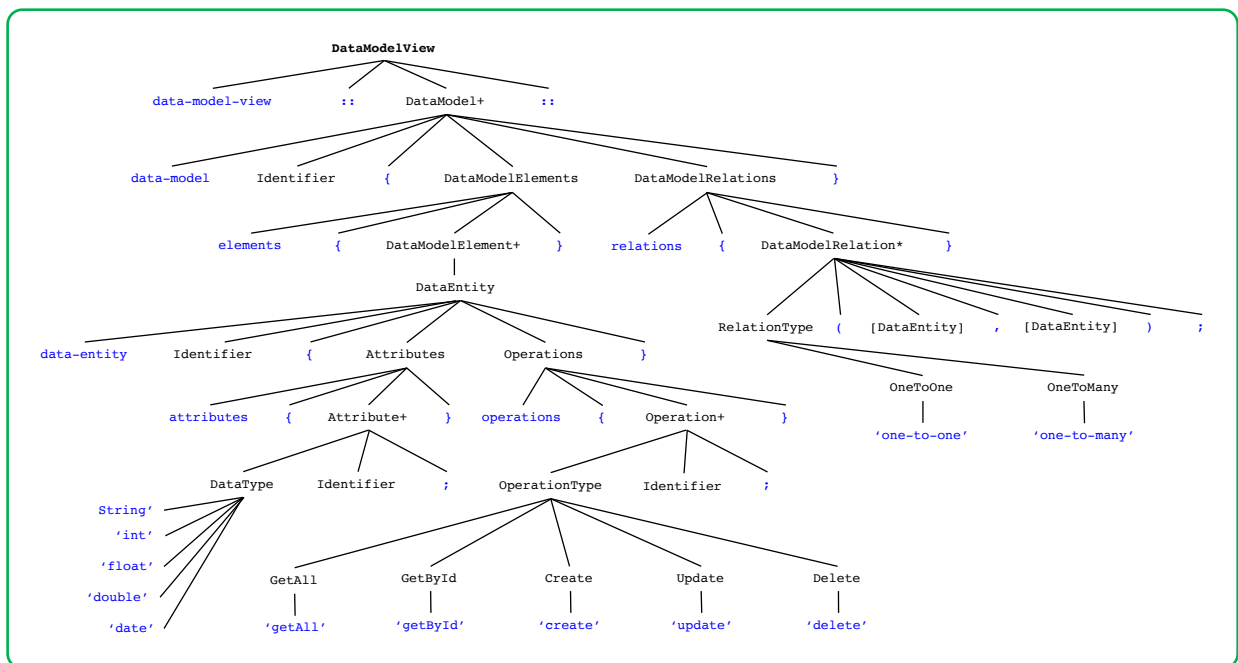


Figure 5-3.: CST for the Data Model view in Sarch language.

5.3. Layered View

This view describes a division of the software into units and each unit represents a group of modules that offers a cohesive set of services [19]. Elements and relations of this view are described below:

- *Elements*: **layer**, which is a fundamental object that holds a set of services to other layers.
- *Relations*: the layered view has a relationship that allows the communication between different layers: **allowed-to-use**. It indicates the way as a layer can use other or others layers.

Based on the above, Sarch defines the Layered view as one of its views. The grammar of this view represents the ability of define a set of layers, and a set of relations between them.

This view has a classification of its layers: *back-end* and *front-end* layers. These are defined by Sarch in order to separate the concerns related to the detailed design of special components of the C&C view. Nevertheless, this condition will be completely described in the next chapter. The details of the view are shown below (as EBNF representation) and in Figure 5-4 (as a CST).

$\langle \text{LayeredView} \rangle ::= \text{'layered-view' '::' } \langle \text{LayeredElements} \rangle \langle \text{LayeredRelations} \rangle \text{'::'}$

$\langle \text{LayeredElements} \rangle ::= \text{'elements' '{' } \langle \text{LayeredElement} \rangle^+ \text{'}'}$

$\langle \text{LayeredElement} \rangle ::= \langle \text{Layer} \rangle$

$\langle \text{Layer} \rangle ::= \text{'layer' '(' } \langle \text{LayerType} \rangle \text{')' } \langle \text{Identifier} \rangle \text{' ;'}$

$\langle \text{LayerType} \rangle ::= \langle \text{BackEndLayer} \rangle \mid \langle \text{FrontEndLayer} \rangle$

$\langle \text{BackEndLayer} \rangle ::= \text{'back-end'}$

$\langle \text{FrontEndLayer} \rangle ::= \text{'front-end'}$

$\langle \text{LayeredRelations} \rangle ::= \text{'relations' '{' } \langle \text{LayeredRelation} \rangle^* \text{'}'}$

$\langle \text{LayeredRelation} \rangle ::= \langle \text{AllowedToUse} \rangle$

$\langle \text{AllowedToUse} \rangle ::= [\text{Layer}] \text{'allowed-to-use' } [\text{Layer}] \text{' ;'}$

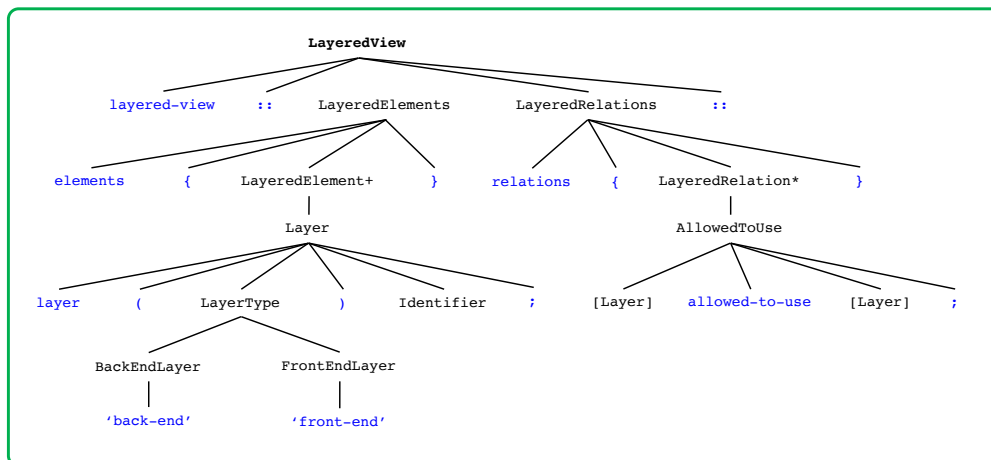


Figure 5-4.: CST for the Layered view in Sarch language.

5.4. Component-and-Connector (C&C) View

This view describes elements that have some runtime presence [19]. Elements and relations of this view are described below:

- *Elements*: **component**, which represents the principal computational element or data store that is present at runtime. And **connector**, which is a runtime pathway of interaction between two or more components.
- *Relations*: the C&C view has a relationship that allows to attach components and connectors: **attachment**. It indicates the way as a connector is associated with two components.

Based on the above, Sarch defines the C&C view as one of its views. The grammar of this view represents the ability of define a set of components and connectors, and a set of relations between them.

This view presents a set of predefined components and connectors. Components can be of type *database*, *back-end*, *front-end* and *load-balancer*. On the other hand, connectors can be of type *jdbc*, *rest*, and *http*. Whenever a component is of type *database*, Sarch allows the possibility of relating a data model designed in the Data Model view. In addition, the attachment relation is defined by a connector and two components. However, these concepts will be completely described in the next chapter. The details of the view are shown below (as EBNF representation) and in Figure 5-5 (as a CST).

```
⟨ComponentAndConnectorView⟩ ::= 'component-and-connector-view' '::' ⟨ComponentAndConnectorElements⟩
    ⟨ComponentAndConnectorRelations⟩ '::'
```

```
⟨ComponentAndConnectorElements⟩ ::= 'elements' '{' ⟨ComponentAndConnectorElement⟩+ '}'
```

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

```
⟨ComponentElement⟩ ::= 'component' ⟨ComponentType⟩ ⟨Identifier⟩ [DataModel]? ';'
```

```
⟨ComponentType⟩ ::= ⟨Database⟩ | ⟨BackEnd⟩ | ⟨FrontEnd⟩ | ⟨LoadBalancer⟩
```

```
⟨Database⟩ ::= 'database'
```

```
⟨BackEnd⟩ ::= 'back-end'
```

```
⟨FrontEnd⟩ ::= 'front-end'
```

```
⟨LoadBalancer⟩ ::= 'load-balancer'
```

```

<ConnectorElement> ::= 'connector' <ConnectorType> <Identifier> ';'
<ConnectorType> ::= <JDBC> | <REST> | <HTTP>
<JDBC> ::= 'jdbc'
<REST> ::= 'rest'
<HTTP> ::= 'http'
<ComponentAndConnectorRelations> ::= 'relations' '{' <ComponentAndConnectorRelation> '*' '}'
<ComponentAndConnectorRelation> ::= <Attachment>
<Attachment> ::= 'attachment' '(' [ConnectorElement] ':' [ComponentElement] ',' [ComponentElement] ')' ';'

```

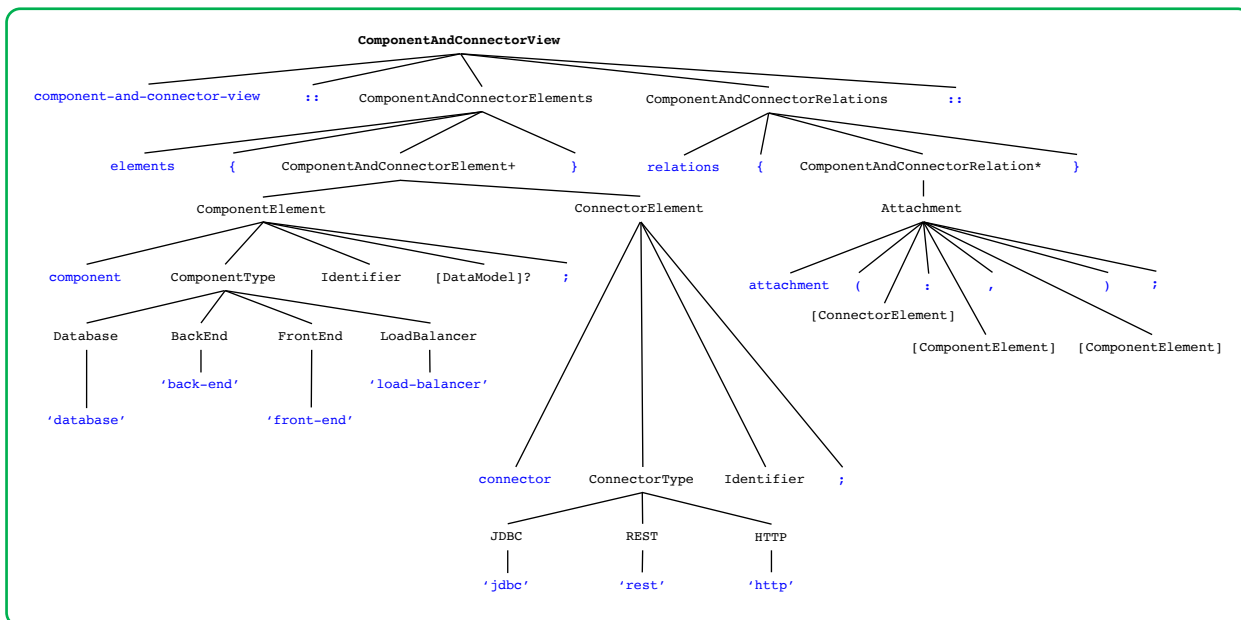


Figure 5-5.: CST for the Component-and-Connector (C&C) view in Sarch language.

5.5. Deployment View

This view describes the way as software elements are allocated to a hardware infrastructure [19].

- *Elements*: **software element** which is a components of the C&C view[19]. And **environmental element** which is a hardware of a computing platform.

- *Relations*: the deployment view has a relationship that allows the mapping between software elements and environmental elements: **allocated-to** (also called **deployed-in**).

Thus, Sarch defines the Deployment view as one of its views. The grammar of this view represents the ability of define a set of software and environmental elements, and a set of relations between them.

This view allows to associate the components defined in the C&C view as software elements. On the other hand, Sarch proposes two types of environmental elements: server-node and *container*. A server-node has the information associated with a server that will act as the infrastructure manager in a deployment scenario. A container expresses the specification of how a software element is deployed. Information like execution environments, ports and scaling values is required, but it will be completely described in the next chapter because this view is the basis of the Model-Driven Deployment (MDDep) process. The details of the view are shown below (as EBNF representation) and in Figures 5-6 (part A) and 5-7 (part B) (as a CST). It is important to mention that the parts A and B conform the same tree, this division is made due to the size of the tree and the difficulty of showing it in a single figure. The element that connects the two parts of the tree is the nonterminal *Container*.

```

<DeploymentView> ::= 'deployment-view' '::' <DeploymentElements> <DeploymentRelations> <PerformanceTests>?
    '::'

<DeploymentElements> ::= <SoftwareElements> | <EnvironmentalElements>

<SoftwareElements> ::= 'software-elements' '{' <SoftwareElement>+ '}'

<SoftwareElement> ::= <Identifier> '( 'component' [ComponentElement] )' ';'

<EnvironmentalElements> ::= 'environmental-elements' '{' <EnvironmentalElement>+ '}'

<EnvironmentalElement> ::= <ServerNode> | <Container>

<ServerNode> ::= 'server-node' <Identifier> '{' <ServerNodeDetails> '}'

<ServerNodeDetails> ::= 'ip' ':' <Identifier> ';' 'port' ':' <Identifier> ';' 'access-key' ':' <Identifier> ';'
    'secret-key' ':' <Identifier> ';'

<Container> ::= 'container' <Identifier> '{' <ContainerDetails> '}'

<ContainerDetails> ::= 'associated-stack' ':' <Identifier> ';' 'execution-environment' ':' <ExecutionEnvironment>
    ';' 'host-label' ':' <Identifier> ';' 'port' ':' <Identifier> ';' 'scale' ':' <Identifier> ';'

<ExecutionEnvironment> ::= <DatabaseEnvironment> | <ApplicationEnvironment> | <LoadBalancingEnvironment>

```

$\langle \text{DatabaseEnvironment} \rangle ::= \langle \text{MySQL} \rangle$

$\langle \text{MySQL} \rangle ::= \text{'mysql'}$

$\langle \text{ApplicationEnvironment} \rangle ::= \langle \text{GlassFish} \rangle$

$\langle \text{GlassFish} \rangle ::= \text{'glassfish'}$

$\langle \text{LoadBalancingEnvironment} \rangle ::= \langle \text{HAProxy} \rangle$

$\langle \text{HAProxy} \rangle ::= \text{'haproxy'}$

$\langle \text{DeploymentRelations} \rangle ::= \text{'relations' } \{ \langle \text{DeploymentRelation} \rangle + \}$

$\langle \text{DeploymentRelation} \rangle ::= [\text{SoftwareElement}] \langle \text{AllocatedTo} \rangle [\text{Container}] \text{' ;'}$

$\langle \text{AllocatedTo} \rangle ::= \text{'allocated-to' } | \text{'deployed-in'}$

$\langle \text{PerformanceTests} \rangle ::= \text{'performance-tests' } (\text{'target-node-ip' } \text{' ;' } \langle \text{Identifier} \rangle \text{' ;' }) \text{' ;'}$

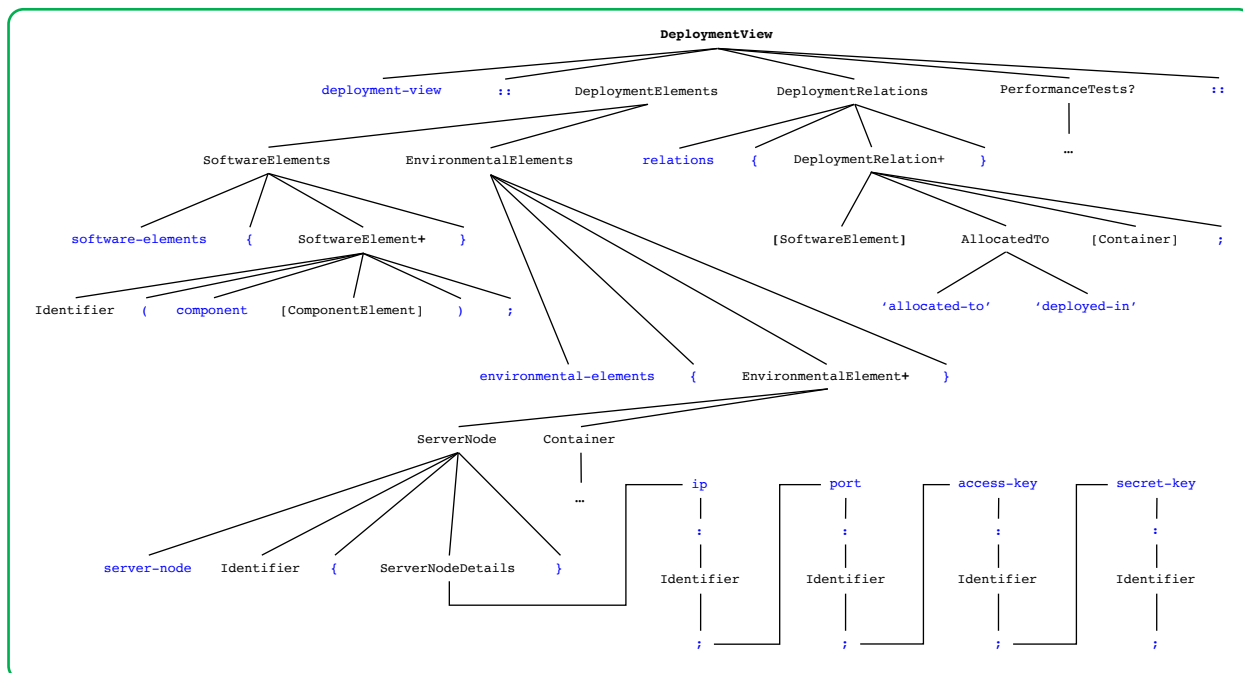


Figure 5-6.: CST for the Deployment view in Sarch language (part A).

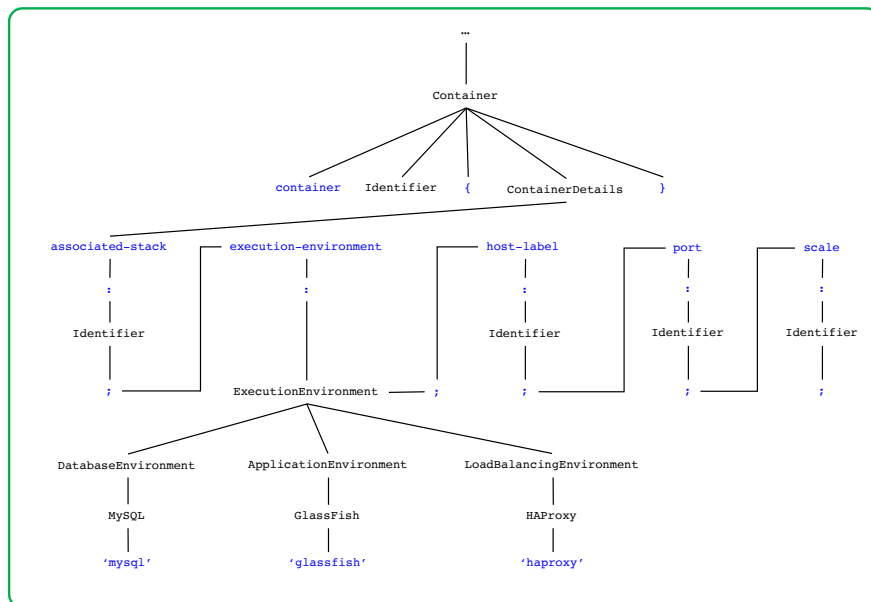


Figure 5-7.: CST for the Deployment view in Sarch language (part B).

6. Sarch-Studio Tool

Sarch-Studio is a tool developed as part of this research work, whose objective is to define a model-driven environment to automate the implementation and deployment of software systems designed from a software architecture in Sarch language.

6.1. Sarch-Studio Architecture

The architecture of Sarch-Studio is composed of two components: the *Core* and the *Editor*. These components were built using a set of tools of the Eclipse Modeling Project ¹. Thus, a representation of the layered architecture of the tool is shown in Figure 6-1.

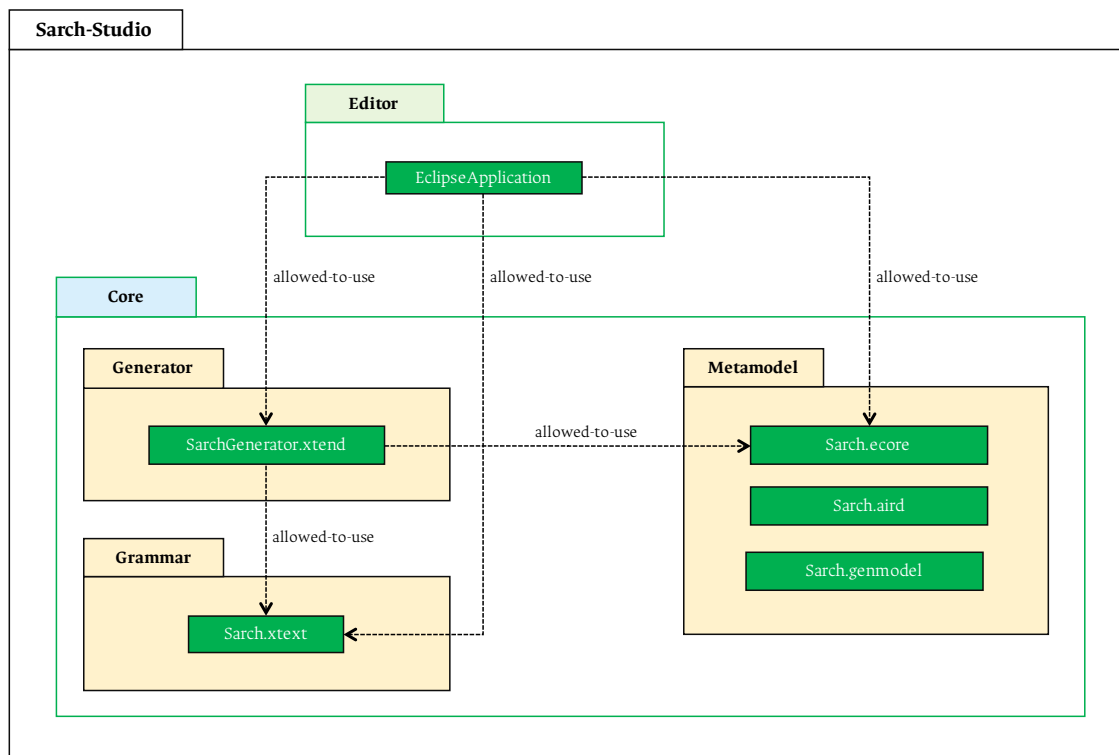


Figure 6-1.: Layered architecture of Sarch-Studio.

¹See: <http://www.eclipse.org/modeling/>

6.1.1. Associated Technologies

The following is a brief description of the tools used in the tool construction.

- **Xtext:** it is an open-source framework for the development of programming languages and DSLs. Xtext has its own infrastructure which comprises an EBNF grammar language, a parser, a linker, a typechecker, a compiler and a textual editor for Eclipse [12].
- **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 [51].
- **Xtend:** it is a dialect of Java programming language that has as one of its main features the support for code generation [11].



Figure 6-2.: Xtext, Eclipse Modeling Framework (EMF), and Xtend.

6.1.2. Components

The following is the description of the two components presented in Sarch-Studio.

6.1.2.1. Core Component

The Core component integrates the basis of the Sarch design with its grammar, the way as will be done the model transformations from Sarch, and the structure of the metamodel that abstracts Sarch in terms of a model-driven context. In this way, the mentioned characteristics are described below as tree different modules:

- **Grammar Module:** this module defines the specifications of the Sarch grammar through the use of Xtext. It allows to define all the syntax rules presented in Sarch, taking into account the set of terminals and nonterminals. Grammar module specification is shown in Figure 6-3.
- **Generator Module:** this module specifies a set of transformation rules through the use of Xtend, i.e., it defines the transformations used by the model-driven processes which will be described in the next subsections. Generator module specification is shown in Figure 6-4.

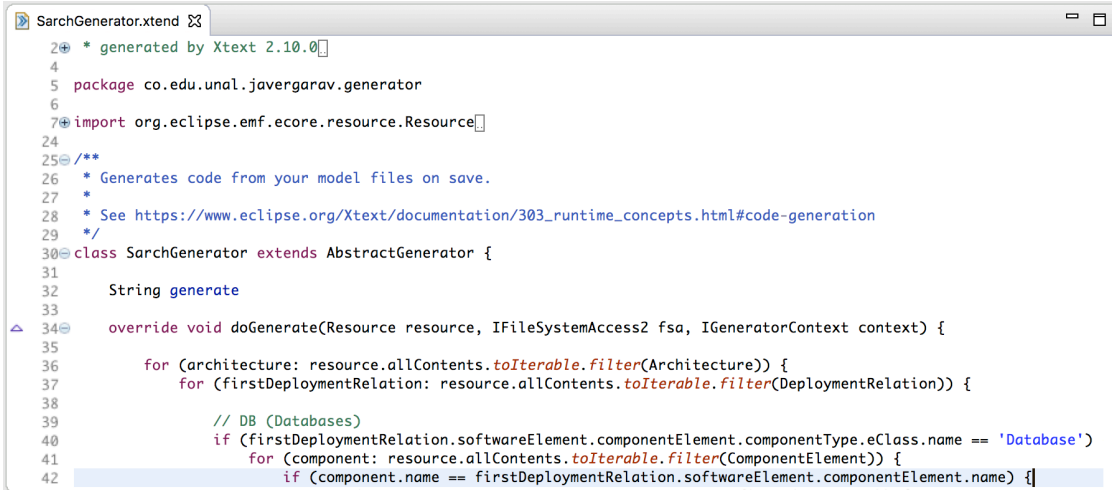


```

1 grammar co.edu.unal.javergarav.Sarch with org.eclipse.xtext.common.Terminals
2
3 generate sarch "http://javergarav.unal.edu.co/Sarch"
4
5 // Architecture
6
7 Architecture:
8   'architecture' name=ID '{'
9     dataModelView = DataModelView
10    layeredView = LayeredView
11    componentAndConnectorView = ComponentAndConnectorView
12    deploymentView = DeploymentView
13   '}'
14 ;|
15

```

Figure 6-3.: Definition of Sarch grammar in the Xtext environment.



```

2+ * generated by Xtext 2.10.0
4
5 package co.edu.unal.javergarav.generator
6
7+ import org.eclipse.emf.ecore.resource.Resource
24
25+ /**
26  * Generates code from your model files on save.
27  *
28  * See https://www.eclipse.org/Xtext/documentation/303_runtime_concepts.html#code-generation
29  */
30+ class SarchGenerator extends AbstractGenerator {
31
32   String generate
33
34+ override void doGenerate(Resource resource, IFileSystemAccess2 fsa, IGeneratorContext context) {
35
36     for (architecture: resource.allContents.toIterable().filter(Architecture)) {
37       for (firstDeploymentRelation: resource.allContents.toIterable().filter(DeploymentRelation)) {
38
39         // DB (Databases)
40         if (firstDeploymentRelation.softwareElement.componentElement.componentType.eClass.name == 'Database')
41           for (component: resource.allContents.toIterable().filter(ComponentElement)) {
42             if (component.name == firstDeploymentRelation.softwareElement.componentElement.name) {}

```

Figure 6-4.: Definition of Sarch transformations in the Xtend environment.

- **Metamodel Module:** this module includes the definition of the Sarch grammar as a metamodel, specifically, the metamodel used by the model-driven processes. This metamodel is designed through the use of EMF. It is important to highlight that the metamodel defines the structure of the elements that allows to carry out the model-driven phases and fully guarantees the syntax defined for Sarch. Sarch language metamodel is presented in Appendix A.

6.1.2.2. Editor Component

The Editor component is an Eclipse application that allows to write in Sarch language with the aim of designing a software architecture for a particular software system. The editor has a set of features like syntax coloring and validation of grammar elements. A view of the editor component is shown in Figure 6-5.

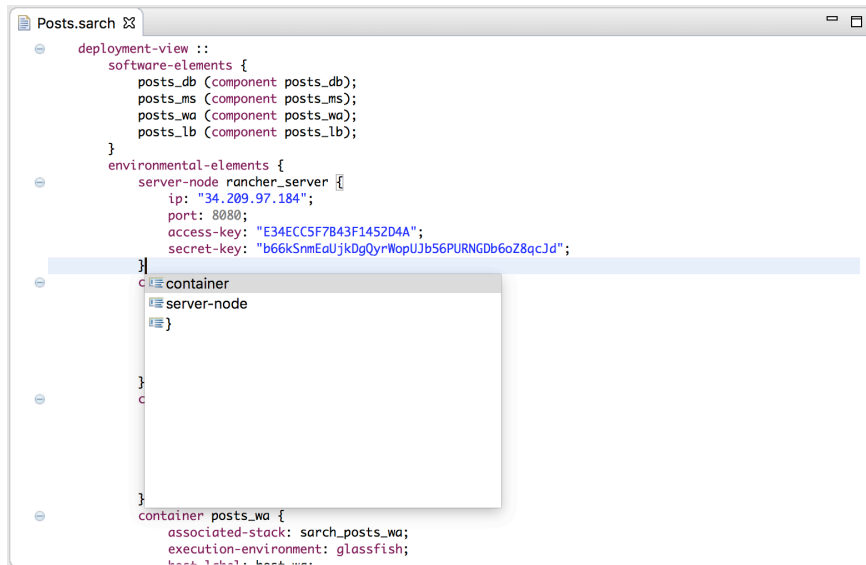


Figure 6-5.: Editor component of Sarch-Studio.

6.2. Target Software Architecture

Sarch-Studio defines target software architecture that supports the model-driven processes. In this way, this architecture includes all the possible elements of a software system that has been designed through Sarch language, specifically, elements related to code generation (implementation) and deployment.

Based on the above, in order to define an adequate software architecture, a *Microservices Architecture (MSA)* was selected. The main reasons for which this architecture was selected is because the microservices architecture offers a complete specifications of the scope of each one of the components in the architecture. This benefits the interaction of the elements presented by Sarch, i.e., the elements associated with each one of the architectural view. In addition, the above also offers great advantages in terms of the application of the architectural tactic of scalability. This is because as a fully distributed architecture, each component can be scaled independently without affecting others, thus increasing the performance of the respective software system functionalities.

6.2.1. Associated Technologies

A set of technologies was selected in order to support the implementation and deployment of a software system based on the target architecture. The following is a brief description of the technologies used for create software systems based on the target architecture.

- **MySQL:** it is a relational Database-Management System (DBMS). MySQL is considered as the most popular open source relational database in the world [8].

- **Java:** it is a general-purpose (GPL), concurrent, and object-oriented programming language that was specifically designed to allow developers to write a program once and run it on any device. [7].
- **Apache Maven:** it is a tool used primarily for the automation build of Java projects. It describes how software is built and how are its dependencies. [3].
- **GlassFish:** it is an open-source application server project focused on the Java EE platform. [5].
- **Docker:** it is a software platform that allows the deployment of different kind of software artifacts in elements called *Containers*. A container wraps a piece of software in a complete file system that contains everything needed to run. [4].



Figure 6-6.: MySQL, Java, Maven, GlassFish, Docker, Rancher, HAProxy, Amazon Web Services (AWS), and JMeter.

- **Rancher:** it is a container management platform that allows to deploy and run containers in production on any infrastructure. [9].
- **HAProxy:** it is a software solution that offers services as high availability, load balancing, and proxying for TCP and HTTP-based applications. [6].
- **Amazon Web Services (AWS):** it is a platform that provides on-demand cloud computing platforms to different kind of consumers. AWS can be considered as one of the most popular cloud providers in the world [1].

- **Apache JMeter:** it is a tool that allows the execution of different types of software testing, specially load tests. Its objective is analyzing and measuring the performance of web-based software systems [2].

6.2.2. Software Architecture Description

The target architecture is shown in Figure 6-7 and described as follows.

- A set of components which can be of type: *database (DB)*, *microservice (MS)*, *web application (WA)*, and *load balancer (LB)*.
- Databases to handle the software system information.
- Microservices to handle the business logic of the software system.
- Web applications to handle the user interaction with the software system.
- Load balancer to redirect requests to web applications
- A set of connectors that can be JDBC (Java Database Connectivity) type for connecting microservices and databases, REST (REpresentational State Transfer) type for connecting web applications and microservices, or HTTP (HyperText Transfer Protocol) type for connecting load balancers and web applications.
- A set of data models in which each one has at least one data entity and a set of relations between the entities presented.
- Each data model is associated with a database. A database has no more than one associated data model.
- A set of layers for representing the division of microservices and web applications into units.
- A set of software elements, defined from the specified components.
- A set of environmental elements to allows the software elements deployment.
- Each software element is deployed in an individual container.
- A set of execution environments that can be GlassFish as application server for microservices and web applications, MySQL as database server for databases, or HAProxy as server for load balancers.
- Each container has an execution environment and a specific port.
- Each component can have more than one instance, i.e., it can be scaled horizontally.

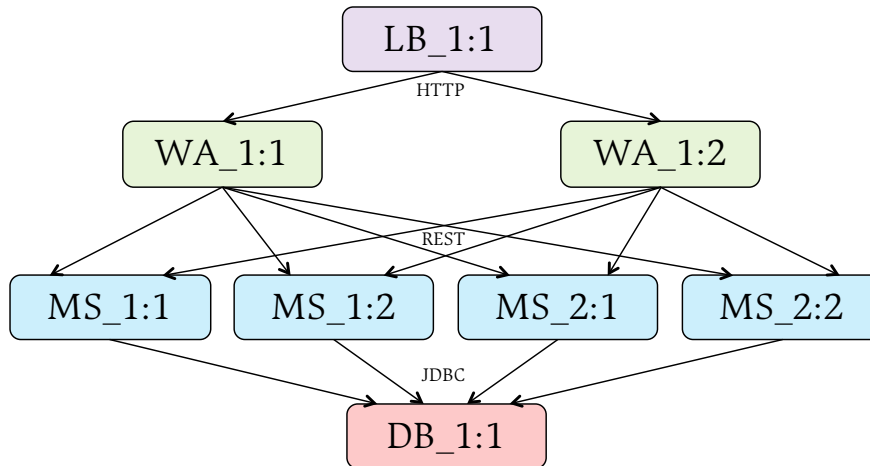


Figure 6-7.: Target architecture from the point of view of the C&C view in a semiformal representation

Notation: {DB, MS, WA, LB}_{identifier}:{instance}. In this case, there is a load balancer (LB) with one instance, there is a web application (WA) with two instances, there are two microservices (MS) with two instances each one, and there is a database (DB) with one instance.

6.3. Model-Driven * Processes

The main approach of Sarch-Studio is to take the advantages of Sarch language in order to automate two important aspects of the software engineering field: *development* and *deployment*. The first, which is automated by means of Model-Driven Development (MDD), and the second, which is automated by means of Model-Driven Deployment (MDDep).

6.3.1. Automation of Development

The transformations of the MDD process are defined using Xtend. Thus, the mapping between a software architecture designed in Sarch and a web-based software system with the target architecture is:

- For each component of type: microservice or web application, a Java project is created.
- The source code of a microservice Java project is organized (through packages) according to the layers defined as back-end type.
- The source code of a web application Java project is organized (through packages) according to the layers defined as front-end.

- For each component of type database, a database schema is created based on the associated Data Model.
- Each data model defines the Java entity classes of the model layer (model packages) of the associated microservice and the web application.
- For each JDBC connector, the database connection between the microservice and the created database is defined.
- For each REST connector, in the associated microservice, a set of Java classes (service layer: service package) are created, according to the respective data entities, for defining the business logic based on the declared CRUD operations. Additionally, a set of Java classes (resource layer: resource package) are created, according to the respective data entities, for defining the available endpoints which allow the consumption of services.
- For each REST connector, in the associated web application, a set of Java classes (service layer: service package) are created, according to the respective data entities, for defining the business logic to consume the services exposed by the microservice. Additionally, a set of HTML and CSS files (web layer: web package) are created, for defining the web pages that allows the user interaction with the different available operations; and finally, a set of Java classes (bean layer: bean package) are created, for defining the connection between web pages and services.

6.3.2. Automation of Deployment

This process is mainly based on the deployment view defined in Sarch. The main characteristic of this view is the capacity to define important information about the way of how the software will be deployed. Sarch requires three kind of information: information about the software elements to be deployed (C&C elements), information about the environmental elements where the software elements will be deployed, and information about the way in which software elements and environmental elements are related.

It is necessary to emphasize in the environmental elements because they are the basis for the MDDep approach. Sarch defines two types of environmental elements: *server-node* and *containers*. The server node must be unique and it describes the information of a server allocated on the cloud which have control over all nodes where the software system can be deployed. This server must be configured previously, using Rancher. Basically, Rancher permits to add a set of nodes (virtual servers) from different cloud providers. For this case, we use an Amazon Web Services (AWS) infrastructure, specifically, using the delivery model of Infrastructure as a Services (IaaS). Once the server node is ready (with nodes added), Sarch can generate the deployment schema defined in the architecture created:

- Sarch uses Apache Maven to build previously created Java projects.
- Sarch uses Docker to generate a schema for DB, MS, WA and LB, by means of three generated files: a "Dockerfile" that defines the information about the execution environment to be created, a "docker-compose" that allows the creation of the environment in the Dockerfile (container name and port where it must be deployed), a "rancher-compose" that defines the scaling value, i.e., the number of times the component will be scaled in its deployment (number of instances to be created), and a "deploy-cloud" that defines the server node information, in order to deploy the respective component on the cloud (url, access key and secret key: information provided by the Rancher API in the server node).
- For each component, Sarch deploy it in a set of containers, depending of the scale value in Sarch.
- For each container, Sarch automates the necessary configurations for its execution environment, depending of the respective component type: MySQL for the databases, GlassFish for the microservices and web applications, and HAProxy for the load balancers.
- Sarch uses the Apache JMeter specifications to generate a set of stress test plans for the POST (Create) operations of the microservices. These plans allows the automatic execution of a set of load tests in order to know the maximum performance of the system in different scenarios of concurrent users.
- Finally, it is important to highlight that for this first work using Sarch language, the scale value for DB and LB is restricted to 1, while the scale value for MS and WA is restricted to $n - 1$, where n is the maximum number of nodes added on the server node. The main reasons of this restriction are: 1) Scaling databases involves the application of architectural tactics for the high availability quality attribute, especially to maintain a synchronization of the information of all instances of the same database. However, the scope of this research work is focused only on performance and scalability; and 2) Scaling load balancers involves having different access points to the deployed software system, i.e., for the scope of this research work, stress tests could be run by any of these points (ports associated with load balancers).

However, in Figure 6-8 is represented the final interaction between Sarch-Studio and the cloud platform (AWS), starting from the definition of the architectural model in Sarch language, until the deployment in a cloud infrastructure previously configured.

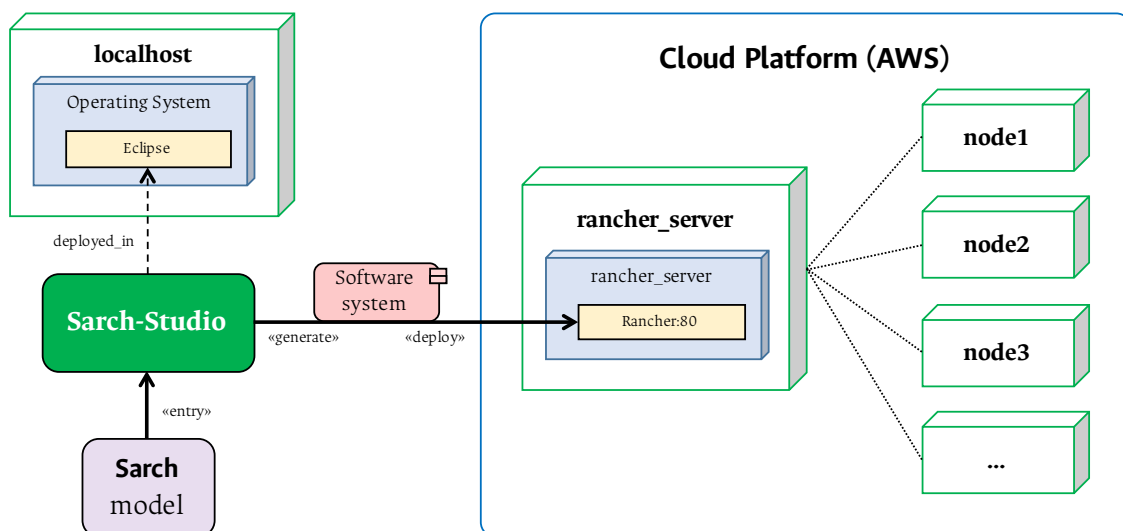


Figure 6-8.: Final interaction between Sarch-Studio and the Cloud Platform.

7. Evaluation

7.1. Conceptual Analysis

It is important to highlight some important aspects about the five approaches presented in the related work and the approach presented with Sarch, in order to compare its scope and its advantages with respect to others. Due to the characteristics of software architecture, it is necessary to check how the main aspects abstracted of the five approaches are covered by Sarch (show Table 7-1). The approaches analyzed are: 1st: ADLs [44], [31], [24], [57], [48], [37], [28], [56], [33]; 2nd: DSLs for documenting SA [23], [22], [21], [55]; 3rd: support the software development process [10], [17], [13]; 4th: support the software deployment process [41], [42], [38], [40], [46], [35], [32], and 5th: deployment in a cloud platform [16], [26].

In the same way, the characteristics analyzed are: A: use of styles/views, B: SA description, C: use of DSLs, D: use of Model-Driven *, E: automatic source code generation, F: automatic deployment, and G: cloud support. In this way, Sarch uses a notation defined from a set of architectural styles/views to describe and model the relevant elements of an architectural design. In addition, the Sarch grammar and its textual representation allow an automatic software architecture documentation. Moreover, Sarch addresses the software development process with the automatic code generation from a set of defined functionalities and a MDD process. In addition, it is important to mention that the main advantages of the Sarch approach is its support for the automatic software deployment in a cloud platform. This allows the ability to take architectural decisions from the model in order to applying the performance and scalability perspective through a dynamic scaling of components.

Table 7-1.: Conceptual analysis of Sarch: approaches vs. main aspects.

	A	B	C	D	E	F	G
Architecture Description Languages		X	X		X		
DSLs for Documenting Software Architectures	X	X	X	X			
Support the Software Development Process	X		X	X	X		
Support the Software Deployment Process						X	
Deployment in a Cloud Platform						X	X
Sarch Approach	X	X	X	X	X	X	X

7.2. Practical Analysis

A case study is presented as part of the approach evaluation. In this way, an architecture has been designed in Sarch, in order to analyze the results of the software system implementation and deployment. The case study consists on a web application, called *PostsApplication*, to handle *posts*. In particular, the application allows to perform the four CRUD operations for *posts*, which have two attributes: *title* and *content*. It is important to highlight that this case study is based on the reference implementation performed as part of the methodology defined to develop the research work, i.e., *PostsApplication* was first developed manually. The graphic representation of the views of this case study are shown in Appendix B. Likewise, its representation in Sarch language is shown in Appendix C. In addition, some fragments of the software system implemented (reference implementation) and generated (Sarch-Studio processes) are shown in Appendix D.

Emphasis is placed on Deployment and C&C styles because they are the basis for the MDDep approach. However, Data Model and Layered styles are the basis for the implementation of the components designed for the case study. The server node was previously configured in an AWS server. Likewise, a set of four nodes with the same hardware specifications was configured: Amazon EC2, t2.large, Ubuntu 16.04.1 as OS, 2x2.4 GHz, memory: 8 GiB, storage: 16 GiB. In the same way, Figure 7-1 shows the architecture visualization from Rancher server for the case study after the transformations. In this figure it is possible to see the number of containers created in each node, corresponds to the one specified in the respective deployment view: one web application instance and three microservice instances.

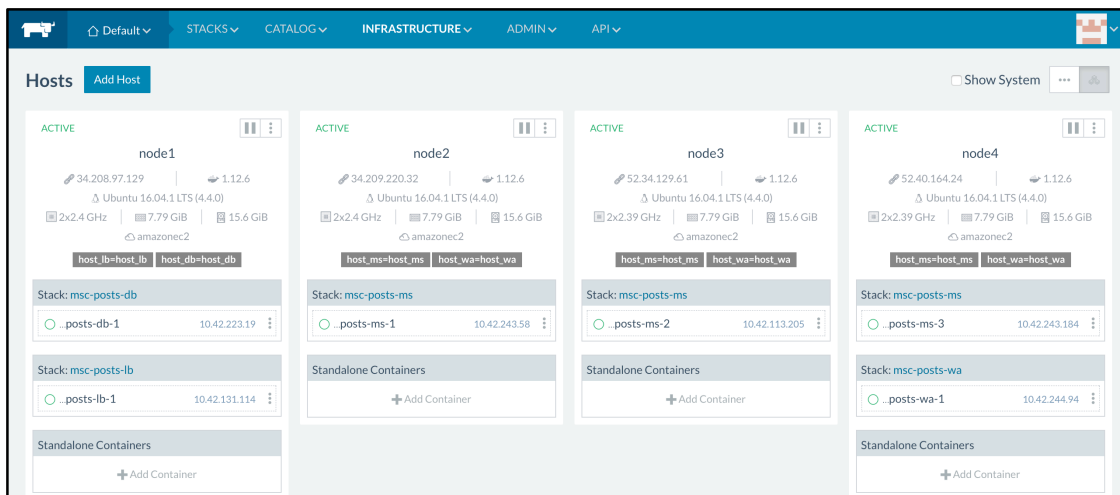


Figure 7-1.: Rancher server environment with 4-nodes infrastructure on the cloud.

For the analysis, a set of test cases was selected to analyze the performance of the system when changes are made to the scale value in Sarch (for microservices and web applications), i.e., when there are different scenarios for the horizontal scaling tactic. In the same way, each scenario

Table 7-2.: Stress test results.

Users	Scenario 1		Scenario 2		Scenario 3	
	MS: 1, WA: 1		MS: 2, WA: 1		MS: 3, WA: 1	
	RT	T	RT	T	RT	T
0	0	0.0	0	0.0	0	0.0
1	210	49.6	199	50.0	194	50.3
5	218	246.3	203	249.4	200	250.0
50	310	2290.1	223	2453.0	235	2429.1
100	503	3992.0	491	4024.1	484	4043.1
200	1185	5492.0	1067	5805.5	1061	5822.4
250	1599	5771.5	1319	6468.3	1228	6732.5
300	-	-	1794	6442.4	1603	6915.1
310	-	-	2010	7948.7	1682	6935.1
360	-	-	-	-	2404	8299.2
400	-	-	-	-	-	-

was tested from the generated stress test plan for the "create posts" functionality in the system, and with different number of concurrent users (requests). The results of this analysis is shown in Table 7-2, where the numeric values represent the average Response Time (RT), i.e., the length of time the system takes to give an answer to a request (time in milliseconds) [49], and the Throughput (T), i.e., the amount of workload the system is capable of handling in a unit time period (transactions/minute) [49]. In addition, a graphical representation of the system performance in each of the scenarios is shown in Figures 7-2 and 7-3, where in the first representation it is also possible to see the knees for each of the curves, that is, the point at which the system reaches its maximum responsiveness (peak load behavior) [49]. For conducting the tests was used JMeter, it allowed the execution of each scenario with the different alternatives in the number of concurrent users, from the stress test plans generated by Sarch.

It is important to mention that the *ramp-up* period (in seconds) used in the tests is 1. This parameter describes the period in which a certain number of users make requests to the system; for example, for a test with 1 user, this represents that a user accesses the system every second, and for a test with 100 users, this represents that 100 users access the system every second.

Based on the above, it can be evidenced that when horizontal scaling is applying, the response times decrease and the throughput increases. This implied that the scalability objective is satisfied correctly and the design initially done with Sarch is validated.

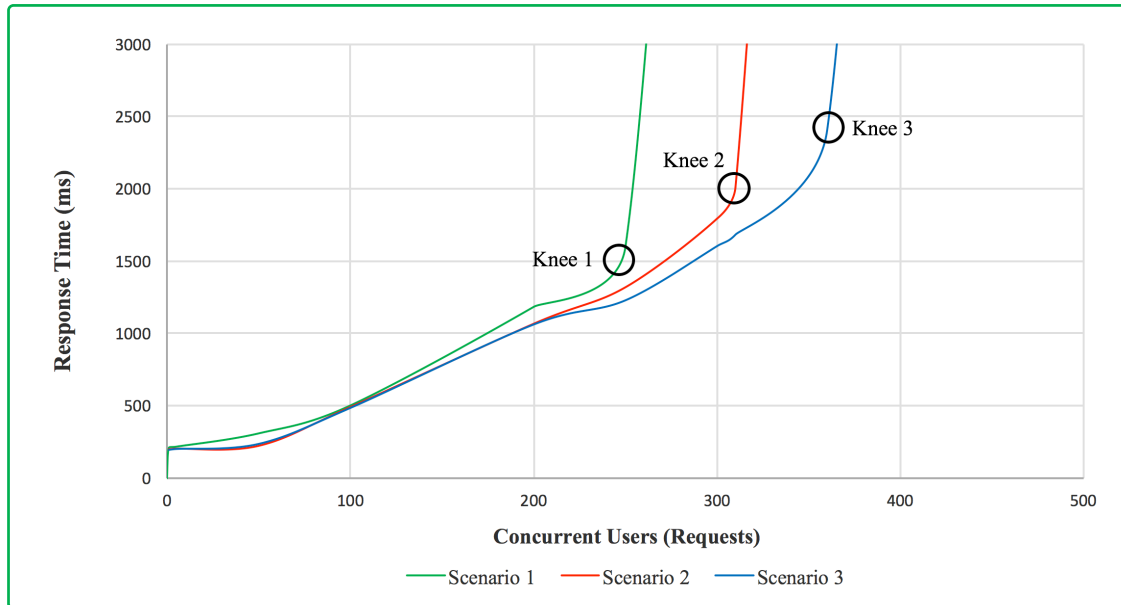


Figure 7-2.: Performance curves with knee for each scenario (requests vs. response time).

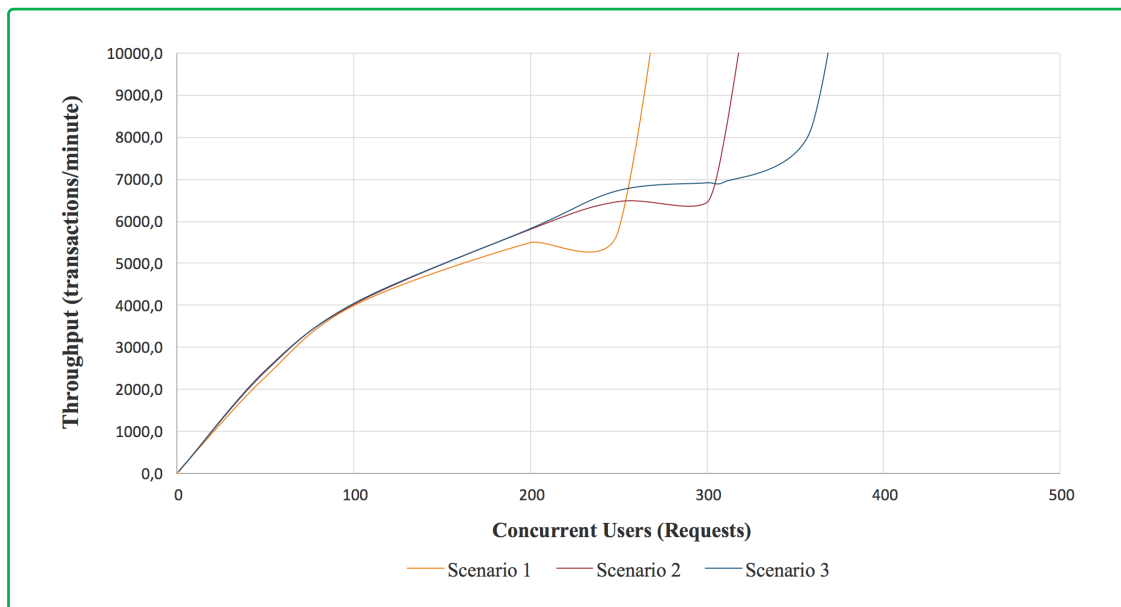


Figure 7-3.: Performance curves for each scenario (requests vs. throughput).

8. Conclusions and Future Work

8.1. Conclusions

Based on the related work with software architecture and software automation, new approaches in the software architecture modeling have been proposed. In the first place, Sarch offers a set of complete options to model software architectures, which helps to have a greater consistency between the design, implementation and deployment of distributed software systems. Sarch can be seen as a first approximation to a new ADL, considering that it covers a wider scope than traditional ADLs that are primarily focused on architectural documentation and a few to the automatic code generation. On the other hand, MDDep is proposed as a new approach which can be considered as a variant of MDD. However, we emphasize that the approach is based entirely on an architectural design based on the conception of quality attributes of the software and how they can be solved from its modeling, e.g., with the approach proposed in this work to take architectural decisions focused on the performance and scalability.

8.2. Future Work

As a future work, we will define a greater scope for Sarch, in which there is a greater collaboration of architectural styles and their views, in order to allow the modeling of other types of software systems, of high and low complexity. In the same way, strengthen the theoretical bases of MDDep to introduce new ways of dealing with other software quality attributes such as high availability and resilience. Finally, to integrate a set of improvements for Sarch-Studio that allow to design the architectures based on Sarch through graphical representations, as well as to add more validations in order to guarantee conditions of good software design practices from its architecture.

B. Appendix: Views of the Case Study

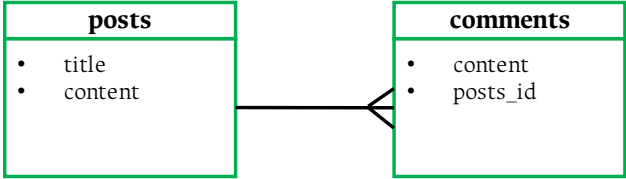


Figure B-1.: Graphic representation of the Data Model view.

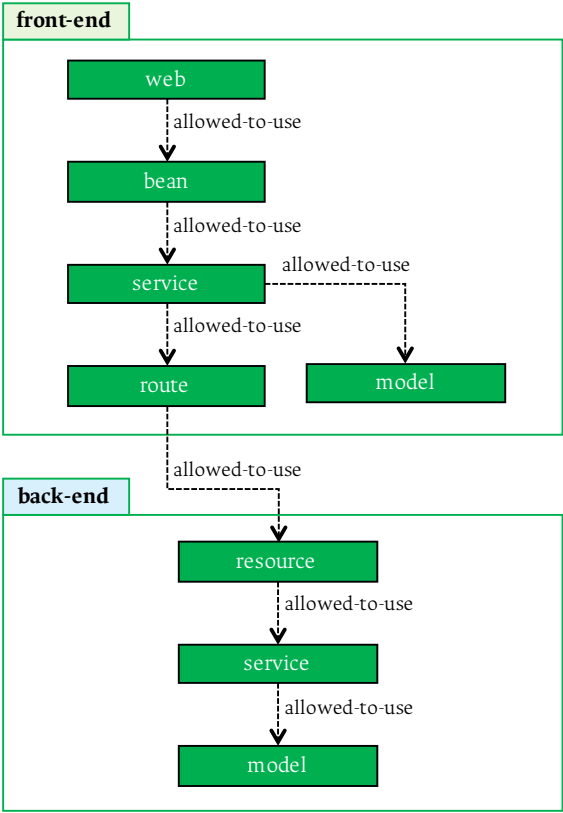


Figure B-2.: Graphic representation of the Layered view.



Figure B-3.: Graphic representation of the C&C view.

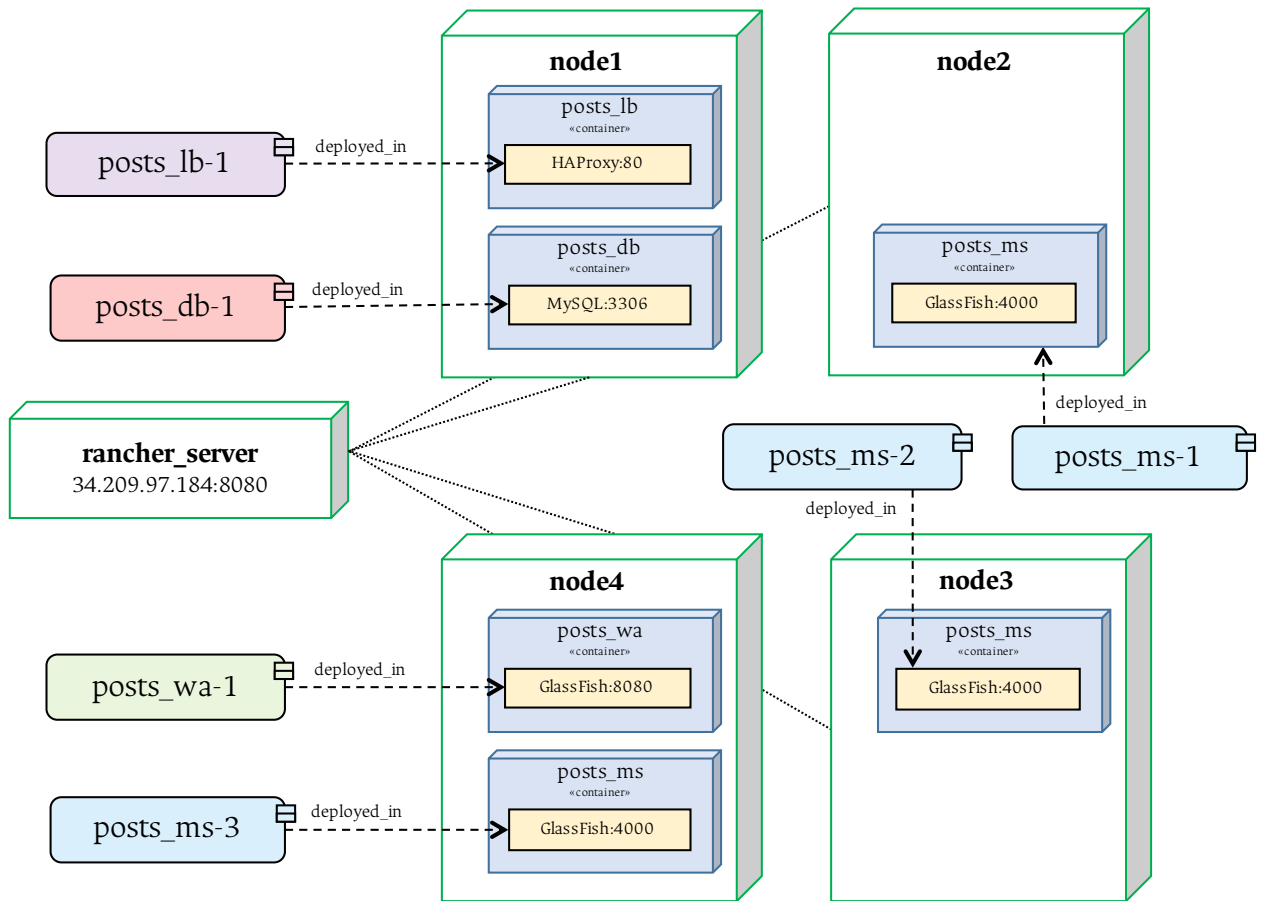


Figure B-4.: Graphic representation of the Deployment view.

C. Appendix: Case Study designed in Sarch Language

```
architecture PostsApplication {  
  
  data-model-view ::  
    data-model posts {  
      elements {  
        data-entity posts {  
          attributes {  
            String title;  
            String content;  
          }  
          operations {  
            getAll getAllPosts;  
            getById getPostById;  
            create createPost;  
            update updatePost;  
            delete deletePost;  
          }  
        }  
        data-entity comments {  
          attributes {  
            String content;  
          }  
          operations {  
            getAll getAllComments;  
            getById getCommentById;  
            create createComment;  
            update updateComment;  
            delete deleteComment;  
          }  
        }  
      }  
      relations {  
        one-to-many (posts, comments);  
      }  
    }  
  ::  
  
  layered-view ::  
    elements {  
      layer (back-end) model;  
      layer (back-end) service;  
      layer (back-end) resource;  
      layer (front-end) model;  
      layer (front-end) route;  
      layer (front-end) service;  
      layer (front-end) bean;  
      layer (front-end) web;  
    }  
    relations {  
      //back-end  
      resource allowed-to-use service;  
      service allowed-to-use model;  
      //front-end  
      web allowed-to-use bean;  
      bean allowed-to-use service;  
      service allowed-to-use route;  
      service allowed-to-use model;  
    }  
  ::  
}
```

Figure C-1.: Software architecture wrote in Sarch (i).

```

component-and-connector-view ::
  elements {
    component database posts_db (
      data-model: posts;
      root-password: "123";
      user: "sarch";
      password: "123"
    );
    component back-end posts_ms;
    component front-end posts_wa;
    component load-balancer posts_lb;
    connector jdbc JDBC;
    connector rest REST;
    connector http HTTP;
  }
  relations {
    attachment (JDBC: posts_ms, posts_db);
    attachment (REST: posts_wa, posts_ms);
    attachment (HTTP: posts_lb, posts_wa);
  }
::

deployment-view ::
  software-elements {
    posts_db (component posts_db);
    posts_ms (component posts_ms);
    posts_wa (component posts_wa);
    posts_lb (component posts_lb);
  }
  environmental-elements {
    server-node rancher_server {
      ip: "34.209.97.184";
      port: 8080;
      access-key: "E34ECCSF7B43F1452D4A";
      secret-key: "b66kSnmEalJkDgQyrWopUJb56PURNGDb6oZ8qcJd";
    }
    container posts_db {
      associated-stack: sarch_posts_db;
      execution-environment: mysql;
      host-label: host_db;
      port: 3306;
      scale: 1;
    }
    container posts_ms {
      associated-stack: sarch_posts_ms;
      execution-environment: glassfish;
      host-label: host_ms;
      port: 4000;
      scale: 3;
    }
    container posts_wa {
      associated-stack: sarch_posts_wa;
      execution-environment: glassfish;
      host-label: host_wa;
      port: 8080;
      scale: 1;
    }
    container posts_lb {
      associated-stack: sarch_posts_lb;
      execution-environment: haproxy;
      host-label: host_lb;
      port: 80;
      scale: 1;
    }
  }
  relations {
    posts_db deployed-in posts_db;
    posts_ms deployed-in posts_ms;
    posts_wa deployed-in posts_wa;
    posts_lb deployed-in posts_lb;
  }
  performance-tests (target-node-ip: '52.40.164.24');
::
}

```

Figure C-2.: Software architecture wrote in Sarch (ii).

D. Appendix: Implementation and Generation of Case Study

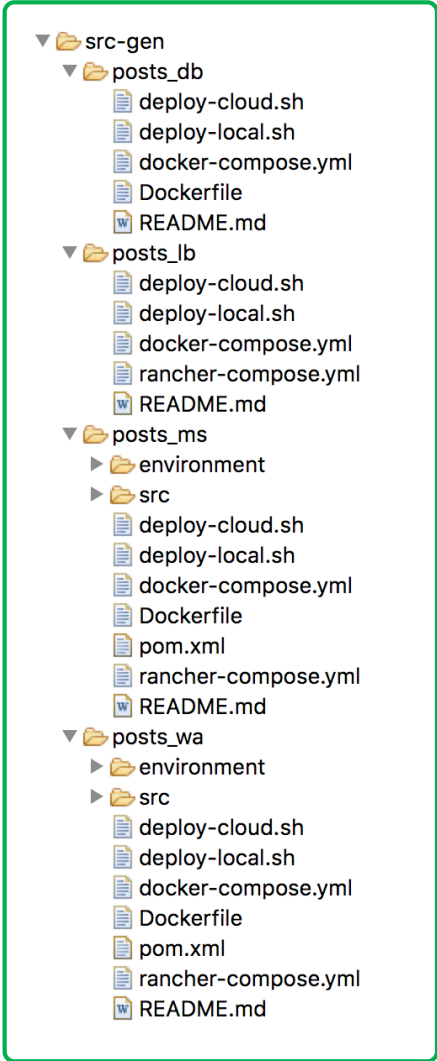


Figure D-1.: Code structure of the project.

```
1 posts-db:
2   build: .
3   ports:
4     - "3306:3306"
5   labels:
6     io.rancher.scheduler.affinity:host_label: host_db=host_db

1 posts-ms:
2   build: .
3   ports:
4     - "4000:4000"
5   external_links:
6     - msc-posts-db/posts-db
7   labels:
8     io.rancher.scheduler.affinity:host_label: host_ms=host_ms

1 posts-wa:
2   build: .
3   ports:
4     - "8080:8080"
5   external_links:
6     - msc-posts-ms/posts-ms
7   labels:
8     io.rancher.scheduler.affinity:host_label: host_wa=host_wa

1 posts-lb:
2   image: rancher/lb-service-haproxy:v0.4.2
3   ports:
4     - "80:80"
5   labels:
6     io.rancher.container.agent.role: environmentAdmin
7     io.rancher.container.create_agent: 'true'
8     io.rancher.scheduler.affinity:host_label: host_lb=host_lb
```

Figure D-2.: Docker structure for database, microservice, web application and load balancer.

```
1 package msc.posts.ms.model;
2
3 import javax.persistence.*;
4
5 /**
6  * Created by javergarav.
7  */
8
9 @Entity
10 @Table(name = "posts")
11 @NamedQueries({@NamedQuery(name = Post.FIND_ALL, query = "SELECT u FROM Post u")})
12 public class Post {
13
14     public static final String FIND_ALL = "Post.findAll";
15
16     @Id
17     @GeneratedValue(strategy = GenerationType.IDENTITY)
18     private long id;
19
20     private String title;
21     private String content;
22
23     public long getId() { return id; }
24
25     public void setId(long id) { this.id = id; }
26
27     public String getTitle() { return title; }
28
29     public void setTitle(String title) { this.title = title; }
30
31     public String getContent() { return content; }
32
33     public void setContent(String content) { this.content = content; }
34 }
35
36
37 package msc.posts.ms.resource;
38
39 import ...
40
41 /**
42  * Created by javergarav.
43  */
44
45 @Path("/posts")
46 public class PostResource {
47
48     @Context
49     UriInfo uriInfo;
50
51     @EJB
52     PostService postService;
53
54     @GET
55     public List<Post> getAllPosts(@QueryParam("firstResult") int firstResult, @QueryParam("maxResults") int maxResults) {
56         System.out.println("Getting ALL Posts...");
57         return postService.getAllPosts(firstResult, maxResults);
58     }
59
60     @GET
61     @Path("/{id}")
62     public Post getPostById(@PathParam("id") long id) {
63         System.out.println("Getting Post By Id...");
64         return postService.getPostById(id);
65     }
66
67     @POST
68     public Response createPost(Post post) {
69         postService.createPost(post);
70         System.out.println("Creating Post...");
71         return Response.status(Response.Status.CREATED).build();
72     }
73 }
```

Figure D-3.: Model and Resource classes of microservice.

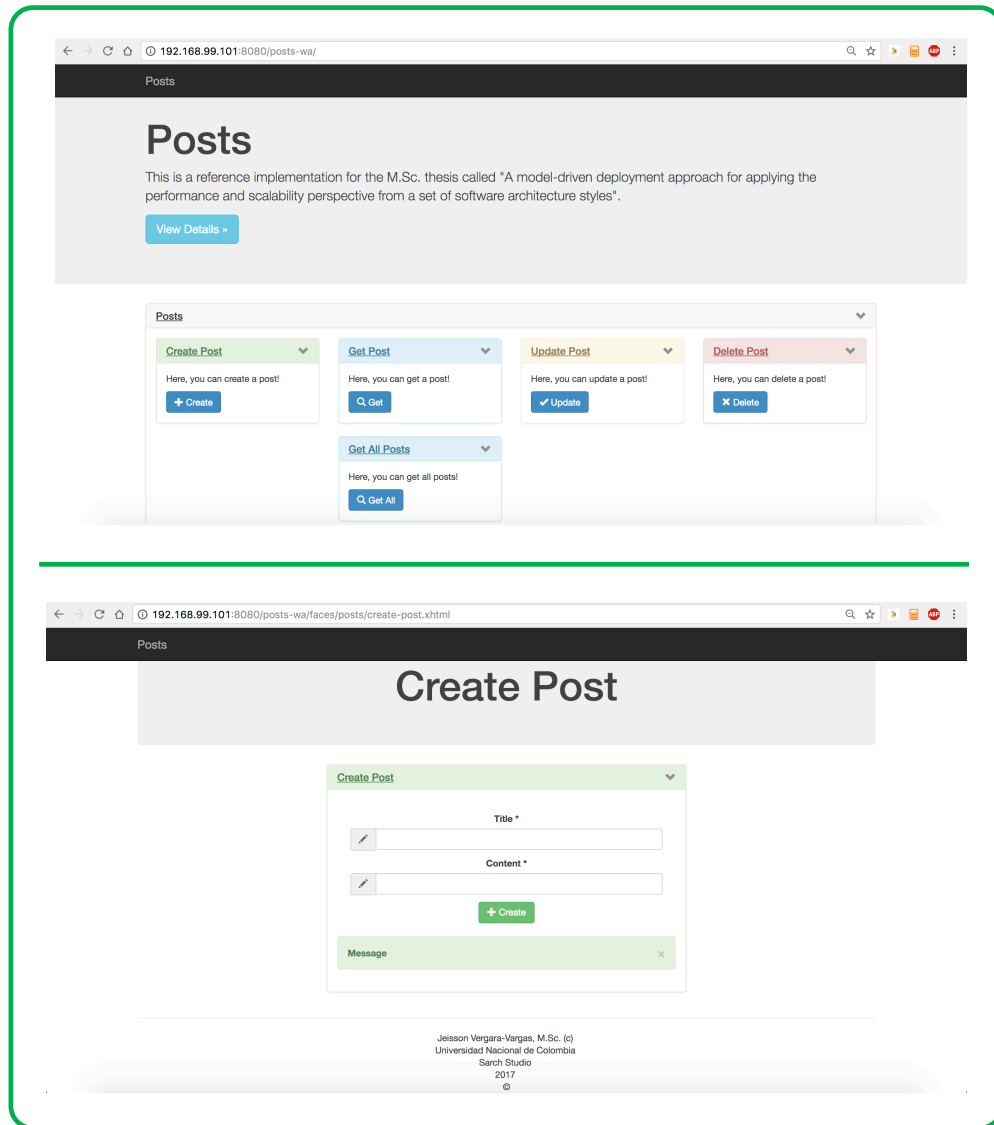


Figure D-4.: Web application visualization (Index and Create Post).

Bibliography

- [1] Amazon Web Services (AWS). <https://aws.amazon.com/>
- [2] Apache JMeter. <http://jmeter.apache.org/>
- [3] Apache Maven. <https://maven.apache.org/>
- [4] Docker. <https://www.docker.com/>
- [5] GlassFish. <https://javaee.github.io/glassfish/>
- [6] HAProxy. <http://www.haproxy.org/>
- [7] Java. <https://www.java.com/>
- [8] MySQL. <https://www.mysql.com/>
- [9] Rancher. <http://rancher.com/>
- [10] WebDSL. <http://webdsl.org/>
- [11] Xtend. <https://www.eclipse.org/xtend/>
- [12] Xtext. <https://eclipse.org/Xtext/>
- [13] ALDRICH, Jonathan ; CHAMBERS, Craig ; NOTKIN, David: ArchJava: Connecting Software Architecture to Implementation. In: *Proceedings of the 24th international conference on Software engineering - ICSE '02* (2002), 187. <http://dx.doi.org/10.1145/581339.581365>. – DOI 10.1145/581339.581365. – ISBN 158113472X
- [14] BASS, Len ; CLEMENTS, Paul ; KAZMAN, Rick: *Software Architecture in Practice*. 3rd. 2013
- [15] BRAMBILLA, Marco ; CABOT, Jordi ; WIMMER, Manuel: *Model-Driven Software Engineering in Practice*. Bd. 1. 1st. 2012. – 1–182 S. <http://dx.doi.org/10.2200/S00441ED1V01Y201208SWE001>. <http://dx.doi.org/10.2200/S00441ED1V01Y201208SWE001>. – ISBN 9781608458820
- [16] CALA, Jacek ; WATSON, Paul: Automatic Software Deployment in the Azure Cloud. In: *Computing* (2010), Nr. June, S. 155–168

-
- [17] CAVALCANTE, Everton ; OQUENDO, Flavio ; BATISTA, Thais: Architecture-Based Code Generation: From π -ADL Architecture Descriptions to Implementations in the Go language. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8627 LNCS (2014), S. 130–145. http://dx.doi.org/10.1007/978-3-319-09970-5_13. - DOI 10.1007/978-3-319-09970-5_13. - ISBN 9783319099699
- [18] CLEMENTS, Paul: A Survey of Architecture Description Languages. In: *Eighth International Workshop on Software Specification and Design* (1996), Nr. March
- [19] CLEMENTS, Paul ; BACHMANN, Felix ; BASS, Len ; GARLAN, David ; IVERS, James ; LITTLE, Reed ; MERSON, Paulo ; NORD, Robert ; STAFFORD, Judith: *Documenting Software Architectures: Views and Beyond*. 2011. - 582 S. - ISBN 0132366258
- [20] DAMYANOV, Ivo ; SUKALINSKA, Mila: Domain Specific Languages in Practice. In: *International Journal of Computer Applications* 115 (2015), Nr. 2, 42–45. <http://dx.doi.org/10.5120/20126-2205>. - DOI 10.5120/20126-2205. - ISSN 09758887
- [21] DEMIRLI, Elif: *Model-Driven Engineering of Software Architecture Viewpoints*, Diss., 2012
- [22] DEMIRLI, Elif ; TEKINERDOGAN, Bedir: SAVE: Software Architecture Environment for Modeling Views. In: *2011 Ninth Working IEEE/IFIP Conference on Software Architecture* (2011), S. 355–358. <http://dx.doi.org/10.1109/WICSA.2011.57>. - DOI 10.1109/WICSA.2011.57. ISBN 978-0-7695-4351-2
- [23] DEMIRLI, Elif ; TEKINERDOGAN, Bedir: Software Language Engineering of Architectural Viewpoints. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6903 LNCS (2011), S. 336–343. http://dx.doi.org/10.1007/978-3-642-23798-0_36. - DOI 10.1007/978-3-642-23798-0_36. - ISBN 9783642237973
- [24] DERBEL, Imen ; JILANI, Lamia L. ; MILI, Ali: ACME+: An ADL for Quantitative Analysis of Quality Attributes. In: *ENASE 2013 Communications in Computer and Information Science* 417 CCIS (2013), 16. <http://www.scopus.com/inward/record.url?eid=2-s2.0-84904749759&partnerID=40&md5=b61c7f962ff2a22c9f9bc183ba7e4028>
- [25] DEURSEN, Arie V. ; KLINT, Paul ; VISSER, Joost: Domain-Specific Languages: An Annotated Bibliography. In: *ACM Sigplan Notices* 35 (2000), Nr. 6, 26–36. <http://dx.doi.org/10.1145/352029.352035>. - DOI 10.1145/352029.352035. - ISBN 0362-1340
- [26] DI COSMO, Roberto ; MAURO, Jacopo ; ZACCHIROLI, Stefano: Automatic Deployment of Services in the Cloud with Aeolus Blender. (2015), S. 397–411. <http://dx.doi.org/10.1007/978-3-662-48616-0>. - DOI 10.1007/978-3-662-48616-0. - ISBN 9783662486153
- [27] ERL, Thomas: *Cloud Computing - Concepts, Technology & Architecture*. 2013

- [28] FEILER, Peter H. ; LEWIS, Bruce A. ; VESTAL, Steve: The SAE Architecture Analysis & Design Language (AADL) - A Standard for Engineering Performance Critical Systems. In: *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE (2006)*, S. 1206–1211. <http://dx.doi.org/10.1109/CACSD.2006.285483>. – DOI 10.1109/CACSD.2006.285483. ISBN 0780397975
- [29] FOWLER, Martin ; PARSONS, Rebecca: *Domain-Specific Languages*. 2011
- [30] FOX, Christopher: *Introduction to Software Engineering Design: Processes, Principles, and Patterns with UML2*. 2006. – 748 S. – ISBN 0321410130
- [31] GARLAN, David ; MONROE, Robert T. ; WILE, David: Acme: Architectural Description of Component-Based Systems. In: *Foundations of Component-Based Systems (2000)*, S. 47–68. ISBN 0521771641
- [32] GASSARA, Amal ; RODRIGUEZ, Ismael B. ; JMAIEL, Mohamed: A multi-scale modeling approach for software architecture deployment. In: *Proceedings of the ACM Symposium on Applied Computing 13-17-April (2015)*, Nr. April, 1405–1410. <http://dx.doi.org/10.1145/2695664.2695721>. – DOI 10.1145/2695664.2695721. ISBN 9781450331968
- [33] GILSON, Fabian ; ENGLEBERT, Vincent: A Domain Specific Language for Stepwise Design of Software Architectures. In: *MODELSWARD 2014 - Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development (2014)*, S. 67–78. ISBN 9789897580079
- [34] GONZALEZ-HERRERA, Inti ; BOURCIER, Johann ; DAUBERT, Erwan ; RUDAMETKIN, Walter ; BARAIS, Olivier ; FOUQUET, François ; JÉZÉQUEL, Jean M.: Scapegoat: an adaptive monitoring framework for component-based systems. In: *Proceedings - Working IEEE/IFIP Conference on Software Architecture 2014, WICSA 2014, 2014*. – ISBN 9781479934126, S. 67–76
- [35] HOENISCH, Philipp ; WEBER, Ingo ; SCHULTE, Stefan ; ZHU, Liming ; FEKETE, Alan: Four-Fold Auto-Scaling on a Contemporary Deployment Platform Using Docker Containers. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 9435 (2015)*, S. 316–323. <http://dx.doi.org/10.1007/978-3-662-48616-0>. – DOI 10.1007/978-3-662-48616-0. – ISBN 9783662486153
- [36] IEEE: *ISO/IEC/IEEE: Systems and Software Engineering - Architecture Description*. 2011
- [37] JIA, Xiangyang ; YING, Shi ; ZHANG, Tao ; CAO, Honghua ; XIE, Dan: A New Architecture Description Language for Service-Oriented Architecture. In: *Sixth International Conference on Grid and Cooperative Computing (GCC 2007) (2007)*, Nr. Gcc, 96–103. <http://dx.doi.org/10.1109/GCC.2007.18>. – DOI 10.1109/GCC.2007.18. ISBN 0-7695-2871-6

-
- [38] KHALGUI, Mohamed ; REBEUF, Xavier: A heuristic based method for automatic deployment of distributed component based applications. In: *Industrial Embedded Systems - IES'2006* (2006). <http://dx.doi.org/10.1109/IES.2006.357471>. - DOI 10.1109/IES.2006.357471. ISBN 142440777X
- [39] KRUCHTEN, Philippe: Architectural Blueprints - The "4+1" View Model of Software Architecture. In: *IEEE Software* 12 (1995), Nr. November, 42-50. <http://dx.doi.org/10.1145/216591.216611>. - DOI 10.1145/216591.216611. - ISBN 0897917057
- [40] LAN, Ling ; HUANG, Gang ; MA, Liya ; WANG, Meng ; MEI, Hong ; ZHANG, Long ; CHEN, Ying: Architecture Based Deployment of Large-Scale Component Based Systems: The Tool and Principles. (2005), 123-138. http://dx.doi.org/10.1007/11424529_9. - DOI 10.1007/11424529_9. - ISSN 03029743
- [41] LASCU, Tudor A. ; MAURO, Jacopo ; ZAVATTARO, Gianluigi: Automatic Component Deployment in the Presence of Circular Dependencies Tudor. (2013), S. 254-272. <http://dx.doi.org/10.1016/j.scico.2015.11.001>. - DOI 10.1016/j.scico.2015.11.001. - ISBN 978-3-319-57665-7
- [42] LASCU, Tudor A. ; MAURO, Jacopo ; ZAVATTARO, Gianluigi: Automatic deployment of component-based applications. In: *Science of Computer Programming* 113 (2015), 261-284. <http://dx.doi.org/10.1016/j.scico.2015.07.006>. - DOI 10.1016/j.scico.2015.07.006. - ISSN 01676423
- [43] LEE, Jaemyoun ; JEONG, Haegwon ; LEE, Won J. ; SUH, Hyo J. ; LEE, Dongeun ; KANG, Kyungtae: Advanced Primary-Backup Platform with Container-Based Automatic Deployment for Fault-Tolerant Systems. In: *Wireless Personal Communications* (2017), S. 1-18. <http://dx.doi.org/10.1007/s11277-017-4282-4>. - DOI 10.1007/s11277-017-4282-4. - ISSN 1572834X
- [44] MEDVIDOVIC, Nenad ; TAYLOR, Richard N.: A Classification and Comparison Framework for Software Architecture Description Languages. In: *Software Engineering, IEEE Transactions on* 26 (2000), Nr. 1, 70-93. <http://dx.doi.org/10.1109/32.825767>. - DOI 10.1109/32.825767. - ISBN 0098-5589
- [45] NEWMAN, Sam: *Building Microservices: Designing Fine-Grained Systems*. 2015. - ISBN 1491950331
- [46] NICOLAS, Alejandro ; POSADAS, Hector ; PENIL, Pablo ; VILLAR, Eugenio: Automatic deployment of component-based embedded systems from UML/MARTE models using MCAPI. In: *Proceedings of the 2014 29th Conference on Design of Circuits and Integrated Systems, DCIS 2014* (2014). <http://dx.doi.org/10.1109/DCIS.2014.7035575>. - DOI 10.1109/DCIS.2014.7035575. ISBN 9781479957439

- [47] 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, 1-14. <http://dx.doi.org/10.1145/986710.986728>. - DOI 10.1145/986710.986728. - ISBN 0163-5948
- [48] QIN, Wei ; MALIK, Sharad: A Study of Architecture Description Languages from a Model-Based Perspective. In: *Proceedings - International Workshop on Microprocessor Test and Verification* (2006), S. 3-11. <http://dx.doi.org/10.1109/MTV.2005.2>. - DOI 10.1109/MTV.2005.2. - ISBN 0769526276
- [49] ROZANSKI, Nick ; WOODS, Eoin: *Software Systems Architecture*. 2nd. 2011
- [50] STAHL, Thomas ; VÖLTER, Markus: *Model-Driven Software Development - Technology, Engineering, Management*. 2006. - ISBN 9780470025703
- [51] STEINBERG, Dave ; BUDINSKY, Frank ; PATERNOSTRO, Marcelo ; MERKS, Ed: *EMF - Eclipse Modeling Framework*. 2nd. 2009. - ISBN 9780321331885
- [52] STEPHENS, Rod: *Beginning Software Engineering*. 2015. - ISBN 9781118969144
- [53] TAYLOR, Richard N. ; MEDVIDOVIC, Nenad ; DASHOFY, Eric M.: *Software Architecture - Foundations, Theory, and Practice*. 2010
- [54] WEBBER, Jim ; PARASTATIDIS, Savas ; ROBINSON, Ian: *REST in Practice*. 2010. - ISBN 9780596805821
- [55] YAZDANSHENAS, Amir R. ; KHOSRAVI, Ramtin: Using domain-specific languages to describe the development viewpoint of software architectures. In: *2009 14th International CSI Computer Conference, CSICC 2009* (2009), S. 146-151. <http://dx.doi.org/10.1109/CSICC.2009.5349322>. - DOI 10.1109/CSICC.2009.5349322. ISBN 9781424442621
- [56] ZDUN, Uwe: A DSL toolkit for deferring architectural decisions in DSL-based software design. In: *Information and Software Technology* 52 (2010), Nr. 7, 733-748. <http://dx.doi.org/10.1016/j.infsof.2010.03.004>. - DOI 10.1016/j.infsof.2010.03.004. - ISBN 0950-5849
- [57] ZHANG, Shifeng ; GODDARD, Steve: xSADL: An architecture description language to specify component-based systems. In: *International Conference on Information Technology: Coding and Computing, ITCC 2* (2005), 443-448. <http://dx.doi.org/10.1109/ITCC.2005.303>. - DOI 10.1109/ITCC.2005.303. ISBN 0-7695-2315-3