GRAN SASSO SCIENCE INSTITUTE
SCHOOL OF ADVANCED STUDIES
Scuola Universitaria Superiore

Doctoral Thesis Research Proposal

Recommender Systems for Reference Architectures

PhD Program in Computer Science: 37 cycle

Author:

Jose Alejandro Concepcion Alvarez
jose.alvarez@gssi.it

Supervisor:

Prof. Ludovico Iovino
ludovico.iovino@gssi.it

Co-supervisor:

Dr. Leonardo Mariani
leonardo.mariani@unimib.it

September 2022

GSSI Gran Sasso Science Institute
Viale Francesco Crispi, 7 - 67100 L'Aquila - Italy

Abstract

A Reference Architecture (**RA**) is a generic architecture that provides guidelines and options for making decisions in developing more specific architectures and implementing solutions. RAs inspire architects when designing concrete architectures, and they help guarantee compliance with architectural decisions, regulatory requirements, and architectural qualities. It is used as a foundation for the design of concrete architectures within a given context or application domain, e.g., automotive, banking, the Internet of things, avionics, and robotics. Due to the lack of common interpretation of the concept of RA, inefficient support for instantiation and refinements, multiple challenges are still unresolved, for instance, the lack of automated compliance checking mechanisms. By automating the compliance checking, a reduction of the effort and the margin of errors may be drastically reduced compared to a task still performed manually.

Model Driven Engineering (**MDE**) is a branch of software engineering focusing on modeling, instead of traditional code-centric techniques, to automate parts of the software production lifecycle, e.g., via code generation. Model Driven Architecture (**MDA**) allows producing architectural models at varying levels of abstraction, from a conceptual view to the smallest implementation detail. Using MDE techniques, tasks such as automated compliance checking can be supported. Assistive modeling tools are software modules usually integrated with ML techniques to help the modeler compose models smartly. In MDE, multiple tasks might be subject to recommendations, and numerous tools have been presented as modeling assistants. These tools are generally inspired by Recommender Systems (**RS**) used in software development. For instance, RS may assist in choosing suitable third-party programming libraries or recommend APIs, suggest refactoring of source code, or complete portions of a model or metamodel like a UML class-based diagram. As far as we know, there is still a lack of RS applications to help with architecture and RA modeling tasks.

We aim to build a project that delivers an innovative way of modeling RAs. We will develop an assistive tool that automates the architectural design process. With the use of real-time suggestions, we aim to support the process of compliance checking, instantiation, coverage, refinements, and refactoring between a RA and its respective concrete architecture.

Contents

List of Figures

Chapter 1

Introduction

Software architecture (SA) is the foundation of a software system. Like other types of engineering, the foundation has a profound effect on the quality of what is built on top of it. As such, it holds a great deal of importance in terms of the successful development and eventual maintenance of the system [2]. The greater the size and complexity of a software system, the more we will need a well-thought-out architecture to succeed. A reference architecture (RA), in essence, can be defined as a general architecture used as a foundation for the design of concrete architectures within a given context or application domain, e.g., automotive, banking, the internet of things, avionics, and robotics [1]. The main difference between them is that a reference architecture is an abstraction, a template that provides the guidelines and solutions for a problem. In contrast, a SA is a fully-fleshed-out implementation or representation of that RA. RAs inspire architects when designing concrete architectures, and they help guarantee compliance with architectural decisions, regulatory requirements, and architectural qualities. Although there are a few attempts to automate the compliance checking of an architectural model w.r.t its RA [1, 3, 4]. There is an evident lack of support in the instantiation, refinement, and refactoring of the architectural models. Some interesting actions would improve the process. For example, when the architect needs to define his software architecture, it would be nice to suggest the minimal architecture structure to comply with the chosen RA. Alternatively, when the architect has already specified a partial architecture, and while selecting a RA, the result is not compliant, suggesting the modeling actions to reach minimal compliance would be effective. Furthermore, when the software architects work with internal guidelines and regulations, they follow implicit RAs specifications. Given multiple declared architectures, an useful feature could be to extract a common RA based on the declared architectural models.

To better support the architectural modeling, we aim to develop a modeling assistant to support the software architectural modeling task. We will plug the developed modeling assistant into a MDE framework presented in [1] that automatically checks compliance and validates a software architectural model against its correspondent reference architecture. The main extension consists of adding continuous modeling assistance to the architectural phase, which is currently uncovered and managed at the beginning and at the end of the process. In this thesis, we will address the problem of developing assistive tools for software architectural modeling that will consider the RA as a blueprint with the validation, compliance checking, and automation that will support the instantiation process. In this context, the first open issue is the lack of support for the instantiation of a RA, which is expensive in terms of effort, time-consuming and error-prone. There are other challenges we have to face, such as the diversity of architectural descriptions language and the lack of datasets that relate a set of SA with its representative RA. The tool's core will be based on the benefit of a recommender system (RS). RSs are information filtering systems that aim to predict users' preferences for a given set of items or entities to offer suggestions.

To validate our approach, we plan to prepare a test suite to check the effectiveness and quality of the recommendation. We will perform an online validation where we can choose different application domains such as (automotive, IoT, microservices, monolithic services, and web browsers) with different stakeholders and verify the accuracy and correctness of the suggestions based on the positive or negative feedback we get from people who know well the domain of the applications under analysis.

The rest of this proposal is organized as follows. Chapter 2 gives the background about RA, MDE, and RS. In chapter 3, we describe our proposal's state-of-the-art and research challenges. Eventually, in chapter 4, we define the research questions and present our solution.

Chapter 2

Background

In this chapter, we introduce in section 2.1 The Reference Architecture main ideas and concepts. In section 2.2 we describe the main concepts of model driven engineering. Finally in section 2.3 we make a study on recommender systems and some use cases in MDE.

2.1 Reference Architecture

Software architecture refers to the fundamental structures of a software system and the discipline of creating those structures and systems. Each structure comprises software elements, relationships between them, and properties of the elements and the relationships. The architecture of a software system is an abstraction representation analogous to the architecture of a building. It functions as a blueprint of the system and the project under development, setting out the necessary tasks to be performed by the design teams. Software architectures play a significant role in determining system quality since they form the backbone of any successful software-intensive system. In this context, **Reference Architecture** is a generic architecture for a class of software systems used as a foundation for the design of concrete architectures from this class [5].

Software reference architectures have emerged as abstractions of concrete architectures from a certain domain [6]. Reference architectures (RA) can be used as an inspiration in the design of concrete architectures or as a standardization tool that guarantees the interoperability between systems and between components of systems [7]. A reference architecture is used for the design of concrete architectures in multiple contexts, affecting different stakeholders in each context [8]. Nowadays, the increasing complexity of software, the need for efficient and effective software design processes and for high levels

of system interoperability lead to an increasing role of reference architectures in the software design process. According to [9] a reference model is *a division of functionality together with data flow between the pieces*, and a reference architecture is *a reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flows between them.* A reference architecture is based on the functionalities and data flows defined in a reference model and applies architectural styles and patterns that help in addressing the main qualities expected from the architecture (see Fig. 2.1). A "good" reference architecture can bring a number of benefits [10]. It may facilitate the design of high-quality concrete architectures.

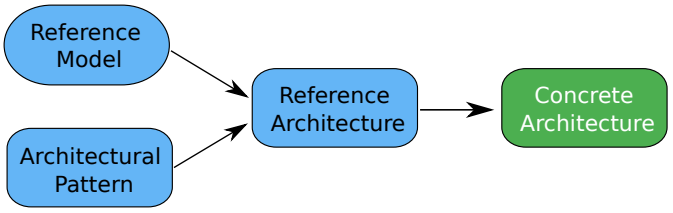

FIGURE 2.1: The relationship between reference models, reference architectures and concrete architectures

Because it is a structural description, usually RAs do not identify the internal mechanisms that drive the information processing among the components, like behavioural diagrams for instance.

Important industrial actors also started to catalog RAs for the benefit of the architects and developers. For instance, Amazon reports the suggested RAs [11] and so does Microsoft for Azure [12], and also IBM [13], in order to assure compliance with their technologies and standard when used as templates and guidelines. Fig. 2.2**a** shows an RA provided by **AWS** for validating in-app purchases and managing refunds with a serverless backend. In Fig. 2.2**b** you can see a RA provide by **Azure Microsoft** that contains and API Gateway as main actor; the API gateway acts as a reverse proxy routing requests from clients to services. Those RA catalogs provided by these leading technology companies are crucial. They help developers and companies address their problems with the standard software architecture solution model. That saves engineering time and reduces the need to reinvent the wheel. In addition, it standardizes the development process and ensures the scalability of the implemented SA.

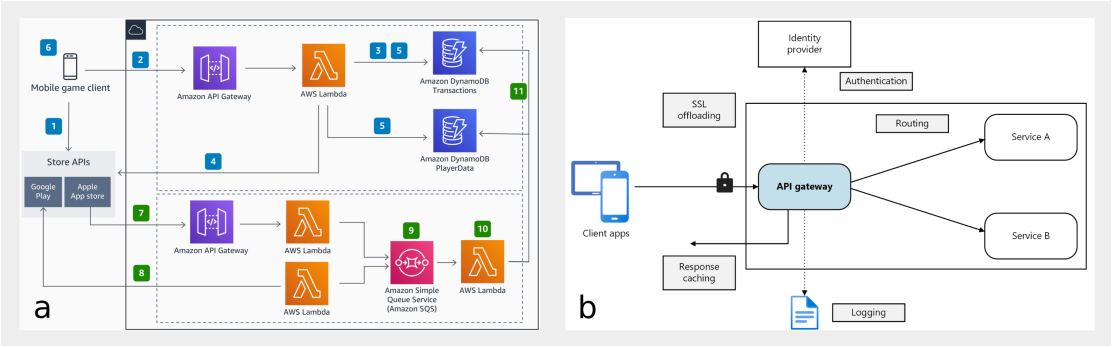

FIGURE 2.2: **a**: Mobile In App Purchase Validation on AWS, **b**: API Gateway application provided by Azure.

2.2 Model Driven Engineering

Model driven engineering **(MDE)** is a software development methodology that focuses on creating and exploiting domain models, which are conceptual models of all the topics related to a specific problem [14]. Here, models are blueprints to provide a complete and detailed system specification. These blueprint models can be further refined to create the system by using code-generation techniques to minimize the coding tasks. In software engineering, abstraction is usually implemented with a computing-oriented focus, making it difficult for people with no language development expertise to understand. MDE combines Domain-Specific Modelling Languages (DSMLs), transformation engines, and generators [15]. In particular, the use of DSMLs allows the involvement of domain experts in the development. To sum up, a relevant feature of MDE approaches is that they enable the description of very complex systems with simplified models. In this way, we can think of MDE as a reduction approach because it allows systems to be described as an aggregation of complementary models. The main advantage of MDE is that models are understandable by the machine and can be processed for multiple purposes. Model Driven Architecture (MDA) allows producing models at varying levels of abstraction, from a conceptual view to the smallest implementation detail. Models are used in architectural design at different stages, e.g., in the early design cycle as a sketch created to examine particular aspects of a design idea. At later stages, architectural models can be used for communication purposes or rapid prototyping [16]. Various studies investigated the use of MDE for RAs.

2.3 Recommender Systems

Recommender systems **(RSs)** are information filtering systems that aim to predict the preferences of users for a given set of items, with the purpose of offering a typically

prioritised list of potentially interesting items [17]. RSs are widely used by commercial applications such as music and video broadcasting platforms, e-commerce sites and social networks, and they are increasingly being used to help developers with software engineering activities [18]. For example, we can find RSs that help in choosing appropriate third-party programming libraries [19], recommend API method invocations [20], suggest code refactorings [21] and propose features for mobile apps [22] to name a few.

In general, RSs follow a process that encompasses three main steps:

1. Collecting relevant user information
2. Learning from the collected information to build user profiles.
3. Applying a heuristic function or a previously built model to select and rank the items that the user is most likely to prefer.

In order to present personalized item suggestions to a user, RSs build a user profile that captures past choices and preferences of the user. This information can be either explicitly provided by the user or implicitly inferred by the system. Explicit feedback refers to user preference statements about items they know, typically stored as numeric ratings or unary/binary values (e.g., likes and thumbs up/down). In contrast, implicit feedback is inferred by observing and mining user interactions within the system, such as previous search queries, product purchases, and mouse clicks. Other features that can be used to model users' preferences include demographic data, personality traits, emotional states, and trust relationships [23].

2.3.1 Types of recommender systems

RSs are commonly classified into the following major categories, depending on how they generate personalised recommendations:

- **Content Based Systems**: Systems recommend similar items to those items the target user liked in the past [17]. They use item attributes or features to represent both user and item profiles and establish the corresponding user/item similarities.
- **Collaborative Filtering Systems**: Systems make suggestions to the target user based on items preferred by like-minded people [17]. They rely on the feedback (commonly ratings) that users give about the items. Hence, user and item similarities are established via explicit or implicit rating-based similarities and patterns [24, 25].

- **Knowledge-Based Systems**: Systems recommend items using domain-specific knowledge about how item attributes and features could meet the user's needs and interests [26]. Many recommendation approaches can be categorised as KB. Among them, two types of approaches have gained great interest in the literature: case-based and constraint-based.

- **Context-Aware Systems**: Systems consider the current user's context (e.g. location, time, weather) to enrich personalised recommendations [27]. This is widely used in social networks.

- **Social-Based Systems**: Systems which analyse and exploit the social network connections of the target user to generate recommendations [28].

- **Hybrid systems**: Systems that make use of two or more recommendation methods, such as *Content based* and *Collaborative filtering*, to take advantage of their benefits and avoid some of their limitations [29]. Because of this, many real world RSs are hybrid.

Chapter 3

State of the Art

3.1 Reference Architectures

A good example of RA modeling and definition, appeared first in [30]. Where a RA for web browsers is reported. They derived it using the source code and available documentation for two mature web browser implementations, Mozilla and Konqueror. The authors defined the basic subsystems that characterize. Bellow you can see the list of those subsystems.

- The *User Interface* subsystem
- The *Browser Engine* subsystem
- The *Rendering Engine* subsystem
- The *Networking* subsystem
- The *JavaScript interpreter* subsystem
- The *XML parser* subsystem
- The *Display Backend* subsystem
- The *Data Persistence* subsystem

You can see the mapping of Mozilla's conceptual architecture onto the reference architecture in Fig. 3.1**a** as well for the Konqueror in Fig. 3.1**b**

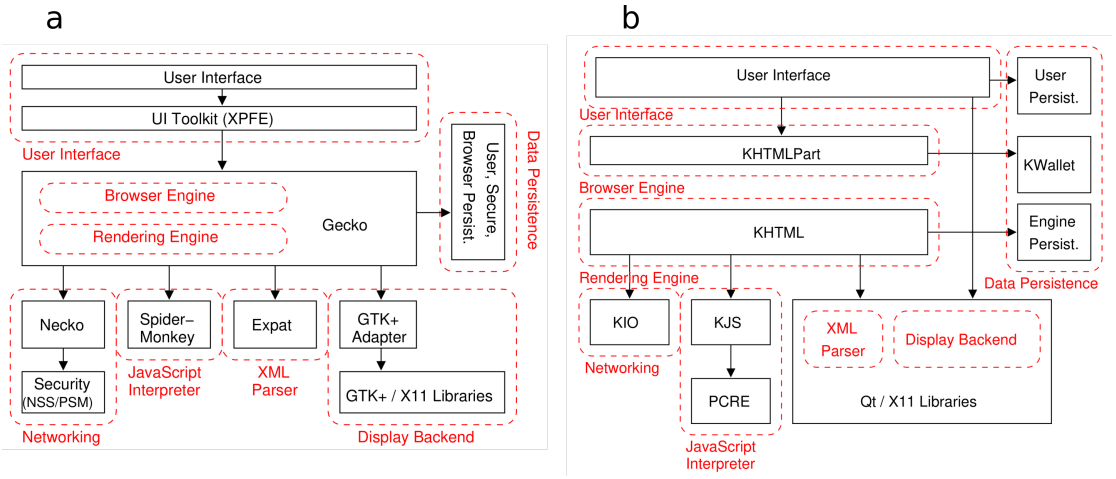

FIGURE 3.1: Mozilla(a) and Konqueror(b) concrete architecture and Web browser RA

The literature on reference architectures is scarce. Definitions and brief explanations of reference architectures are provided in [6]. The authors acknowledge the lack of a clear understanding of reference architectures and provide a multidimensional classification of reference architectures. Therefore, we have found several works related to the classification and understanding of RAs. Several frameworks have been developed to test the compliance between a particular domain-specific and generic reference architecture. Attempts are made to generalize reference architecture categories to organize and study them. Angelov *et al.* [6] present a three-dimensional framework for the classification of reference architectures. The three dimensions are based on the need for congruent goals, context, and the design of reference architectures. They have used the framework to define five types of reference architectures that have congruent values in the three dimensions. The match of a reference architecture with one of these types is a pre-condition for its practical usage in the design of concrete architectures.

Software architectures capture the most significant properties and design constraints of software systems [31]. The modifications to a system that violate its architectural principles can degrade system performance and shorten its useful lifetime. This phenomenon is known as **Architecture Erosion** [31], and it has negative effects on the maintainability of software systems and other quality attributes. In [3], the authors report an application of a rule-based architecture conformance checking approach in an industrial case study where they derive a RA for the German public administration software. The reference architecture and its implementation constraints are formalized as architecture rules enabling automatic conformance checking tool support. The results from the case study show that the approach can check reference architecture conformance in realistic settings and helps avoid software architecture erosion.

3.2 Use of MDE for modeling software architectures and RAs

Several studies investigated the use of MDE for modeling architectures and RA. There are works focused on creating general architectures for a specific application domain and using model-driven techniques like model to model and model to code transformations; the source code can be validated and generated automatically. Here in [32], the authors present a model-driven fast prototyping approach for web applications that exploit methods and technologies proper of the MDE domain implemented upon the Eclipse Modeling Project. The approach is based on the definition of a meta-model that enables designing a Web Application (WA) to adopt a Model-View-Controller (MVC) architectural design pattern with a Java Server as a target platform. This approach allows to effortlessly and quickly carry out a modeling-generation-validation process to validate and refine a WA design before actually implementing it. The process and the technologies used to develop this approach can be reused to develop a fast prototyping approach for a different design model and a different target technology platform.

Another example of use of MDE for RA can be seen in [1]. Where a model-driven approach for modeling concrete architectures and RAs, assuring the automated compliance checking concerning to the defined RA has been presented. This tool provide a precise definition of reference architectures and an automatic virtual conformance checking mechanism. The definition of reference architectures is entrusted to the proposed *Generic Reference Architecture Metamodel*. The conformance checking mechanism is entrusted to the *Model Promotion Transformation*, the *Architectural Style Validation Script*, *the RATagging Weaving metamodel*, and *the Generated RA Validation Script* [1]. To demonstrate the applicability of this approach and its constituent artifacts, the authors applied it to two scenarios. The application scenario involved defining a reference architecture for web browsers, the definition of concrete architecture for the Mozilla Firefox browser and their virtual conformance checking, and a cloud RA for connected and autonomous vehicles for attack surface analysis. See the Fig. 3.2.

We will use this work as a baseline for developing our solution. One of the issues the above work has is that, unfortunately, the architect must always know perfectly the RA specification provided by senior architects or technology affiliates to compose the concrete architecture and check the compliance. This task, even if it is a considerable advancement w.r.t. the manual check, can present multiple difficulties, such as selecting the most suitable RA, instantiating and composing the architecture, or even refining it without a deep knowledge of the RA, resolving violations in the right way without

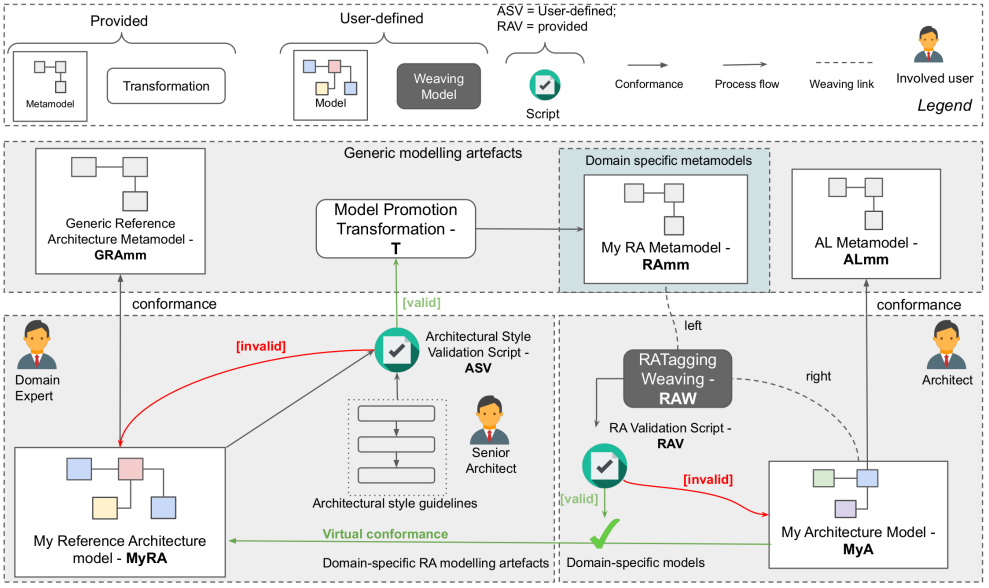

FIGURE 3.2: MORE: An approach that enables software architects to automatically check the compliance of their architecture description to previously modelled reference architectures [1]

disrupting other parts of the architecture. In order to reduce the burden of these essential tasks and further automate and support the architectural specification, a modeling assistant can propose modeling actions with the intent of composing not only a valid architectural model but a compliant one w.r.t. the chosen RA.

3.3 Recommender systems to support software modeling tasks

Recommender systems have been widely used to support software tasks. Following the trend in software engineering in recent years, there have been proposals for RSs to assist in modeling tasks. Most of them focus on models, especially model completion and repair. We have looked for those related to assisting modelers by providing suggestions for completing their models or suggesting generated code to perform some tasks. Based on this, we have found that most of them are oriented to modeling UML class-based diagrams using Machine Learning (ML) and Natural Language Processing (NLP) techniques. In addition, others use hybrid and collaborative filtering techniques.

Di Rocco *et al.* [33], proposes a recommender system based on a graph neural network (GNN) to assist modelers in performing the specification of metamodels and models. The (meta)model being specified, and the training data are encoded in a graph-based format by exploiting natural language processing (NLP) techniques. Afterward, a graph

kernel function uses the extracted graphs to provide modelers with relevant recommendations to complete the partially specified (meta)models. To train this ML system with a large set of models, they mined popularly Java software projects stored in the Maven repository and use the MoDisco reverse engineering tool [34] to extract a UML class based model representation for each project. As the first step, the tool produces a textual representation of the considered (meta)model by employing tailored parsers. Then, relevant features are encoded in a graph-based representation that preserves the internal structure of the represented model, i.e., relationships among defined entities. Afterward, the underpinning GNN model computes a graph kernel function used to assess the similarity among nodes by relying on the extracted graphs. The tool eventually retrieves the missing structural features that can be used to complete the models under construction. Furthermore, the system can recommend a ranked list of similar classes that modelers can embed to enrich their designs. One limitation is related to the time-consuming of the recommendation. The authors employ a technique that relies on graph kernels to compute similarity values. Such a comparison can be time-consuming in the case of large graphs. Furthermore, some relationships might be missed due to the presented encoding mechanism.

Burgueno *et al.* [35] proposes a NLP-based architecture for the autocompletion of partial UML class-based models has been presented in . From a set of textual documents related to the initial model, relevant terms are extracted to train a contextual model using the primary NLP pipeline, i.e., tokenization, splitting, and stop-word removal. Afterward, the system is fed with the model under construction to automatically slice it following a set of predefined patterns used to search for recommendations. Modelers can give feedback that can be used to update the model and improve the recommendations. The process starts by preprocessing all the available documentation about the project to use as input for the NLP training process. This step provides a corpus of text that satisfies the requirements imposed by the NLP algorithm chosen to create the NLP models. The input to this step is a partial domain model (e.g., a UML model). To improve efficiency, they do not generate autocomplete suggestions using the complete working model. Instead, a model slice is taken according to multiple dimensions and generates suggestions for each slice. For instance, to provide attribute suggestions for a class, the authors slice the model isolating the class for which a suggestion will be made. After the splicing process, the system extracts the element names, which become the list of positive words employed to query the NLP model. As a final step, a transformation of the refined lists of words into potential new model elements is performed. The proposed approach requires extracting relevant sources of general knowledge to gap the lack of contextual knowledge, as it may not always be accurate when a modeling activity requires knowledge about a specific application domain. Furthermore, the authors report that for a large

dataset of general knowledge, its current Python implementation takes around 20s to load the model and between 1-3s for querying; they register as a future work the re-implementation in c++. Also, the suggestions do not offer replacement and removal of parts.

Kuschke *et al.* [36], provides another approach for supporting UML-based-class. Here the authors have created an assistive tool for recommending valid and relevant completions of modeling activities for structural UML models while a developer changes a model. First, the system uses a pattern recognition algorithm for detecting editing events. Incoming events are matched against predefined activity patterns AP to detect partly performed modeling activities. For instance, an action made by the developer could match with the event: *"Replacing an association between two classes by an interface realization"*. It is worth noting that a previous work where the authors created a catalog of predefined modeling activities for structural UML models was used here. The filtered set of activity candidates is then ranked concerning the relevance of each candidate for the user. Relevant are activities that the user wants to perform. Finally, the filtered and ranked activity candidates are reduced to a comprehensible number of recommendations and presented within the modeling environment. This approach relies on a catalog of UML modeling that contains 19 activities. Here there is manual work that must evolve as long as the UML class modeling does. Another limitation is that it is environment-dependent, which makes the scalability of this framework difficult.

When creating a new software system or evolving an existing one, developers seek available libraries that suit their purpose. In such a context, open source software repositories contain rich resources that can provide developers with helpful advice to support their tasks [19]. In this mark, there are works related to the use of recommender systems to assist open source software developers in selecting suitable third-party libraries. Nguyen [19], creates a system for providing developers with third-party library recommendations was developed by using collaborative filtering techniques. The authors called **CrossRec**, it codes the relationships among various Open Source Systems (OSS) artifacts using a semantic graph and utilizes a collaborative filtering technique to recommend third-party libraries. This work relies on the premise that *"if projects share some libraries in common, then they will probably share other common libraries."*. This approach relies upon the user-matrix representation to get the similarities among libraries. It has a computational complexity limitation because more data has to be settled to increase the performance, which means more rows in the matrix.

3.4 Research Challenges

In this section, we describe the research challenges that we have to face in the reference architecture context and the use of recommender systems to support modellers with suggestions.

The modeling of software architectures and the use of modeling assistant to support it, ensuring conformance between the specific RA by suggesting modeling actions has to address issues related to:

- **I1**: The diversity of several Architecture Description Language for representing(**ADL**) SA.
- **I2**: The inefficient support for the instantiation a specific SA based on a RA.
- **I3** The complete absence of assistive modeling tools for reference architectural modeling.
- **I4**: RAs are domain specific and can be collected in repositories. As far as we know, there is no formal repository where we can find an RA with a set of software architectures that comply with and instantiate it.

We made a preliminary analysis of the literature related to this context to find out the issues highlighted in the reviewed works. We also mapped them to the above mentioned issues (in bolded brackets in the following paragraphs).

Too many ADLs have been developed over the past two decades; each of these ADLs follow a specific approach to the specification and evolution of the architecture [37]. ADL applications are wide and range in multiple directions. At least three areas where ADL is defined in different contexts can be found. These areas or communities are system engineering, software engineering, and enterprise modeling. They have their own specific description of ADL with one thing in common, they all provide the architectural representation. For system engineering, ADL describes and represents the system architecture. For software engineering, ADL describes and represents the software architecture. For enterprise modeling and engineering, ADL describes and refers to the application architecture. Software engineering itself contains two levels of architecture these are technical architecture and functional architecture. Technical architecture is targeted for the software developers while functional architecture is for the stakeholder. All these three communities have developed several different ADLs, which is one of the main reasons we do not see a common standard for ADLs. **(I1)**

One of the main success factors for RAs is using an appropriate abstraction level. Indeed, a RA can be specified at a different level of abstraction with incremental fine-grained details and is left to architects who personally decide the abstraction level and details needed. This decision is often rigid and can influence the entire architecture life-cycle. To date, there is no consensus on how to represent RAs precisely, and as a result, both practitioners and researchers tend to represent RAs only informally, also due to the lack of support in the entire process. Existing ADLs support the specification of concrete architectures, but maybe the wrong tool to declare RA and their usage for this purpose may result in misuse. Again this aspect enforces the need for the concretization of RA, not only as mere documentation and guidelines but at the same level of technology readiness of the ADLs. ADLs are not considering RAs in the architectural design, resulting in an unsupported instantiation process. **(I2)**

Recommender systems have been widely used for software engineering and modeling tasks, as it is described above. There are applications of these for suggesting third-party libraries in programming projects, suggesting and auto-completing API calls, with the goal of speeding up the development process. In software modeling, related to MDE, there are recommender systems that complete and repair models and suggest model transformations based on data sources represented as UML class diagrams. Even if we consider these examples as recommendation systems to build software, at the best of our knowledge, there is still a lack of application of RSs to help in architectural software modeling. **(I3)**

The main goal of our research is to develop a modeling assistant that helps the modeling process of software architectures to conform to its reference architecture. An essential issue for developing it, is the data, the source of knowledge. If we use some supervised learning techniques, these require an adequate data set. In this case, it is a challenge to obtain repositories where there are already examples of software architectures that seniors have verified correctly. In case we can obtain enough repositories, they will undoubtedly be modeled with different ADLs. Another critical issue would be how to classify these architectural models based on clustering techniques and how to derive the reference architecture that represents those. **(I4)**

Chapter 4

Research Proposal

In this chapter, we define the problem that arises from the challenges highlighted in the previous chapter in section 4.1. In section 4.2, we describe our research proposal and how it can represent a solution to the issues presented in the previous section. Hints and directions about validating our approach are discussed in section 4.2.2.

4.1 Problem Statement

In section 3.4 we mentioned the different challenges that we have to face in the reference architecture context and the use of recommender systems to supports modellers with suggestions.

We have to deal with the lack of a common interpretation for RA modeling and the inefficient support for the instantiation it. To date, there is no consensus on how to represent RAs accurately. Consequently, practitioners and researchers tend to represent RAs only informally due to the lack of support throughout the process. In the case of modeling software architecture, it can be observed that there are many ADLs developed by different companies and researchers, which have various applications and follow a specific approach for architecture specification and evolution. The applications of ADLs are broad and go in multiple directions. At least three areas where ADL is defined in different contexts can be found. They have their own specific description with one thing in common, they all provide an architectural representation. That gives us a problem which is what would be the correct representation of them and how to establish a mapping between the specific set of SAs with their general RA.

There are repositories on GitHub for various software architectural models. However, as far as we know, there is a lack of repositories that relate RAs to specific SAs, which is

necessary to create a dataset that is the source of knowledge for a recommender system to assist architectural modeling tasks.

Another major problem is the lack of applications of RSs to support architectural modeling tasks like instantiation and refactoring. This leaves us at a point where an in-depth study should be carried out on the correct tools and methodologies to support what we have previously described. Especially emphasizing the use of recommender system techniques, as these are useful for both assistive and automatic tasks. What are their advantages and disadvantages, and what are the data requirements and inputs of the different algorithms. In the case of using variants based on machine learning models and classification formulas, a good data set is a necessary requirement. In the case of algorithmic variants, a good rule system or good heuristic algorithm can help when there is not enough data. That confirms the complex that could be to archive a good RS that provides suggestions for completions and repair of a Software architectural model. Given the aforementioned challenges, the following research questions arise:

- **RQ1**: How are reference architectures (RAs) modeled and represented in the literature?
- **RQ2**: How can a correspondence between a set of software architectures (SA) and their general RA be defined?
- **RQ3**: What methodologies, tools and techniques are more suitable for building assistive architectural modeling tools?
- **RQ4**: How much a developed architectural modeling assistant is effective in supporting the architectural composition?
- **RQ5**: How much a developed architectural modeling assistant is effective in supporting other architectural modeling task, e.g., architectural switch, refactoring, etc?

This proposal aims to develop an assistive tool that supports software architecture modeling. We define the problems that arise with this, and explain the previous work that this proposal will be based on [1]. In that framework, unfortunately, the architect must always know the RA specification, specified or provided by senior architects or technology affiliates, to compose the concrete architecture and check the compliance. This task, even if it is a considerable improvement compared to the manual check, can present multiple difficulties, such as selecting the most suitable RA, instantiating and composing the architecture, or even refining it without a deep knowledge of the RA. In order to reduce the burden of these essential tasks and further automate and support the

architectural specification, we propose to perform these tasks under the guidance of a modeling assistant that can propose modeling actions with the intent of composing not only a valid architectural model but a compliant one w.r.t. the chosen RA.

4.2 Solution

Our vision is that effective RA modeling must be guided by a modeling assistant that provides interactive recommendations to support the instantiation, refinement, evolution, and migration of architectures with visualization and suggestions, continuously considering compliance with the RA. In this way, the software architect is supported in the definition of new architectural models and in their evolution while respecting the chosen RA.

The central core of this modeling assistant will be based on a recommender system. As a source of knowledge of the recommender engine, we will provide a cloud-based repository of software architectures and reference architectures. Combined with an algorithmic or rule-based approach, this dataset can derive a hybrid implementation of the RS that could archive better results.

The specific approach that we envision is sketched in Fig. 4.1. We rely on a modeling framework in charge of supporting all the modeling activities related to RAs and concrete architectures, which has been proposed in [1]. This is a prerequisite to support syntactic recommendations based on the typing relationship between the RA and the modeled architecture. A modeling editor supports the modeling phase of both the software architectures and RAs. The resulting models for both of them can be stored in modeling repositories, locally or on the cloud. When the architect composes an architectural model, at all stages, suggestions for the addition or removal of components, not only in case of violations but also considering existing architectural models defined by (other) architects, will be provided. Some edits or even more complex patterns can be suggested to reach compliance.

For instance, in Fig. 4.1 (the zoomed part), a violation is identified by a not allowed connector (red), and redirection is then suggested (in orange) to reach the minimal conformance w.r.t. the RA (on top). In the architectural model, the dotted elements are not part of the partial model but are generated based on suggestions of the recommender engine.

It is worth noting that without a live continuous assistance in modeling the architectures, the software architect would understand the violations at the end of the process, and she would need to face all the issues in an already developed architecture. This fact

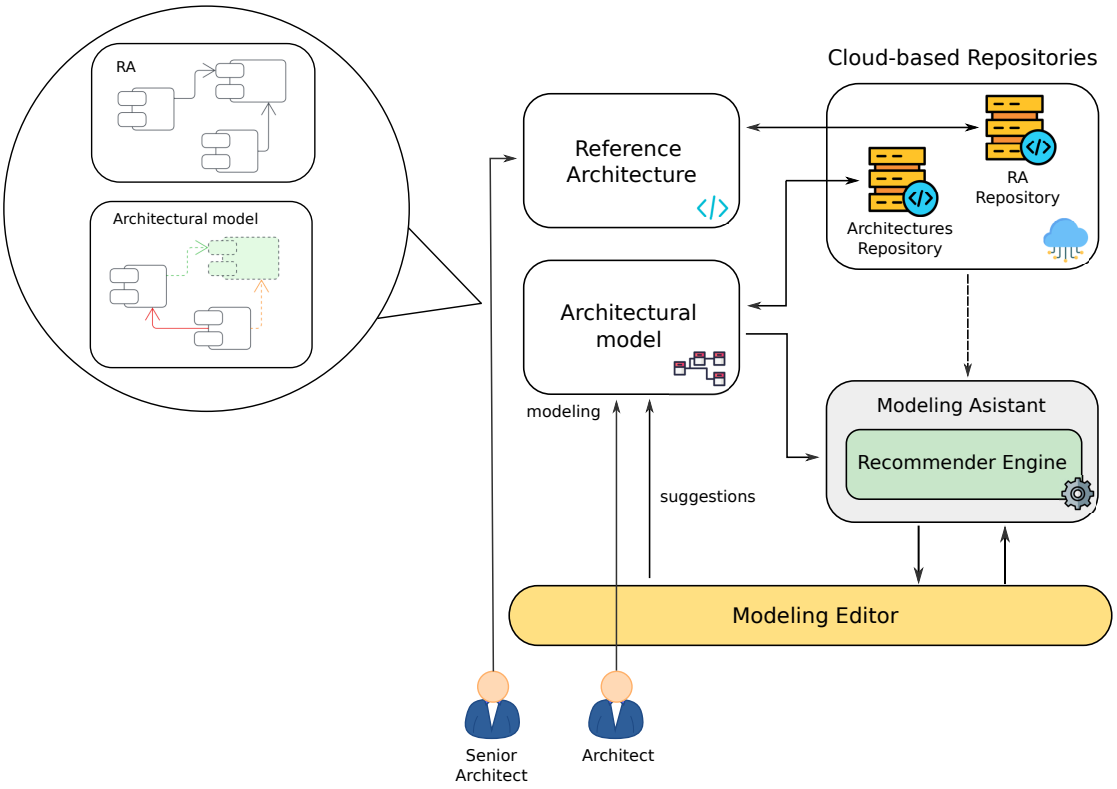

FIGURE 4.1: The proposed solution.

can be an absolute nightmare when violations are chained in a waterfall decision model, where removing a violation could introduce another problem. For this reason, it is crucial to support the entire process by allowing smaller architectural decisions rather than dealing with a complex resolution later. This is also often identified with the postponement of architectural improvements, which could lead to architectural erosion and technical debt. In conclusion, the proposed approach will automate and improve the process of modeling software architectures. When the architect needs to define her software architecture, the system can suggest the minimal architecture structure to comply with the chosen RA. Alternatively, when the architect has already specified a partial architecture, and while selecting a RA, the result is not compliant, the proposed solution can suggest the modeling actions to reach minimal compliance.

4.2.1 Research Plan

Figure 4.2 depicts the activity diagram describing the flow of activities we plan to perform to develop our approach. We will divide it into four phases: *Architectural Domain Analysis Phase*, *Recommender Systems Domain Phase*, *Engineering Phase*, and *Validation Phase*. In a preliminary phase, we want to perform a domain analysis of the reference architecture. We will conduct a literature review on RA modeling and software architectures. We will see different alternatives and ADLs used for representing

them. Furthermore, we focus on the approaches for mapping and validating a specific model of SA with its respective RA. Another critical study that we will perform will be related to the use of MDE techniques to address the problems mentioned 4.1. This is important as we will be based on previous work in which a framework for modeling RA and SA that checks compliance and validates them was developed.

After the first phase, we will start to do a deep study of recommender systems. First, we will see the different recommender systems we find in the literature, their advantages, disadvantages, and input settings. Then we will study their algorithms and their implementations in open sources. In the final step, we focus on the applications of RSs for supporting software engineering tasks, specifically those related to software modeling.

During the Engineering phase, we will start building the proposed tool. This phase will be divided into two sub-phases. First, we will create a model repository containing all the software architecture models and reference architectures we can use. This will be the knowledge base that the RS will use. Here we will considerably revise the literature and open sources to reuse models and approaches for deriving RA from a set of SAs. After that, we will develop the assistive tool.

The last phase will consist of the overall Evaluation and validation of our approach, which we will describe in detail in section 4.3.

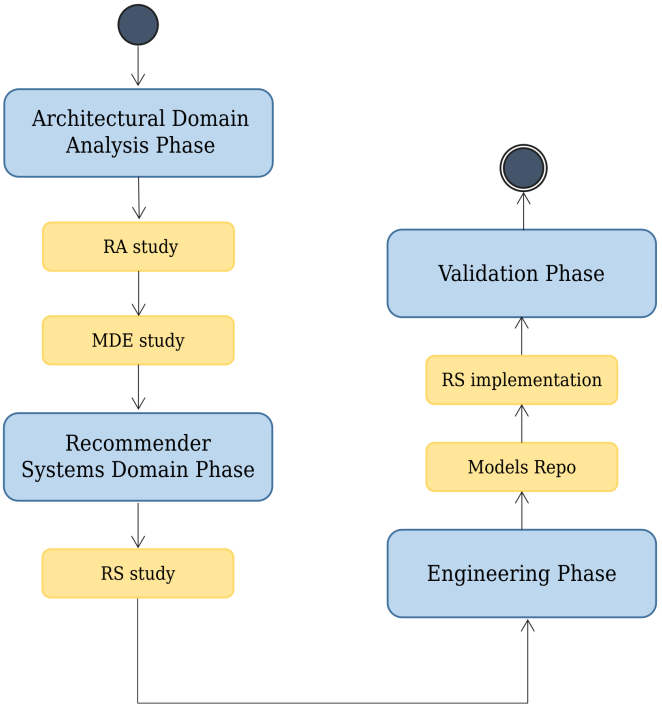

FIGURE 4.2: The proposed Research Plan

4.2.2 Research Contributions

The thesis will contribute to the definition and implementation of an assistive tool for supporting software architecture modeling tasks. In figure 4.3, we mapped the research questions listed in section 4.1 on the activities specified in the research plan described in section 4.2.1. Lastly, we schedule the publications we envisage producing for each activity. In particular, we plan to develop a survey paper from the architectural domain analysis phase. Instead, the recommender system domain and engineering phase's output will probably consist of at least a few conference papers. This is because, in this phase, we will provide results with the central research questions of this proposal. The tool's implementation and the subsequent approach evaluation will allow us to contribute to both conference and demo papers. Eventually, the overall contribution arising from this research work will be proposed in a journal paper, besides the corresponding PhD thesis.

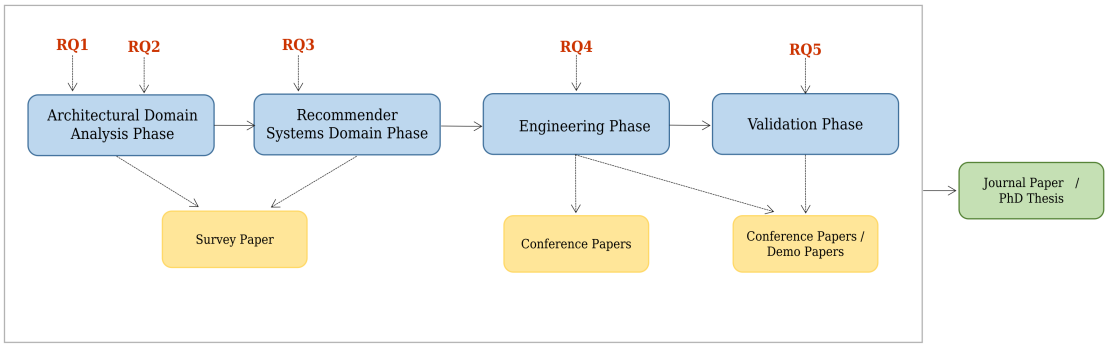

FIGURE 4.3: The proposed research contribution.

4.3 Research Validation

This proposal will build on the previous work presented here. We aim to introduce this RS into that framework to automate and improve the modeling process. After finishing our first implementation, the conformance and validation between the RA provided by the lead architect and the specific architecture modeled by the designer should not be affected. We will prepare a test suite to check the quality of the recommendation. From there, we can perform an online validation where we can choose different application domains such as (automotive, IoT, microservices, monolithic services, web browsers, etc.) with different stakeholders and verify the accuracy and correctness of the suggestions. Here we can archive this with the following:

- We can measure the effectiveness and correctness of the tool by counting the number of selections the user made in each suggestion list.
- Performing a qualitative analysis based on the positive or negative feedback we get from people who know well the domain of the applications under analysis.

Another critical point is the delay in the runtime execution. If the algorithm is inefficient or has a considerable delay in providing new suggestions, the user experience will be affected. This will cause the modeling process's automation will not be reflected, and users will never use the method. In order to test this requirement, we will do stress testing with complex scenarios and large models to measure the execution time.

Bibliography

[1] Alessio Bucaioni, Amleto Di Salle, Ludovico Iovino, Ivano Malavolta, and Patrizio Pelliccione. Reference architectures modelling and compliance checking. *Software and Systems Modeling*, pages 1–27, 2022.

[2] Linkedin. Reasons why software architecture is important, 2022. URL https://www.linkedin.com/pulse/8-reasons-why-software-architecture-important-ahad-khan-csm/.

[3] Sebastian Herold, Matthias Mair, Andreas Rausch, and Ingrid Schindler. Checking conformance with reference architectures: A case study. In *2013 17th IEEE International Enterprise Distributed Object Computing Conference*, pages 71–80. IEEE, 2013.

[4] Sandra Schröder and Matthias Riebisch. Architecture conformance checking with description logics. In *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings*, pages 166–172, 2017.

[5] Milena Guessi, Lucas Bueno Ruas de Oliveira, and Elisa Yumi Nakagawa. Representation of reference architectures: A systematic review. In *SEKE*, pages 782–785. Citeseer, 2011.

[6] Samuil Angelov, Paul Grefen, and Danny Greefhorst. A classification of software reference architectures: Analyzing their success and effectiveness. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*, pages 141–150. IEEE, 2009.

[7] Gerrit Muller. A reference architecture primer. *Eindhoven Univ. of Techn., Eindhoven, White paper*, 2008.

[8] Samuil Angelov, Jos JM Trienekens, and Paul Grefen. Towards a method for the evaluation of reference architectures: Experiences from a case. In *European Conference on Software Architecture*, pages 225–240. Springer, 2008.

[9] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. Addison-Wesley Professional, 2003.

[10] P Reed. Reference architecture: The best of best practices (2002). *Online.¡ http://www. ibm*, 2002.

[11] Amazon. Aws architecture center, 2022. URL https://aws.amazon.com/architecture/?nc1=h_ls.

[12] Microsoft. Azure architectures, 2022. URL https://docs.microsoft.com/en-us/azure/architecture/browse/.

[13] IBM. Ibm architecture center, 2022. URL https://www.ibm.com/cloud/architecture.

[14] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. ice model-driven software engineering in practice.

[15] Douglas C Schmidt. Model-driven engineering. *Computer-IEEE Computer Society-*, 39(2):25, 2006.

[16] Ian Gibson, Thomas Kvan, and Ling Wai Ming. Rapid prototyping for architectural models. *Rapid prototyping journal*, 2002.

[17] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005.

[18] Martin Robillard, Robert Walker, and Thomas Zimmermann. Recommendation systems for software engineering. *IEEE software*, 27(4):80–86, 2009.

[19] Phuong T Nguyen, Juri Di Rocco, Davide Di Ruscio, and Massimiliano Di Penta. Crossrec: Supporting software developers by recommending third-party libraries. *Journal of Systems and Software*, 161:110460, 2020.

[20] Phuong T Nguyen, Juri Di Rocco, Davide Di Ruscio, Lina Ochoa, Thomas Degueule, and Massimiliano Di Penta. Focus: A recommender system for mining api function calls and usage patterns. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1050–1060. IEEE, 2019.

[21] Marcos César de Oliveira, Davi Freitas, Rodrigo Bonifácio, Gustavo Pinto, and David Lo. Finding needles in a haystack: Leveraging co-change dependencies to recommend refactorings. *Journal of Systems and Software*, 158:110420, 2019.

[22] He Jiang, Jingxuan Zhang, Xiaochen Li, Zhilei Ren, David Lo, Xindong Wu, and Zhongxuan Luo. Recommending new features from mobile app descriptions. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 28(4): 1–29, 2019.

[23] Francesco Ricci, Lior Rokach, and Bracha Shapira. Recommender systems: introduction and challenges. In *Recommender systems handbook*, pages 1–34. Springer, 2015.

[24] Lizi Liao, Xiangnan He, Hanwang Zhang, and Tat-Seng Chua. Attributed social network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 30 (12):2257–2270, 2018.

[25] Lissette Almonte, Esther Guerra, Iván Cantador, and Juan De Lara. Recommender systems in model-driven engineering. *Software and Systems Modeling*, 21(1):249–280, 2022.

[26] Robin Burke. Knowledge-based recommender systems. *Encyclopedia of library and information systems*, 69(Supplement 32):175–186, 2000.

[27] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer, 2011.

[28] Ido Guy. Social recommender systems. In *Recommender systems handbook*, pages 511–543. Springer, 2015.

[29] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[30] Alan Grosskurth and Michael W Godfrey. A reference architecture for web browsers. In *21st IEEE International Conference on Software Maintenance (ICSM'05)*, pages 661–664. IEEE, 2005.

[31] Lakshitha De Silva and Dharini Balasubramaniam. Controlling software architecture erosion: A survey. *Journal of Systems and Software*, 85(1):132–151, 2012.

[32] Mario Luca Bernardi, Giuseppe Antonio Di Lucca, and Damiano Distante. A model-driven approach for the fast prototyping of web applications. In *2011 13th IEEE International Symposium on Web Systems Evolution (WSE)*, pages 65–74. IEEE, 2011.

[33] Juri Di Rocco, Claudio Di Sipio, Davide Di Ruscio, and Phuong T Nguyen. A gnn-based recommender system to assist the specification of metamodels and models. In *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 70–81. IEEE, 2021.

[34] Hugo Bruneliere, Jordi Cabot, Grégoire Dupé, and Frédéric Madiot. Modisco: A model driven reverse engineering framework. *Information and Software Technology*, 56(8):1012–1032, 2014.

[35] Loli Burgueño, Robert Clarisó, Sébastien Gérard, Shuai Li, and Jordi Cabot. An nlp-based architecture for the autocompletion of partial domain models. In *International Conference on Advanced Information Systems Engineering*, pages 91–106. Springer, 2021.

[36] Tobias Kuschke, Patrick Mäder, and Patrick Rempel. Recommending auto-completions for software modeling activities. In *International conference on model driven engineering languages and systems*, pages 170–186. Springer, 2013.

[37] Sajjad Hussain. Investigating architecture description languages (adls) a systematic literature review. 2013.