A Recommender Plug-in for Enterprise Architecture Models

Sashikanth Raavikanti[1], Simon Hacks[2][a] and Sotirios Katsikeas[1][b]
[1] *KTH Royal Institute of Technology, Stockholm, Sweden*
[2] *Stockholm University, Stockholm, Sweden*

Keywords: Enterprise Architecture, Archi, ArchiMate, Enterprise Modeling, Recommender Systems.

Abstract: IT has evolved over the decades, where its role and impact have transitioned from being a tactical tool to a more strategic one for driving business strategies to transform organizations. The right alignment between IT strategy and business has become a compelling factor for Chief Information Officers and Enterprise Architecture (EA) in practice is one of the approaches where this alignment can be achieved. Enterprise Modeling complements EA with models that are composed of enterprise components and relationships, that are stored in a repository. Over time, the repository grows which opens up research avenues to provide data intelligence. Recommender Systems is a field that can take different forms in the modeling domain and each form of recommendation can be enhanced with sophisticated models over time. Within this work, we focus on the latter problem by providing a recommender architecture framework eases the integration of different Recommender Systems. Thus, researchers can easily compare the performance of different recommender systems for EA models. The framework is developed as a distributed plugin for Archi, a widely used modeling tool to create EA models in the ArchiMate notation.

1 INTRODUCTION

IT has evolved over the decades, where its role and impact have transitioned from being a tactical tool to a more strategic one for driving business strategies to transform organizations (Henderson and Venkatraman, 1993). With increased digitization of the interconnection among products, processes, and services over the last two decades, the role of IT has been revisited to not just view it as an assistance to business strategy but rather a fusion that creates industry disruptions. Bharadwaj et al.(Bharadwaj et al., 2013) termed this phenomenon as ”Digital Business Strategy”. The right alignment between IT strategy and business has become a compelling factor for Chief Information Officers where misalignment could lead to a degradation of organization performance and slow them down in market competition (Coltman et al., 2015). Enterprise Architecture (EA) allows to achieve such an alignment (Alaeddini et al., 2017). Enterprise Modeling (EM) complements EA by providing models that represent the structure, behavior, and organization of the enterprise with different stakeholder viewpoints (Sandkuhl et al., 2014). In practice, EM is achieved through a language that provides a formal syntax, semantics and notation (Sandkuhl et al., 2014). A popular language for EA modeling (Barbosa et al., 2019) is ArchiMate that is provided by The Open Group (The Open Group, 2021a) and Archi[1] is an open source modeling tool that supports ArchiMate.

EA models are composed of enterprise components and relationships, and are stored in a repository (Sandkuhl et al., 2014). Over time, the repository grows as architects design new models or update existing models. Recommender Systems (RS) can help to manage the challenges related to growing repositories by providing meaningful suggestions to the user based on context (Ricci et al., 2011); i.e., by providing modeling assistance (Shilov et al., 2021) and component recommendations (Borozanov et al., 2019). A plethora of different algorithms can be used as backbone for a RS in EA modeling. Therefore, we as researchers are interested in a platform that eases the scientific comparison of the performance of these different algorithms. In line, we implement a recommender framework as a plugin for Archi. As the recommendation algorithms can be implemented in various technologies, the framework must be able to in-

[a] https://orcid.org/0000-0003-0478-9347
[b] https://orcid.org/0000-0001-8287-3160

[1] https://www.archimatetool.com/

Raavikanti, S., Hacks, S. and Katsikeas, S.
A Recommender Plug-in for Enterprise Architecture Models.
DOI: 10.5220/0011709000003467
In *Proceedings of the 25th International Conference on Enterprise Information Systems (ICEIS 2023) - Volume 2*, pages 474-480
ISBN: 978-989-758-648-4; ISSN: 2184-4992

tegrate to external RS outside Archi. The framework must be resilient to errors during the execution or any erroneous input. For instance, there can be latency issues when connecting to systems outside Archi, or the recommendation systems might not contain appropriate Archi model repository information to generate recommendations. We define this resilience as the robustness of the framework. We also focus on the ease of integration aspect where the framework can provide the needed information, and the recommendation system can connect to the framework with as minimal changes as possible. We, thus, explore the research question:

"How to design an architecture for Archi that provides easy integration to different recommendation systems?"

Using the steps of Design Science Research Methodology (DSRM) (Peffers et al., 7 12), we identify our research problem to be design-centric. The objective of the solution is to design a plugin with an architecture that satisfies our research problem, which also constitutes the main contribution of this work: **A framework to integrate different recommendation systems into the open source EA modeling tool Archi**. The design and development stage of DSRM consumed the most effort. We started with the theoretical foundations of EA to understand how Archi supports Enterprise Architects as a modeling toolkit. We studied the existing implementations of RS in the Software Engineering (Elkamel et al., 2016; Agt-Rickauer et al., 2018) and the EM domain (Borozanov et al., 2019; Shilov et al., 2021). We found that different forms of recommendations are possible and recommendation algorithms become more sophisticated over time. We considered a component recommendation strategy (Borozanov et al., 2019; Elkamel et al., 2016) as most suitable to base our framework on. We design our framework to contain a Graphical User Interface (GUI) component to show recommendations, a connector that can integrate the plugin to external systems, and a model to show the relationship between an object selection and its corresponding recommendations.

The rest of this paper is structured as following: Next, we present our extensions to the Archi tool. Then, we describe how the framework can be used for different RS and provide first insights from potential users. Before concluding our work, we shortly shed light on related work.

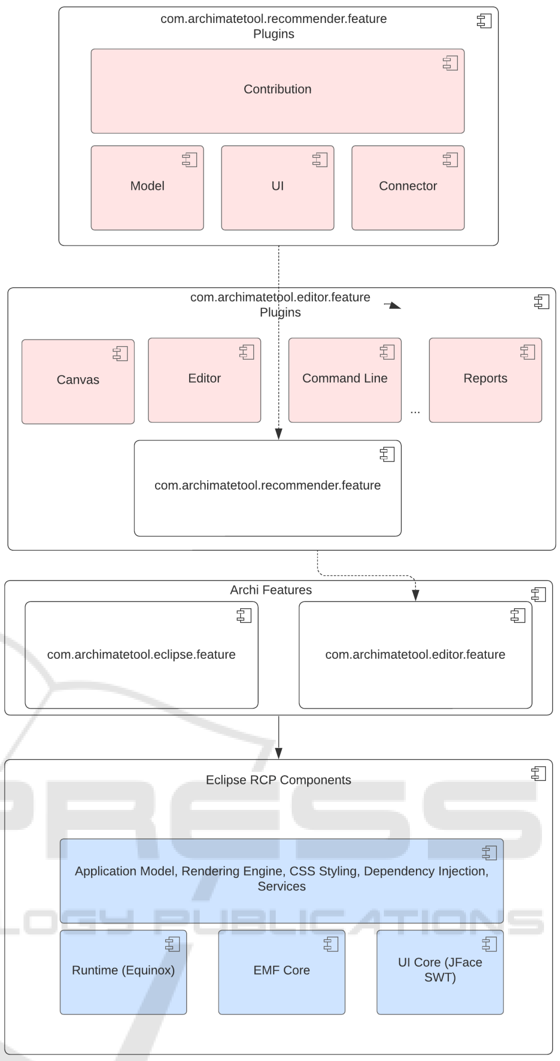

com.archimatetool.recommender.feature
Plugins
Contribution
Model
UI
Connector
com.archimatetool.editor.feature
Plugins
Canvas
Editor
Command Line
Reports
...
com.archimatetool.recommender.feature
Archi Features
com.archimatetool.eclipse.feature
com.archimatetool.editor.feature
Eclipse RCP Components
Application Model, Rendering Engine, CSS Styling, Dependency Injection, Services
Runtime (Equinox)
EMF Core
UI Core (JFace SWT)

Figure 1: Archi Architecture.

2 AN ARCHI FRAMEWORK FOR RECOMMENDER SYSTEMS

To answer our research question, we rely on the open-source solution Archi[2], which has been designed to model ArchiMate models. Archi is built on Eclipse Rich Client Platform (RCP) which provides a modular architecture. On a high level, Archi is composed of features that in turn contain sub-features and plugins. We implemented our RS framework as a sub-feature to Archi's **editor** feature. The high-level architecture diagram is shown in figure 1.

With Eclipse RCP's modular plugin concept and using the Common Closure principle and Common

[2]https://www.archimatetool.com/

Reuse principle, we divided our plugin into multiple plugins based on their capabilities such as GUI, Model, and Connector. We treat each plugin to be a component. Each component contains classes and interfaces that are grouped based on The Single Responsible Principle. These classes change for the same reason and serve a similar purpose. The plugins are defined with a fully qualified name following the naming conventions of Archi.

The plugins **Model (com.archimatetool.recommender.model)** and **GUI (com.archimatetool.recommender.ui)** are on the top-level in the hierarchy that contain abstract classes or interfaces. This adheres to Stable Abstractions Principle. The **CompletableRecommender** class use the CompletableFuture concept in Java for asynchronous communication when connecting external systems. The **RecommenderPreferenceAggregator** class provides an abstract implementation to capture user preferences. New preferences can be added by providing an implementation of this class. The **RecommenderView** class provides an abstraction of displaying recommendations in a Tree-based view. These plugins expose their classes and interfaces for other plugins to extend and provide their implementation. This also includes other Archi plugins that provide recommendation algorithms. The classes are loosely coupled and changes to a certain plugin will not force the change to other plugins.

The **Connector (com.archimatetool.recommender.connector)** plugin contains an abstract Connector interface and provides **HTTPConnector** as a default implementation. This addresses the objective to integrate with external RS. As multiple RS are also possible, we implemented an observer pattern where a plugin can subscribe to a certain RS using the **RecommendationSubscriber** class. This class maintains the subscription information and communicates it back to the plugin once it receives recommendation information. The **HTTPConnector** provides both synchronous and asynchronous communication between the plugin and the external system, with synchronous as default.

To see the recommender feature in action within Archi, a concrete implementation of GUI and Model classes is needed. This is addressed using **Contribution (com.archimatetool.recommender.contribution)** plugin. It provides an implementation to show component recommendations in a tabular format. In the case of Archi, the component can be an Element within a model. The table shows a grid where the columns provide the recommended component information along with a similarity score (cf. Figure 2).

The Contribution plugin subscribes to the Connector plugin. With the Contributor plugin, we defined a simple JSON schema. This is to address the robustness objective of the framework where any external RS adheres to this contract and the behavior of the plugin will not change when we connect different RS. As part of the contract, the outgoing request will send the **id** of the element or relationship. The JSON response must adhere to the contract by validating against the JSON schema. The schema expects an array of recommendations, where every recommendation must contain its **id**, **type**, **name**, **model-id** and a similarity **score**. The Contributor plugin uses a **RecommendationJSONParser** class to parse the response. Figure 3 shows the UML class diagram of the entire plugin.

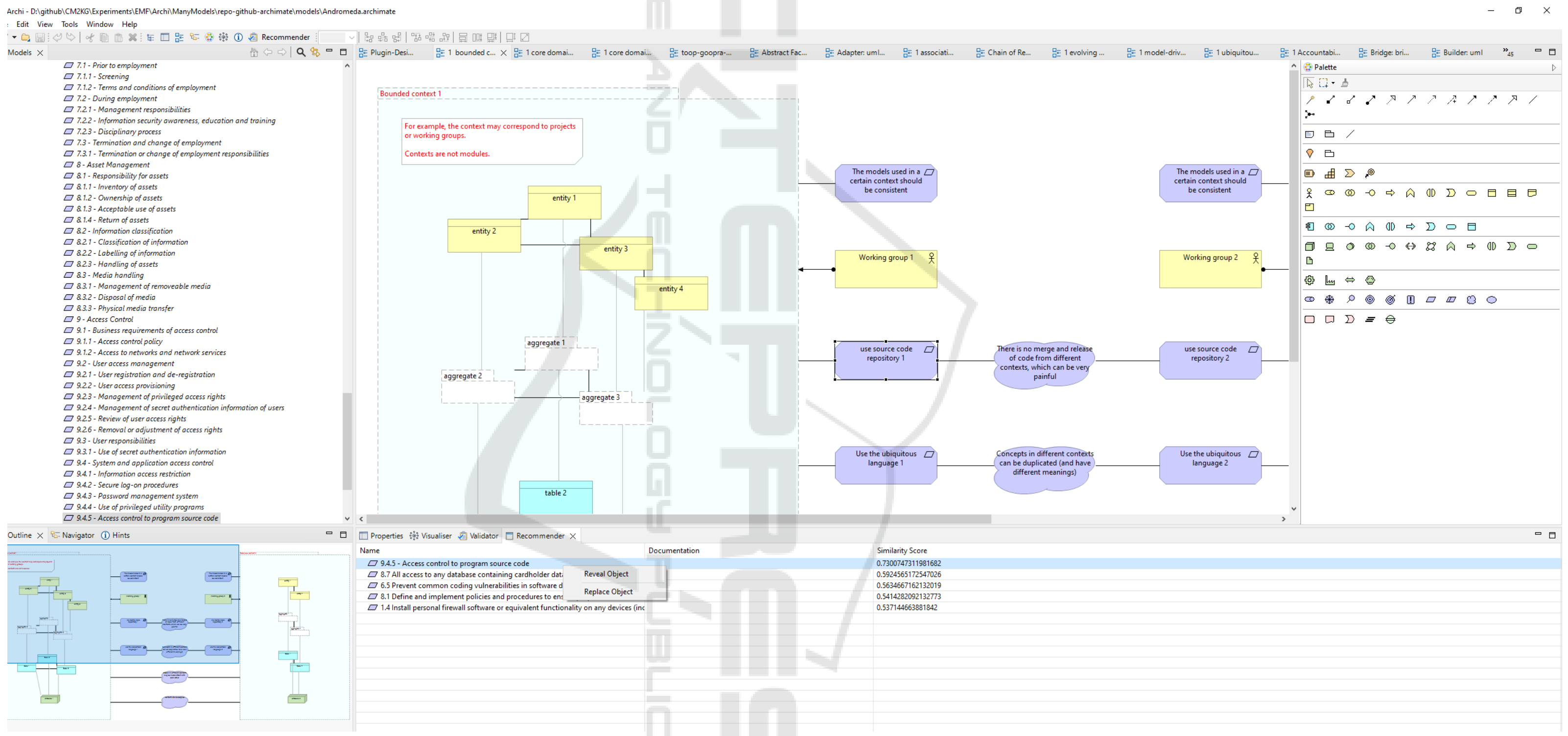
3 USAGE OF THE FRAMEWORK

The plugin can connect to any RS that provides an endpoint over HTTP and provides a JSON response that complies with the schema. The current implementation of the plugin supports only one active connection to the external RS. Also, the current implementation supports only synchronous communication, and we left asynchronous communication as an avenue for future research.

3.1 Demonstration

As our focus was only on the framework, we have simulated the external recommendation system by hosting a REST service using the Java Spring Boot framework. We have loaded some Archi Model elements to this external recommendation system database. Whenever the REST endpoint receives a GET request, it generates a JSON response with mock recommendations. The recommendations are displayed in a table within Archi as shown in the figure 2. It contains 3 columns with the name of the element, the documentation property corresponding to that element and a similarity score calculated from the recommendation model.

For each recommendation, we can perform two different actions: **Reveal** and **Replace** using the **ComponentViewActions** class in the **Contribution** plugin. As the recommended component could belong to other models also, **Reveal** would point to the corresponding model of a recommended component. The element belongs to **1 bounded context** model


Archi - D:\github\CM2KG\Experiments\EMF\Archi\ManyModels\repo-github-archimate\models\Andromeda.archimate
Edit
View
Tools
Window
Help
Recommender
Models
Plugin-Desi...
1 bounded c...
1 core domai...
1 core domai...
toop-goopra-...
Abstract Fac...
Adapter: uml...
1 associati...
Chain of Re...
1 evolving ...
1 model-driv...
1 ubiquitou...
1 Accountabi...
Bridge: bri...
Builder: uml
Palette
Bounded context 1
For example, the context may correspond to projects or working groups.
Contexts are not modules.
entity 1
entity 2
entity 3
entity 4
aggregate 1
aggregate 2
aggregate 3
table 2
The models used in a certain context should be consistent
Working group 1
use source code repository 1
Use the ubiquitous language 1
There is no merge and release of code from different contexts, which can be very painful
Concepts in different contexts can be duplicated (and have different meanings)
The models used in a certain context should be consistent
Working group 2
use source code repository 2
Use the ubiquitous language 2
Outline
Navigator
Hints
Properties
Visualiser
Validator
Recommender
Reveal Object
Replace Object

- 7.1 - Prior to employment
- 7.1.1 - Screening
- 7.1.2 - Terms and conditions of employment
- 7.2 - During employment
- 7.2.1 - Management responsibilities
- 7.2.2 - Information security awareness, education and training
- 7.2.3 - Disciplinary process
- 7.3 - Termination and change of employment
- 7.3.1 - Termination or change of employment responsibilities
- 8 - Asset Management
- 8.1 - Responsibility for assets
- 8.1.1 - Inventory of assets
- 8.1.2 - Ownership of assets
- 8.1.3 - Acceptable use of assets
- 8.1.4 - Return of assets
- 8.2 - Information classification
- 8.2.1 - Classification of information
- 8.2.2 - Labelling of information
- 8.2.3 - Handling of assets
- 8.3 - Media handling
- 8.3.1 - Management of removeable media
- 8.3.2 - Disposal of media
- 8.3.3 - Physical media transfer
- 9 - Access Control
- 9.1 - Business requirements of access control
- 9.1.1 - Access control policy
- 9.1.2 - Access to networks and network services
- 9.2 - User access management
- 9.2.1 - User registration and de-registration
- 9.2.2 - User access provisioning
- 9.2.3 - Management of privileged access rights
- 9.2.4 - Management of secret authentication information of users
- 9.2.5 - Review of user access rights
- 9.2.6 - Removal or adjustment of access rights
- 9.3 - User responsibilities
- 9.3.1 - Use of secret authentication information
- 9.4 - System and application access control
- 9.4.1 - Information access restriction
- 9.4.2 - Secure log-on procedures
- 9.4.3 - Password management system
- 9.4.4 - Use of privileged utility programs
- 9.4.5 - Access control to program source code

Name	Documentation	Similarity Score
9.4.5 - Access control to program source code		0.7300747311981682
8.7 All access to any database containing cardholder dat		0.5924565172547026
6.5 Prevent common coding vulnerabilities in software d		0.5634667162132019
8.1 Define and implement policies and procedures to ens		0.5414282092132773
1.4 Install personal firewall software or equivalent functionality on any devices (inc		0.537144663881842

Figure 2: Recommender Plugin.

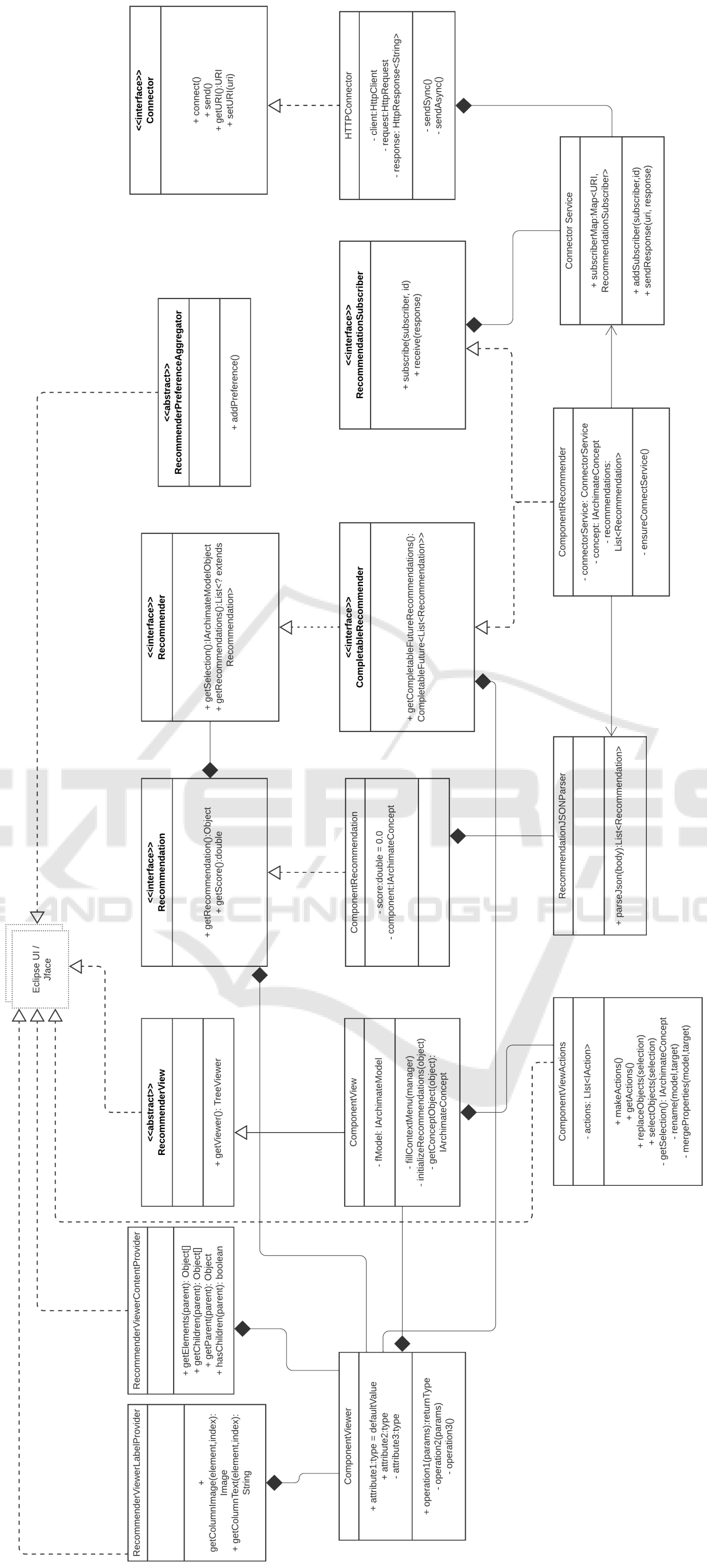

<<interface>>
Connector
+ connect()
+ send()
+ getURI():URI
+ setURI(uri)
HTTPConnector
- client:HttpClient
- request:HttpRequest
- response: HttpResponse<String>
- sendSync()
- sendAsync()
<<abstract>>
RecommenderPreferenceAggregator
+ addPreference()
<<interface>>
RecommendationSubscriber
+ subscribe(subscriber, id)
+ receive(response)
Connector Service
+ subscriberMap:Map<URI, RecommendationSubscriber>
+ addSubscriber(subscriber,id)
+ sendResponse(uri, response)
<<interface>>
Recommender
+ getSelection():IArchimateModelObject
+ getRecommendations():List<? extends Recommendation>
<<interface>>
CompletableRecommender
+ getCompletableFutureRecommendations(): CompletableFuture<List<Recommendation>>
ComponentRecommender
- connectorService: ConnectorService
- concept: IArchimateConcept
- recommendations: List<Recommendation>
- ensureConnectService()
<<interface>>
Recommendation
+ getRecommendation():Object
+ getScore():double
ComponentRecommendation
- score:double = 0.0
- component:IArchimateConcept
RecommendationJSONParser
+ parseJson(body):List<Recommendation>
Eclipse UI / Jface
<<abstract>>
RecommenderView
+ getViewer(): TreeViewer
ComponentView
- fModel: IArchimateModel
- fillContextMenu(manager)
- initializeRecommendations(object)
- getConceptObject(object): IArchimateConcept
ComponentViewActions
- actions: List<IAction>
+ makeActions()
+ getActions()
+ replaceObjects(selection)
+ selectObjects(selection)
- getSelection(): IArchimateConcept
- rename(model,target)
- mergeProperties(model,target)
RecommenderViewerContentProvider
+ getElements(parent): Object[]
+ getChildren(parent): Object[]
+ getParent(parent): Object
+ hasChildren(parent): boolean
RecommenderViewerLabelProvider
+ getColumnImage(element,index): Image
+ getColumnText(element,index): String
ComponentViewer
+ attribute1:type = defaultValue
+ attribute2:type
- attribute3:type
+ operation1(params):returnType
- operation2(params)
- operation3()

Figure 3: Archi Recommender - Class Diagram.

and Plugin Connectors is revealed in the **Plugin Design** model.

Replace would replace the name of the selected component with the recommended component. It also appends the properties from the recommended component to the selected component. This action is to demonstrate that we can perform modifications through the framework.

The plugin is extensible to provide any kind of new actions by extending this class such as to Revert the change that was made.

We simulated scenarios with erroneous input and induced latency to test the robustness of the architecture. We noticed that the scenarios related to erroneous input have no impact, but the thread within Archi waits for a response which blocks the execution within Archi. We perceive this to be a limitation within the Archi framework. This is left as an avenue for future research.

3.2 First User Feedback

To evaluate the usage of our framework, we considered technical aspects (ease of integration and robustness of the architecture) and gathered first feedback from Archi users. For the technical evaluation, we collaborated with a developer who possesses knowledge on Natural Language Processing for working with text data. We implemented a new RS that recommends components that have similar names. The exemplary implementation of a RS with our framework by the developer showed that some minor changes were needed: On the RS side, additional functions were implemented to provide more information within the request. For example, the developer expected more parameters such as the type, name, component id, and model id. To evaluate the robustness, we provided the plugin with incorrect data and simulated different types of responses with various HTTP codes to verify if the plugin breaks or affects the performance of Archi. There were no latency issues when accessing RS on localhost or the internet through ngrok[3]. The developer suggested the implementation of a queue in between the plugin and the external RS to make it more robust when dealing with multiple requests. Generally, the developer was positive on the plugin's error handling, besides that the synchronous connection caused waiting times. The integration was perceived as efficient.

We have also collected feedback from Archi users regarding the usability of the plugin and possible improvements. We demonstrated the plugin and received positive feedback regarding its usability. One suggested improvement to the plugin was to provide a possibility to include repository information for models when requesting recommendations. For instance, the current implementation only shows recommendations for models open within the Archi workspace. Instead, several repositories should be considered. Most of the discussion was based on the recommendation capabilities within the plugin such as the user interface. As our focus is only on the architecture, we have not captured those discussions as part for this first step. Providing different GUI capabilities for the framework is left as an avenue for future research.

[3]https://ngrok.com/

4 RELATED WORK

In Software Engineering (SE) and EA, RS is a recent trend (Elkamel et al., 2016; Shilov et al., 2021). We used related research to modeling in SE and EA as rationale for our framework.

In SE, we took a closer look at the UML. E.g., Elkamel et al. (Elkamel et al., 2016) developed a prototype for recommending UML classes that are semantically similar in terms of their characteristics. Once the recommendations are displayed, the user can accept one or more of the recommended classes. For each class, the user can accept all the attributes and methods or only some of them. Another contribution is RS for domain modeling (DoMoRe) (Agt-Rickauer et al., 2018). DoMoRe provides context-sensitive information during domain modeling and suggests names for model elements that are ordered by relevance and are semantically similar. The DoMoRe system integrates with various knowledge bases using mediator-based information. A typical example to demonstrate the usage of DoMoRe is to take a modeling use-case for the HealthCare domain. Consider there are 2 UML classes Patient and Doctor, when an association relationship is set from Doctor class to Patient class, DoMoRe takes the contextual information based on the class names, refers to its knowledge base, and comes up with some recommendations for the name of the association such as treats, examines, visits, etc., These contributions used the applications of RS to address different research problems. The former addresses redundant classes and the latter helps in having a shared understanding of the problem domain across different stakeholders.

Within EA research, Borozonaov et al. used machine learning (ML) techniques to reduce repository pollution which arises when models have redundant components that are semantically similar (Borozanov et al., 2019). This is demonstrated with a client-server architecture, and as a plugin for Archi, where

Archi sends model information in an XML format (The Open Group, 2021b) to a recommendation system outside Archi. Once the response is received, the recommendations are displayed to the Enterprise Architects. Another research uses ML for EA model prediction which reduces the manual effort during the modeling process (Shilov et al., 2021). With the help of Graph Neural Networks, models have been built for node classification and edge prediction. This helps modeling by extracting patterns and best practices.

5 CONCLUSION

As main contribution of this work, we designed an architecture for a recommender plugin for Archi that could easily integrate with an external RS. The current implementation of the plugin is lightweight in terms of the amount of data exchange. Hence, there were no latency issues. For evaluation, we implemented a connection to an external RS and collected feedback from the user and research community regarding the usability of the plugin. Further evaluation can be done, with other RS that have a different implementation and have other requirements for data exchange. The plugin framework is also extensible to add new features. However, we only implemented a simple RS using the current framework to showcase its use. The current implementation is tailored for a component-based RS. With a more sophisticated recommender framework in terms of GUI, the plugin can be more generic to accommodate different forms of recommendation. For instance, context-based recommendations (Agt-Rickauer et al., 2018) and enterprise model prediction (Shilov et al., 2021). This is left as an avenue for future research.

6 ADDITIONAL MATERIAL

The source code of the plugin can be accessed via our github repository[4]. Additionally, we prepared a short video to present the tool[5].

REFERENCES

Agt-Rickauer, H., Kutsche, R.-D., and Sack, H. (2018). DoMoRe-a recommender system for domain modeling. In *MODELSWARD*, pages 71–82.

Alaeddini, M., Asgari, H., Gharibi, A., and Rashidi Rad, M. (2017). Leveraging business-IT alignment through enterprise architecture—an empirical study to estimate the extents. *Information Technology and Management*, 18(1):55–82.

Barbosa, A., Santana, A., Hacks, S., and Stein, N. v. (2019). A taxonomy for enterprise architecture analysis research. In *21st International Conference on Enterprise Information Systems*, volume 2, pages 493–504. SciTePress.

Bharadwaj, A., El Sawy, O. A., Pavlou, P. A., and Venkatraman, N. (2013). Digital business strategy: Toward a next generation of insights. *MIS Quarterly*, 37(2):471–482.

Borozanov, V., Hacks, S., and Silva, N. (2019). Using machine learning techniques for evaluating the similarity of enterprise architecture models. In Giorgini, P. and Weber, B., editors, *Advanced Information Systems Engineering*, pages 563–578. Springer International Publishing.

Coltman, T., Tallon, P., Sharma, R., and Queiroz, M. (2015). Strategic IT alignment: Twenty-five years on. *Journal of Information Technology*, 30(2):91–100.

Elkamel, A., Gzara, M., and Ben-Abdallah, H. (2016). An UML class recommender system for software design. In *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pages 1–8.

Henderson, J. C. and Venkatraman, H. (1993). Strategic alignment: Leveraging information technology for transforming organizations. *IBM Systems Journal*, 32(1):472–484.

Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. (2007-12). A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45–77.

Ricci, F., Rokach, L., and Shapira, B. (2011). Introduction to recommender systems handbook. In Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors, *Recommender Systems Handbook*, pages 1–35. Springer US.

Sandkuhl, K., Stirna, J., Persson, A., and Wißotzki, M. (2014). *Enterprise modeling*. Springer.

Shilov, N., Othman, W., Fellmann, M., and Sandkuhl, K. (2021). Machine learning-based enterprise modeling assistance: Approach and potentials. In Serral, E., Stirna, J., Ralyté, J., and Grabis, J., editors, *The Practice of Enterprise Modeling*, pages 19–33. Springer International Publishing.

The Open Group (2021a). ArchiMate® 3.1 specification.

The Open Group (2021b). ArchiMate® model exchange file format for the ArchiMate modeling language, version 3.0.

[4]https://github.com/sashikanthr/archi-recommender
[5]https://youtu.be/iA51FG39omE