UNIVERSITY OF LEICESTER

Department of Informatics University of Leicester CO7201 Individual Project

Final Report Comparit - A Syntactic and Semantic Model Comparison Tool

Jawad Mustafa
jm982@student.le.ac.uk
239036872

Project Supervisor: Dr. Artur Boronat
Second Marker: Dr. Furqan Aziz

September 6th, 2024
Word Count: 10885

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor and mentor, Dr. Artur Boronat, for their invaluable guidance, continuous support, and patience throughout this research. Their expertise and insight have been fundamental to the success of this work, and I am immensely grateful for the countless hours they dedicated to refining my ideas and improving the quality of this thesis.

I also want to extend my heartfelt thanks to my co-advisor Dr Furqan Aziz for taking significant interest in my project, and for their constructive feedback, challenging questions, and insightful suggestions that helped strengthen my research.

I am deeply indebted to my family for their unconditional love and support. To my parents, Dara and Saeeda, my cousin, Ghazanfar, Fizza, and little Azlan, thank you for your constant encouragement, belief in me, and understanding during the demanding periods of this journey.

Finally, to my friends, Shiraz and Khawaja, and loved ones, thank you for your patience, positivity, and support, which helped me persevere through the toughest moments. Your understanding, especially when I was consumed by this research, means the world to me.

This thesis is dedicated to everyone who has contributed in some way to the successful completion of this work.

Thank you all!

Abstract

Model matching or model differencing is the process of comparing software models and reporting the degree of similarity between them. Model Comparison tools are used in the field of Model Driven Engineering and Model Driven Reverse Engineering to gain insights and perform analysis on software systems. We investigate the existing model comparison approaches, identify their gaps, and propose a mature IDE-agnostic model comparison tool named as "Comparit" with configurable granularity of comparison that could help further research in MDE and MDRE by allowing for comparison of model pairs. In addition to that, we provide an interface that could be implemented to accommodate a variety of model comparison algorithms. To assert the credibility of our tool, we have used an elaborate testing scheme. YAMTL (Yet Another model transformation language) is used to create mutants of models; each mutant labeled with the correct metrics. The test dataset is used to evaluate the credibility of the proposed tool.

Contents

Chapter 1

Introduction

Software models are abstract representations of a software system that help understand the design and behavior of different components in a software system. They serve as blueprints for the development of new systems and as documentation for a project that could be used by newcomers to gain familiarity with the internal functioning of the system.

Software models are generally categorized into Structural Models and Behavioral Models; The former serve as representatives of the data model; showcasing classes, attributes, methods and their relationships while the latter capture the flow of information in the system and depict the interaction of components and sequence of events.

There are various modelling languages and frameworks for software modelling including and not limited to: Eclipse Modelling Framework (EMF) [9], UML [16] and Moose [3]. These frameworks allow users to create and edit software models via code as well as drawings. The source code of the diagrams opens doors for code generation and automated software analysis. For example, EMF contains the feature to generate code from ecore models. On the contrary, Modular Moose, Papyrus and Modisco can be used to convert java projects to Moose Models.

The comparison of software models is a fundamental task in MDE (Model Driven Engineering) and MDRE (Model Driven Reverse Engineering) research, essential for various purposes such as software evolution, model-driven development, model clone detection, and quality assurance. Model Comparison tools can assist researchers working in the field of MDRE to validate the results of their reverse engineering algorithms by comparing them with baseline models. When it comes to software evolution and quality assurance;

model comparison tools can help track changes between different version of software.

There are two aspects of model comparison; syntactic and semantic. **Syntactic comparison** focuses on the structural similarity between models, disregarding semantic aspects, and plays a crucial role in identifying similarities and differences between different versions of models or models created by different stakeholders. **Semantic comparison** focuses on the resemblance of two models with respect to their meaning or semantics. From the perspective of MDE, this could help identify model clones and aid with model versioning. Additionally, semantic comparison of models could automate the grading of software based assignments in academia; extracted models from the student's work could be compared with the benchmark models to check the validity. This would not be the same as plagiarism detection, because the student might have chosen a different coding style to achieve the required semantics.

Somogyi [22] defines two goals for model comparison frameworks:

- **Accuracy**: Refers to how accurately the algorithm captures the differences. The accuracy could be gauged by several evaluation techniques for model comparison algorithms such as utilizing model generation to create benchmarks for evaluation [14].

- **Generality**: Refers to how adaptable the algorithm is for a variety of modelling languages.

There exist multiple model comparison algorithms that would be detailed in the literature review of this report. Kolovos et al. [13] categorizes these algorithms into four types:

- **Static Identity-Based Matching:** This method relies on persistent and non-volatile unique identifiers associated with model elements, usually in the form of universally unique identifiers (UUIDs). While static matching ensures accuracy by always providing determinable matches, it may encounter difficulties when attempting to generalize to more complex modeling languages due to its dependence on unique identifiers.

- **Signature-Based (Dynamic) Matching:** Pre-selected subset of model features are used to compute signatures, and the signatures are compared to determine model similarity.

- **Similarity-Based Matching:** Differing from previous approaches, similarity based algorithms do not yield a binary yes or no answer to match queries. Instead, a degree of similarity is computed, providing fuzzy comparison.

- **Custom Language-Specific Matching:** These algorithms are tailored to specific modeling languages, utilizing the particular semantics of the language for matching. While such algorithms offer high accuracy, they lack generality.

In this paper, we propose an IDE-agnostic model comparison tool named as "Comparit" that allows the user to choose the type of matching; similarity based or signature based, and specify the similarity threshold, modelling language of the models to be compared, and the granularity of comparison. The granularity of comparison refers to the elements that should be included in the computation of syntactic similarity. The comparison would result in the computation of metrics that reflect the degree of syntactic similarity; precision, recall, and f1-score, and semantic similarity.

Paper organization: Chapter 2 discusses the state-of-the-art model comparison techniques and how they could be improved. Additionally, existing evaluation techniques of model comparison algorithms is discussed later in the chapter. Chapter 3 enlists the objectives of this tool in light of the literature review. Chapter 4 presents the architecture of our model comparison tool using descriptive diagrams. Chapter 6 explains in detail the algorithms used for model comparison in our tool followed by Chapter 6 that contains the workflow of the user-interface. Chapter 7 demonstrates a case study carried out using Compair, followed by Chapter 8 that presents the evaluation methodology and corresponding results for our tool. At the end, we conclude and provide a list of actionable items with regards to this tool for the future.

Chapter 2

Literature Review

In this literature review, we explore state-of-the-art model comparison techniques; focusing on the granularity aspect, evaluation methodology and the challenges associated with model comparison. We identify gaps in the literature and opportunities for innovation in this domain.

2.1 Research Questions

The research questions were formulated to incorporate an understanding of existing work related to model comparison. The following questions allow us to navigate the literature to find existing model comparison tools and the challenges associated with their usage and evaluation.

- RQ1: What are the common challenges faced by state-of-the-art model comparator tools when comparing Ecore and UML2 models, and how does the proposed tool address these challenges?
- RQ2: What are the performance implications of using the proposed comparator tool in large-scale model comparison scenarios?
- RQ3: What metrics and criteria should be used to evaluate the effectiveness and accuracy of model comparison tools? (This could be an entirely new area: 'evaluating model comparator tools')
- RQ4: What metrics can be representative of the syntactic and semantic similarity of two models?
- RQ5: How can text-based matching help with model comparison?

2.2 Search Strings

The following search strings were created in accordance with each research question to explore relevant articles. These strings were used on google scholar and ACM to find existing literature:

- Query String for RQ1: "(challenges OR problems) AND (""model comparison"" OR ""model matching""OR ""model alignment"") AND (""software engineering"" OR ""software development"" OR ""Model Driven Engineering"" OR ""Model-Driven Engineering"" OR ""Model-Driven Reverse Engineering"" OR ""Model Driven Reverse Engineering"" OR ""MDRE"" or ""MDE"") "

- Query String for RQ2: ("Ecore" OR "UML") AND ("model comparison" OR "model matching" OR "model alignment") AND (performance OR runtime OR "run time" OR "run-time" OR scalability)

- Query String for RQ3:("metrics" OR "criteria") AND ("evaluation" OR "assessment" OR "measurement") AND ("Ecore" OR "UML") AND ("model comparison" OR "model matching" OR "model alignment")

- Query String for RQ4: ("metrics" OR "criteria") AND ("syntactic similarity" OR "semantic similarity") AND ("Ecore" OR "UML") AND ("model comparison" OR "model matching" OR "model alignment")

- Query String for RQ5: ("text-based" OR "textual") AND ("model comparison" OR "model matching" OR "model alignment") AND ("software engineering" OR "software development" OR "Model Driven Engineering" OR "Model-Driven Engineering" OR "Model-Driven Reverse Engineering" OR "Model Driven Reverse Engineering" OR "MDRE" or "MDE")

2.3 Review of Model Comparison Techniques

There have been several systematic literature reviews [[13] [17] [22]] done on existing techniques for model matching; be it syntactic or semantic. The latest systematic literature review was done by Somogyi [22] in 2019 that analyzes 72 model matching algorithms to find that 8 of them are text-based, while the other 64 are graph based algorithms. The text-based algortihms refer to raw text differencing or the XML based serialization of models.

Below is the summary of text baed algorithms surveyed by Somogyi [22]:

- Badreddin et al [4]: The goal of this tool was to provide textual differencing between models similar to a version control system.
- Alwanain et al [2]: Used for comparison of sequence diagram based on Alloy; a modelling tool.
- Maoz et al. [15]: is a semantic differencing operator for UML class diagrams. The tool works for models based on Alloy language. The authors have mentioned the application of semantic differencing on models based on other languages as future work.
- Foucault et al. [10]: The matching algorithm is baesd on static identifiers. The algorithm is specific to EBNF based modelling language. The author of the literature review remarks that while the algorithm aims to achieve both accuracy and generality, it is limited in the latter due the usage of static identifiers.
- Somogyi and Aztalos [21] [20]: The algorithm compares and merges text based models. The algorithm provides generality by allowing the user to specify he parser for the text-based models. The matching process depends significantly on the parser. The parser must meet several algorithmic requirements, which involves some configuration effort. However, this setup allows the algorithm to achieve both accuracy and generality.
- Barrett et al. [5]: The algorithm proposed by Barrett et al. [19] is designed for merging use case models provided in textual form, where use cases are typically described in natural language. This operation-based algorithm tracks changes as they happen, eliminating the need for explicit model matching. It involves constructing finite state machines from the textual descriptions, which are then utilized in the merging process. The authors emphasize the accuracy of merging textual use case models, a problem that previously lacked a proposed solution, and introduce a novel algorithm to address this challenge.

- Rozen and van der Storm [26]: The proposed algorithm employs standard textual differencing to track the origin between the textual representation and the model. It is also operation-based, which means it does not require explicit matching because it relies on operation-based change tracking. However, it still uses raw text differencing for origin tracking, specifically for mapping between text and model. The algorithm is implemented in Rascal, and a parser is used to parse text into an abstract syntax tree. The authors highlight a notable issue in text-based model matching: non-semantic information (such as comments) can impact the algorithms and must be managed appropriately.

- Rivera and Vallecillo [18]: Text-based algorithms baed on Maude are compared using static identifiers and similarity metrics.

The individual description of the graph-based model comparison algorithms have not been discussed by Somogyi [22] due to space limitations. The general comments about graph-based algorithms suggest that there is more variety in graph-based algorithms compared to text-based algorithms. 25% of the algorithms are focused on generality, 22% 42 on accuracy, 22% on both accuracy and generality. As a part of this review; we discuss the most popular graph-based algorithm; EMF-Compare and draw inspiration for our tool. The key findings of the review are summarized below. Our tool is designed to specifically address the second, third, and fourth points of these findings.

- To assess the effectiveness of matching algorithms, metrics such as precision, recall, and F-score - commonly used in pattern recognition - are employed to measure matching accuracy. These same metrics can also be applied to evaluate the accuracy of conflict detection and resolution. Additionally, performance and scalability are important considerations, although large-scale models often receive less attention in this regard.

- Another significant challenge in model matching is the absence of automated benchmarking. A pressing need exists for the development of technology-independent benchmarking model datasets that can be automatically generated, thereby addressing this longstanding issue in the field.

- The field of model matching suffers from a scarcity of empirical studies, which are essential for verifying certain aspects of an algorithm, such as the human effort required to use it, that would be challenging to quantify otherwise.

- Another significant limitation is the lack of technology-independent evaluation frameworks for model matching algorithms. While some evaluation frameworks do exist, they are often tied to specific technologies, which can hinder their broader applicability.

Somogyi [22] enlists open questions that open doors for future research. Some of the questions that are particularly helpful for us to set the direction for our project are as follows.

- **Absence of a mature tool**: Most of the model matching algorithms have a working prototype but the existence of mature tools is rare.
- **Semantic Model Matching**: The independent semantic matching of models as well as semantic matching informed from syntactic differences is a new research direction.
- **Large-scale model matching**: There is room for research when it comes to optimization of matching algorithms with minimal compromise on accuracy.

Based on the literature review done by the author in 2019, they have proposed a text-based model differencing and merging algorithm [23]. Since, it is an MDM (Model Differencing and Merging) algorithm, it consists of two fundamental phases; model matching phase, and model merging phase. The AST matching phase involves pairing subtrees that represent the same elements in Abstract Syntax Trees (ASTs) of two models. This process uses the "same-level heuristic," prioritizing subtree matches on the same level before moving to deeper levels, as matches are commonly found at the same level. The algorithm takes two ASTs as input and produces matched pairs (MP) and unmatched subtrees. It uses the "IS MATCH" operation, which is considered to be an external operation that could utilize any matching technique (similarity based, static matching, etc), to determine if subtrees are pairs. The algorithm first checks if the entire ASTs match; if not, it terminates. It then recursively matches children subtrees, avoiding retries of previously tried pairs to enhance performance. The author also describes a "MATCH ASTS" operation, which attempts to match children subtrees on the same level, using the IS MATCH operation. For optimal performance, the children that have already been matched are stored in a cache to avoid redundant matching. After completing MATCH ASTS, the algorithm gathers all unmatched subtrees and attempts further matching among them. Matched pairs can be labeled, allowing AST traversal in linear time using methods like Depth-First Search (DFS).

The algorithm proposed by Somogyi [23] exists as a proof of concept and the paper mentions that there is a need for the development of a mature tool that implements the algorithm.

In the light of the literature review conducted by Somogyi [22], Xiao He [12] enhanced the comparison mechanism of EMF-Compare by introducing the idea of hashing.

EMF Compare is a state-of-the-art generic model merging tool based on the Eclipse Modeling Framework (EMF). Its workflow comprises four major steps:

1. **Model Resolving**: Loading models from files.
2. **Match**: Computing match results.
3. **Diff**: Differencing models based on match results.
4. **Post-process**: Refining the comparison.

EMF-Compare compares model elements using a 2-step process. Firstly, potential candidates for a model element are compared by finding the edit distance between model elements. A set threshold is used to compute candidate pairs. The second step is the double-check stage where the model element that is closest to the base element qualifies as a match.

Xiao He [12] has mentioned two types of hashing; Similarity-preserving hashing, and Integrity-based hashing.

Similarity-preserving hashing maps data to numerical hash values. Examples of Similarity-preserving hashing are SimHash [19] and MinHash [7]. These generated hashes of two models can be compared using Cosine Similarity, Jaccard Similarity, and Sorensen Similarity.

Integrity-based hashing refers to techniques including cryptographic hashing algorithms. This hashing approach is used in cases where there is a high chance of hash collisions, which is rare in the case of model comparison.

Xiao He has used a combination of these two types of hashing approaches to propose a dual-hashing-based model matching algorithm; an extension of

EMF-C. "HMo", addressed as the similarity-preserving hashing algorithm; iterates through the model to get the model elements and computes the hash of individual features. The model element's hash is weighted sum of the individual 64-bit hashes of each feature of the model element. The mechanism to compute a hash for features that represent a name is to compute the hash of individual NGrams of the feature. "Hchk" has been labelled as the Integrity-based hashing algorithm; it is used for the computation of of the checksum of a collective string of the features.

To briefly put the entire algorithm in picture, Xiao He [12] selects a query element "Q" for which a match is to be found. The candidtes are filtered out using the S-hashes (computed using HMo). The filtered candidates are double checked to find the closest match for each query element. In-order to speed up the algorithm; the distances are cached corresponding to the "C-Hashes" (Compute using Hchk) of the element to avoid redundant computations.

As a result of introducing hashing to avoid the tedious task of comparing raw elements, [12] was able to succeed in surpassing the performance benchmark set by EMF compare. As a part of future improvements, the author has suggested using semantic word embeddings; such as word2vec in place of "hashNGram" function used in the computation of HMo.

While several approaches exist for syntactic model comparison, a common challenge is the lack of granularity in the comparison process. Existing techniques often provide limited control over the level of detail considered during comparison, leading to either oversimplified or overly complex results.

2.4 Review of Evaluation Techniques for Model Comparison Algorithms

Another challenge in field of model comparison is the evaluation of the algorithms used. There are existing works that explore this topic, and suggest possible methods of evaluation of model comparison approaches.

Lorenzo [1] talks about evaluation of existing model comparison tools. The author points out the fact that existing literature contains a multitude of model matching algorithms but there is an absence of systematic evaluation. To address this issue, the author has proposed a way to evaluate model comparison algorithms in a systematic fashion. This is achieved by creating model comparison benchmarks. This is achieved using the proposed tool in the paper called Benji[14]; that allows for the specification of the preconditions, actions, and post conditions to produce mutants of a model.

Xiao [12] has developed a mutator as well, to create mutants of model elements that can be used to evaluate model comparison tool.

Brand [25] has suggested a method for assessing the quality of model comparison tools by using model generation to produce benchmark data. They have performed evaluation of pairs generated by mutators for model elements individually which could prove to be challenging when specifying benchmark datasets for evaluation. Additionally, their evaluation is restricted to attributes and references. They have curated a benchmark dataset by amalgamation of manually defined data, and generated data using model mutator. The manually defined dataset has been created by developers who manually identified the differences between models. This data set could have been valuable but it does not exist at the given URL[1] in the paper anymore.

There are several other eclipse-based evaluation tools [[11]] [24]]. The takeaway from the approaches described in these evaluation tools is that we can use precision, recall, and F-Score as metrics for the evaluation.

[1]http: //www.win.tue.nl/ zprotic/benchmark.html

Chapter 3

Objectives

In the light of the literature review; we have identified the following key objectives for the model comparison tool:

- API endpoints that returns metrics obtained as a result of syntactic and semantic model comparison: The APIs will be dockerized and can be plugged into any system; thus achieving the goal of a technology independent tool for model comparison.

- A configuration file to specify the granularity of comparison for syntactic similarity. The configuration will allow the user to specify the elements, as well as their features that merit comparison for the particular usecase.

- Model Conversion from Ecore to Emfatic, Emfatic to Ecore, and UML2 to Ecore to provide interoperability between these 3 modelling languages.

- A benchmark dataset constructed using the model transformation tool [6]. The dataset will contain triples (model1, model2, syntactic metrics), that will help us gauge the accuracy of the model comparison tool.

- Generate visualizations for the metrics obtained after comparison: Separate visualizations for comparison of a single pair of models, as well as bulk of pairs.

Chapter 4

System Specification

This chapter describes in detail the working of the model comparison tool. We begin with the discussion of the architecture; broken down into usecases in 4.1, Component Diagram in 4.2, and Deployment Diagram in 4.3, followed by sequence diagrams in 4.4 that capture the flow of information in the system. Later, we present a class diagram representing the structure of the key components in 4.5. At the end of this chapter, we describe the choice of technology, steps to deploy the project, and run it locally for development purposes.

4.1 Use Cases

Figure 4.1 shows the usecases of the comparit user inteface. The description of the usecases is as follows:

- **Compare a single pair of Ecore/Emfatic/UML2 models**: The user should be able to input a pair of valid models to be compared. As an extension of this usecase, the user should be able to specify the granularity of comparison. For example, if the user wants to compare two ecore models such that they want to match elements based on a defined subset of features (name and type); the user should be able to specify the granularity.

- **View Comparison Results**: The user should be able to see detailed results of comparison including precision, recall, f1-score of the resulting comparison.

- **Bulk Comparison**: The user should be able to input a zip folder containing pairs of ecore/emf/uml models and get aggregate as well as individual comparison results for the uploaded models.
- **Ecore to Emfatic**: The user should be able to specify a valid ecore model and get the corresponding emfatic model. While this usecase is not directly related with model comparison, this helps facilitate analysis of models.
- **Emfatic to Ecore**: The user should be able to specify a valid emfatic model and get the corresponding ecore model.
- **UML2 to Ecore**: The user should be able to specify a valid UML2 model and get the corresponding ecore model.

Figure 4.2 depicts how the user could interact with the API of the tool. The description of the usecases is as follows:

- **Make a POST Request to syntactic Comparator**: The user should be able to send a pair of valid ecore/emfatic/uml models along with a configuration file as a POST request to the syntactic comparator endpoint and obtain comparison results.
- **Make a POST Request to Semantic Comparator**: The user should be able to send a pair of valid ecore/emfatic/uml models as a POST request to the semantic comparator endpoint and obtain comparison results.
- **Make a POST Request to get Emfatic from Ecore**: The user should be able to attach a valid emfatic model in a POST Request to the API endpoint and get the corresponding Ecore model.
- **Make a POST Request to get UML2 from Ecore**: The user should be able to attach a valid UML2 model in a POST Request to the API endpoint and get the corresponding Ecore model.
- **Make a POST Request to get Ecore from Emfatic**: The user should be able to attach a valid ecore model in a POST Request to the API endpoint and get the corresponding Emfatic model.

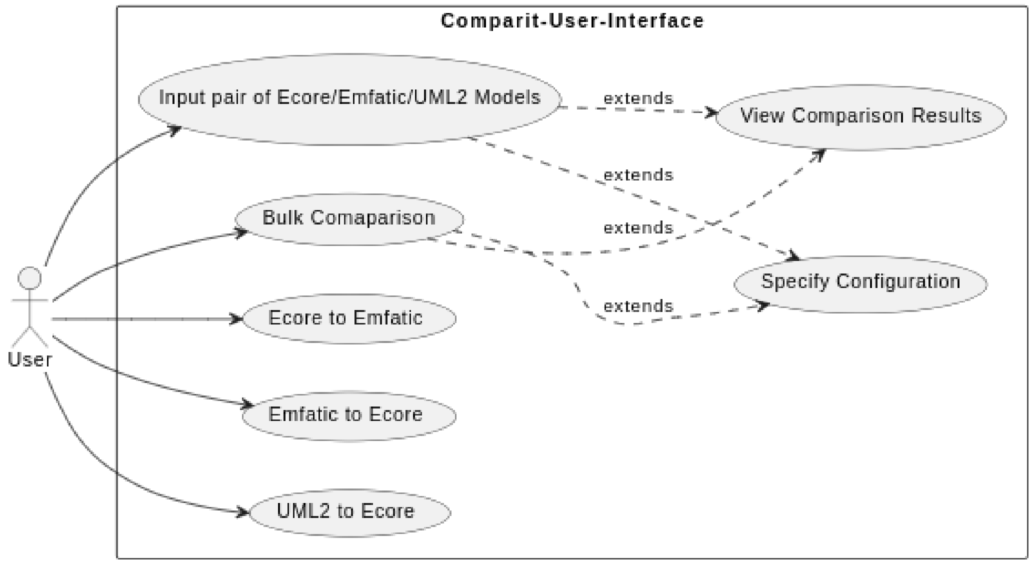


Figure 4.1: User Interface Usecase

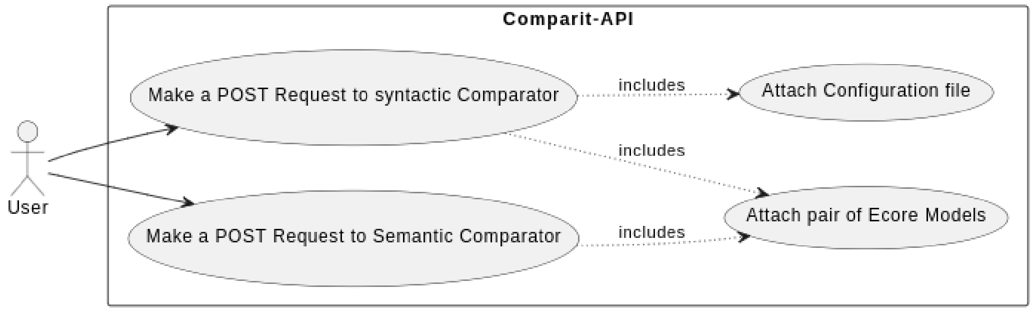

Comparit-API
Make a POST Request to syntactic Comparator
Attach Configuration file
Make a POST Request to Semantic Comparator
Attach pair of Ecore Models
includes
includes
includes
User

Figure 4.2: Api Usecase

4.2 Component Diagram

This section describes the components of our tool that help fullfil the requirements specified in the usecases. The component diagram 4.3 identifies 3 main components that could be used by a "Usage Script"; a script making direct API calls to the endpoints, or a user interface.

- Semantic-comparator: Compares two emfatic models semantically.

- Syntactic-comparator: Compares two ecore models syntactically.
- Model-convertor: Allows for conversion of models to allow inter-operable behavior.

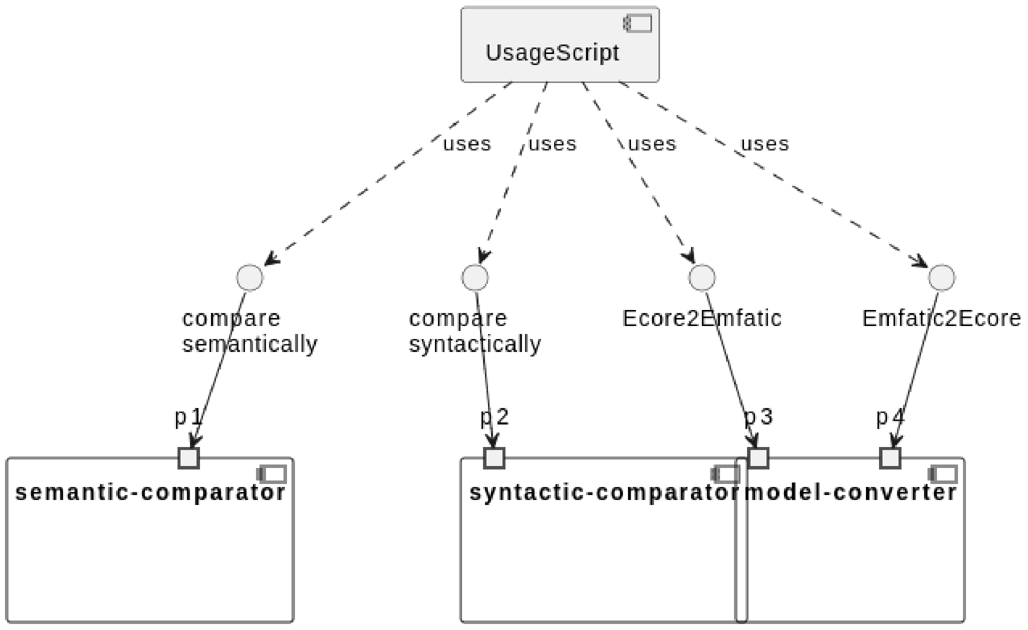


Figure 4.3: Component Diagram

4.3 Deployment Diagram

The deployment diagram 4.4 showcases how the components have been packaged. The ports contained in component diagrams for model-convertor and **syntactic comparator** have been implemented as API-endpoints in a spring-boot service. The port that allows communication with the **semantic comparator** has been implemented as an API endpoint in a Flask service. The two services have been dockerized. Steps to obtain the docker images and run them locally can be found in 4.7. To run the tool in production environments where scalability and high availability is improtant, we recommend deploying the containers inside pods in a kubernetes cluster. In-order to accelerate the process of comparison; we have segregated the semantic copmarator and syntactic comparator; allowing for parallel computation of the two versions of similarity.

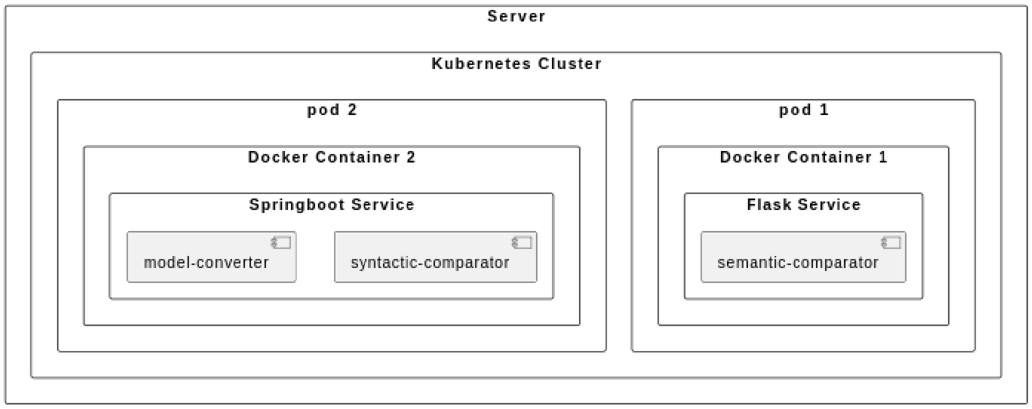


Figure 4.4: Deployment Diagram

4.4 Sequence Diagrams

The sequence diagrams in this section depict the flow of information between the user and the system.

Figure 4.5 describes the flow of events when the user inputs a pair of ecore models, they are sent to model-converter to obtain the corresponding emfatic models. The emfatic models are sent to the semantic-comparator while the ecore models are sent to the syntactic comparator. The results are then combined and returned to the user with the relevant visualizations displayed on the user interface.

As described in figure 4.6, when the user inputs a pair of emfatic models, they are sent to model-converter to obtain the corresponding ecore models. The emfatic models are sent to the semantic-comparator while the ecore models are sent to the syntactic comparator. The results are then combined and returned to the user with the relevant visualizations displayed on the user interface.

As described in figure 4.7, when the user inputs a pair of uml models, they are sent to model-converter to obtain the corresponding ecore models. The obtained ecore models are sent to model-convertor to obtain the corresponding emfatic models. After this, the emfatic models are sent to the semantic-comparator while the ecore models are sent to the syntactic comparator. The results are then combined and returned to the user with the relevant visualizations displayed on the user interface.

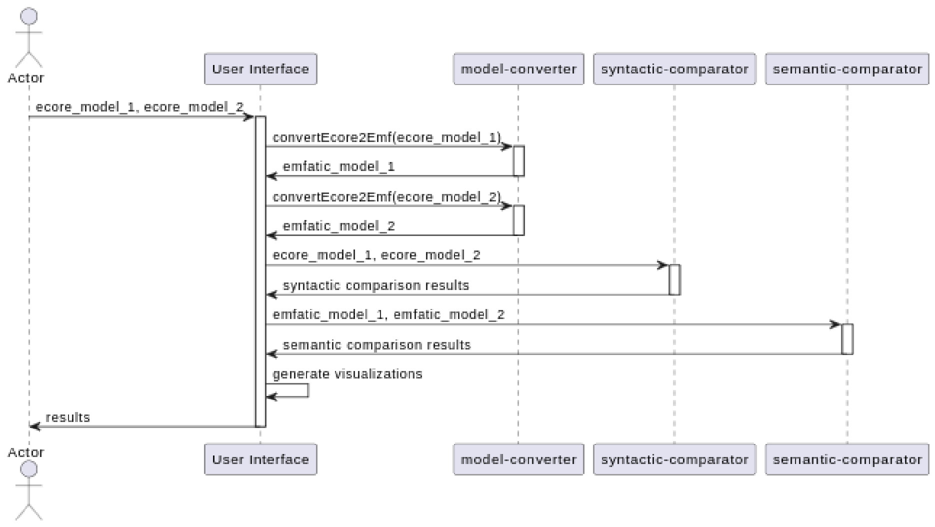

Actor
User Interface
model-converter
syntactic-comparator
semantic-comparator
ecore_model_1, ecore_model_2
convertEcore2Emf(ecore_model_1)
emfatic_model_1
convertEcore2Emf(ecore_model_2)
emfatic_model_2
ecore_model_1, ecore_model_2
syntactic comparison results
emfatic_model_1, emfatic_model_2
semantic comparison results
generate visualizations
results

Figure 4.5: Sequence Diagram Ecore Model Comparison

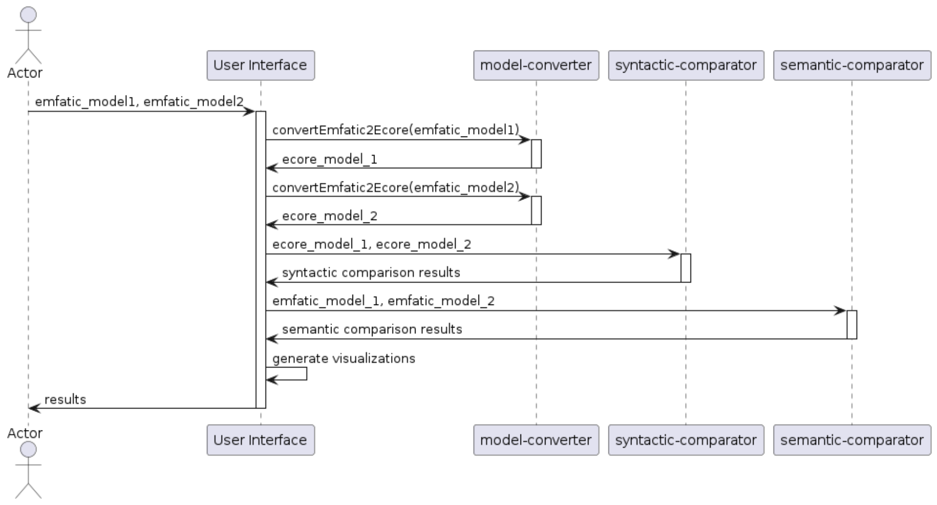

Actor
User Interface
model-converter
syntactic-comparator
semantic-comparator
emfatic_model1, emfatic_model2
convertEmfatic2Ecore(emfatic_model1)
ecore_model_1
convertEmfatic2Ecore(emfatic_model2)
ecore_model_2
ecore_model_1, ecore_model_2
syntactic comparison results
emfatic_model_1, emfatic_model_2
semantic comparison results
generate visualizations
results

Figure 4.6: Sequence Diagram Emfatic Model Comparison

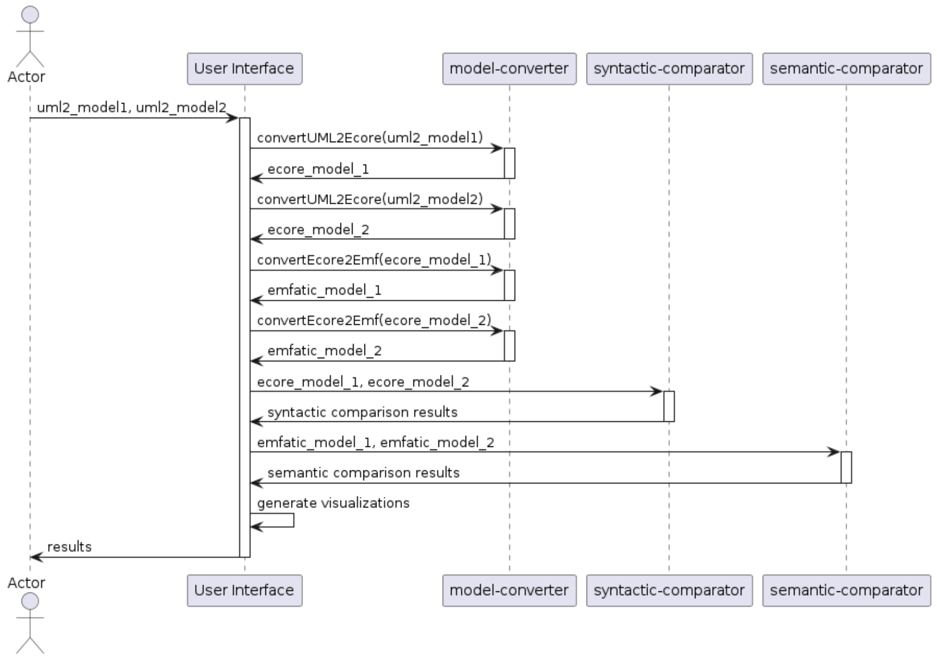

Actor
User Interface
model-converter
syntactic-comparator
semantic-comparator
uml2_model1, uml2_model2
convertUML2Ecore(uml2_model1)
ecore_model_1
convertUML2Ecore(uml2_model2)
ecore_model_2
convertEcore2Emf(ecore_model_1)
emfatic_model_1
convertEcore2Emf(ecore_model_2)
emfatic_model_2
ecore_model_1, ecore_model_2
syntactic comparison results
emfatic_model_1, emfatic_model_2
semantic comparison results
generate visualizations
results

Figure 4.7: Sequence Diagram UML2 Model Comparison

4.5 Understanding Codebase: Class Diagrams and Directory Structure

In this section, we present the class diagrams for the springboot and flask service. The class diagrams capture the architecture of the two services; providing insight into how different components of the code work together. Additionally, the diagrams depict the design patterns and best practices implemented in the codebase. This section is particularly directed to researchers and developers who are interested in contributing to this tool.

The following tree captures the directory structure of ”src” folder of the springboot service (syntactic-comparator). We have followed the controller-service-repository pattern to streamline data flow across the application from the api to services. We do not have repositories because we do not maintain user’s state.

```
src/
└── main/
    └── java/
└── com/
    └── mdre/
        └── evaluation/
            ├── config/
            │   └── Constants.java
            ├── controller/
            │   ├── EcoreToEmfaticController.java
            │   ├── EmfaticToEcoreController.java
            │   ├── HeartbeatController.java
            │   ├── ModelComparisonController.java
            │   └── UML2ToEcoreController.java
            ├── dtos/
            │   ├── DigestConfigurationDTO.java
            │   ├── HashingConfigurationDTO.java
            │   ├── MatchedElementsDTO.java
            │   ├── ModelComparisonConfigurationDTO.java
            │   ├── MutantCreationDTO.java
            │   └── VenDiagramDTO.java
            ├── schemas/
            │   └── configuration.json
            ├── services/
            │   ├── modelComparisonService/
            │   │   ├── AbstractClassComparisonService.java
            │   │   ├── DigestService.java
            │   │   ├── EMFCompareService.java
            │   │   ├── HashingService.java
            │   │   ├── MetricsComputationService.java
            │   │   ├── ModelComparisonService.java
            │   │   ├── ModelElementsFetcher.java
            │   │   ├── ModelMutator.groovy
            │   │   ├── ModelMutatorJavaWrapper.java
            │   │   ├── YamlModelComparator.groovy
            │   │   └── YamlModelComparatorWrapper.java
            │   ├── EcoreToEmfaticService.java
            │   ├── EmfaticToEcoreService.java
            │   ├── JsonSchemaValidatorService.java
            │   └── UML2ToEcoreService.java
            └── utils/
```

```
├── FileUtils.java
├── JSONtoDTOMapper.java
├── ModelComparisonUtils.java
├── Tuple.java
└── App.java
```

The entrypoint of the application is in src/App.java file. The springoot application is loaded alongwith the controllers. The controllers contain the implementation of endpoints defined in the component diagram in section 4.2. The controllers send the the request to the corresponding service. The naming convention for controllers and services is such that it is intuitive for programmers to infer the service called by a controller by looking at the directory structure. For example, the ModelComparisonController calls the ModelComparisonService and the EmfaticToEcoreController calls the EmfaticToEcoreService.

The schema defined in the schemas folder is enforced on the configuration file for granularity of comparison sent via the API. This is to ensure that the API correctly hanldes cases when configuration files having unexpected syntax are provided. The configuration schema is described in appendix table 9.1

The class diagram 4.8 represents the relationships, attributes and operations of classes within the modelComparisonService package. To remain concise in our explanation, we will only explain the ModelComparisonService in the class diagram because this package contains the classes that define the algorithms for model comparison. Later in Chapter 6, we will discuss the implementation of the algorithms in detail.

The AbstractClassComparisonService class provides a blueprint for the implementation of a model comparison algorithm. It can be extended to provide implementations of different algorithms. For this tool, we have implemented two algorithms; hashing based algorithm (explained in section 6.1) and digest based algorithm (explained in section 6.2).

The ModelComparisonService class uses the algorithm based on the specified configuration; the configuration file comes from the controller. It uses ModelElementsFetcher to fetch model elements. ModelElementsFetcher uses YAMTLCounter; that extends from an external dependency "YAMTL Module" which is a powerful model transformation and manipulation tool. Apart from using it for creating mutants, we use YAMTL for fetching elements as well because it provides a convinient API to fetch all elements on model level.

The ModelMutator class uses Yamtl Module to create mutants. Section

8.2 explains how the model mutator is used to create a benchmark dataset.

The following tree captures the directory structure of the semantic-comparator. Similar to the syntactic-comparator we have followed the controller-service-repository pattern to streamline data flow across the application from the api to services.

```
│ archive/
│ ├── glove.6B/
│ │   └── glove.6B.50d.txt
│ └── GoogleNews-vectors-negative300.bin
├── controllers/
│   └── semantic_similarity_controller.py
├── services/
│   └── semantic_similarity_services.py
├── app.py
└── wsgi.py
```

The class diagram in figure 4.9 represents the relationships, attributes, and operations in the semantic-comparator service. Depending on the configuration of the algorithm at the time of starting the service, the embedding model is chosen for converting emfatic models to vector embeddings. These vector embeddings are used to compute cosine similarity betweeen the two models.

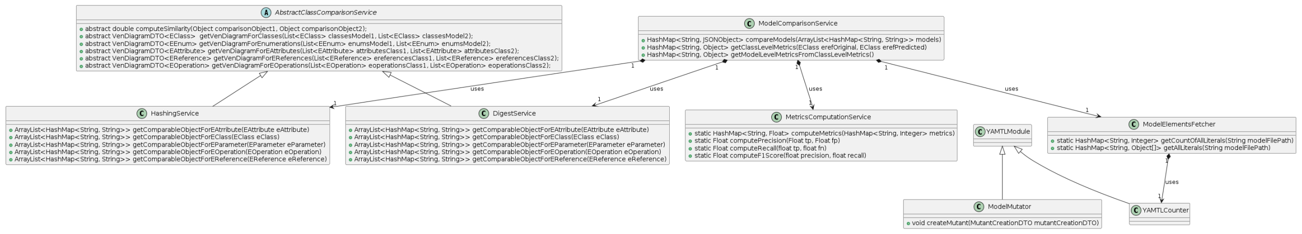

AbstractClassComparisonService
+ abstract double computeSimilarity(Object comparisonObject1, Object comparisonObject2);
+ abstract VenDiagramDTO<EClass> getVenDiagramForClasses(List<EClass> classesModel1, List<EClass> classesModel2);
+ abstract VenDiagramDTO<EEnum> getVenDiagramForEnumerations(List<EEnum> enumsModel1, List<EEnum> enumsModel2);
+ abstract VenDiagramDTO<EAttribute> getVenDiagramForEAttributes(List<EAttribute> attributesClass1, List<EAttribute> attributesClass2);
+ abstract VenDiagramDTO<EReference> getVenDiagramForEReferences(List<EReference> ereferencesClass1, List<EReference> ereferencesClass2);
+ abstract VenDiagramDTO<EOperation> getVenDiagramForEOperations(List<EOperation> eoperationsClass1, List<EOperation> eoperationsClass2);
ModelComparisonService
+ HashMap<String, JSONObject> compareModels(ArrayList<HashMap<String, String>> models)
+ HashMap<String, Object> getClassLevelMetrics(EClass erefOriginal, EClass erefPredicted)
+ HashMap<String, Object> getModelLevelMetricsFromClassLevelMetrics()
uses
1
HashingService
+ ArrayList<HashMap<String, String>> getComparableObjectForEAtrribute(EAttribute eAttribute)
+ ArrayList<HashMap<String, String>> getComparableObjectForEClass(EClass eClass)
+ ArrayList<HashMap<String, String>> getComparableObjectForEParameter(EParameter eParameter)
+ ArrayList<HashMap<String, String>> getComparableObjectForEOperation(EOperation eOperation)
+ ArrayList<HashMap<String, String>> getComparableObjectForEReference(EReference eReference)
DigestService
+ ArrayList<HashMap<String, String>> getComparableObjectForEAtrribute(EAttribute eAttribute)
+ ArrayList<HashMap<String, String>> getComparableObjectForEClass(EClass eClass)
+ ArrayList<HashMap<String, String>> getComparableObjectForEParameter(EParameter eParameter)
+ ArrayList<HashMap<String, String>> getComparableObjectForEOperation(EOperation eOperation)
+ ArrayList<HashMap<String, String>> getComparableObjectForEReference(EReference eReference)
MetricsComputationService
+ static HashMap<String, Float> computeMetrics(HashMap<String, Integer> metrics)
+ static Float computePrecision(Float tp, Float fp)
+ static Float computeRecall(float tp, float fn)
+ static Float computeF1Score(float precision, float recall)
YAMTLModule
ModelElementsFetcher
+ static HashMap<String, Integer> getCountOfAllLiterals(String modelFilePath)
+ static HashMap<String, Object[]> getAllLiterals(String modelFilePath)
ModelMutator
+ void createMutant(MutantCreationDTO mutantCreationDTO)
YAMTLCounter

Figure 4.8: Class Diagram for Springboot Service

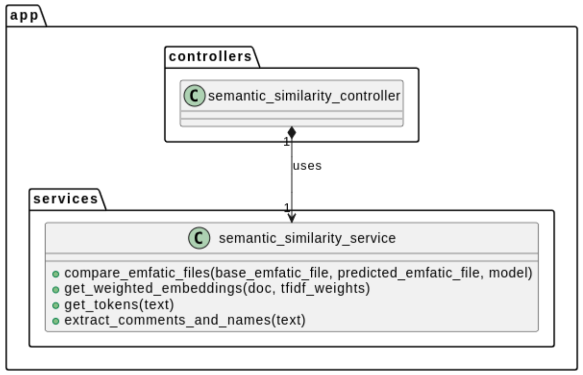

app
controllers
semantic_similarity_controller
1
uses
1
services
semantic_similarity_service
+ compare_emfatic_files(base_emfatic_file, predicted_emfatic_file, model)
+ get_weighted_embeddings(doc, tfidf_weights)
+ get_tokens(text)
+ extract_comments_and_names(text)

Figure 4.9: Class Diagram for Flask Service

4.6 Summary of Technologies Used

The tool has been developed using an amalgam of technologies. We have used Java, Groovy, and Python as the primary languages. Java and Groovy have been used interchangeably to experiment with existing technologies and to manipulate Ecore models in particular because of the rich EMF API that is available as a Gradle/Maven dependency. Python has been used for rare situations where the Java-based libraries shall not suffice. Each service that is built as a part of this tool has been packaged as a microservice; Spring Boot for Java/Groovy, and Flask for Python. These services are containerized so they can be plugged into systems and readily utilized.

Diving deeper into some of the libraries that we have used:

- **YAMTL** [6]: A tool that comes packed with packages allowing us to perform powerful transformations to obtain meaningful results. We have used YAMTL to generate mutants for a vetted dataset of ecore models to create a test dataset for evaluation. The evaluation chapter expands on the creation of test data.
- **Java UML2** [8]: Library used to convert UML2 models to ecore.
- **EMF API**: to traverse and manipulate Ecore models, convert Ecore models to Emfatic and Emfatic to Ecore.
- **Spring Boot**: To develop Java-based services.
- **Flask**: To develop Python-based services.
- **Streamlit**: to develop the web interface.

4.7 Steps to Run

The services developed as a part of the tool have been packaged as docker images and made public on docker hub. Table 4.1 describes the docker images. In this section, we will specify the system requirements, and the steps to run the tool using docker-compose.

Image Name	Tag	Description
jawad571/comparit-syntactic	1.0.0	Syntactic Comparator
jawad571/comparit-semantic	1.0.0	Semantic Comparator
jawad571/comparit-user-interface	1.0.0	User Interface

Table 4.1: Docker Images

4.7.1 System Requirements

- OS: Linux
- Memory: 2GB Minimum (4GB Recommended)
- Storage: 10GB
- Python version: 3.8
- docker-compose version: v2.17.2
- docker version: 27.1.1

4.7.2 Steps to run using docker compose

The docker-compose file's contents listed below. If your system does not have atleast 4GB of spare RAM, then we recommend setting the "EMBEDDING_MODEL" environment variable for "comparit-semantic" container to "glove". Glove is a light weight embedding model compared to the Google News embedding model. Further details regarding the embedding models are explained later in Chapter 6.

- Create a docker-compose file with the following contents

```
version: '3.8'

services:
  comparit-syntactic:
    image: jawad571/comparit-syntactic:1.0.0
    ports:
      - "8080:8080"

  comparit-semantic:
    image: jawad571/comparit-semantic:1.0.0
    ports:
      - "9090:9090"
    environment:
      - EMBEDDING_MODEL=gnews

  user-interface:
    image: jawad571/comparit-user-interface:1.0.0
    ports:
      - "8501:8501"
    environment:
      - COMPARIT_SYNTACTIC_URL=http://comparit-syntactic:8080
      - COMPARIT_SEMANTIC_URL=http://comparit-semantic:9090
```

- Run "docker-compose up –build –pull always"
- Access the user-interface in your browser on "localhost:8501"

4.7.3 Steps to run in Development Environment

To ensure that this project is easily extensible, we provide the details of running this project in the development environment. The following steps demonstrate how to run this project in development environment on a linux system.

- Clone the repository [1]. Note: This repository will be made public after the dissertation has been graded.
- Enter command: "cd syntactic-similarity"
- Enter command: "./gradlew bootrun". This will start the springboot service on port 8080.

[1]https://github.com/jawad571/model-comparator

- Open new terminal and navigate to root directory of the project, then enter command: "cd semantic-simiarlity". Create a folder named "archive" inside this directory. Download the google news word2vec embedding model (GoogleNews-vectors-negative300.bin)[2] in the archive folder. Create a sub directory inside the archive folder and name it "glove.6B". Download the glove.6B.50d.txt embedding [3] in this folder. You can specify which embedding model you want to use in the .env file with the variable name "EMBEDDING_MODEL"; that could hold two values "gnews" and "glove".

- Enter command: "python3 -m venv env" to create a new virtual environment

- Enter command: "source env/bin/activate" to activate the environment

- Enter command: "pip install -r requirements.txt" to install the libraries and then enter "python app.py". This will start the flask based service on port 9090.

- Open new terminal and navigate to root directory of the project, then enter command: "cd user-interface".

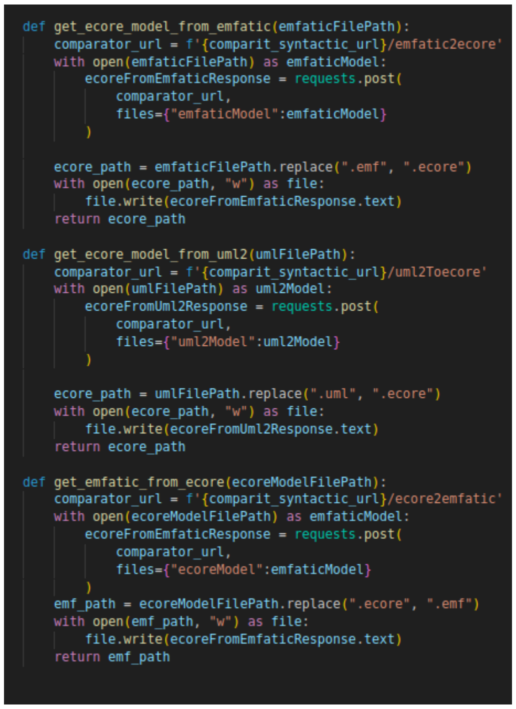
- Repeat steps 5-7, and run "streamlit run main.py". This will run the user interface.

As a result of performing these steps, you should have 3 terminal windows running syntactic-comparator on port 8080, semantic-comparator on port 9090, and user-interface on port 8501.

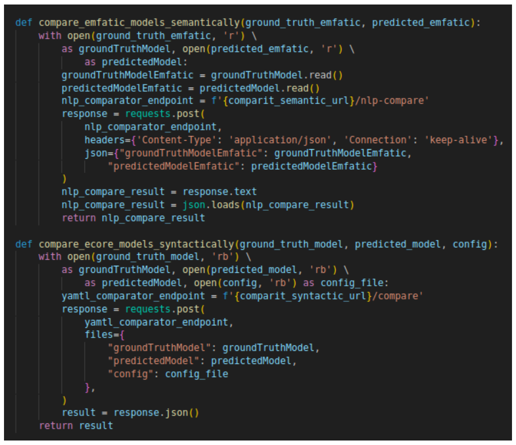
4.8 Python Adapter to use the API

We have written an adapter as a part of the user-interface that makes API calls the springboot and flask services. Figure 4.10 showcases 5 functions; one for each api defined in the component diagram. The adapter is decoupled from the user interface and can be plugged into any software; and its functions can be imported and readily invoked.

[2]https://github.com/mmihaltz/word2vec-GoogleNews-vectors?tab=readme-ov-file
[3]https://nlp.stanford.edu/projects/glove/



```python
def get_ecore_model_from_emfatic(emfaticFilePath):
    comparator_url = f'{comparit_syntactic_url}/emfatic2ecore'
    with open(emfaticFilePath) as emfaticModel:
        ecoreFromEmfaticResponse = requests.post(
            comparator_url,
            files={"emfaticModel":emfaticModel}
        )

    ecore_path = emfaticFilePath.replace(".emf", ".ecore")
    with open(ecore_path, "w") as file:
        file.write(ecoreFromEmfaticResponse.text)
    return ecore_path

def get_ecore_model_from_uml2(umlFilePath):
    comparator_url = f'{comparit_syntactic_url}/uml2Toecore'
    with open(umlFilePath) as uml2Model:
        ecoreFromUml2Response = requests.post(
            comparator_url,
            files={"uml2Model":uml2Model}
        )

    ecore_path = umlFilePath.replace(".uml", ".ecore")
    with open(ecore_path, "w") as file:
        file.write(ecoreFromUml2Response.text)
    return ecore_path

def get_emfatic_from_ecore(ecoreModelFilePath):
    comparator_url = f'{comparit_syntactic_url}/ecore2emfatic'
    with open(ecoreModelFilePath) as emfaticModel:
        ecoreFromEmfaticResponse = requests.post(
            comparator_url,
            files={"ecoreModel":emfaticModel}
        )
    emf_path = ecoreModelFilePath.replace(".ecore", ".emf")
    with open(emf_path, "w") as file:
        file.write(ecoreFromEmfaticResponse.text)
    return emf_path
```



```python
def compare_emfatic_models_semantically(ground_truth_emfatic, predicted_emfatic):
    with open(ground_truth_emfatic, 'r') \
        as groundTruthModel, open(predicted_emfatic, 'r') \
            as predictedModel:
        groundTruthModelEmfatic = groundTruthModel.read()
        predictedModelEmfatic = predictedModel.read()
        nlp_comparator_endpoint = f'{comparit_semantic_url}/nlp-compare'
        response = requests.post(
            nlp_comparator_endpoint,
            headers={'Content-Type': 'application/json', 'Connection': 'keep-alive'},
            json={"groundTruthModelEmfatic": groundTruthModelEmfatic,
                "predictedModelEmfatic": predictedModelEmfatic}
        )
        nlp_compare_result = response.text
        nlp_compare_result = json.loads(nlp_compare_result)
        return nlp_compare_result

def compare_ecore_models_syntactically(ground_truth_model, predicted_model, config):
    with open(ground_truth_model, 'rb') \
        as groundTruthModel, open(predicted_model, 'rb') \
            as predictedModel, open(config, 'rb') as config_file:
        yamtl_comparator_endpoint = f'{comparit_syntactic_url}/compare'
        response = requests.post(
            yamtl_comparator_endpoint,
            files={
                "groundTruthModel": groundTruthModel,
                "predictedModel": predictedModel,
                "config": config_file
            },
        )
        result = response.json()
    return result
```

Figure 4.10: Python Adapter

Chapter 5

User Interface

In this chapter we describe the features of the interface.

5.1 Home

The homepage in appendix figure 9.1 features a drop-down menu that allows the user to select from the various options and features that this tool offers.

5.2 Compare Models

In the drop-down, as show in appendix figure 9.2, there are options to compare Ecore, Emfatic, and UML models. Upon the selection of any of these options,the user can specify the models to be compared and select a configuration from the configuration panel as show in the appendix figure 9.3 that displays the Ui as a result of selecting the "Compare Ecore" option. A similar user-interface can be seen for Emfatic and Ecore model comparison as well. For ease of testing, sample models are provided. The ground truth model and the predicted model are displayed side by side on the screen. Pressing the "Compare" button allows you to visualize the model-level metrics which indicate how closely the predicted model matches the ground truth model. A further breakdown of class-level metrics is also provided to analyze the model's performance in greater detail. This includes a comparison of predicted classes, operations, attributes, references, and supertypes as can be seen in appenfix figure 9.4.

5.3 Bulk Comparison

This option can be chosen from the dropdown on the home page and allows for the comparison of multiple model pairs simultaneously. The user interface can be seen in appenfix figure 9.5. You can upload folders or select the provided sample. Each folder should contain a base model and a predicted model, which will then be compared. The type of base and predicted model can be chosen according to your preferences from the available options (Ecore, Emfatic, and UML2). The sample folder, downloaded as a zip file, can be uploaded to test the tool.

After clicking the "Compare" button, you can visualize metrics to assess how well the predicted model performs relative to the ground truth. The aggregate visualization section provides overall metrics for the bulk comparison. Additionally, a CSV file is provided for inspecting specific model pairs. From a dropdown menu in the individual pair visualization section, you can select the specific pair you want to inspect, and metrics will be available for that pair, similar to the previous individual model comparison options.

5.4 Model Conversion

The tool allows the user to compare models. The user can perform Ecore to Emfatic, Emfatic to Ecore, and UML2 to Ecore conversion. Appendix figure 9.7 shows how the ecore to emfatic conversion user interface looks like. A similar interface is displayed for the other options as well. It should be noted that the model to be converted must be syntactically accurate, or the conversion may fail.

Chapter 6

Algorithms

In this chapter, we will abstractly define the algorithms that have been used for syntactic and semantic comparison. Highlights of their implentation have been showcased in form of screenshots in the appendix of this report in figures 9.8, 9.9, and 9.10. These figures display the function signatures and first few lines; the entire code can be viewed from source code.

The tool provides an an abstract java class that could be extended to implement an algorithm of choice. The instance of the algorithm of choice is created and used to compare models as represented abstractly in Algorithm 1.

The compareModels function expects a hashmap (config) that contains information about the granularity of comparison and choice of algorithm. The hashmap can contain the configuration variables lisetd in table 9.1. Based on the choice of algorithm, if implemented in the tool, the relevant instance of the class is initialized; that instance is named as "alg". The function "getClassLevelMetrics" is called that uses the instance of the algorithm to compute the ven diagram for each of the elements of the model. The ven diagram provides information about the true positives, false positives, and false negatives. if the MODEL-LEVEL-COMPARISON-DERIVED-FROM-CLASS-LEVEL-COMPARISON [1] configuration variable (refer to appendix table 9.1) is set to True then the classLevelMetrics and the Ven Diagram for enumerations (computed separately because they are present at the root of the model instead of being part of a class), are used to generate the the model level metrics; else, the model level metrics are generated by comparing

[1]The configuration variable is used as a short form "aggregateClassLevelMetrics" in Algorithm 1 due to space limitations

model elements on model level. We have not described the functions "get-ConfusionMatrixForAllElements()" and "getMLM(clm, VDEnum")because they are trivial.

Algorithm 1 Compare Models using Algorithm of Choice

$conf$:	configuration	bM:	baseModel
pM:	predictedModel	alg:	Algorithm of choice
VD:	Ven Diagram	Att:	attribute
Op:	operation	ref:	reference
Sup:	supertype	$enum$:	enumeration
CLM:	classLevelMetrics	MLM:	modelLevelMetrics

```
function COMPAREMODELS(conf, bM, pM)
    $alg \leftarrow conf.alg$                ▷ of Type AbstractClassComparisonService
    $VDEnum \leftarrow alg.getVDEnum(bM.Enum, pM.Enum)$
    $clm \leftarrow getCLM(bM.Classes, pM.Classes, alg, conf)$
    if conf.aggregateClassLevelMetrics is True then
        $mlm \leftarrow getMLM(clm, VDEnum)$
    else
        $mlm \leftarrow getMLM(bM, pM)$
    end if
end function

function GETCLM(bc, pc, alg, conf)
    $VDAtt \leftarrow alg.getVDAtt(bc.Att, pc.Att)$ if conf.Att is true
    $VDOp \leftarrow alg.getVDOp(bc.Op, pc.Op)$ if conf.Op is true
    $VDRef \leftarrow alg.getVDRef(bc.Ref, pc.Ref)$ if conf.Ref is true
    $VDSup \leftarrow alg.getVDSup(bc.Sup, pc.Sup)$ if conf.Sup is true
    $classLevelMetrics \leftarrow getConfusionMatrixForAllElements()$
    return $classLevelMetrics$
end function
function GETMLM(bm, pm)
    $VDAtt \leftarrow alg.getVDAtt(bm.Att, pm.Att)$ if conf.Att is true
    $VDOp \leftarrow alg.getVDOp(bm.Op, pm.Op)$ if conf.Op is true
    $VDRef \leftarrow alg.getVDRef(bm.Ref, pm.Ref)$ if conf.Ref is true
    $VDSup \leftarrow alg.getVDSup(bm.Sup, pm.Sup)$ if conf.Sup is true
    $modelLevelMetrics \leftarrow getConfusionMatrixForAllElements()$
    return $modelLevelMetrics$
end function
```

The algorithm of choice can be implemented such that it extends from the abstract class defined in figure 4.8. Each algorithm should contain functions to extract venDiagrams for the elements as used on lines [8 - 11] in Algorithm 1. This section describes 2 algorithms that have been integrated as part of the tool.

6.1 Hashing Based Comparison

The hashing based algorithm has been implemented as an extension of the abstract model comparison class (figure 1). The inspiration for this algorithm comes from the Xiao He's [12] paper; an extension of EMF-Compare. Our algorithm computes the venn diagrams for each of the following elements; classes, references, attributes, operations, superTyes, and Enums. VennDiagram is a data structure (implemented as a DTO (Data Transfer Object)) containing a triple such that:

Venn Diagram = <onlyInModel1, intersection, onlyInModel2>

The pseudocode in 2 details the steps to compute the venn diagrams provided with two arrays of the same types of elements. At first, a hash value of each of the elements is computed, which is use to index the element. This is followed by finding pairs of elements that have the maximum similarity score; these pairs are included in the intersection section of the venn diagram. For the elements in model 1 that were not paired with any element in model 2; are included in the set of elements only in model 1, and vice versa.

The algorithm used to compute similarity is detailed in pseudocode 3. The "computeSimilarity" function computes a dot product of the two hash values. It expects the hash to be a 64 bit binary value. The usage of this function can be seen in algorithm 2. The "computeHash" function takes input an element and computes a sum of the hash values of each of its features. The function "getHashValue" is ued to compute a 64 bit binary hash of each individual feature in the element. If the feature is a text-based feature then the "hashNGram" function is used to compute the 64 bit binary hash for that feature value. This technique is inspired from the "hashNGram" function proposed by Xiao He in [12]. The hashNGram function breaks the string into bi-grams and for each bi-gram computes a 64 bit binary hash; these hash values are summed up to get a 64-bit hash value for the feature. We have introduced a variation to accommodate for classes that are composed of only 1 letter, for example "public class A ". We have added an if condition to check if such is the case, then the hashCode for that letter is returned instead of computing the bigrams.

Algorithm 2 Get Venn Diagram

$conf$:	configuration	VD:	Ven Diagram
DP:	DotProduct	thv:	totalHashValue
$em1$:	elementsOnlyInModel1	$em2$:	elementsOnlyInModel2
$HiM1$:	hashIndexModel1	$HiM2$:	hashIndexModel2

```
1: function GETVD(elementsM1, elementsM2, conf)
2:     $venDiagram \leftarrow triple < intersection, em1, em2 >$
3:     $HiM1, HiM2 \leftarrow \{\}$
4:     for $i \leftarrow 1$ in elementsM1 do
5:         $HiM1.put(computeHash(i, conf), i)$
6:     end for
7:     for $i \leftarrow 1$ in elementsM2 do
8:         $HiM2.put(computeHash(j, conf), j)$
9:     end for
10:
11:     for $i \leftarrow 1$ in HiM1.keys() do
12:         $maxSimilarity \leftarrow -1$
13:         for $j \leftarrow 1$ in HiM2.keys() do
14:             $similarity \leftarrow computeSimilarity(i, j)$
15:             $maxSimilarity \leftarrow max(similarity, maxSimilarity)$
16:         end for
17:         if maxSimilarity >= conf.hashingThreshold then ▷ match found
18:             HiM1.keys().remove(i)
19:             HiM2.keys().remove(j)
20:             venDiagram.intersection.add(HiM1.get(i))
21:         else
22:             venDiagram.em1.add(HiM1.get(i))
23:         end if
24:     end for
25:
26:     for $j \leftarrow 1$ in HiM2.keys() do         ▷ Unmatched elements in model2
27:         venDiagram.em2.add(HiM2.get(j))
28:     end for
29:     return venDiagram
30: end function
```

Algorithm 3 Similarity Computation

thv: totalHashValue DP: DotProduct

```
function hashNGram(featureValue)
    $hashValue \leftarrow 0$;
    if $featureValue.length < 2$ then
        return $featureValue.hashCode()$
    end if
    for $i \leftarrow 0$ in featureValue.length(() do
        String bigram= input.substring(i, i + 2);
        hashValue += (long) bigram.hashCode();
    end for
    return $hashValue$
end function

function computeHash(element, conf)
    $thv \leftarrow 0$
    for $i \leftarrow 1$ in elementNamesToBeCompared do
        if conf.$i$ is True then
            if $element.i$ is text based then
                $thv \leftarrow thv + hashNGram(element.i)$
            else
                $thv \leftarrow thv + getHashValue(element.i)$
            end if
        end if
    end for
    return $thv$
end function

function computeSimilarity(hash1, hash2)
    $dp \leftarrow DP(hash1, hash2)$
    return $dp$
end function
```

6.2 Digest Based Comparison

The digest based algorithm has been implemented in a similar way to that of the hashing based algorithm except for the matching process; where the elements' raw feature values are matched instead of their hash. If an exact match is found; the elements are paired together.

6.3 Semantic Similarity

This algorithm defines a class SemanticSimilarity to compute the semantic similarity between two emfatic files. The process begins by extracting and tokenizing text from the given code using NLTK, then removes packages, namespace declaration and comments. Package declerations are removed because the focus of the tools is to find semantic similarity between class diagrams; it is not necessary for the two models to have the same package structure for them to be semantically similar. The algorithm tokenizes the remaining text and converts it into a list of tokens. Then, it uses TF-IDF (Term Frequency-Inverse Document Frequency) to calculate the importance of each word in the documents and obtains word embeddings using pre-trained Word2Vec vectors from the GoogleNews-vectors-negative300 word embedding model. We have provided support for 2 models; glove6b50d and GoogleNews-vectors-negative300 that can be configured from the environment variable of the api. These embeddings are weighted by the TF-IDF scores to get a weighted average embedding for each document. Finally, the cosine similarity between the embeddings of the original and predicted code files is computed to determine their semantic similarity, which is then returned as a score. The implementation of the semantic similairty algorithm can be viewed in the trivial code presented in appendix figure 9.17.

Chapter 7

Case Study

This chapter contains a case study carried out using Comparit. The motivation for the case study is to compare the effectiveness of Modisco [1]; a reverse engineering tool. We will manually create the ecore model for the ecommerce-backend project on github [2] using EMF and use it as a baseline for comparison.

7.1 Source Code and its Ecore Model

The project is built on Spring Boot[3], a popular framework for Java backend development, making it an excellent platform for demonstrating reverse engineering testing tools.

The widespread use and inherent complexity of the E-Commerce domain make it an ideal candidate for showcasing the capabilities of a reverse engineering algorithm. The selected case study incorporates a wide variety of Java elements including attributes, operations, references, and superTypes. This diversity provides a thorough basis for evaluating how well our model comparison tool is able to determine the differences with regards to these elements between the baseline and reverse engineered models.

The core domain model classes in our E-Commerce application include `Order`, `Product`, and `Category`, which represent key elements of the E-Commerce domain, such as product listings, customer orders, and categories.

[1]https://github.com/atlanmod/modisco
[2]
[3]`https://spring.io/projects/spring-boot`

Figure 7.1 presents the conceptual domain model using a class diagram.

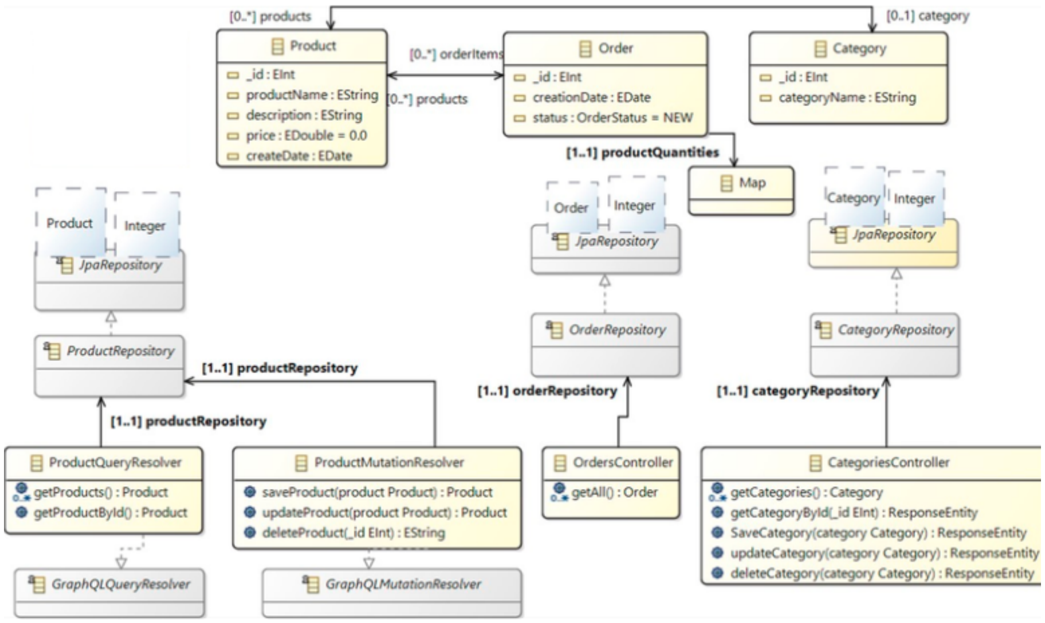


Figure 7.1: Ecommerce Application Baseline Model

Within the code, classes are annotated with Spring MVC and JPA annotations to define their behavior and persistence characteristics. For example, the `Product` class is annotated with `@Entity` and `@Data`, marking it as a persistent entity and auto-generating boilerplate methods like getters and setters.

The application includes Spring Boot controllers that handle HTTP requests related to orders and categories. For example, the `OrdersController` provides endpoints for retrieving orders, while the `CategoriesController` manages category-related actions. Additionally, repository interfaces such as `OrderRepository` and `CategoryRepository` define methods for interacting with the underlying database, utilizing Spring Data JPA to streamline data access. The figure also includes the `JpaRepository` interface, which is extended by these repository interfaces. While the class diagram omits the main entry point and test class of the Java project, it is important to note that the `CategoriesController` class makes use of the `ResponseEntity` class provided by Spring Framework.

7.2 Modisco Model

Figure 7.2 shows the ecore model extracted by Modisco. Both the baseline and modisco models contain similar class structures. As can be seen from the ecore model extracted by Modisco; it has correctly identified the following classes; CategoriesController, OrdersController, ProductMutationResolver, ProductQueryResolver, Category, Order, Product, CategoryRepository, OrderRepository, ProductRepository.

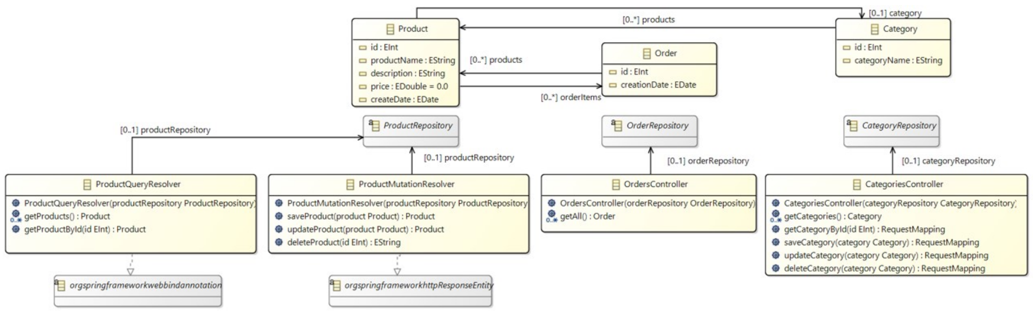

[0..1] category
[0..*] products
Product
id : EInt
productName : EString
description : EString
price : EDouble = 0.0
createDate : EDate
[0..*] products
Order
id : EInt
creationDate : EDate
[0..*] orderItems
Category
id : EInt
categoryName : EString
[0..1] productRepository
ProductRepository
OrderRepository
CategoryRepository
[0..1] productRepository
[0..1] orderRepository
[0..1] categoryRepository
ProductQueryResolver
ProductQueryResolver(productRepository ProductRepository)
getProducts() : Product
getProductById(id EInt) : Product
ProductMutationResolver
ProductMutationResolver(productRepository ProductRepository
saveProduct(product Product) : Product
updateProduct(product Product) : Product
deleteProduct(id EInt) : EString
OrdersController
OrdersController(orderRepository OrderRepository)
getAll() : Order
CategoriesController
CategoriesController(categoryRepository CategoryRepository)
getCategories() : Category
getCategoryById(id EInt) : RequestMapping
saveCategory(category Category) : RequestMapping
updateCategory(category Category) : RequestMapping
deleteCategory(category Category) : RequestMapping
orgspringframeworkwebbindannotation
orgspringframeworkhttpResponseEntity

Figure 7.2: Ecore Model Extracted by Modisco

The baseline includes two additional abstract classes: GraphQLMutationResolver and GraphQLQueryResolver, which are not explicitly defined in the predicted model. However, in the predicted model, both ProductMutationResolver and ProductQueryResolver extend other classes (orgspringframeworkhttpResponseEntity and orgspringframeworkwebbindannotation), suggesting an alternative abstraction.

The attributes are mostly captured in both models, but there are some significant differences in how they are represented. The matching attributes class-wise are listed below:

- Category Class: _id, categoryName.
- Product Class: _id, productName, description, price, createdData
- Order Class: _id, creationDate.

In the predicted model, many attributes are marked as ref even when they are clearly simple types (e.g., int, String, double). This includes attributes like _id and categoryName in the Category class and creationDate in the

Order class. The ground truth uses the datatype EDate for dates, while the predicted model uses Date, which is inferred from java.util.Date. The predicted model uses a simplified approach to the attributes by setting the "ordered" feature to false (observed by inspecting ecore file) for operations, references and attributes, which doesn't exist in the ground truth.

References have been mostly captured correctly in Modisco's model. The matching references are; Category.products referencing Product, Order.products referencing Product, Product.category referencing Category, and Product.orderItems referencing Order.

Most operation names are accurately captured, but there are some changes in return types and operation parameters. The operations getCategories(), getAll(), saveProduct(Product product), updateProduct(Product product), deleteProduct(int _id) are well captured. The operations in ProductMutationResolver and ProductQueryResolver mostly align, although with some differences in the return types. In Modisco's model, the return type for getCategoryById(int _id), saveCategory(Category category), and other similar methods in CategoriesController is RequestMapping instead of ResponseEntity, as defined in the baseline model. The operation ProductQueryResolver.getProductById() in the ground truth does not take any parameters, while in the predicted model, it takes int _id as a parameter.

Since the classes GraphQLMutationResolver, GraphQLQueryResolver and JPA Repository were not captured; 2 of the superTypes present in the baseline model were not included in Modisco's model.

7.3 Comparison using Comparit

In this section, we will compare the ecore model generated by Modisco with the baseline ecore model. We will try various combinations of configurations to see how that affects the desired outcome.

The results of matching using hashing algorithm with a threshold of 0.95 and all the rest configuration variables (refer to appendix table 9.1) to true, can be seen in figures 7.3, 7.4, and 7.5.

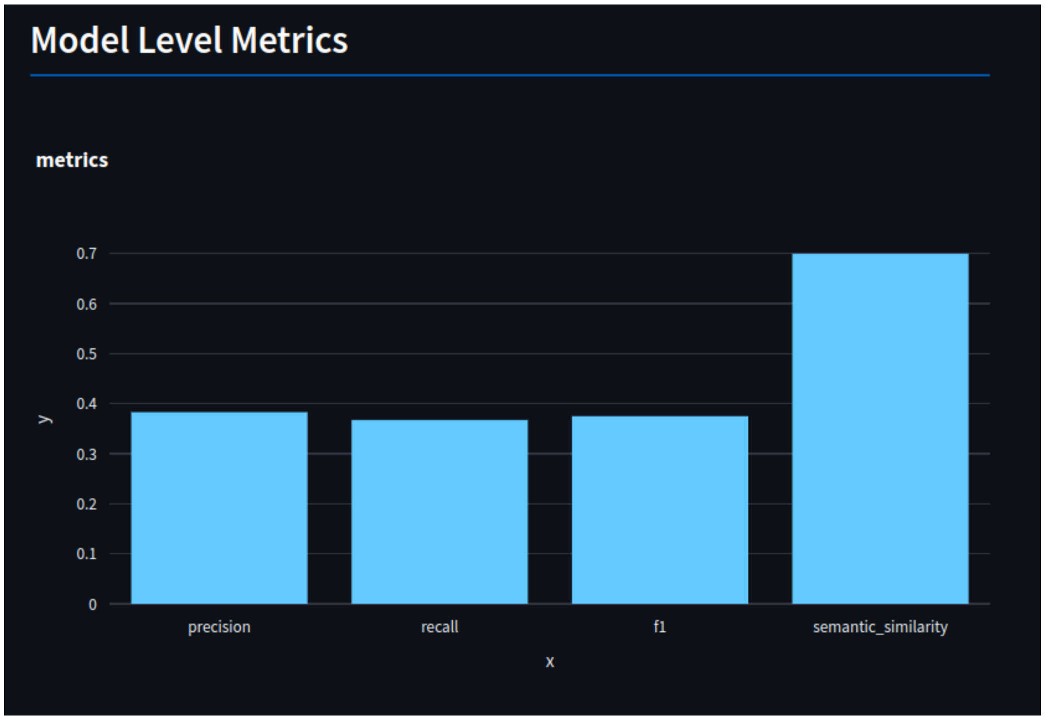


Figure 7.3: Model Level Metrics. *hashing=0.95, rest config variables set to True*

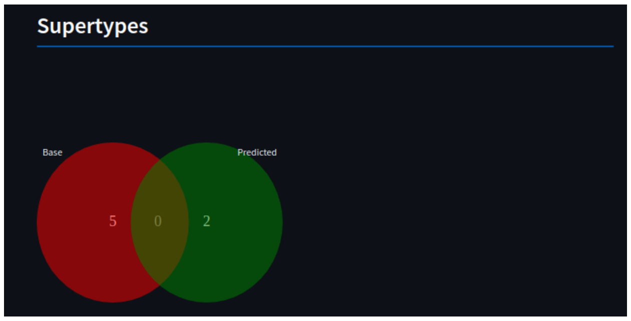


Figure 7.5: Ven Diagrams for Elements (2/2). *hashing=0.95, rest config variables set to True*

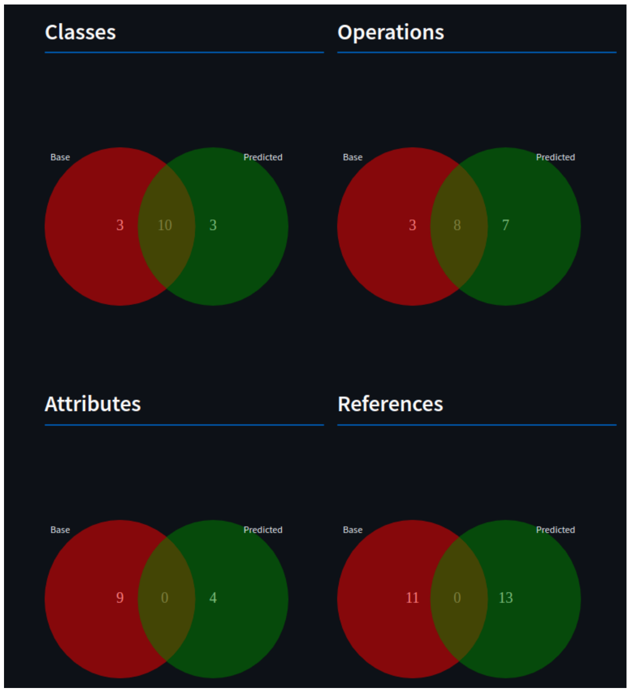


Figure 7.4: Ven Diagrams for Elements (1/2). *hashing=0.95, rest config variables set to True*

By looking at the ven diagrams, it can be inferred that attributes, references, and superTypes are the major players in bringing the precision down for the model. There are no matching sueprtypes because they were misidentified by Modisco as discussed in the manual insepction of the two models earlier in this chapter. One of the resaons for the absence of any true positives in attributes is the fact that Modisco idenitified the primitive types as references; and hence resulting in an increase false positive values for references. No true positives in the references was a result of the fact that modisco had set the ”ordered” feature for references to false. To investigate this further, we set the configuration variable ”ordered” for attributes and references to false; which means the comparator will ignore that element feature when performing the comparison. The results are presented in figures 7.6 and 7.7.

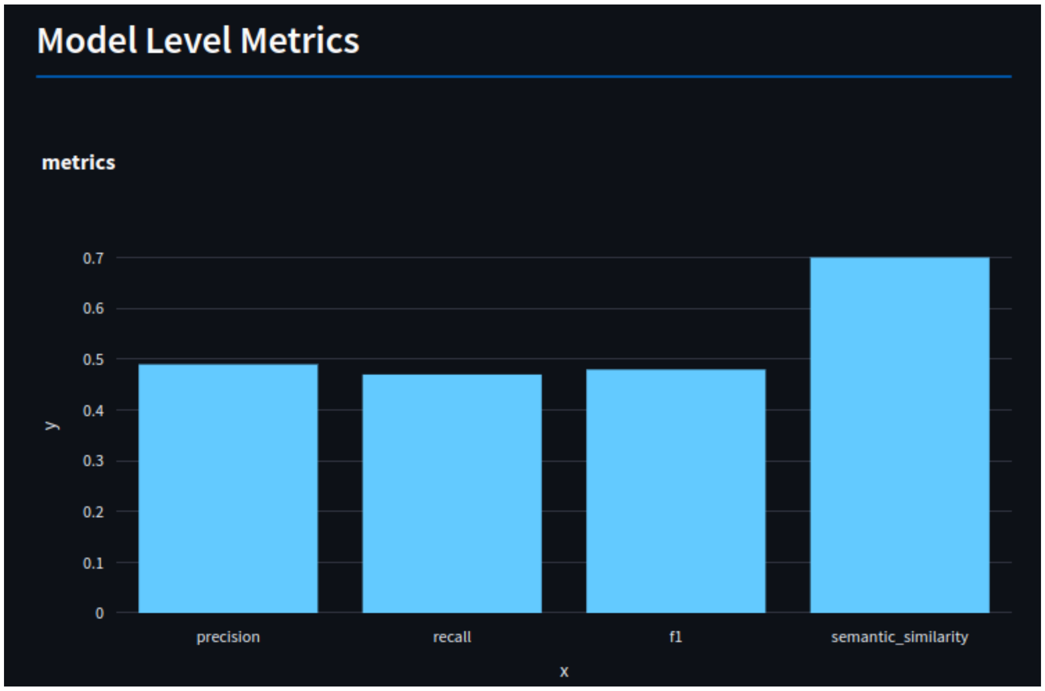


Figure 7.6: Model Level Metrics. *hashing=0.95, order=false for references and attributes, Rest config variables set to True*

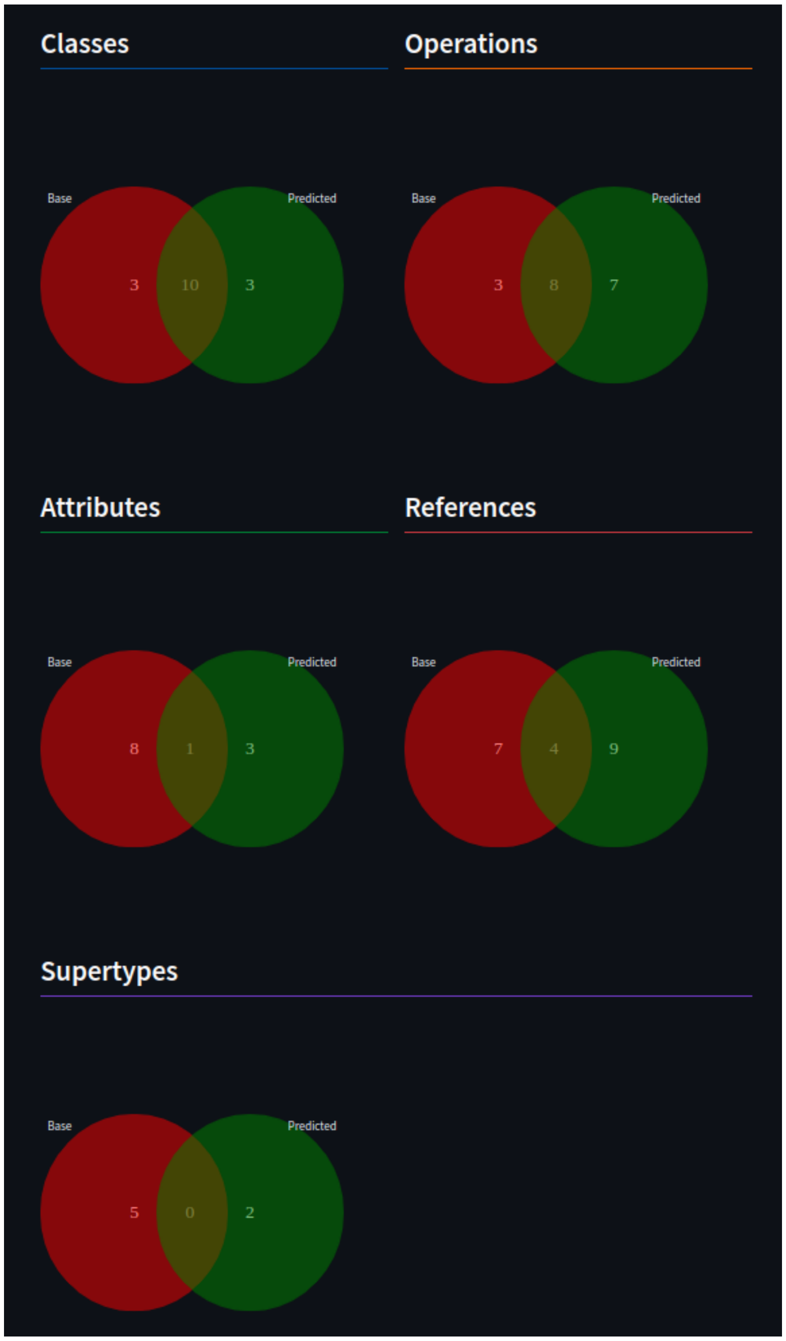


Figure 7.7: Venn Diagrams *hashing=0.95, order=false for references and attributes, Rest config variables set to True*

The new configuration resulted in improved metrics. Now, we can see 1 true positive attribute and 4 true positive references. There seem to be 3 false negative operations; that is because the parameters did not accurately match with the baseline model as described in the manual inspection. If we wish to ignore the operation parameters, we can set the `INCLUDE-OPERATION-PARAMETERS` to false. The results are depicted in figures 7.8 and 7.9.

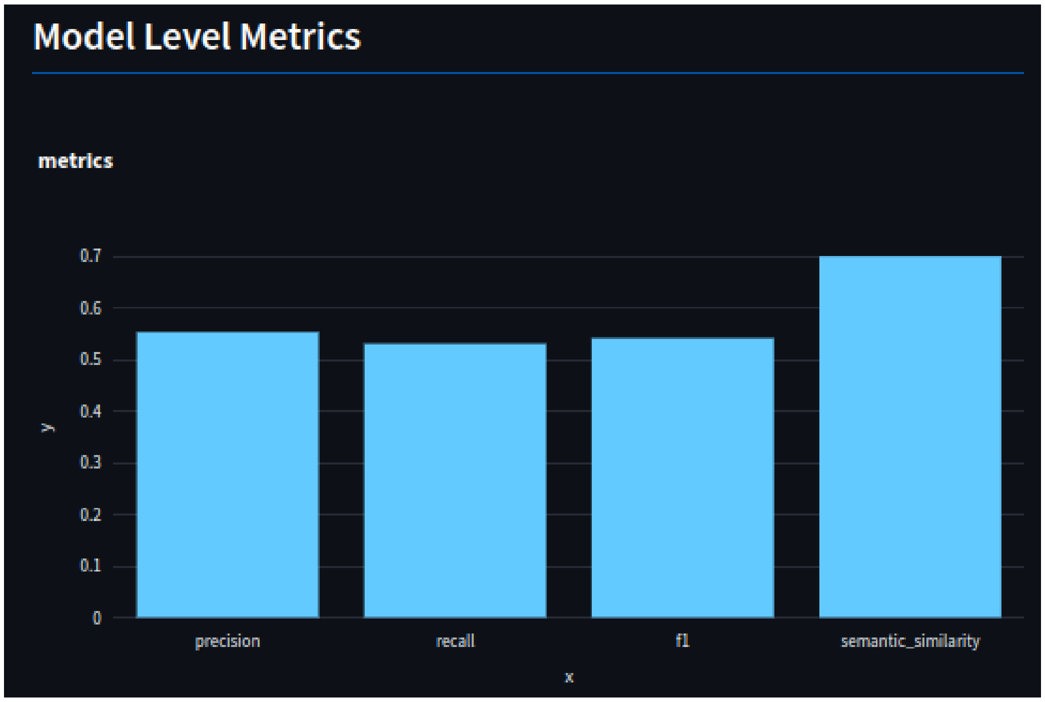


Figure 7.8: Model Level Metrics. *hashing=0.95, order=false for references and attributes, include operation parameters=False. Rest config variables set to True*

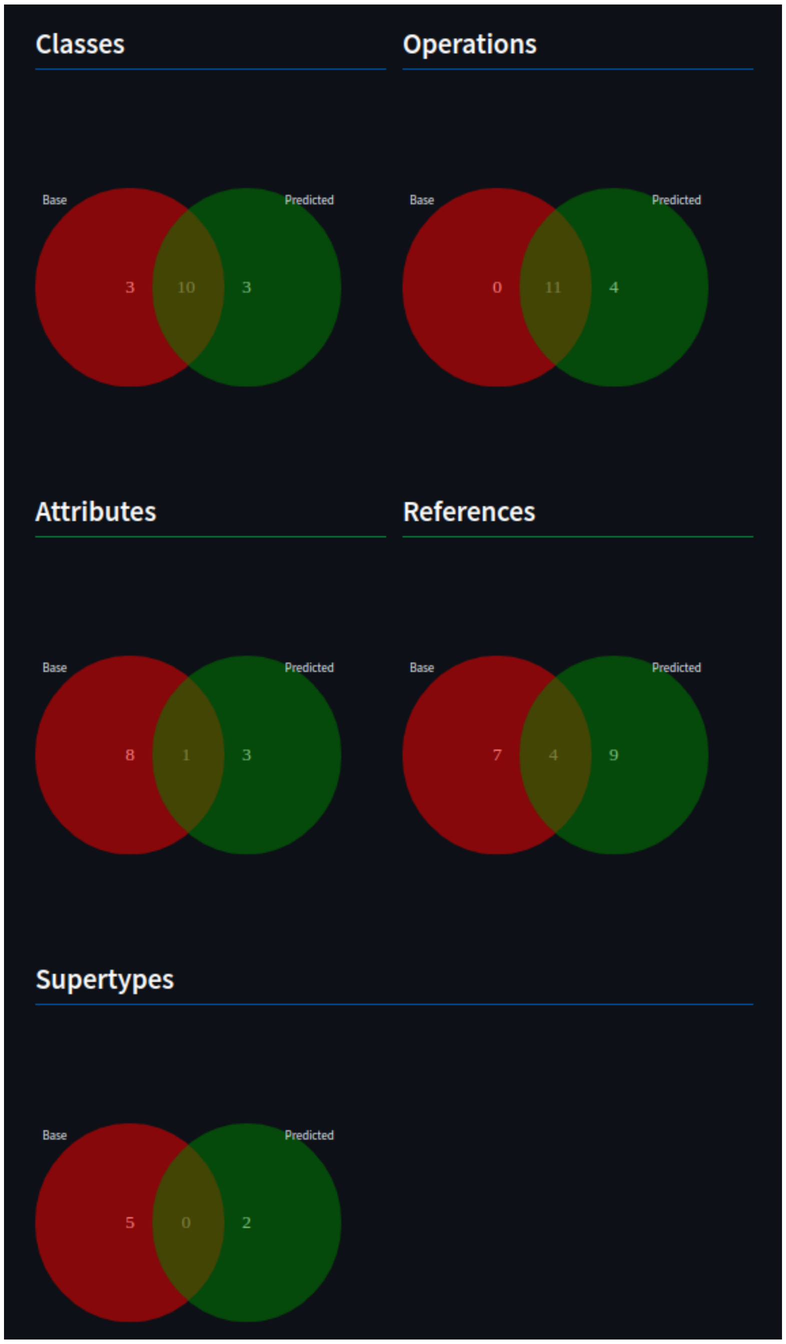


Figure 7.9: Venn Diagrams. *hashing=0.95, order=false for references and attributes, include operation parameters=False. Rest config variables set to True*

We can conclude that one of the key factors contributing to Modisco's lower accuracy was its handling of primitive types as references rather than attributes. While this impacted the syntactic similarity score, it's worth noting that the semantic similarity score remained high. This highlights the tool's strength in not overly penalizing minor syntactic discrepancies, as long as the underlying meaning stays consistent.

Chapter 8

Evaluation

8.1 Methodology

To validate the correctness of the model matching algorithms used in the tool, we prepared a benchmark dataset containing triples; base ecore model, mutant, and their similarity metrics. The benchmark dataset's structure is described in detail in section 8.2.

The model comparison tool was then executed with difference combinations of configuration to extract the similarity metrics for each ecore model pair and compared against the given similarity metrics to gauge the accuracy of the tool. The results are presented in 8.3.

8.2 Benchmark Dataset

The preliminary preparation of the dataset involves downloading the modelset zip folder [1]. We have used the Dec 14, 2023 release of modelset (size: 104MBs). Then we extract the all the ecore models present in the repo-ecore-all folder.

For each ecore model, 10 mutants were generated using YAMTL. The details of each mutant can be found in the table 8.2; each entry represents the percentage of similar element tags between the base model and predicted model. The mutants cover edge cases and one hybrid case described as "mutant 10" in the table. For each mutant; we computed the percentage

[1]https://github.com/modelset/modelset-dataset/releases

of false negatives by subtracting the specified percentage of similar elements from 100. For example, if we want 90 percent of operations to be true positives, then the number of false negatives that would be generated for that mutant would be 10 percent (100 - 90 = 10) of the total operations. The false negatives are generated in a way such that the elements in the base model are kept intact; and no corresponding element is generated for the mutant. However, to avoid having equal precision and recall, we generate a non-matching element in the mutant for 40 percent of the false negatives. This introduces a non-zero value for false positives.

Figure 8.1 (Screenshot taken from the Ui for evaluation results; as described in section 8.4) depicts the distribution of precision, recall, and f1-score across the model-pairs used for evaluation. It can be seen that the distribution of precision and recall differs, indicating that there is a variety of false positive and false negative values across the model pairs.

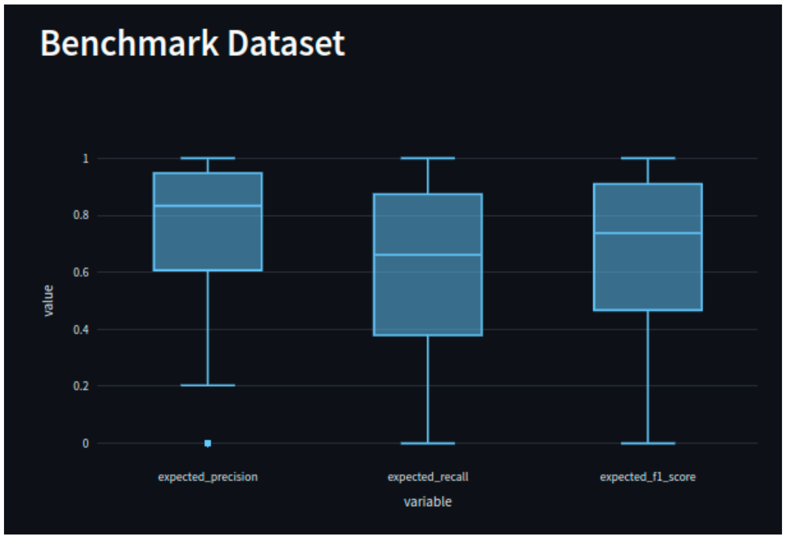


Figure 8.1: Benchmark Dataset Metrics Distribution

Figure 8.2: Table for description of mutants. Each entry represents the percentage of similar elements between base model and predicted model.

Mutant	Classes	Attributes	Operations	References
1	100	100	100	100
2	0	0	0	0
3	100	100	0	0
4	100	0	0	100
5	100	0	100	0
6	100	100	100	0
7	100	100	0	100
8	100	100	0	100
9	100	0	100	100
10	100	70	90	80

When processing ecore models from the modelset dataset and generating mutants, we only included models that are greater than or equal to 20kbs in file size. This ensured that the models are big and hence have more variety when it comes to model elements; attributes, references, operations, parameters, supertypes and enumerations. Figure 8.3 showcsaes the diversity of the size of models in the benchmark dataset. The distribution of model elements in the benchmark dataset can be seen from a bird's eye view in figure 8.4; each point in the box plot represents the total number of elements for a model present in the dataset.

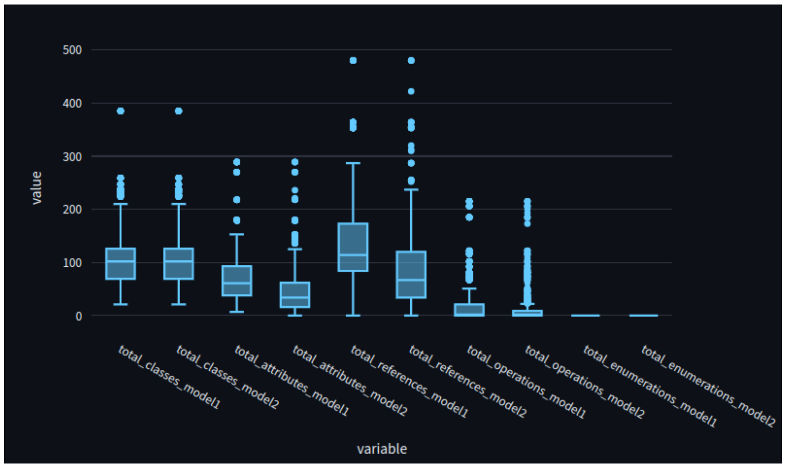


Figure 8.4: Model Elements Distribution

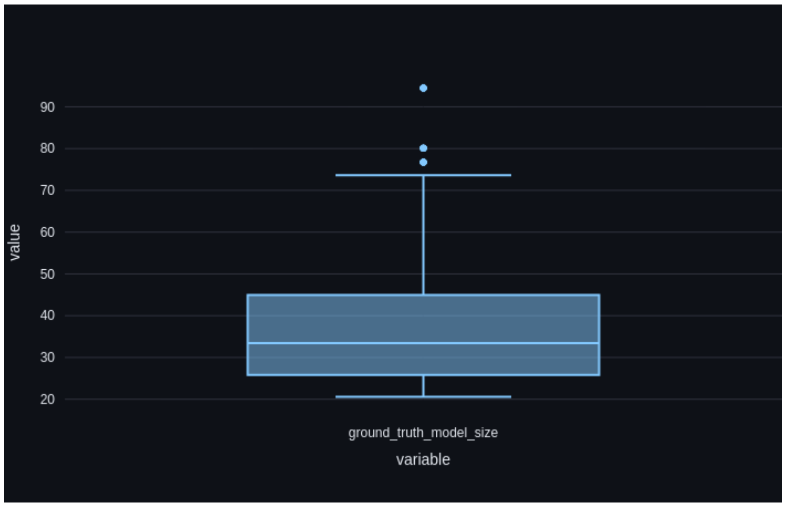


Figure 8.3: Models Size Distribution

The model comparator was executed on the benchmark dataset with four different configurations:

- Using Hashing based algorithm with a hashing threshold of 0.95 and setting the MODEL-LEVEL-COMPARISON-DERIVED-FROM-CLASS-LEVEL-COMPARISON to true
- Using Hashing based algorithm with a hashing threshold of 0.95 and

setting the MODEL-LEVEL-COMPARISON-DERIVED-FROM-CLASS-LEVEL-COMPARISON to false

- Using Digest based algorithm and setting the MODEL-LEVEL-COMPARISON-DERIVED-FROM-CLASS- LEVEL-COMPARISON to true
- Using Digest based algorithm and setting the MODEL-LEVEL-COMPARISON-DERIVED-FROM-CLASS- LEVEL-COMPARISON to false

The results of the evaluation are reported in section 8.3.

8.3 Results

Figure 8.5 reports the mean of absolute errors and MSQE (Mean squared error) for the four configurations.

The mean of absolute errors is computed using the following formula:

$$\frac{\sum |expected\_f1\_score - obtained\_f1\_score|}{totalModelPairs}$$

The MSQE is computed using the following formula:

$$\frac{\sum |expected\_f1\_score - obtained\_f1\_score|^2}{totalModelPairs}$$

The figure displays four bar graphs comparing ther results of different configurations ,in terms of their "Mean" and "Mean Squared Error (MSQE)", for the two components of the model comparator: **Syntactic Comparator** and **Semantic Comparator**. Each graph has bars representing four configurations: *digest without aggregate*, *digest with aggregate*, *hashing.95 without aggregate*, and *hashing.95 with aggregate*. The vertical axis shows the error values, while the horizontal axis lists the configurations, with labels rotated for readability.

Syntactic Comparator Error

- **Mean of Syntactic Comparator Error**: The *hashing.95 with aggregate* configuration has the highest mean error (552.217μ), followed by *digest with aggregate* (369.15μ). The configurations *digest without aggregate* and *hashing.95 with aggregate* exhibit the lowest mean errors (72.36μ and 351.69μ, respectively).

- **MSQE of Syntactic Comparator Error**: Similarly, *hashing .95 with aggregate* and *digest with aggregate* yield the highest MSQE (14.12μ and 13.47μ). The *hashing.95 without aggregate* has the lowest MSQE (1.68μ), while *digest without aggregate* has an intermediate MSQE (4.47μ).

Semantic Comparator Error

The semantic copmarator doesn't take into account the configuration and hence no change in error is observed upon changing the configuration.

- **Mean of Semantic Comparator Error**: All four configurations show very similar mean error values (0.153), with slight variations but no significant differences.

- **MSQE of Semantic Comparator Error**: The MSQE results are also similar across all configurations, ranging around 0.048, with minimal differences between them.

The syntactic comparator's mean of absolute error decreased from 522.13μ to 369.15μ and the MSQE decresed from 13.12μ to 1.61μ as a result of setting the `MODEL-LEVEL-COMPARISON-DERIVED-FROM-CLASS-LEVEL-COMPARISON` to "false" while keeping the hashing threshold same (0.95). AFter inspecting the individual results of model comparison, it was observed that the supertypes are duplicated when the class level metrics are being aggregated. This is because 1 class can be extended my multiple classes. The same argument holds explains the fall in error for the digest based algorithm upon turning setting the `MODEL-LEVEL-COMPARISON-DERIVED-FROM-CLASS-LEVEL-COMPARISON` to "false".

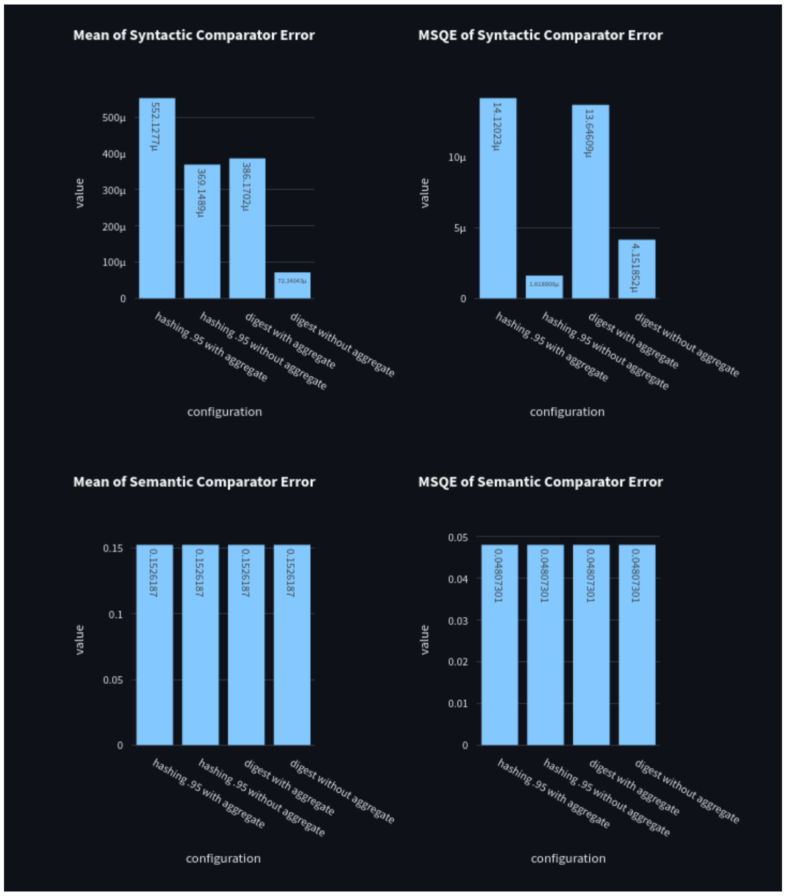


Figure 8.5: Bar Chart representing Errors

Scatter Plot Analysis

The figure 8.6 consists of four scatter plots showing the relationship between **Model Size** and **Time (s) for Syntactic Comparison** using different configurations of hashing and digest methods, with and without aggregation.

Hashing with Aggregation

This plot represents the time taken for syntactic comparison using hashing with aggregation. As model size increases, the time also increases, showing a positive correlation.

Hashing without Aggregation

This plot shows the time for syntactic comparison using hashing without aggregation. Similar to the previous plot, there is a positive correlation between model size and comparison time, but the increase in time is more significant than in the case with aggregation.

Digest without Aggregation

This plot illustrates the comparison time when using digest based algorithm without aggregation. There is a positive relationship between model size and comparison time, but the time values are generally lower than in the hashing configurations.

Digest with Aggregation

This plot shows the syntactic comparison time using the digest based algorithm with aggregation. As model size increases, there is again a clear positive correlation with time.

Conclusively, in all plots, model size has a clear positive correlation with comparison time, indicating that larger models require more time for syntactic comparison. However, the configuration with hashing (without aggregation) produces the highest comparison times, while the digest method results in generally lower comparison times. It is speculated that a steeper rise in time taken for execution is higher for hashing based algorithm because of the cache. The comparison was executed in one go, and thus the cache bloated. This could be improved by implementing a more intelligent caching mechanism that deletes less frequently used cached values. We have listed this improvement in the future works section.

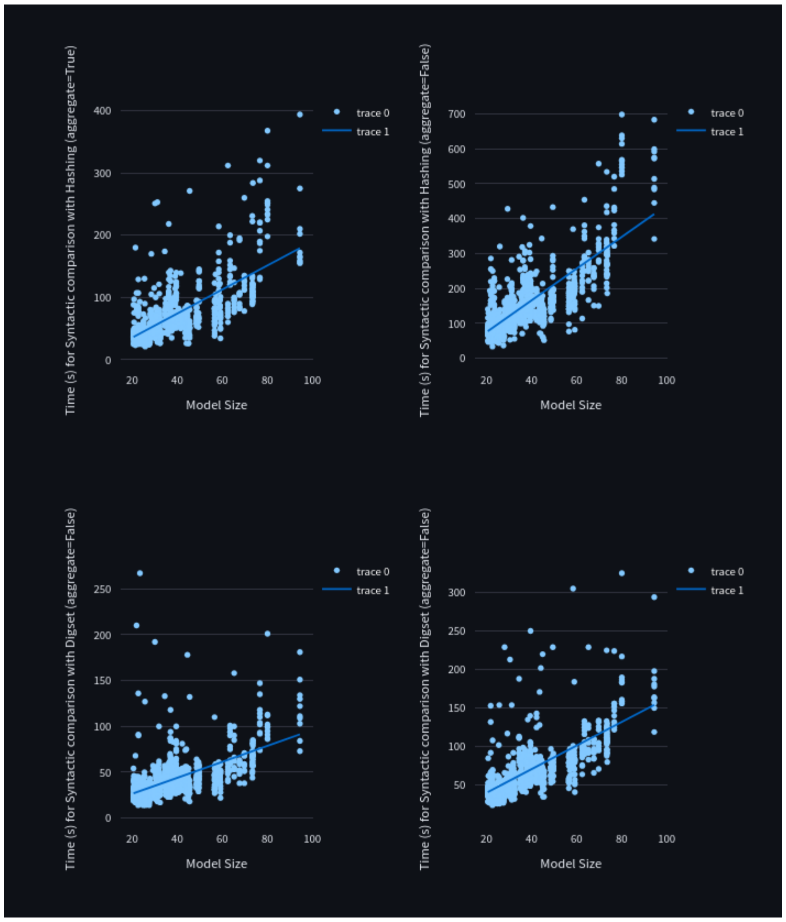


Figure 8.6: Scatter Plot

8.4 Steps to Reproduce Evaluation Results

The modelset dataset and its corresponding generated mutants amount to a large file size and hence are not included in the source code. Instead, the data is located in the artifacts folder for the tool that can be found. here[2].

The benchmark dataset can be reproduced and re-evaluated by following the steps below:

- Download the modelset zip folder and copy it in functional-tests/modelset

[2]`https://uniofleicester-my.sharepoint.com/:f:/g/personal/jm982_student_le_ac_uk/Eq8ttmKwGiZGhUX8Ujdr75IBJwI2soEHRAxqw-IaRO9nCw?e=R9m0xc`

folder

- Create a virtual environment, install dependencies and run migrate.py script. This will generate a folder named ”ecore_models_modelset”
- copy the folder `"ecore_models_modelset"` in functional-tests/
- Go to `syntactic-comparator/src/main/java/com/mdre/evaluation/services/modelComparisonService/ModelMutator.groovy` and sepcify the directory of `"ecore_models_modelset"` folder inside the ”run()” function.
- run ”./gradlew runModelMutatorJavaWrapper: This will run the model mutator that will generate mutants within the ”ecore_models_modelset” fodler.
- To execute Comparit on the generated dataset, create a virtual env inside functional-tests folder and install dependencies using requirements.txt file. Then run functional-tests/api-tests.py. This should take 10-15 mins depending on your system's capacity. This script runs sanity tests for the available APIs for ecore model comparison, then executes model comparison for each model with its mutants. For example, if we have 5 models then the comparison would run 50 times (10 times for each model with its 10 mutants). Once the execution of this script finishes, the following csv files will be produced:
  - evaluation_results_digest_true.csv: This file will contain the resutls for the comparison with the MODEL-LEVEL-COMPARISON-DERIVED-FROM-CLASS- LEVEL-COMPARISON (refer to appendix table 9.1) flag set to True and USE-HASHING set to false. This implies that the comparator has used the digest based comparison algorithm 6.2 and carried out the model level comparison by aggregating the class level comparison results.
  - evaluation_results_digest_false.csv: This file will contain the resutls for the comparison with the MODEL-LEVEL-COMPARISON-DERIVED-FROM-CLASS- LEVEL-COMPARISON flag set to False and USE-HASHING set to false. This implies that the comparator has used the digest based comparison algorithm 6.2 and carry out the model level comparison seperately instead of aggregating the class level comparison results.

- evaluation_results_hasing_.95_aggregate_true.csv: This file will contain the resutls for the comparison with the MODEL-LEVEL-COMPARISON-DERIVED-FROM-CLASS- LEVEL-COMPARISON flag set to True and USE-HASHING set to true. This implies that the comparator will use the hashing based comparison algorithm 6.1 with a hashing threshold of 0.95 and carry out the model level comparison by aggregating the class level comparison results.
- evaluation_results_hasing_.95_aggregate_false.csv: This file will contain the resutls for the comparison with the MODEL-LEVEL-COMPARISON-DERIVED-FROM-CLASS- LEVEL-COMPARISON flag set to False and USE-HASHING set to true. This implies that the comparator will use the hashing based comparison algorithm 6.1 with a hashing threshold of 0.95 and carry out the model level comparison seperately instead of aggregating the class level comparison results.

- To view the visualizations for results, navigate to functional-tests/ and execute "streamlit run evaluation_ui.py". This will open a user-interface providing insights into the evaluation results. The summary of evaluation results are presented earlier in Section 8.3.

Chapter 9

Conclusion Future Works

In this paper, we proposed an IDE-agnostic model comparison tool that is highly extensible, facilitating the implementation of various model comparison algorithms. We have demonstrated its capabilities by implementing both digest-based and hashing-based comparison algorithms. The tool enables users to specify the granularity of the comparison process, offering greater control and flexibility. We evaluated the tool using a benchmark dataset created by applying model transformations to generate mutants for the modelset dataset.

Moving forward, we aim to enhance the tool in several key areas. First, we plan to expand its compatibility to support additional popular modeling languages beyond Ecore, Emfatic, and UML, such as PlantUML and Moose. This will increase the tool's versatility and applicability across diverse modeling environments. Second, we intend to introduce functionality for identifying matched and unmatched model elements, providing users with more granular insights into their model comparisons.

To further improve the user experience, we will integrate a syntax highlighter to validate model syntax and pinpoint specific line numbers causing issues. Currently, the tool does not handle syntactically invalid models well, and adding this feature will significantly enhance usability.

We also aim to address performance concerns by enabling parallel processing of models. This can be achieved by allocating separate nodes for processing different sets of model elements, thereby distributing the workload and reducing memory consumption associated with syntactic comparison. Additionally, optimizing the caching mechanism is a priority. The current system experiences cache bloating and increased memory usage during bulk comparisons of large models. By refining the caching strategy, we can improve both efficiency and scalability.

To showcase the applicability and usability of the tool in the software industry, we intend to include a case study where we will use Comparit to compare meta models of different versions of a Java project. For example, using two versions of the same Java project from two different releases.

Currently, the tool only incorporates the configuration specified by the user when computing syntactic similarity. Going further, it would be a nice addition to the tool to specify configuration for semantic similarity; which would include the elements to be included and the embedding model to be used.

Lastly, we plan to leverage Large Language Models (LLMs) for semantic model comparison. The idea is to utilize LLMs to analyze metamodels and perform comparisons, providing a more nuanced and context-aware analysis of model similarities and differences. This approach promises to enhance the tool's ability to handle complex semantic comparisons and offer deeper insights into model relationships.

Bibliography

[1] Lorenzo Addazi and Antonio Cicchetti. Systematic evaluation of model comparison algorithms using model generation. *The Journal of Object Technology*, 19:11:1, 01 2020.

[2] Mohammad Alwanain, Behzad Bordbar, and Julian K. F. Bowles. Automated composition of sequence diagrams via alloy. In *Proceedings of the 2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 384–391, 2014.

[3] Nicolas Anquetil, Anne Etien, Mahugnon H. Houekpetodji, Benoit Verhaeghe, Stéphane Ducasse, Clotilde Toullec, Fatiha Djareddir, Jerôme Sudich, and Moustapha Derras. Modular moose: A new generation of software reverse engineering platform. In *ICSR: International Conference on Software and Software Reuse*, 2020.

[4] Omar Badreddin, Timothy C. Lethbridge, and Andrew Forward. A novel approach to versioning and merging model and code uniformly. In *Proceedings of the 2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 254–263, 2014.

[5] Steve Barrett, Dirk Sinnig, Philippe Chalin, and Graham Butler. Merging of use case models: semantic foundations. In *Proceedings of the 2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering*, pages 182–189. IEEE, 2009.

[6] Artur Boronat. Expressive and Efficient Model Transformation with an Internal DSL of Xtend. 10 2018.

[7] Andrei Z. Broder. On the resemblance and containment of documents. In *Proceedings of the International Conference on Compression and Complexity of Sequences*, pages 21–29, 1997.

[8] Eclipse Foundation. Eclipse Foundation website. `https://eclipse.dev/modeling/mdt/?project=uml2`. Accessed: 2024-06-28.

[9] Eclipse Foundation. Eclipse modeling framework (emf), 2024. Accessed: 2024-08-12.

[10] Mathieu Foucault, Frédéric Barbier, and Denis Lugato. Enhancing version control with domain-specific semantics. In *Proceedings of the 5th International Workshop on Modeling in Software Engineering (MiSE '13)*, pages 31–36, Piscataway, 2013. IEEE Press.

[11] V. García-Díaz, B.C. Pelayo G-Bustelo, O. Sanjuán-Martínez, E.R. Nú nez Valdez, and J.M. Cueva Lovelle. Mctest: Towards an improvement of match algorithms for models. *IET Software*, 6(2):127–139, April 2012.

[12] Xiao He, Yi Liu, and Huihong He. Accelerating similarity-based model matching using dual hashing. 2024. Received: 18 April 2023 / Revised: 14 December 2023 / Accepted: 14 March 2024.

[13] Dimitrios S. Kolovos, Davide Di Ruscio, Alfonso Pierantonio, and Richard F. Paige. Different models for model matching: An analysis of approaches to support model differencing. In *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, pages 1–6. IEEE Computer Society, 2009.

[14] Antonio Cicchetti Lorenzo Addazi. Using benji to systematically evaluate model comparison algorithms. 2020.

[15] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Cddiff: semantic differencing for class diagrams. In Mira Mezini, editor, *ECOOP 2011–Object Oriented Programming*, pages 230–254, Berlin, 2011. Springer.

[16] PlantUML. Plantuml: Open-source tool for drawing uml diagrams, 2024. Accessed: 2024-08-12.

[17] Junaid Rashid, Waqar Mehmood, and Muhammad Wasif Nisar. A survey of model comparison strategies and techniques in model driven engineering. *International Journal of Software Engineering and Technology (IJSET)*, 1(3):165–176, December 2016. Received Jun 12th, 2016, Revised Aug 20th, 2016, Accepted Aug 26th, 2016.

[18] Juan E. Rivera and Antonio Vallecillo. Representing and operating with model differences. In Robert F. Paige and Bernhard Meyer, editors, *Objects, Components, Models and Patterns*, pages 141–160. Springer, Berlin, 2008.

[19] Caitlin Sadowski and Greg Levin. Simhash: hash-based similarity detection. Technical report, Google Inc., 2007.

[20] Ferenc A. Somogyi. Merging textual representations of software models. In T. Kékesi, editor, *The Publications of the MultiScience—XXX. microCAD International Multidisciplinary Scientific Conference*. MultiScience, Miskolc, 2016.

[21] Ferenc A. Somogyi and Marcell Asztalos. Formal description and verification of a text-based model differencing and merging method. In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development–Volume 1: AMARETTO*, pages 657–667. INSTICC, SciTePress, 2018.

[22] Ferenc A. Somogyi and Miklós Asztalos. Systematic review of matching techniques used in model-driven methodologies. *Software and Systems Modeling*, 2019.

[23] Ferenc Attila Somogyi. *Investigating Text-Based Domain-Specific Modeling Techniques (Szöveg alapú szakterületi modellezési módszerek vizsgálata)*. Ph.d. dissertation, Budapest University of Technology and Economics, Budapest, 2022. Associate Professor.

[24] S. Uhrig and F. Schwägerl. Tool support for the evaluation of matching algorithms in the eclipse modeling framework. In *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development—Volume 1: MODELSWARD*, pages 101–110. INSTICC, SciTePress, 2013.

[25] Mark van den Brand, Albert Hofkamp, Tom Verhoeff, and Zvezdan Protić. Assessing the quality of model-comparison tools: a method and a benchmark data set. In *Proceedings of the 2nd International Workshop on Model Comparison in Practice*, IWMCP '11, page 2–11, New York, NY, USA, 2011. Association for Computing Machinery.

[26] Ronald van Rozen and Tom van der Storm. Origin tracking+text differencing=textual model differencing. In *Proceedings of the 8th International Conference on Theory and Practice of Model Transformations*, volume 9152 of *Lecture Notes in Computer Science*, pages 18–33, New York, 2015. Springer.

Appendix

Table 9.1: Model Comparison Configuration Properties

Property	**Type**	**Default Value**	**Description**
USE-HASHING	boolean	true	Whether to use hashing in the comparison
INCLUDE-DEPENDENCIES	boolean	true	Whether to include dependencies in the comparison
MODEL-LEVEL-COMPARISON-DERIVED-FROM-CLASS-LEVEL-COMPARISON	boolean	true	Model-level comparison derived from class-level comparison. If false, elements will be compared on model level
HASHING-THRESHOLD	number	0.5	The threshold for hashing similarity
INCLUDE-ENUMS	boolean	true	Whether to include enums in the comparison
INCLUDE-ENUM-NAME	boolean	true	Whether to include enum names in the comparison
INCLUDE-CLASS-ATTRIBUTES	boolean	true	Whether to include class attributes in the comparison
INCLUDE-CLASS-OPERATIONS	boolean	true	Whether to include class operations in the comparison
INCLUDE-CLASS-PARAMETERS	boolean	true	Whether to include class parameters in the comparison
INCLUDE-CLASS-REFERENCES	boolean	true	Whether to include class references in the comparison

Property	Type	Default Value	Description
INCLUDE-CLASS-SUPERTYPES	boolean	true	Whether to include class supertypes in the comparison
INCLUDE-ATTRIBUTE-NAME	boolean	true	Whether to include attribute names in the comparison
INCLUDE-ATTRIBUTE-CONTAINING-CLASS	boolean	true	Whether to include the class containing the attribute in the comparison
INCLUDE-ATTRIBUTE-TYPE	boolean	true	Whether to include attribute types in the comparison
INCLUDE-ATTRIBUTE-LOWER-BOUND	boolean	true	Whether to include attribute lower bounds in the comparison
INCLUDE-ATTRIBUTE-UPPER-BOUND	boolean	true	Whether to include attribute upper bounds in the comparison
INCLUDE-ATTRIBUTE-IS-ORDERED	boolean	true	Whether to include if the attribute is ordered
INCLUDE-ATTRIBUTE-IS-UNIQUE	boolean	true	Whether to include if the attribute is unique
INCLUDE-REFERENCES-NAME	boolean	true	Whether to include reference names in the comparison
INCLUDE-REFERENCES-CONTAINING-CLASS	boolean	true	Whether to include the class containing the reference in the comparison
INCLUDE-REFERENCES-IS-CONTAINMENT	boolean	true	Whether to include if the reference is containment
INCLUDE-REFERENCES-LOWER-BOUND	boolean	true	Whether to include reference lower bounds in the comparison
INCLUDE-REFERENCES-UPPER-BOUND	boolean	true	Whether to include reference upper bounds in the comparison
INCLUDE-REFERENCES-IS-ORDERED	boolean	true	Whether to include if the reference is ordered

Property	Type	Default Value	Description
INCLUDE-REFERENCES-IS-UNIQUE	boolean	true	Whether to include if the reference is unique
INCLUDE-OPERATION-NAME	boolean	true	Whether to include operation names in the comparison
INCLUDE-OPERATION-CONTAINING-CLASS	boolean	true	Whether to include the class containing the operation in the comparison
INCLUDE-OPERATION-PARAMETERS	boolean	true	Whether to include operation parameters in the comparison
INCLUDE-PARAMETER-NAME	boolean	true	Whether to include parameter names in the comparison
INCLUDE-PARAMETER-TYPE	boolean	true	Whether to include parameter types in the comparison
INCLUDE-PARAMETER-OPERATION-NAME	boolean	true	Whether to include the operation name associated with the parameter in the comparison

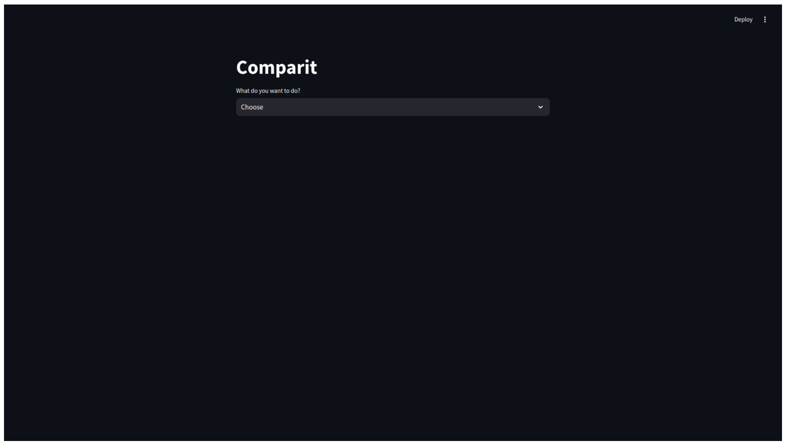


Figure 9.1: Home Page

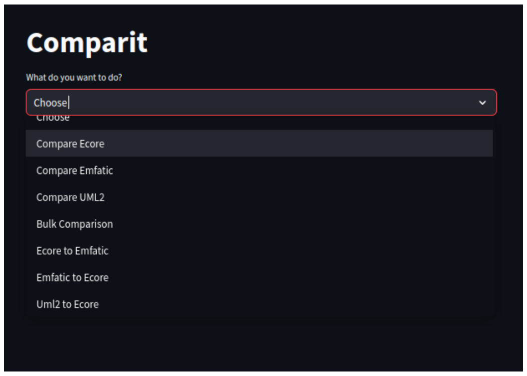
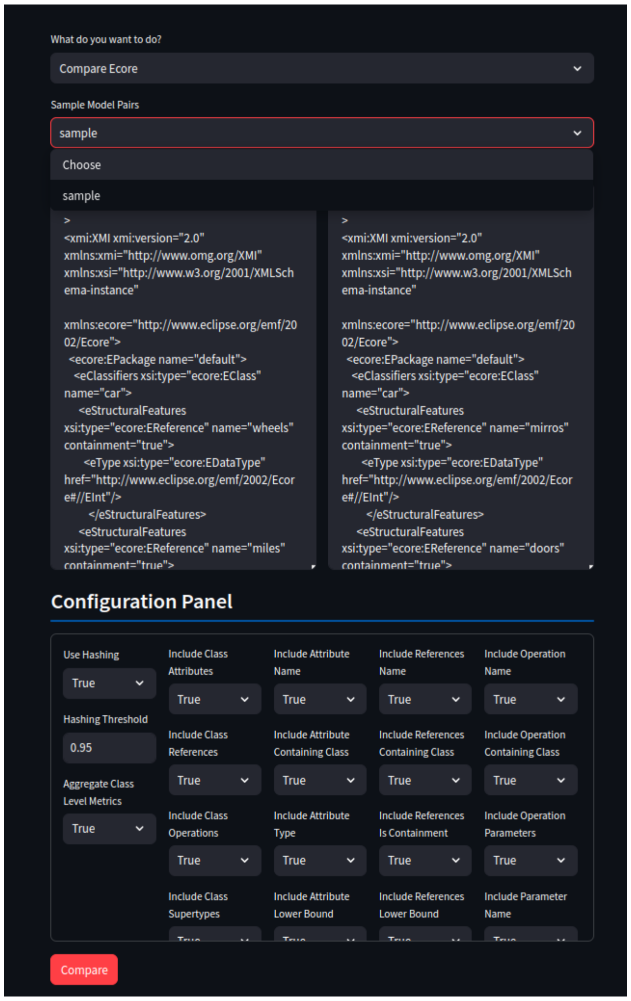


Figure 9.2: Dropdown


What do you want to do?

Compare Ecore

Sample Model Pairs

sample

Choose

sample

```
>
<xmi:XMI xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSch
ema-instance"

xmlns:ecore="http://www.eclipse.org/emf/20
02/Ecore">
 <ecore:EPackage name="default">
  <eClassifiers xsi:type="ecore:EClass"
name="car">
   <eStructuralFeatures
xsi:type="ecore:EReference" name="wheels"
containment="true">
    <eType xsi:type="ecore:EDataType"
href="http://www.eclipse.org/emf/2002/Ecor
e#//EInt"/>
     </eStructuralFeatures>
   <eStructuralFeatures
xsi:type="ecore:EReference" name="miles"
containment="true">
```

```
>
<xmi:XMI xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSch
ema-instance"

xmlns:ecore="http://www.eclipse.org/emf/20
02/Ecore">
 <ecore:EPackage name="default">
  <eClassifiers xsi:type="ecore:EClass"
name="car">
   <eStructuralFeatures
xsi:type="ecore:EReference" name="mirros"
containment="true">
    <eType xsi:type="ecore:EDataType"
href="http://www.eclipse.org/emf/2002/Ecor
e#//EInt"/>
     </eStructuralFeatures>
   <eStructuralFeatures
xsi:type="ecore:EReference" name="doors"
containment="true">
```

Configuration Panel

Use Hashing: True

Hashing Threshold: 0.95

Aggregate Class Level Metrics: True

Include Class Attributes: True

Include Attribute Name: True

Include References Name: True

Include Operation Name: True

Include Class References: True

Include Attribute Containing Class: True

Include References Containing Class: True

Include Operation Containing Class: True

Include Class Operations: True

Include Attribute Type: True

Include References Is Containment: True

Include Operation Parameters: True

Include Class Supertypes

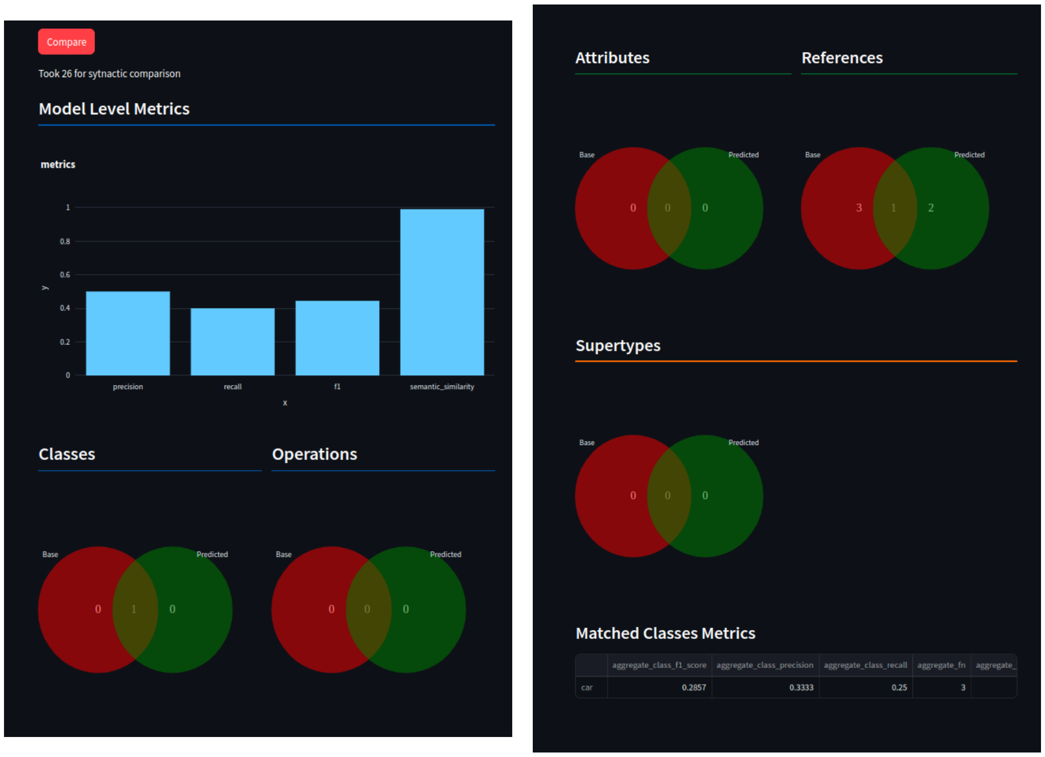
Include Attribute Lower Bound

Include References Lower Bound

Include Parameter Name

Compare

Figure 9.3: Model Comparison UI



Took 26 for sytnactic comparison

Model Level Metrics

Classes

Operations

Attributes

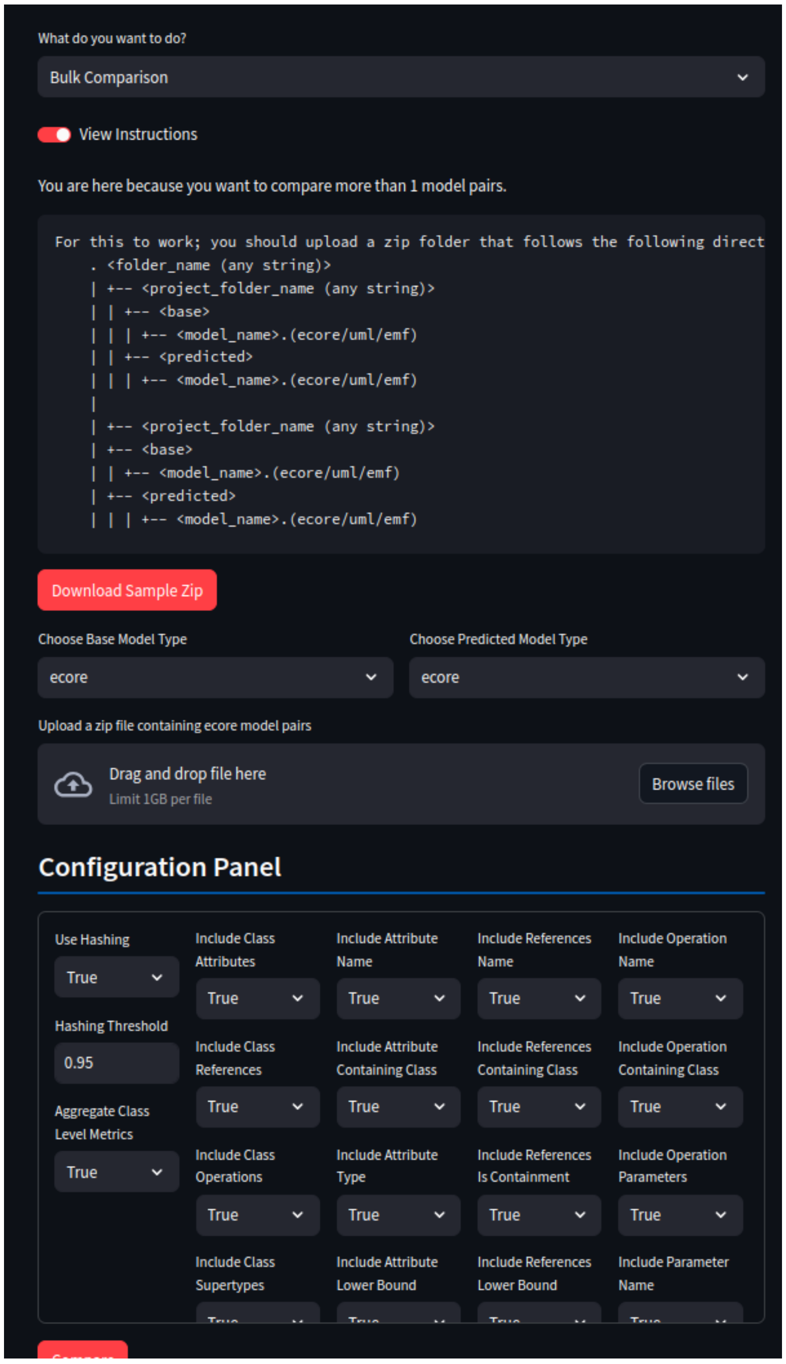
References

Supertypes

Matched Classes Metrics

	aggregate_class_f1_score	aggregate_class_precision	aggregate_class_recall	aggregate_fn	aggregate_
car	0.2857	0.3333	0.25	3	

Figure 9.4: Model Comparison Results



What do you want to do?

Bulk Comparison

View Instructions

You are here because you want to compare more than 1 model pairs.

```
For this to work; you should upload a zip folder that follows the following direct
   . <folder_name (any string)>
   | +-- <project_folder_name (any string)>
   | | +-- <base>
   | | | +-- <model_name>.(ecore/uml/emf)
   | | +-- <predicted>
   | | | +-- <model_name>.(ecore/uml/emf)
   |
   | +-- <project_folder_name (any string)>
   | +-- <base>
   | | +-- <model_name>.(ecore/uml/emf)
   | +-- <predicted>
   | | | +-- <model_name>.(ecore/uml/emf)
```

Download Sample Zip

Choose Base Model Type: ecore

Choose Predicted Model Type: ecore

Upload a zip file containing ecore model pairs

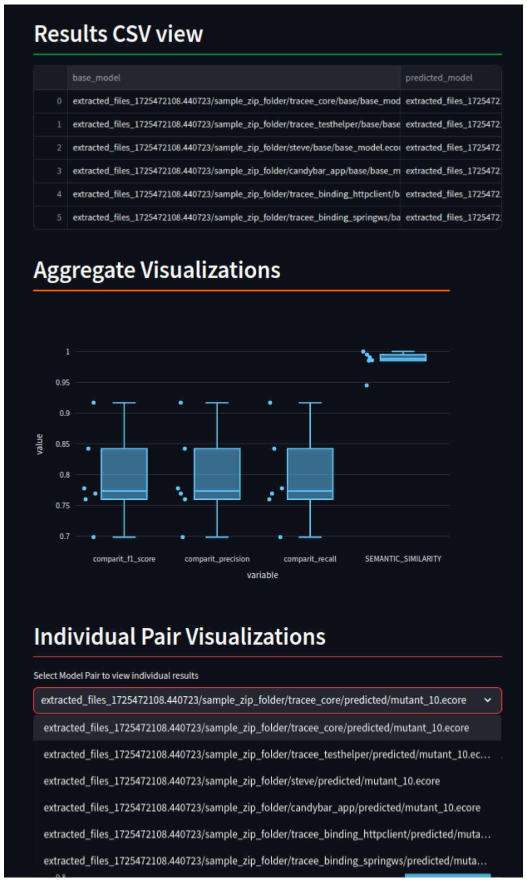
Drag and drop file here
Limit 1GB per file

Browse files

Configuration Panel

Use Hashing: True	Include Class Attributes: True	Include Attribute Name: True	Include References Name: True	Include Operation Name: True
Hashing Threshold: 0.95	Include Class References: True	Include Attribute Containing Class: True	Include References Containing Class: True	Include Operation Containing Class: True
Aggregate Class Level Metrics: True	Include Class Operations: True	Include Attribute Type: True	Include References Is Containment: True	Include Operation Parameters: True
	Include Class Supertypes	Include Attribute Lower Bound	Include References Lower Bound	Include Parameter Name

Figure 9.5: Bulk Comparison UI



Results CSV view

	base_model	predicted_model
0	extracted_files_1725472108.440723/sample_zip_folder/tracee_core/base/base_mod	extracted_files_1725472
1	extracted_files_1725472108.440723/sample_zip_folder/tracee_testhelper/base/base	extracted_files_1725472
2	extracted_files_1725472108.440723/sample_zip_folder/steve/base/base_model.ecoı	extracted_files_1725472
3	extracted_files_1725472108.440723/sample_zip_folder/candybar_app/base/base_m	extracted_files_1725472
4	extracted_files_1725472108.440723/sample_zip_folder/tracee_binding_httpclient/b	extracted_files_1725472
5	extracted_files_1725472108.440723/sample_zip_folder/tracee_binding_springws/ba	extracted_files_1725472

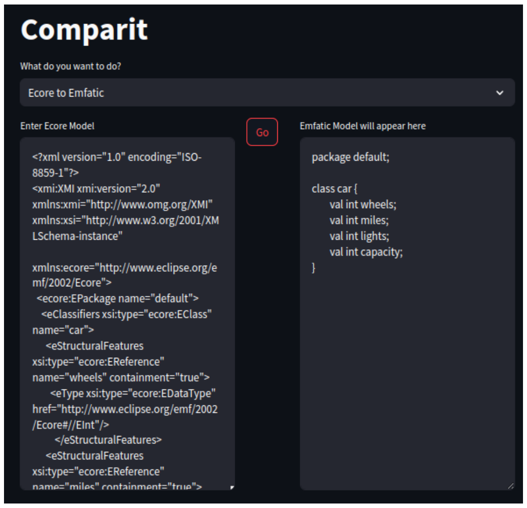
Aggregate Visualizations

Individual Pair Visualizations

Select Model Pair to view individual results

```
extracted_files_1725472108.440723/sample_zip_folder/tracee_core/predicted/mutant_10.ecore
extracted_files_1725472108.440723/sample_zip_folder/tracee_core/predicted/mutant_10.ecore
extracted_files_1725472108.440723/sample_zip_folder/tracee_testhelper/predicted/mutant_10.ec…
extracted_files_1725472108.440723/sample_zip_folder/steve/predicted/mutant_10.ecore
extracted_files_1725472108.440723/sample_zip_folder/candybar_app/predicted/mutant_10.ecore
extracted_files_1725472108.440723/sample_zip_folder/tracee_binding_httpclient/predicted/muta…
extracted_files_1725472108.440723/sample_zip_folder/tracee_binding_springws/predicted/muta…
```

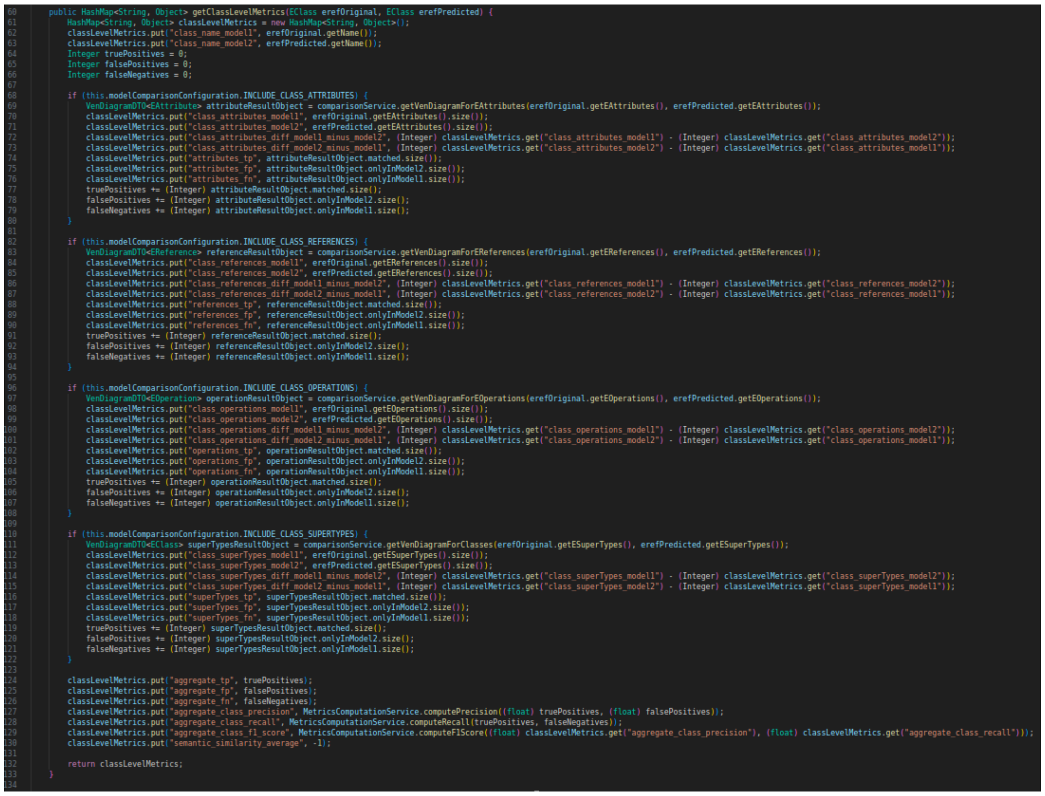
Figure 9.6: Bulk Comparison Results



```
<?xml version="1.0" encoding="ISO-
8859-1"?>
<xmi:XMI xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XM
LSchema-instance"

xmlns:ecore="http://www.eclipse.org/e
mf/2002/Ecore">
 <ecore:EPackage name="default">
  <eClassifiers xsi:type="ecore:EClass"
name="car">
   <eStructuralFeatures
xsi:type="ecore:EReference"
name="wheels" containment="true">
    <eType xsi:type="ecore:EDataType"
href="http://www.eclipse.org/emf/2002
/Ecore#//EInt"/>
     </eStructuralFeatures>
   <eStructuralFeatures
xsi:type="ecore:EReference"
name="miles" containment="true">
```

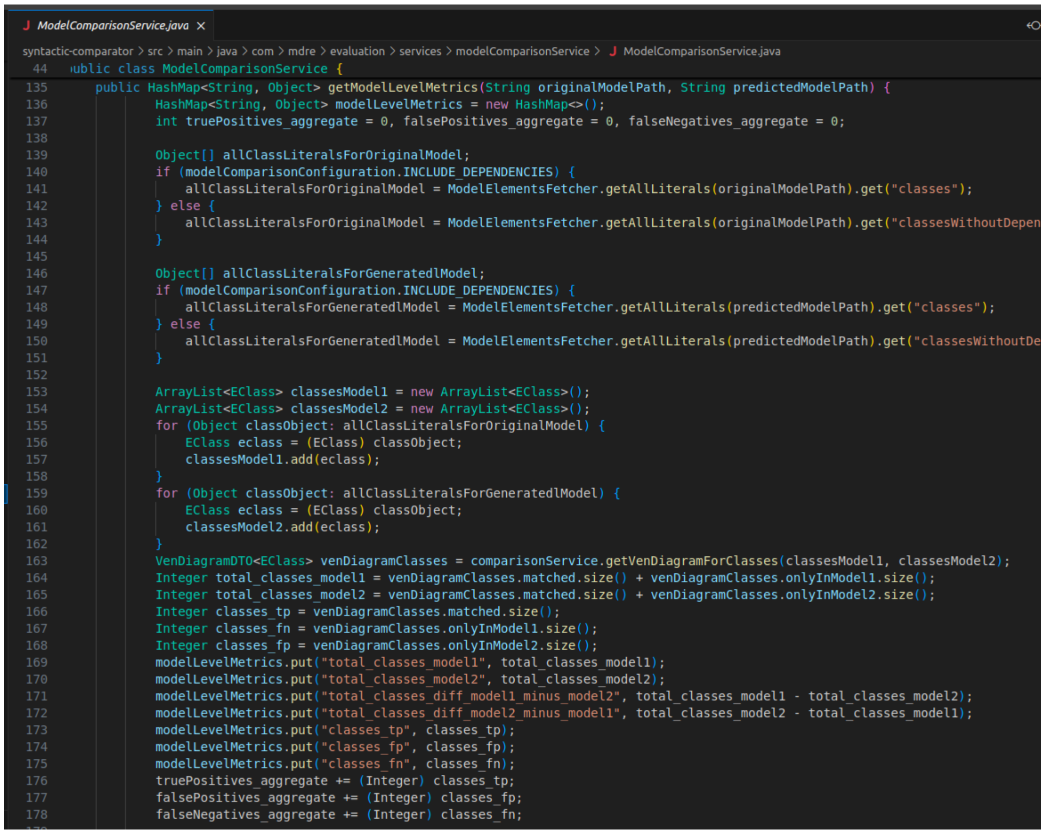
```
package default;

class car {
    val int wheels;
    val int miles;
    val int lights;
    val int capacity;
}
```

Figure 9.7: Model Conversion Ecore to Emfatic



```java
    public HashMap<String, Object> getClassLevelMetrics(EClass erefOriginal, EClass erefPredicted) {
        HashMap<String, Object> classLevelMetrics = new HashMap<String, Object>();
        classLevelMetrics.put("class_name_model1", erefOriginal.getName());
        classLevelMetrics.put("class_name_model2", erefPredicted.getName());
        Integer truePositives = 0;
        Integer falsePositives = 0;
        Integer falseNegatives = 0;

        if (this.modelComparisonConfiguration.INCLUDE_CLASS_ATTRIBUTES) {
            VenDiagramDTO<EAttribute> attributeResultObject = comparisonService.getVenDiagramForEAttributes(erefOriginal.getEAttributes(), erefPredicted.getEAttributes());
            classLevelMetrics.put("class_attributes_model1", erefOriginal.getEAttributes().size());
            classLevelMetrics.put("class_attributes_model2", erefPredicted.getEAttributes().size());
            classLevelMetrics.put("class_attributes_diff_model1_minus_model2", (Integer) classLevelMetrics.get("class_attributes_model1") - (Integer) classLevelMetrics.get("class_attributes_model2"));
            classLevelMetrics.put("class_attributes_diff_model2_minus_model1", (Integer) classLevelMetrics.get("class_attributes_model2") - (Integer) classLevelMetrics.get("class_attributes_model1"));
            classLevelMetrics.put("attributes_tp", attributeResultObject.matched.size());
            classLevelMetrics.put("attributes_fp", attributeResultObject.onlyInModel2.size());
            classLevelMetrics.put("attributes_fn", attributeResultObject.onlyInModel1.size());
            truePositives += (Integer) attributeResultObject.matched.size();
            falsePositives += (Integer) attributeResultObject.onlyInModel2.size();
            falseNegatives += (Integer) attributeResultObject.onlyInModel1.size();
        }

        if (this.modelComparisonConfiguration.INCLUDE_CLASS_REFERENCES) {
            VenDiagramDTO<EReference> referenceResultObject = comparisonService.getVenDiagramForEReferences(erefOriginal.getEReferences(), erefPredicted.getEReferences());
            classLevelMetrics.put("class_references_model1", erefOriginal.getEReferences().size());
            classLevelMetrics.put("class_references_model2", erefPredicted.getEReferences().size());
            classLevelMetrics.put("class_references_diff_model1_minus_model2", (Integer) classLevelMetrics.get("class_references_model1") - (Integer) classLevelMetrics.get("class_references_model2"));
            classLevelMetrics.put("class_references_diff_model2_minus_model1", (Integer) classLevelMetrics.get("class_references_model2") - (Integer) classLevelMetrics.get("class_references_model1"));
            classLevelMetrics.put("references_tp", referenceResultObject.matched.size());
            classLevelMetrics.put("references_fp", referenceResultObject.onlyInModel2.size());
            classLevelMetrics.put("references_fn", referenceResultObject.onlyInModel1.size());
            truePositives += (Integer) referenceResultObject.matched.size();
            falsePositives += (Integer) referenceResultObject.onlyInModel2.size();
            falseNegatives += (Integer) referenceResultObject.onlyInModel1.size();
        }

        if (this.modelComparisonConfiguration.INCLUDE_CLASS_OPERATIONS) {
            VenDiagramDTO<EOperation> operationResultObject = comparisonService.getVenDiagramForEOperations(erefOriginal.getEOperations(), erefPredicted.getEOperations());
            classLevelMetrics.put("class_operations_model1", erefOriginal.getEOperations().size());
            classLevelMetrics.put("class_operations_model2", erefPredicted.getEOperations().size());
            classLevelMetrics.put("class_operations_diff_model1_minus_model2", (Integer) classLevelMetrics.get("class_operations_model1") - (Integer) classLevelMetrics.get("class_operations_model2"));
            classLevelMetrics.put("class_operations_diff_model2_minus_model1", (Integer) classLevelMetrics.get("class_operations_model2") - (Integer) classLevelMetrics.get("class_operations_model1"));
            classLevelMetrics.put("operations_tp", operationResultObject.matched.size());
            classLevelMetrics.put("operations_fp", operationResultObject.onlyInModel2.size());
            classLevelMetrics.put("operations_fn", operationResultObject.onlyInModel1.size());
            truePositives += (Integer) operationResultObject.matched.size();
            falsePositives += (Integer) operationResultObject.onlyInModel2.size();
            falseNegatives += (Integer) operationResultObject.onlyInModel1.size();
        }

        if (this.modelComparisonConfiguration.INCLUDE_CLASS_SUPERTYPES) {
            VenDiagramDTO<EClass> superTypesResultObject = comparisonService.getVenDiagramForClasses(erefOriginal.getESuperTypes(), erefPredicted.getESuperTypes());
            classLevelMetrics.put("class_superTypes_model1", erefOriginal.getESuperTypes().size());
            classLevelMetrics.put("class_superTypes_model2", erefPredicted.getESuperTypes().size());
            classLevelMetrics.put("class_superTypes_diff_model1_minus_model2", (Integer) classLevelMetrics.get("class_superTypes_model1") - (Integer) classLevelMetrics.get("class_superTypes_model2"));
            classLevelMetrics.put("class_superTypes_diff_model2_minus_model1", (Integer) classLevelMetrics.get("class_superTypes_model2") - (Integer) classLevelMetrics.get("class_superTypes_model1"));
            classLevelMetrics.put("superTypes_tp", superTypesResultObject.matched.size());
            classLevelMetrics.put("superTypes_fp", superTypesResultObject.onlyInModel2.size());
            classLevelMetrics.put("superTypes_fn", superTypesResultObject.onlyInModel1.size());
            truePositives += (Integer) superTypesResultObject.matched.size();
            falsePositives += (Integer) superTypesResultObject.onlyInModel2.size();
            falseNegatives += (Integer) superTypesResultObject.onlyInModel1.size();
        }

        classLevelMetrics.put("aggregate_tp", truePositives);
        classLevelMetrics.put("aggregate_fp", falsePositives);
        classLevelMetrics.put("aggregate_fn", falseNegatives);
        classLevelMetrics.put("aggregate_class_precision", MetricsComputationService.computePrecision((float) truePositives, (float) falsePositives));
        classLevelMetrics.put("aggregate_class_recall", MetricsComputationService.computeRecall(truePositives, falseNegatives));
        classLevelMetrics.put("aggregate_class_f1_score", MetricsComputationService.computeF1Score((float) classLevelMetrics.get("aggregate_class_precision"), (float) classLevelMetrics.get("aggregate_class_recall")));
        classLevelMetrics.put("semantic_similarity_average", -1);

        return classLevelMetrics;
    }
```

Figure 9.8: Model Comparison Service Implementation (1/3)

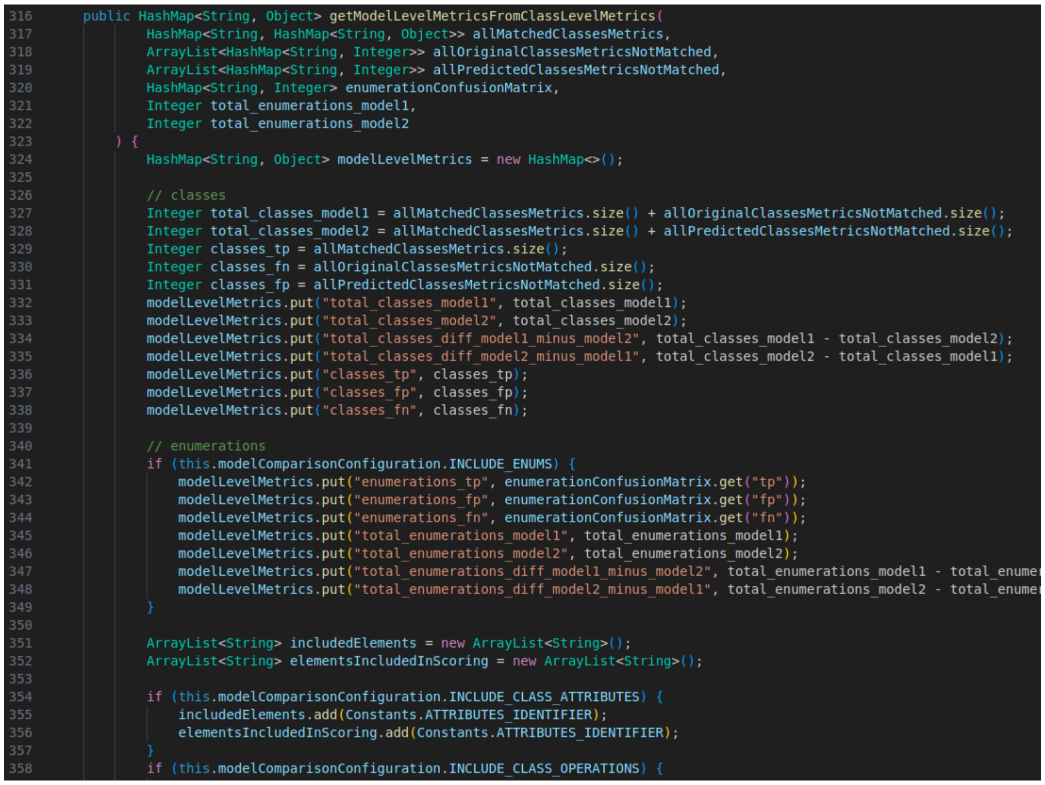


ModelComparisonService.java

syntactic-comparator > src > main > java > com > mdre > evaluation > services > modelComparisonService > ModelComparisonService.java

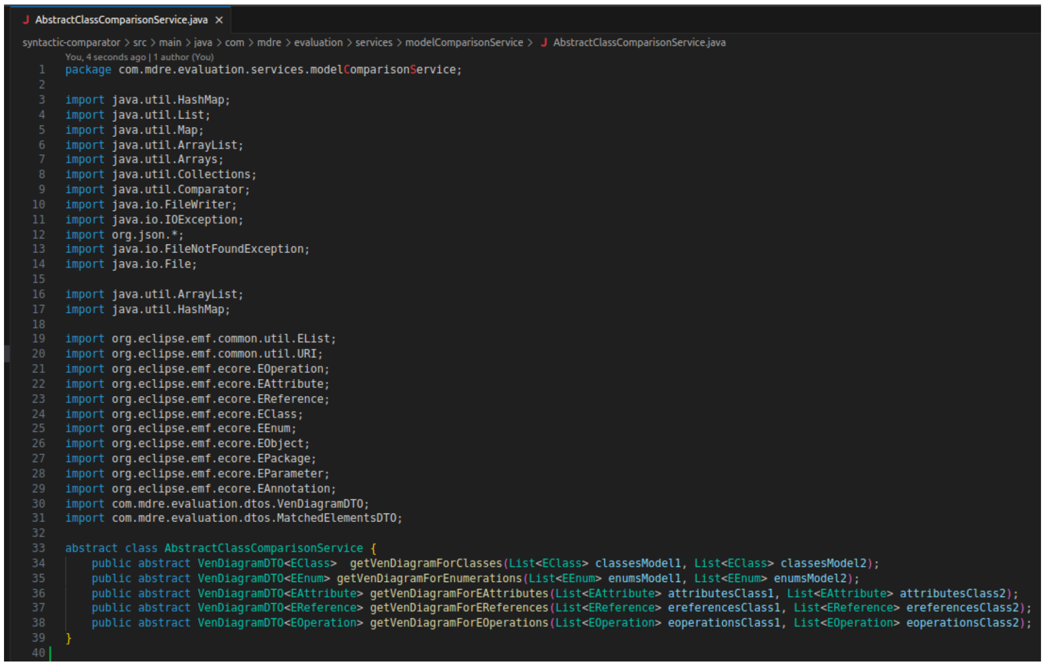
```java
ublic class ModelComparisonService {
    public HashMap<String, Object> getModelLevelMetrics(String originalModelPath, String predictedModelPath) {
            HashMap<String, Object> modelLevelMetrics = new HashMap<>();
            int truePositives_aggregate = 0, falsePositives_aggregate = 0, falseNegatives_aggregate = 0;

            Object[] allClassLiteralsForOriginalModel;
            if (modelComparisonConfiguration.INCLUDE_DEPENDENCIES) {
                allClassLiteralsForOriginalModel = ModelElementsFetcher.getAllLiterals(originalModelPath).get("classes");
            } else {
                allClassLiteralsForOriginalModel = ModelElementsFetcher.getAllLiterals(originalModelPath).get("classesWithoutDepen
            }

            Object[] allClassLiteralsForGeneratedlModel;
            if (modelComparisonConfiguration.INCLUDE_DEPENDENCIES) {
                allClassLiteralsForGeneratedlModel = ModelElementsFetcher.getAllLiterals(predictedModelPath).get("classes");
            } else {
                allClassLiteralsForGeneratedlModel = ModelElementsFetcher.getAllLiterals(predictedModelPath).get("classesWithoutDe
            }

            ArrayList<EClass> classesModel1 = new ArrayList<EClass>();
            ArrayList<EClass> classesModel2 = new ArrayList<EClass>();
            for (Object classObject: allClassLiteralsForOriginalModel) {
                EClass eclass = (EClass) classObject;
                classesModel1.add(eclass);
            }
            for (Object classObject: allClassLiteralsForGeneratedlModel) {
                EClass eclass = (EClass) classObject;
                classesModel2.add(eclass);
            }
            VenDiagramDTO<EClass> venDiagramClasses = comparisonService.getVenDiagramForClasses(classesModel1, classesModel2);
            Integer total_classes_model1 = venDiagramClasses.matched.size() + venDiagramClasses.onlyInModel1.size();
            Integer total_classes_model2 = venDiagramClasses.matched.size() + venDiagramClasses.onlyInModel2.size();
            Integer classes_tp = venDiagramClasses.matched.size();
            Integer classes_fn = venDiagramClasses.onlyInModel1.size();
            Integer classes_fp = venDiagramClasses.onlyInModel2.size();
            modelLevelMetrics.put("total_classes_model1", total_classes_model1);
            modelLevelMetrics.put("total_classes_model2", total_classes_model2);
            modelLevelMetrics.put("total_classes_diff_model1_minus_model2", total_classes_model1 - total_classes_model2);
            modelLevelMetrics.put("total_classes_diff_model2_minus_model1", total_classes_model2 - total_classes_model1);
            modelLevelMetrics.put("classes_tp", classes_tp);
            modelLevelMetrics.put("classes_fp", classes_fp);
            modelLevelMetrics.put("classes_fn", classes_fn);
            truePositives_aggregate += (Integer) classes_tp;
            falsePositives_aggregate += (Integer) classes_fp;
            falseNegatives_aggregate += (Integer) classes_fn;
```

Figure 9.9: Model Comparison Service Implementation (2/3)



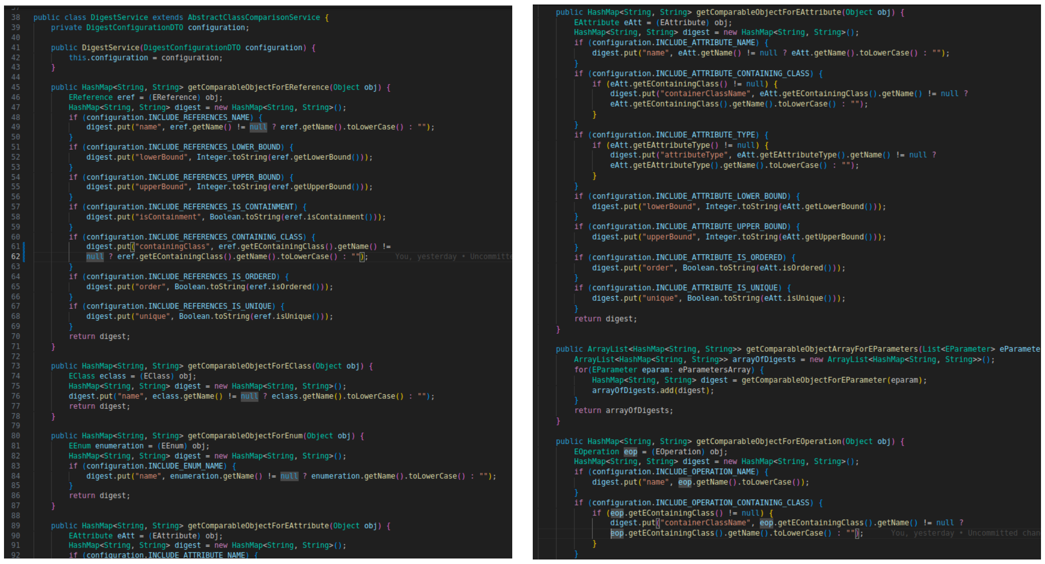
```java
    public HashMap<String, Object> getModelLevelMetricsFromClassLevelMetrics(
            HashMap<String, HashMap<String, Object>> allMatchedClassesMetrics,
            ArrayList<HashMap<String, Integer>> allOriginalClassesMetricsNotMatched,
            ArrayList<HashMap<String, Integer>> allPredictedClassesMetricsNotMatched,
            HashMap<String, Integer> enumerationConfusionMatrix,
            Integer total_enumerations_model1,
            Integer total_enumerations_model2
        ) {
            HashMap<String, Object> modelLevelMetrics = new HashMap<>();

            // classes
            Integer total_classes_model1 = allMatchedClassesMetrics.size() + allOriginalClassesMetricsNotMatched.size();
            Integer total_classes_model2 = allMatchedClassesMetrics.size() + allPredictedClassesMetricsNotMatched.size();
            Integer classes_tp = allMatchedClassesMetrics.size();
            Integer classes_fn = allOriginalClassesMetricsNotMatched.size();
            Integer classes_fp = allPredictedClassesMetricsNotMatched.size();
            modelLevelMetrics.put("total_classes_model1", total_classes_model1);
            modelLevelMetrics.put("total_classes_model2", total_classes_model2);
            modelLevelMetrics.put("total_classes_diff_model1_minus_model2", total_classes_model1 - total_classes_model2);
            modelLevelMetrics.put("total_classes_diff_model2_minus_model1", total_classes_model2 - total_classes_model1);
            modelLevelMetrics.put("classes_tp", classes_tp);
            modelLevelMetrics.put("classes_fp", classes_fp);
            modelLevelMetrics.put("classes_fn", classes_fn);

            // enumerations
            if (this.modelComparisonConfiguration.INCLUDE_ENUMS) {
                modelLevelMetrics.put("enumerations_tp", enumerationConfusionMatrix.get("tp"));
                modelLevelMetrics.put("enumerations_fp", enumerationConfusionMatrix.get("fp"));
                modelLevelMetrics.put("enumerations_fn", enumerationConfusionMatrix.get("fn"));
                modelLevelMetrics.put("total_enumerations_model1", total_enumerations_model1);
                modelLevelMetrics.put("total_enumerations_model2", total_enumerations_model2);
                modelLevelMetrics.put("total_enumerations_diff_model1_minus_model2", total_enumerations_model1 - total_enume
                modelLevelMetrics.put("total_enumerations_diff_model2_minus_model1", total_enumerations_model2 - total_enume
            }

            ArrayList<String> includedElements = new ArrayList<String>();
            ArrayList<String> elementsIncludedInScoring = new ArrayList<String>();

            if (this.modelComparisonConfiguration.INCLUDE_CLASS_ATTRIBUTES) {
                includedElements.add(Constants.ATTRIBUTES_IDENTIFIER);
                elementsIncludedInScoring.add(Constants.ATTRIBUTES_IDENTIFIER);
            }
            if (this.modelComparisonConfiguration.INCLUDE_CLASS_OPERATIONS) {
```

Figure 9.10: Model Comparison Service Implenetation (3/3)



```
AbstractClassComparisonService.java ×
syntactic-comparator > src > main > java > com > mdre > evaluation > services > modelComparisonService > AbstractClassComparisonService.java
You, 4 seconds ago | 1 author (You)
package com.mdre.evaluation.services.modelComparisonService;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.io.FileWriter;
import java.io.IOException;
import org.json.*;
import java.io.FileNotFoundException;
import java.io.File;

import java.util.ArrayList;
import java.util.HashMap;

import org.eclipse.emf.common.util.EList;
import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.ecore.EOperation;
import org.eclipse.emf.ecore.EAttribute;
import org.eclipse.emf.ecore.EReference;
import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.EEnum;
import org.eclipse.emf.ecore.EObject;
import org.eclipse.emf.ecore.EPackage;
import org.eclipse.emf.ecore.EParameter;
import org.eclipse.emf.ecore.EAnnotation;
import com.mdre.evaluation.dtos.VenDiagramDTO;
import com.mdre.evaluation.dtos.MatchedElementsDTO;

abstract class AbstractClassComparisonService {
    public abstract VenDiagramDTO<EClass>  getVenDiagramForClasses(List<EClass> classesModel1, List<EClass> classesModel2);
    public abstract VenDiagramDTO<EEnum> getVenDiagramForEnumerations(List<EEnum> enumsModel1, List<EEnum> enumsModel2);
    public abstract VenDiagramDTO<EAttribute> getVenDiagramForEAttributes(List<EAttribute> attributesClass1, List<EAttribute> attributesClass2);
    public abstract VenDiagramDTO<EReference> getVenDiagramForEReferences(List<EReference> ereferencesClass1, List<EReference> ereferencesClass2);
    public abstract VenDiagramDTO<EOperation> getVenDiagramForEOperations(List<EOperation> eoperationsClass1, List<EOperation> eoperationsClass2);
}
```

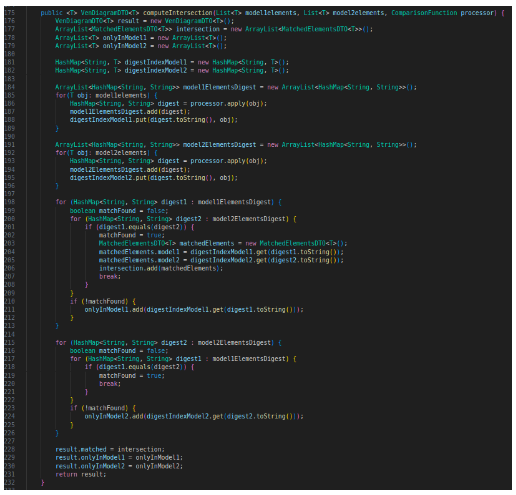
Figure 9.11: Abstract Class for Syntactic Comparison



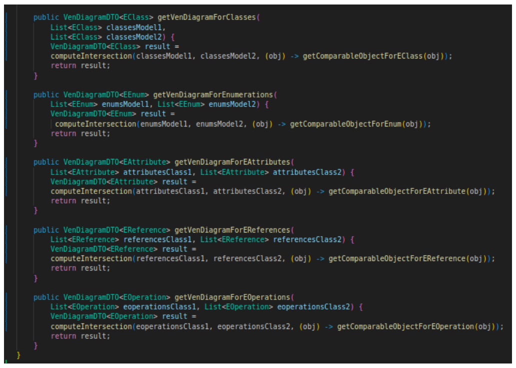
```
public class DigestService extends AbstractClassComparisonService {
    private DigestConfigurationDTO configuration;

    public DigestService(DigestConfigurationDTO configuration) {
        this.configuration = configuration;
    }

    public HashMap<String, String> getComparableObjectForEReference(Object obj) {
        EReference eref = (EReference) obj;
        HashMap<String, String> digest = new HashMap<String, String>();
        if (configuration.INCLUDE_REFERENCES_NAME) {
            digest.put("name", eref.getName() != null ? eref.getName().toLowerCase() : "");
        }
        if (configuration.INCLUDE_REFERENCES_LOWER_BOUND) {
            digest.put("lowerBound", Integer.toString(eref.getLowerBound()));
        }
        if (configuration.INCLUDE_REFERENCES_UPPER_BOUND) {
            digest.put("upperBound", Integer.toString(eref.getUpperBound()));
        }
        if (configuration.INCLUDE_REFERENCES_IS_CONTAINMENT) {
            digest.put("isContainment", Boolean.toString(eref.isContainment()));
        }
        if (configuration.INCLUDE_REFERENCES_CONTAINING_CLASS) {
            digest.put("containingClass", eref.getEContainingClass().getName() !=
            null ? eref.getEContainingClass().getName().toLowerCase() : "");      You, yesterday • Uncommitte
        }
        if (configuration.INCLUDE_REFERENCES_IS_ORDERED) {
            digest.put("order", Boolean.toString(eref.isOrdered()));
        }
        if (configuration.INCLUDE_REFERENCES_IS_UNIQUE) {
            digest.put("unique", Boolean.toString(eref.isUnique()));
        }
        return digest;
    }

    public HashMap<String, String> getComparableObjectForEClass(Object obj) {
        EClass eclass = (EClass) obj;
        HashMap<String, String> digest = new HashMap<String, String>();
        digest.put("name", eclass.getName() != null ? eclass.getName().toLowerCase() : "");
        return digest;
    }

    public HashMap<String, String> getComparableObjectForEnum(Object obj) {
        EEnum enumeration = (EEnum) obj;
        HashMap<String, String> digest = new HashMap<String, String>();
        if (configuration.INCLUDE_ENUM_NAME) {
            digest.put("name", enumeration.getName() != null ? enumeration.getName().toLowerCase() : "");
        }
        return digest;
    }

    public HashMap<String, String> getComparableObjectForEAttribute(Object obj) {
        EAttribute eAtt = (EAttribute) obj;
        HashMap<String, String> digest = new HashMap<String, String>();
        if (configuration.INCLUDE_ATTRIBUTE_NAME) {
```

```
    public HashMap<String, String> getComparableObjectForEAttribute(Object obj) {
        EAttribute eAtt = (EAttribute) obj;
        HashMap<String, String> digest = new HashMap<String, String>();
        if (configuration.INCLUDE_ATTRIBUTE_NAME) {
            digest.put("name", eAtt.getName() != null ? eAtt.getName().toLowerCase() : "");
        }
        if (configuration.INCLUDE_ATTRIBUTE_CONTAINING_CLASS) {
            if (eAtt.getEContainingClass() != null) {
                digest.put("containerClassName", eAtt.getEContainingClass().getName() != null ?
                eAtt.getEContainingClass().getName().toLowerCase() : "");
            }
        }
        if (configuration.INCLUDE_ATTRIBUTE_TYPE) {
            if (eAtt.getEAttributeType() != null) {
                digest.put("attributeType", eAtt.getEAttributeType().getName() != null ?
                eAtt.getEAttributeType().getName().toLowerCase() : "");
            }
        }
        if (configuration.INCLUDE_ATTRIBUTE_LOWER_BOUND) {
            digest.put("lowerBound", Integer.toString(eAtt.getLowerBound()));
        }
        if (configuration.INCLUDE_ATTRIBUTE_UPPER_BOUND) {
            digest.put("upperBound", Integer.toString(eAtt.getUpperBound()));
        }
        if (configuration.INCLUDE_ATTRIBUTE_IS_ORDERED) {
            digest.put("order", Boolean.toString(eAtt.isOrdered()));
        }
        if (configuration.INCLUDE_ATTRIBUTE_IS_UNIQUE) {
            digest.put("unique", Boolean.toString(eAtt.isUnique()));
        }
        return digest;
    }

    public ArrayList<HashMap<String, String>> getComparableObjectArrayForEParameters(List<EParameter> eParamete
        ArrayList<HashMap<String, String>> arrayOfDigests = new ArrayList<HashMap<String, String>>();
        for(EParameter eparam: eParametersArray) {
            HashMap<String, String> digest = getComparableObjectForEParameter(eparam);
            arrayOfDigests.add(digest);
        }
        return arrayOfDigests;
    }

    public HashMap<String, String> getComparableObjectForEOperation(Object obj) {
        EOperation eop = (EOperation) obj;
        HashMap<String, String> digest = new HashMap<String, String>();
        if (configuration.INCLUDE_OPERATION_NAME) {
            digest.put("name", eop.getName().toLowerCase());
        }
        if (configuration.INCLUDE_OPERATION_CONTAINING_CLASS) {
            if (eop.getEContainingClass() != null) {
                digest.put("containerClassName", eop.getEContainingClass().getName() != null ?
                eop.getEContainingClass().getName().toLowerCase() : "");      You, yesterday • Uncommitted chan
            }
        }
```

Figure 9.12: Digest Based Algorithm Implementation (figure 1/2)

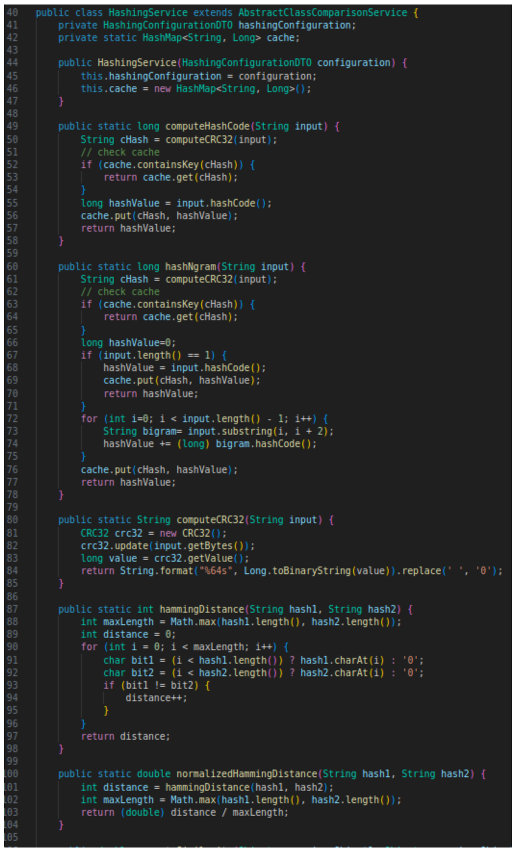


```java
    public <T> VenDiagramDTO<T> computeIntersection(List<T> modellelements, List<T> model2elements, ComparisonFunction processor) {
        VenDiagramDTO<T> result = new VenDiagramDTO<T>();
        ArrayList<MatchedElementsDTO<T>> intersection = new ArrayList<MatchedElementsDTO<T>>();
        ArrayList<T> onlyInModel1 = new ArrayList<T>();
        ArrayList<T> onlyInModel2 = new ArrayList<T>();

        HashMap<String, T> digestIndexModel1 = new HashMap<String, T>();
        HashMap<String, T> digestIndexModel2 = new HashMap<String, T>();

        ArrayList<HashMap<String, String>> model1ElementsDigest = new ArrayList<HashMap<String, String>>();
        for(T obj: modellelements) {
            HashMap<String, String> digest = processor.apply(obj);
            model1ElementsDigest.add(digest);
            digestIndexModel1.put(digest.toString(), obj);
        }

        ArrayList<HashMap<String, String>> model2ElementsDigest = new ArrayList<HashMap<String, String>>();
        for(T obj: model2elements) {
            HashMap<String, String> digest = processor.apply(obj);
            model2ElementsDigest.add(digest);
            digestIndexModel2.put(digest.toString(), obj);
        }

        for (HashMap<String, String> digest1 : model1ElementsDigest) {
            boolean matchFound = false;
            for (HashMap<String, String> digest2 : model2ElementsDigest) {
                if (digest1.equals(digest2)) {
                    matchFound = true;
                    MatchedElementsDTO<T> matchedElements = new MatchedElementsDTO<T>();
                    matchedElements.model1 = digestIndexModel1.get(digest1.toString());
                    matchedElements.model2 = digestIndexModel2.get(digest2.toString());
                    intersection.add(matchedElements);
                    break;
                }
            }
            if (!matchFound) {
                onlyInModel1.add(digestIndexModel1.get(digest1.toString()));
            }
        }

        for (HashMap<String, String> digest2 : model2ElementsDigest) {
            boolean matchFound = false;
            for (HashMap<String, String> digest1 : model1ElementsDigest) {
                if (digest1.equals(digest2)) {
                    matchFound = true;
                    break;
                }
            }
            if (!matchFound) {
                onlyInModel2.add(digestIndexModel2.get(digest2.toString()));
            }
        }

        result.matched = intersection;
        result.onlyInModel1 = onlyInModel1;
        result.onlyInModel2 = onlyInModel2;
        return result;
    }
```

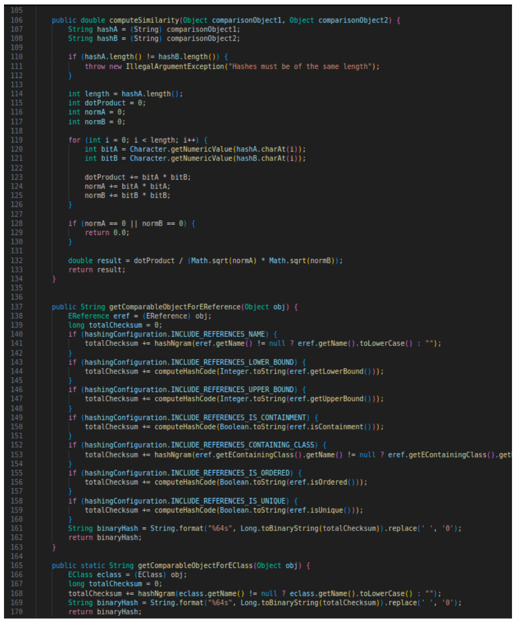


```java
    public VenDiagramDTO<EClass> getVenDiagramForClasses(
        List<EClass> classesModel1,
        List<EClass> classesModel2) {
        VenDiagramDTO<EClass> result =
        computeIntersection(classesModel1, classesModel2, (obj) -> getComparableObjectForEClass(obj));
        return result;
    }

    public VenDiagramDTO<EEnum> getVenDiagramForEnumerations(
        List<EEnum> enumsModel1, List<EEnum> enumsModel2) {
        VenDiagramDTO<EEnum> result =
         computeIntersection(enumsModel1, enumsModel2, (obj) -> getComparableObjectForEnum(obj));
        return result;
    }

    public VenDiagramDTO<EAttribute> getVenDiagramForEAttributes(
        List<EAttribute> attributesClass1, List<EAttribute> attributesClass2) {
        VenDiagramDTO<EAttribute> result =
        computeIntersection(attributesClass1, attributesClass2, (obj) -> getComparableObjectForEAttribute(obj));
        return result;
    }

    public VenDiagramDTO<EReference> getVenDiagramForEReferences(
        List<EReference> referencesClass1, List<EReference> referencesClass2) {
        VenDiagramDTO<EReference> result =
        computeIntersection(referencesClass1, referencesClass2, (obj) -> getComparableObjectForEReference(obj));
        return result;
    }

    public VenDiagramDTO<EOperation> getVenDiagramForEOperations(
        List<EOperation> eoperationsClass1, List<EOperation> eoperationsClass2) {
        VenDiagramDTO<EOperation> result =
        computeIntersection(eoperationsClass1, eoperationsClass2, (obj) -> getComparableObjectForEOperation(obj));
        return result;
    }
}
```

Figure 9.13: Digest Based Algorithm Implementation (figure 2/2)

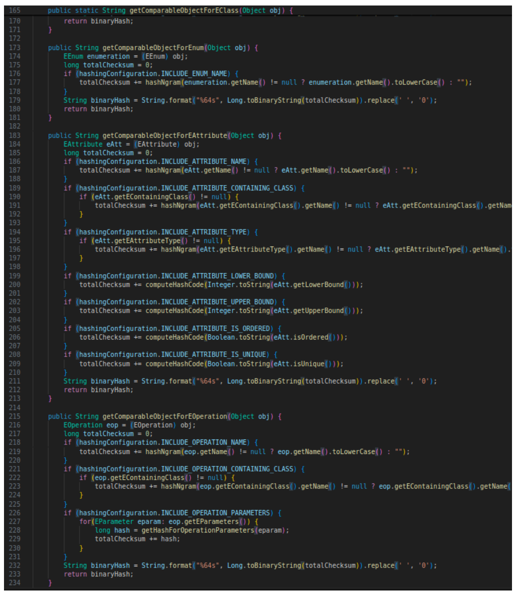


```java
public class HashingService extends AbstractClassComparisonService {
    private HashingConfigurationDTO hashingConfiguration;
    private static HashMap<String, Long> cache;

    public HashingService(HashingConfigurationDTO configuration) {
        this.hashingConfiguration = configuration;
        this.cache = new HashMap<String, Long>();
    }

    public static long computeHashCode(String input) {
        String cHash = computeCRC32(input);
        // check cache
        if (cache.containsKey(cHash)) {
            return cache.get(cHash);
        }
        long hashValue = input.hashCode();
        cache.put(cHash, hashValue);
        return hashValue;
    }

    public static long hashNgram(String input) {
        String cHash = computeCRC32(input);
        // check cache
        if (cache.containsKey(cHash)) {
            return cache.get(cHash);
        }
        long hashValue=0;
        if (input.length() == 1) {
            hashValue = input.hashCode();
            cache.put(cHash, hashValue);
            return hashValue;
        }
        for (int i=0; i < input.length() - 1; i++) {
            String bigram= input.substring(i, i + 2);
            hashValue += (long) bigram.hashCode();
        }
        cache.put(cHash, hashValue);
        return hashValue;
    }

    public static String computeCRC32(String input) {
        CRC32 crc32 = new CRC32();
        crc32.update(input.getBytes());
        long value = crc32.getValue();
        return String.format("%64s", Long.toBinaryString(value)).replace(' ', '0');
    }

    public static int hammingDistance(String hash1, String hash2) {
        int maxLength = Math.max(hash1.length(), hash2.length());
        int distance = 0;
        for (int i = 0; i < maxLength; i++) {
            char bit1 = (i < hash1.length()) ? hash1.charAt(i) : '0';
            char bit2 = (i < hash2.length()) ? hash2.charAt(i) : '0';
            if (bit1 != bit2) {
                distance++;
            }
        }
        return distance;
    }

    public static double normalizedHammingDistance(String hash1, String hash2) {
        int distance = hammingDistance(hash1, hash2);
        int maxLength = Math.max(hash1.length(), hash2.length());
        return (double) distance / maxLength;
    }

```

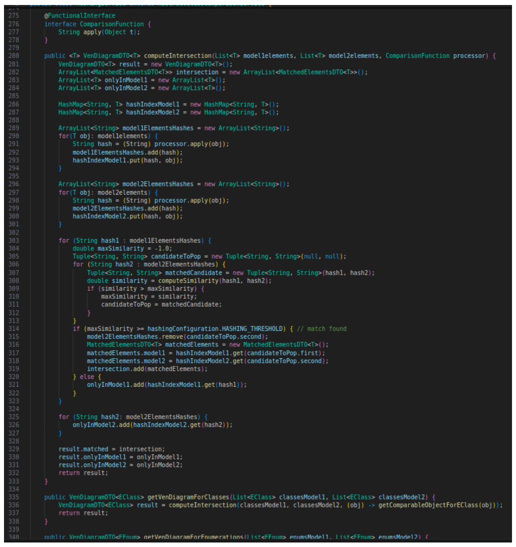


```java

    public double computeSimilarity(Object comparisonObject1, Object comparisonObject2) {
        String hashA = (String) comparisonObject1;
        String hashB = (String) comparisonObject2;

        if (hashA.length() != hashB.length()) {
            throw new IllegalArgumentException("Hashes must be of the same length");
        }

        int length = hashA.length();
        int dotProduct = 0;
        int normA = 0;
        int normB = 0;

        for (int i = 0; i < length; i++) {
            int bitA = Character.getNumericValue(hashA.charAt(i));
            int bitB = Character.getNumericValue(hashB.charAt(i));

            dotProduct += bitA * bitB;
            normA += bitA * bitA;
            normB += bitB * bitB;
        }

        if (normA == 0 || normB == 0) {
            return 0.0;
        }

        double result = dotProduct / (Math.sqrt(normA) * Math.sqrt(normB));
        return result;
    }


    public String getComparableObjectForEReference(Object obj) {
        EReference eref = (EReference) obj;
        long totalChecksum = 0;
        if (hashingConfiguration.INCLUDE_REFERENCES_NAME) {
            totalChecksum += hashNgram(eref.getName() != null ? eref.getName().toLowerCase() : "");
        }
        if (hashingConfiguration.INCLUDE_REFERENCES_LOWER_BOUND) {
            totalChecksum += computeHashCode(Integer.toString(eref.getLowerBound()));
        }
        if (hashingConfiguration.INCLUDE_REFERENCES_UPPER_BOUND) {
            totalChecksum += computeHashCode(Integer.toString(eref.getUpperBound()));
        }
        if (hashingConfiguration.INCLUDE_REFERENCES_IS_CONTAINMENT) {
            totalChecksum += computeHashCode(Boolean.toString(eref.isContainment()));
        }
        if (hashingConfiguration.INCLUDE_REFERENCES_CONTAINING_CLASS) {
            totalChecksum += hashNgram(eref.getEContainingClass().getName() != null ? eref.getEContainingClass().get
        }
        if (hashingConfiguration.INCLUDE_REFERENCES_IS_ORDERED) {
            totalChecksum += computeHashCode(Boolean.toString(eref.isOrdered()));
        }
        if (hashingConfiguration.INCLUDE_REFERENCES_IS_UNIQUE) {
            totalChecksum += computeHashCode(Boolean.toString(eref.isUnique()));
        }
        String binaryHash = String.format("%64s", Long.toBinaryString(totalChecksum)).replace(' ', '0');
        return binaryHash;
    }

    public static String getComparableObjectForEClass(Object obj) {
        EClass eclass = (EClass) obj;
        long totalChecksum = 0;
        totalChecksum += hashNgram(eclass.getName() != null ? eclass.getName().toLowerCase() : "");
        String binaryHash = String.format("%64s", Long.toBinaryString(totalChecksum)).replace(' ', '0');
        return binaryHash;
```

Figure 9.14: Hashing Based Algorithm Implementation (figure 2/2)

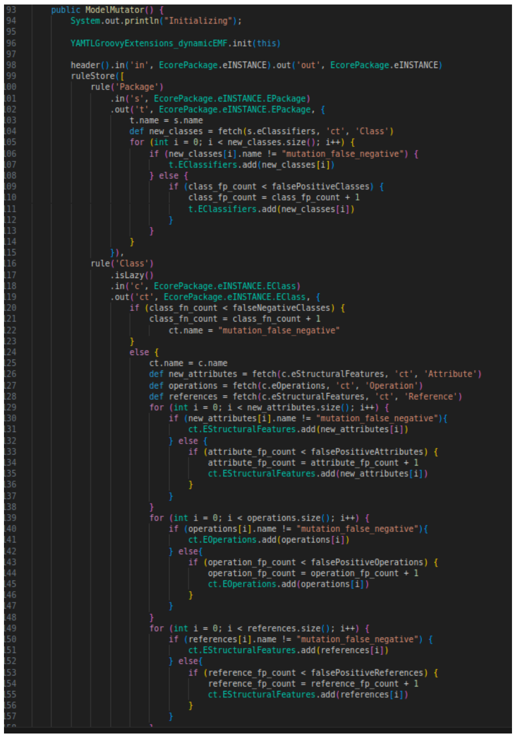


```java
    public static String getComparableObjectForEClass(Object obj) {
        return binaryHash;
    }

    public String getComparableObjectForEnum(Object obj) {
        EEnum enumeration = (EEnum) obj;
        long totalChecksum = 0;
        if (hashingConfiguration.INCLUDE_ENUM_NAME) {
            totalChecksum += hashNgram(enumeration.getName() != null ? enumeration.getName().toLowerCase() : "");
        }
        String binaryHash = String.format("%64s", Long.toBinaryString(totalChecksum)).replace(' ', '0');
        return binaryHash;
    }

    public String getComparableObjectForEAttribute(Object obj) {
        EAttribute eAtt = (EAttribute) obj;
        long totalChecksum = 0;
        if (hashingConfiguration.INCLUDE_ATTRIBUTE_NAME) {
            totalChecksum += hashNgram(eAtt.getName() != null ? eAtt.getName().toLowerCase() : "");
        }
        if (hashingConfiguration.INCLUDE_ATTRIBUTE_CONTAINING_CLASS) {
            if (eAtt.getEContainingClass() != null) {
                totalChecksum += hashNgram(eAtt.getEContainingClass().getName() != null ? eAtt.getEContainingClass().getNam
            }
        }
        if (hashingConfiguration.INCLUDE_ATTRIBUTE_TYPE) {
            if (eAtt.getEAttributeType() != null) {
                totalChecksum += hashNgram(eAtt.getEAttributeType().getName() != null ? eAtt.getEAttributeType().getName().
            }
        }
        if (hashingConfiguration.INCLUDE_ATTRIBUTE_LOWER_BOUND) {
            totalChecksum += computeHashCode(Integer.toString(eAtt.getLowerBound()));
        }
        if (hashingConfiguration.INCLUDE_ATTRIBUTE_UPPER_BOUND) {
            totalChecksum += computeHashCode(Integer.toString(eAtt.getUpperBound()));
        }
        if (hashingConfiguration.INCLUDE_ATTRIBUTE_IS_ORDERED) {
            totalChecksum += computeHashCode(Boolean.toString(eAtt.isOrdered()));
        }
        if (hashingConfiguration.INCLUDE_ATTRIBUTE_IS_UNIQUE) {
            totalChecksum += computeHashCode(Boolean.toString(eAtt.isUnique()));
        }
        String binaryHash = String.format("%64s", Long.toBinaryString(totalChecksum)).replace(' ', '0');
        return binaryHash;
    }

    public String getComparableObjectForEOperation(Object obj) {
        EOperation eop = (EOperation) obj;
        long totalChecksum = 0;
        if (hashingConfiguration.INCLUDE_OPERATION_NAME) {
            totalChecksum += hashNgram(eop.getName() != null ? eop.getName().toLowerCase() : "");
        }
        if (hashingConfiguration.INCLUDE_OPERATION_CONTAINING_CLASS) {
            if (eop.getEContainingClass() != null) {
                totalChecksum += hashNgram(eop.getEContainingClass().getName() != null ? eop.getEContainingClass().getName(
            }
        }
        if (hashingConfiguration.INCLUDE_OPERATION_PARAMETERS) {
            for(EParameter eparam: eop.getEParameters()) {
                long hash = getHashForOperationParameters(eparam);
                totalChecksum += hash;
            }
        }
        String binaryHash = String.format("%64s", Long.toBinaryString(totalChecksum)).replace(' ', '0');
        return binaryHash;
    }
```

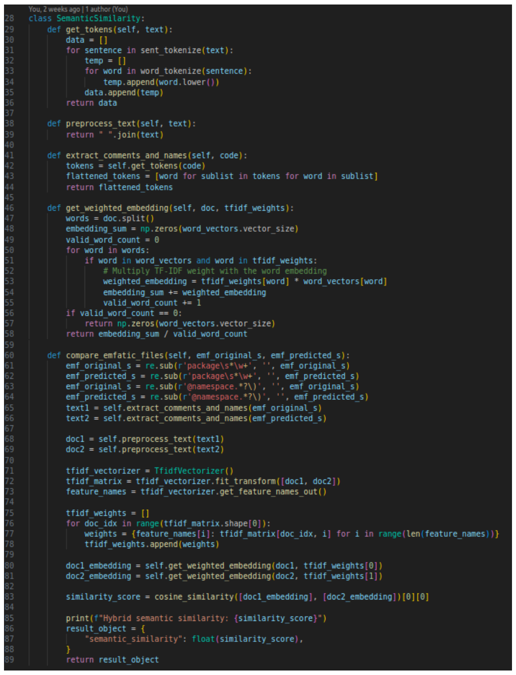


```java
    @FunctionalInterface
    interface ComparisonFunction {
        String apply(Object t);
    }

    public <T> VenDiagramDTO<T> computeIntersection(List<T> model1elements, List<T> model2elements, ComparisonFunction processor) {
        VenDiagramDTO<T> result = new VenDiagramDTO<T>();
        ArrayList<MatchedElementsDTO<T>> intersection = new ArrayList<MatchedElementsDTO<T>>();
        ArrayList<T> onlyInModel1 = new ArrayList<T>();
        ArrayList<T> onlyInModel2 = new ArrayList<T>();

        HashMap<String, T> hashIndexModel1 = new HashMap<String, T>();
        HashMap<String, T> hashIndexModel2 = new HashMap<String, T>();

        ArrayList<String> model1ElementsHashes = new ArrayList<String>();
        for(T obj: model1elements) {
            String hash = (String) processor.apply(obj);
            model1ElementsHashes.add(hash);
            hashIndexModel1.put(hash, obj);
        }

        ArrayList<String> model2ElementsHashes = new ArrayList<String>();
        for(T obj: model2elements) {
            String hash = (String) processor.apply(obj);
            model2ElementsHashes.add(hash);
            hashIndexModel2.put(hash, obj);
        }

        for (String hash1 : model1ElementsHashes) {
            double maxSimilarity = -1.0;
            Tuple<String, String> candidateToPop = new Tuple<String, String>(null, null);
            for (String hash2 : model2ElementsHashes) {
                Tuple<String, String> matchedCandidate = new Tuple<String, String>(hash1, hash2);
                double similarity = computeSimilarity(hash1, hash2);
                if (similarity > maxSimilarity) {
                    maxSimilarity = similarity;
                    candidateToPop = matchedCandidate;
                }
            }
            if (maxSimilarity >= hashingConfiguration.HASHING_THRESHOLD) { // match found
                model2ElementsHashes.remove(candidateToPop.second);
                MatchedElementsDTO<T> matchedElements = new MatchedElementsDTO<T>();
                matchedElements.model1 = hashIndexModel1.get(candidateToPop.first);
                matchedElements.model2 = hashIndexModel2.get(candidateToPop.second);
                intersection.add(matchedElements);
            } else {
                onlyInModel1.add(hashIndexModel1.get(hash1));
            }
        }

        for (String hash2: model2ElementsHashes) {
            onlyInModel2.add(hashIndexModel2.get(hash2));
        }

        result.matched = intersection;
        result.onlyInModel1 = onlyInModel1;
        result.onlyInModel2 = onlyInModel2;
        return result;
    }

    public VenDiagramDTO<EClass> getVenDiagramForClasses(List<EClass> classesModel1, List<EClass> classesModel2) {
        VenDiagramDTO<EClass> result = computeIntersection(classesModel1, classesModel2, (obj) -> getComparableObjectForEClass(obj));
        return result;
    }

    public VenDiagramDTO<EEnum> getVenDiagramForEnumerations(List<EEnum> enumsModel1, List<EEnum> enumsModel2) {
```

Figure 9.15: Hashing Based Algorithm Implementation (figure 2/2)

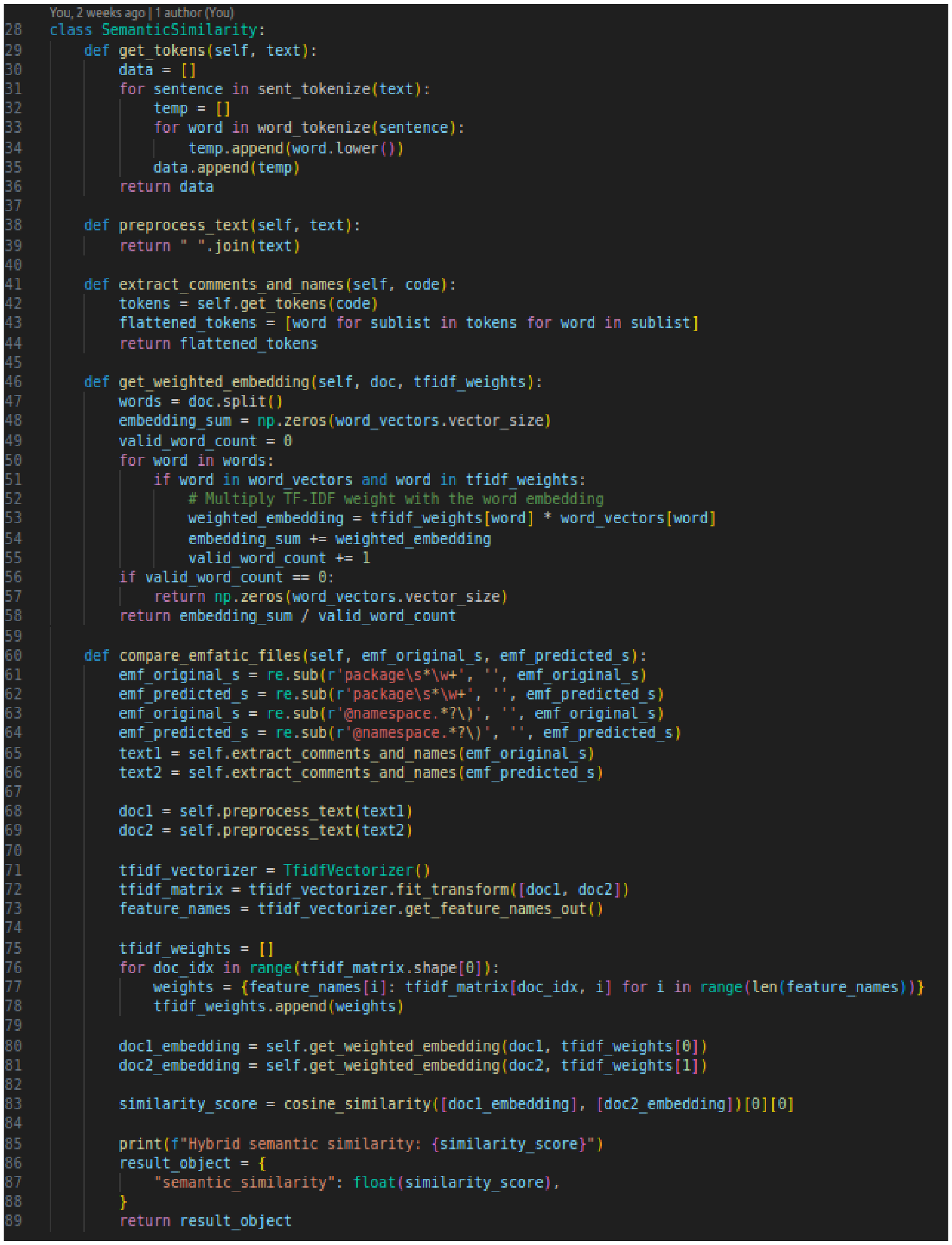


```
public ModelMutator() {
    System.out.println("Initializing");

    YAMTLGroovyExtensions_dynamicEMF.init(this)

    header().in('in', EcorePackage.eINSTANCE).out('out', EcorePackage.eINSTANCE)
    ruleStore([
        rule('Package')
            .in('s', EcorePackage.eINSTANCE.EPackage)
            .out('t', EcorePackage.eINSTANCE.EPackage, {
                t.name = s.name
                def new_classes = fetch(s.eClassifiers, 'ct', 'Class')
                for (int i = 0; i < new_classes.size(); i++) {
                    if (new_classes[i].name != "mutation_false_negative") {
                        t.EClassifiers.add(new_classes[i])
                    } else {
                        if (class_fp_count < falsePositiveClasses) {
                            class_fp_count = class_fp_count + 1
                            t.EClassifiers.add(new_classes[i])
                        }
                    }
                }
            }),
        rule('Class')
            .isLazy()
            .in('c', EcorePackage.eINSTANCE.EClass)
            .out('ct', EcorePackage.eINSTANCE.EClass, {
                if (class_fn_count < falseNegativeClasses) {
                    class_fn_count = class_fn_count + 1
                        ct.name = "mutation_false_negative"
                }
                else {
                    ct.name = c.name
                    def new_attributes = fetch(c.eStructuralFeatures, 'ct', 'Attribute')
                    def operations = fetch(c.eOperations, 'ct', 'Operation')
                    def references = fetch(c.eStructuralFeatures, 'ct', 'Reference')
                    for (int i = 0; i < new_attributes.size(); i++) {
                        if (new_attributes[i].name != "mutation_false_negative"){
                            ct.EStructuralFeatures.add(new_attributes[i])
                        } else {
                            if (attribute_fp_count < falsePositiveAttributes) {
                                attribute_fp_count = attribute_fp_count + 1
                                ct.EStructuralFeatures.add(new_attributes[i])
                            }
                        }
                    }
                    for (int i = 0; i < operations.size(); i++) {
                        if (operations[i].name != "mutation_false_negative"){
                            ct.EOperations.add(operations[i])
                        } else{
                            if (operation_fp_count < falsePositiveOperations) {
                                operation_fp_count = operation_fp_count + 1
                                ct.EOperations.add(operations[i])
                            }
                        }
                    }
                    for (int i = 0; i < references.size(); i++) {
                        if (references[i].name != "mutation_false_negative") {
                            ct.EStructuralFeatures.add(references[i])
                        } else{
                            if (reference_fp_count < falsePositiveReferences) {
                                reference_fp_count = reference_fp_count + 1
                                ct.EStructuralFeatures.add(references[i])
                            }
                        }
```



```
class SemanticSimilarity:
    def get_tokens(self, text):
        data = []
        for sentence in sent_tokenize(text):
            temp = []
            for word in word_tokenize(sentence):
                temp.append(word.lower())
            data.append(temp)
        return data

    def preprocess_text(self, text):
        return " ".join(text)

    def extract_comments_and_names(self, code):
        tokens = self.get_tokens(code)
        flattened_tokens = [word for sublist in tokens for word in sublist]
        return flattened_tokens

    def get_weighted_embedding(self, doc, tfidf_weights):
        words = doc.split()
        embedding_sum = np.zeros(word_vectors.vector_size)
        valid_word_count = 0
        for word in words:
            if word in word_vectors and word in tfidf_weights:
                # Multiply TF-IDF weight with the word embedding
                weighted_embedding = tfidf_weights[word] * word_vectors[word]
                embedding_sum += weighted_embedding
                valid_word_count += 1
        if valid_word_count == 0:
            return np.zeros(word_vectors.vector_size)
        return embedding_sum / valid_word_count

    def compare_emfatic_files(self, emf_original_s, emf_predicted_s):
        emf_original_s = re.sub(r'package\s*\w+', '', emf_original_s)
        emf_predicted_s = re.sub(r'package\s*\w+', '', emf_predicted_s)
        emf_original_s = re.sub(r'@namespace.*?\)', '', emf_original_s)
        emf_predicted_s = re.sub(r'@namespace.*?\)', '', emf_predicted_s)
        text1 = self.extract_comments_and_names(emf_original_s)
        text2 = self.extract_comments_and_names(emf_predicted_s)

        doc1 = self.preprocess_text(text1)
        doc2 = self.preprocess_text(text2)

        tfidf_vectorizer = TfidfVectorizer()
        tfidf_matrix = tfidf_vectorizer.fit_transform([doc1, doc2])
        feature_names = tfidf_vectorizer.get_feature_names_out()

        tfidf_weights = []
        for doc_idx in range(tfidf_matrix.shape[0]):
            weights = {feature_names[i]: tfidf_matrix[doc_idx, i] for i in range(len(feature_names))}
            tfidf_weights.append(weights)

        doc1_embedding = self.get_weighted_embedding(doc1, tfidf_weights[0])
        doc2_embedding = self.get_weighted_embedding(doc2, tfidf_weights[1])

        similarity_score = cosine_similarity([doc1_embedding], [doc2_embedding])[0][0]

        print(f"Hybrid semantic similarity: {similarity_score}")
        result_object = {
            "semantic_similarity": float(similarity_score),
        }
        return result_object
```

Figure 9.16: Model Mutator



```
You, 2 weeks ago | 1 author (You)
class SemanticSimilarity:
    def get_tokens(self, text):
        data = []
        for sentence in sent_tokenize(text):
            temp = []
            for word in word_tokenize(sentence):
                temp.append(word.lower())
            data.append(temp)
        return data

    def preprocess_text(self, text):
        return " ".join(text)

    def extract_comments_and_names(self, code):
        tokens = self.get_tokens(code)
        flattened_tokens = [word for sublist in tokens for word in sublist]
        return flattened_tokens

    def get_weighted_embedding(self, doc, tfidf_weights):
        words = doc.split()
        embedding_sum = np.zeros(word_vectors.vector_size)
        valid_word_count = 0
        for word in words:
            if word in word_vectors and word in tfidf_weights:
                # Multiply TF-IDF weight with the word embedding
                weighted_embedding = tfidf_weights[word] * word_vectors[word]
                embedding_sum += weighted_embedding
                valid_word_count += 1
        if valid_word_count == 0:
            return np.zeros(word_vectors.vector_size)
        return embedding_sum / valid_word_count

    def compare_emfatic_files(self, emf_original_s, emf_predicted_s):
        emf_original_s = re.sub(r'package\s*\w+', '', emf_original_s)
        emf_predicted_s = re.sub(r'package\s*\w+', '', emf_predicted_s)
        emf_original_s = re.sub(r'@namespace.*?\)', '', emf_original_s)
        emf_predicted_s = re.sub(r'@namespace.*?\)', '', emf_predicted_s)
        text1 = self.extract_comments_and_names(emf_original_s)
        text2 = self.extract_comments_and_names(emf_predicted_s)

        doc1 = self.preprocess_text(text1)
        doc2 = self.preprocess_text(text2)

        tfidf_vectorizer = TfidfVectorizer()
        tfidf_matrix = tfidf_vectorizer.fit_transform([doc1, doc2])
        feature_names = tfidf_vectorizer.get_feature_names_out()

        tfidf_weights = []
        for doc_idx in range(tfidf_matrix.shape[0]):
            weights = {feature_names[i]: tfidf_matrix[doc_idx, i] for i in range(len(feature_names))}
            tfidf_weights.append(weights)

        doc1_embedding = self.get_weighted_embedding(doc1, tfidf_weights[0])
        doc2_embedding = self.get_weighted_embedding(doc2, tfidf_weights[1])

        similarity_score = cosine_similarity([doc1_embedding], [doc2_embedding])[0][0]

        print(f"Hybrid semantic similarity: {similarity_score}")
        result_object = {
            "semantic_similarity": float(similarity_score),
        }
        return result_object
```

Figure 9.17: Semantic Similarity