International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2018): 0.28 | SJIF (2018): 7.426

Semantic Harmonizer: Matching Algorithm to Validate Data Transformations & Migrations

Mahidhar Mullapudi[1], Aditya Mamidi[2], Abhishek Shende[3]

Abstract: ***Data transformation and migration processes demand robust mechanisms to ensure semantic consistency between datasets. This paper introduces a sophisticated algorithmic solution encapsulated in the Semantic Harmonizer* [1] *- a versatile library designed for matching data after transformation or migration. The Semantic Harmonizer employs a comprehensive approach, leveraging asynchronous processing, customizable resolution strategies, and a set of predefined semantic matching issues to address differences systematically* [1]. *Key components of this library include interfaces for basic semantic units, generators for unit production, and utility functions for handling complex data structures. The library contributes a vital tool to enhance the reliability and accuracy of semantic matching in diverse scenarios involving data transformation or migration. The proposed algorithm offers flexibility, efficiency, and a systematic approach to resolving semantic differences, making it an invaluable asset in the realm of data management and system evolution* [2]. *This paper delves into the overall architecture, explicates design choices, and imparts insights into best practices for implementing and maintaining such a library, leveraging contemporary tools. The discussion encompasses critical aspects of the library's functionality, emphasizing the need for scalability, flexibility, and resilience to meet the demands of modern data - driven applications* [3][4].

Keywords: Data Matching Algorithms, Semantic Harmonizer, Object Graph Traversal, Semantic Units

1. Introduction

In contemporary software development, the dynamic nature of systems often necessitates the transformation and migration of data to satisfy evolving requirements and technological advancements. However, ensuring the coherence of data semantics throughout these processes poses a formidable challenge [5]. Semantic consistency is imperative to prevent discrepancies, errors, and disruptions in the functionality of software systems.

Recognizing the significance of semantic matching in data transformation and migration, this paper introduces the "Semantic Harmonizer, " a cutting - edge algorithmic solution poised to revolutionize the approach to semantic matching. The Semantic Harmonizer is meticulously crafted to address the intricacies of matching data structures after they undergo transformations or migrations [6].

The Challenge of Semantic Consistency: as data undergoes transformations, ranging from schema modifications to migration across different platforms, maintaining semantic consistency becomes a critical concern. Traditional approaches often fall short in capturing nuanced semantic differences that may arise during these processes. The Semantic Harmonizer is conceived as a response to this challenge, aiming to provide a comprehensive and efficient solution.

Purpose and Significance: the primary purpose of the Semantic Harmonizer is to systematically identify and resolve semantic differences between datasets, ensuring that the transformed or migrated data adheres to predefined semantic standards [7]. By doing so, the library contributes to the overall reliability, accuracy, and integrity of data, safeguarding the functionality of software systems in the face of change.

Key Features and Components: at its core, the Semantic Harmonizer encompasses a set of interfaces defining basic semantic units, generators facilitating unit production, and utility functions for managing complex data structures. The algorithmic approach embedded in the library leverages asynchronous processing, customizable resolution strategies, and a predefined set of semantic matching issues to offer a robust and adaptable solution.

Roadmap to the Semantic Harmonizer: the subsequent sections of this paper delve into the intricacies of the Semantic Harmonizer, exploring its matching process, resolution strategies, and utility functions. Through a detailed examination of its key components, the paper aims to establish the Semantic Harmonizer as an indispensable tool in the arsenal of software developers, architects, and data engineers engaged in data transformation and migration endeavors.

2. Systems Overview

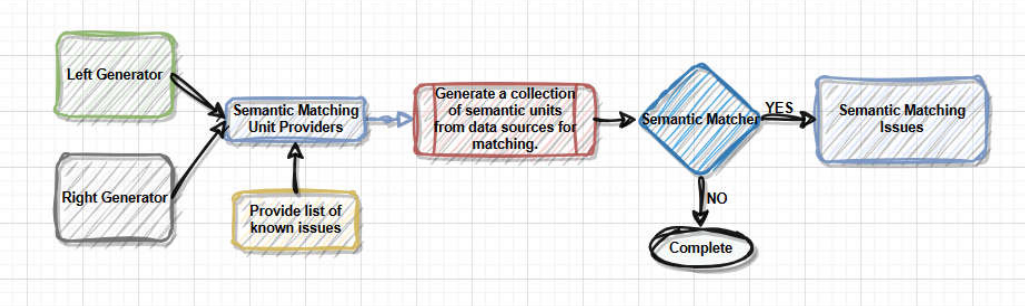

Figure 1: Semantic Harmonizer Architecture Overview

At its core, the Semantic Harmonizer is conceived with the primary purpose of systematically identifying and resolving semantic differences between datasets. By ensuring that the transformed or migrated data adheres to predefined semantic standards, the library becomes a linchpin in maintaining the reliability, accuracy, and integrity of data. Its significance extends to safeguarding the functionality of software systems amidst the constant evolution of data structures. As illustrated in Figure 1 – below are the key components:

- **Basic Semantic Units and Generators**: the Semantic Harmonizer defines a set of interfaces for basic semantic units, laying the foundation for the matching process. Generators, encapsulated in interfaces like *IBasic Semantic Unit Generator*, facilitate the production of basic semantic units through asynchronous processing [6].
- **Algorithmic Matching Process**: the heart of the Semantic Harmonizer lies in its matching process, orchestrated by the *Semantic Matcher* class. Asynchronous generation of basic semantic units is followed by a meticulous identification of differences using the powerful Diff utility [1] [2] [8].
- **Customizable Resolution Strategies**: the library offers flexibility through customizable resolution strategies, allowing users to tailor the handling of semantic differences based on specific needs.
- **Predefined Semantic Matching Issues**: a set of predefined semantic matching issues enhances the library's adaptability, offering solutions for a diverse range of semantic disparities.
- **Utility Functions**: *SemanticMatchingUtils* introduces utility functions, such as *Get Member NamesInChain*, providing practical tools for managing complex data structures during the matching process [9].

DEEP DIVE

In this section we dive deep into some of the interfaces and classes that act as building blocks. We take an opinated approach to implement this matching algorithm and return the results.

***IBasicSemanticUnit*:**
IBasicSemanticUnit interface defines a common structure for semantic units, introducing a property named Key. Any data unit that should be part of this validation or data check must implement this interface and adhere to the contract to display that information. This property returns an object implementing the *IBasic Semantic Unit Key* interface.

IBasicSemanticUnitGenerator<TKey, TUnit>
This interface defines a generic generator for creating basic semantic units. It is parameterized by *TKey* and *TUnit*, where *TKey* is constrained to be a reference type *IBasic Semantic Unit Key*, and *T Unit* is expected to be of *IBasic Semantic Unit<TKey>*. The *Generate Async* method is provided for generation of *Basic Semantic Unit Keyed Collection*.

ISemanticMatcher
This interface defines a contract for semantic matching operations. This is the core of this Matching algorithm which takes the generators for left and right paths, create an object graph of that data and perform the matching to get the results. It then resolves those issues and segregate them to corresponding types [10]. It includes two members:

Property: *Name*
Represents the name of the semantic matcher, providing a human - readable identifier.

Method: *MatchAsync ()*

Declares an asynchronous method for performing semantic matching. The implementation of this method is expected to handle the matching process.

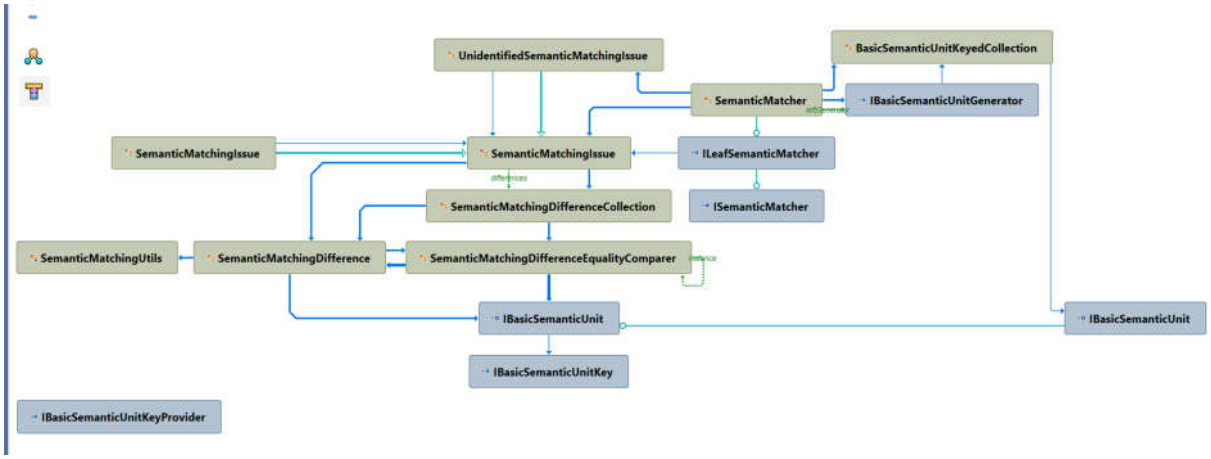

Figure 2: Dependency Diagram for the Semantic Harmonizer library

Semantic Matching Difference
This class represents semantic difference between two data structures [8]. Key characteristics and functionalities include:

Properties:

- *LeftPath*: Represents the pointer of the left - side structure where the difference occurs.
- *LeftValue*: Represents the optional value associated with the left - side structure.

- *Raw Differences*: Provides a collection of raw differences associated with the semantic difference.
- *RightPath*: Represents the pointer of the right - side structure where the difference occurs.
- *RightValue*: Represents the optional value associated with the right - side structure.
- Type: Represents the type of modification (e. g., *OnlyInLeft*, *OnlyInRight*, *Changed*).

Overall, *Semantic Matching Difference* encapsulates information about semantic differences, making it suitable for use in semantic matching algorithms where detailed differences between data structures need to be identified and processed [9].

Semantic Matching Issue
The abstract class serves as a foundation for defining specific semantic matching issues. Key characteristics and functionalities include:

Differences: Represents a collection of semantic matching differences associated with the issue.

TryResolve () method: Virtual method that attempts to resolve a semantic difference. It receives a Difference object and adds a corresponding *Semantic Matching Difference* to the collection. The method returns true, indicating a successful resolution. Derived classes can override this method to customize the resolution logic.

Overall, *Semantic Matching Issue* establishes a framework for creating specific semantic matching issues. It encourages encapsulation of semantic differences and related messages, providing a base structure that derived classes can build upon to address various semantic disparities in data structures.

SemanticMatcher<TKey, TUnit>
The *SemanticMatcher* class represents a semantic matching component designed to identify and resolve differences between two collections of basic semantic units. Key characteristics and functionalities include:

Issues: Represents a list of semantic matching differences and aggregate them based on the type of issue, that can be found during the matching process.

Left Generator and *right Generator*: instances of *IBasic Semantic Unit Generato*r responsible for generating basic semantic units from the left and right collections, respectively.

Name: A string representing the name of the semantic matcher.

MatchAsync (): asynchronous method responsible for executing the semantic matching algorithm. The matching algorithm involves the following steps:

- Generation of basic semantic units for the left and right collections using the provided generators.
- Identification of differences between the generated units using a diffing utility.
- Generate object graph for each semantic unit of data for both left and right items and then create a path for all the properties to be compared.
- Iterate through the object graph, compare corresponding properties for left and right graphs and add differences to the collection.
- Iterate through the identified differences, attempting to resolve each one using the specified semantic matching issues.

Overall, the *SemanticMatcher* class encapsulates the logic for comparing and resolving semantic differences between two collections of basic semantic units. It allows for flexibility by supporting customizable semantic matching issues and generator implementations.

3. Conclusion

In this paper, we proposed and implemented a library that takes in different data sources that can be converted into semantic units that are compared using object graph traversal, get the differences comparing corresponding properties, resolve those to identify the issue type based on given known issues and aggregate those results. By encapsulating differences and related messages, it provides a structured approach to handling various semantic disparities. The asynchronous nature of the algorithm ensures efficient handling of large datasets [11].

As technology advances, future developments should focus on refining performance, enhancing extensibility, exploring advanced pattern matching techniques, and embracing dynamic and intelligent matching approaches. By combining these considerations, the semantic matching framework can evolve into a sophisticated and adaptable solution capable of handling diverse data scenarios in the ever - changing landscape of information technology.

References

[1] "Semantic Matching Wiki, " [Online]. Available: https: //en. wikipedia. org/wiki/Semantic_matching.
[2] P. S. L. &. Z. S. Bouquet, "Semantic coordination: A new approach and an application, " in *In International Semantic Web Conference*, Springer, Berlin, Heidelberg, 2003.
[3] P. S. &. M. Y. Fausto Giunchiglia, "S - Match: an Algorithm and an Implementation of Semantic Matching, " *Springer, Berlin, Heidelberg,* vol.3053, pp.61 - 75, 2004.
[4] J. L. W. S. I. A. J. R. L. Guoqiang Jerry Chen, "Realtime Data Processing at Facebook, " *SIGMOD,* p.1, 2016.
[5] "Why the Data You Use Is More Important Than the Model Itself, " [Online]. Available: https: //medium. com/swlh/why - the - data - you - use - is - more - important - than - the - model - itself - 4a49736ea70c.
[6] "Is Data More Important Than Algorithms In AI?, " [Online]. Available: https: //www.forbes. com/sites/quora/2017/01/26/is - data - more - important - than - algorithms - in - ai/?sh=111664a542c1.

[7] P. G. F. &. Y. M. Avesani, "A large scale taxonomy mapping evaluation, " in *In International Semantic Web Conference*, Springer, Berlin, Heidelberg, 2005.
[8] F. Y. M. &. S. P. Giunchiglia, "Semantic Matching: Algorithms and Implementation, " *Journal on Data Semantics,* pp.1 - 38, 2007.
[9] "Object - Oriented Design and Data Structures - Grpah Traversals, " [Online]. Available: https: //andrewcmyers. github. io/oodds/lecture. html?id=traversals.
[10] "What is object graph, " [Online]. Available: https: //stackoverflow. com/questions/2046761/what - is - object - graph - in - java.
[11] Kleppmann, Martin, Designing Data - Intensive Applications, O'Reilly Media, 2017.