FEDERAL UNIVERSITY OF CEARA

SCIENCE CENTER

COMPUTER DEPARTMENT

MASTER AND DOCTORATE PROGRAM IN COMPUTER SCIENCE

DOCTORATE IN COMPUTER SCIENCE

MARCOS VINÍCIUS DE FREITAS BORGES

A SYNTACTIC-SEMANTIC ANALYSIS METHOD FOR AUTOMATIC API CONNECTION POINTS DISCOVERY IN SYSTEMS-OF-INFORMATION SYSTEMS

FORTALEZA

2025

MARCOS VINÍCIUS DE FREITAS BORGES

A SYNTACTIC-SEMANTIC ANALYSIS METHOD FOR AUTOMATIC API CONNECTION POINTS DISCOVERY IN SYSTEMS-OF-INFORMATION SYSTEMS

Thesis presented to the Master and Doctorate Program in Computer Science of Science Center (Federal University of Ceara) as a requirement to obtain the title of Ph.D. in Computer Science. Concentration area: Information Systems.

Supervisor: Lincoln Souza Rocha, PhD.
Co-supervisor: Paulo Henrique Mendes Maia, PhD.

FORTALEZA

2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

B733s Borges, Marcos Vinícius de Freitas.
A Syntactic-Semantic Analysis Method for Automatic API Connection Points Discovery in Systems-of-Information Systems / Marcos Vinícius de Freitas Borges. – 2025.
135 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação , Fortaleza, 2025.
Orientação: Prof. Dr. Lincoln Souza Rocha.
Coorientação: Prof. Dr. Paulo Henrique Mendes Maia.

1. Systems-of-information systems. 2. Connection points discovery. 3. Syntactic analysis. 4. Semantic analysis. I. Título.

CDD 005

MARCOS VINÍCIUS DE FREITAS BORGES

A SYNTACTIC-SEMANTIC ANALYSIS METHOD FOR AUTOMATIC API CONNECTION POINTS DISCOVERY IN SYSTEMS-OF-INFORMATION SYSTEMS

Thesis presented to the Master and Doctorate Program in Computer Science of Science Center (Federal University of Ceara) as a requirement to obtain the title of Ph.D. in Computer Science. Concentration area: Information Systems.

Approved on: 30/01/2025.

COMMITTEE

Lincoln Souza Rocha, PhD (Supervisor)
Federal University of Ceará (UFC)

Paulo Henrique Mendes Maia,
PhD (Co-supervisor)
State University of Ceará (UECE)

Elisa Yumi Nakagawa, PhD
University of São Paulo (USP)

Rodrigo Pereira dos Santos, PhD
Federal University of the State of Rio de Janeiro (UNIRIO)

João Paulo Pordeus Gomes, PhD
Federal University of Ceara (UFC)

Paulo Antonio Leal Rêgo, PhD
Federal University of Ceara (UFC)

To God, my wife, my son, my parents, my grandmother, my sisters and my niece, for believing in me and always supporting me.

ACKNOWLEDGEMENTS

Firstly, I would like to thank God, whose infinite goodness and love gave me the intelligence, willpower, and perseverance necessary to complete this thesis. Only God and my family know the countless hours of dedication, the sleepless nights, the nights that stretched until dawn, the weekends and holidays that I gave up to dedicate myself to writing this thesis and articles that marked my doctoral career. Each effort was sustained by the faith and unconditional support of those who were by my side throughout this journey.

I would also like to thank my family, especially my wonderful wife, friend, companion, psychologist, encourager and helper: Márcia Cristina dos Santos Paulo. Her dedication, tireless support and constant encouragement were fundamental for me to reach the end of this journey. I would also like to thank my son, Marcos Vinícius de Freitas Borges Júnior, whose eyes and smiles were inexhaustible sources of strength and inspiration for me to continue steadfast in my mission to complete my doctorate. You are my foundation and my greatest gift!

I would like to thank my family of origin, represented by my parents, Maria de Jesus de Freitas Borges and Salvador de Melo Borges, my grandmother, Margarida Pereira Lima, my sisters, Márcia Angélica de Freitas Borges and Jéssica Laís de Freitas Borges, and my niece, Ana Beatriz de Freitas Mousinho, for all the support and for the many moments shared throughout this journey. This victory is also yours!

I would also like to thank my wife's family of origin. My mother-in-law, Maria Cristina dos Santos Paulo, whom I consider a second mother, has always been by my side, offering help, encouragement and sharing my moments of concern. I am also grateful to my father-in-law, Ângelo Paulo, for the support and strength he has always shown. I would also like to thank my brothers-in-law, Marcos and his family, and Marcius and his children, for their support and encouragement throughout this journey.

I would also like to thank my advisor, Lincoln Souza Rocha, for the immense knowledge I have acquired throughout this doctoral journey. His precise questions and objectivity made me reflect deeply and be more incisive in what needed to be done, without wasting time on beating around the bush. I am fully convinced that I have evolved a lot. Looking back at the beginning of my doctorate, I realize how much I have grown as a researcher, thanks to his help, guidance and partnership. I am grateful for all the teachings and constant support throughout my career at UFC. Thank you very much for everything!

I would also like to thank my co-advisor, Paulo Henrique Mendes Maia. I can say

that much of my success is the result of your support and guidance. You were like a father to me in the academic world, and I carry with me your teachings, your responsibility, dedication and commitment, as well as many of your principles, which I now share with my own students. If I am where I am today, it is because you believed in me since my master's degree, trusting that together we could overcome the challenge at UFC, even before the doctorate program was established at UECE. During my doctorate journey, I had the opportunity to learn from you again, and I end with a phrase that expresses my admiration: you are my reference as a teacher! Thank you very much for everything!

I would also like to thank the professors on the committee, João Paulo Pordeus Gomes and Paulo Antônio Leal Rêgo, for their valuable contributions and suggestions that helped to improve the thesis.

I would also like to thank Professor Rodrigo Pereira dos Santos for his valuable partnership in the articles, for his precise notes, suggestions, constructive provocations, commitment and constant availability. His meticulous and detailed corrections, as well as the enriching conversations via email and chat, were fundamental to my learning and growth. I am immensely grateful to God for the opportunity to work with you, one of the main references in the SoS area. I learned a lot from your dedication and excellence, and I intend to follow the model of article corrections that I learned from you. Thank you very much for everything!

I would also like to thank Professor Elisa Yumi Nakagawa for her trust, respect and belief in me, even though I was still a "beginner" in my academic career, in the face of such an admirable role model as you. I am grateful for the invitation to visit USP, for the attention and care in showing me the main spaces, detailing where I would be staying and what would be necessary during my stay. I am also grateful for the enriching meetings, for the "paper-driven" method I adopted for academic life and for the article we managed to approve together. Professor Elisa, your humility, even with such an impressive academic trajectory, is a great inspiration to me. Thank you very much for everything!

I would like to thank my Ph.D. friend, Pedro Almir Martins de Oliveira, for all the support he has given me since I arrived at UFC, for sharing the apartment, for the conversations, for the valuable suggestions, for the support in my doubts, and for sharing articles and documents in general. I am very grateful for his friendship and for everything along this journey!

Finally, I would like to thank the professors of the subjects I took at UFC for the valuable knowledge they transmitted and for their significant contribution to my education.

RESUMO

Estabelecer *links* de interoperabilidade é um desafio significativo na engenharia de sistemas de sistemas-de-informação (SoIS, do inglês *systems-of-information systems*). Mesmo com a documentação das interfaces dos sistemas constituintes (CS, do inglês *constituent systems*), alcançar esses *links* é uma tarefa difícil, demorada e suscetível a erros, que requer atenção dos desenvolvedores de CS, especialmente se realizada manualmente. Para contribuir com essa tarefa, esta tese apresenta um método semiautomático que utiliza análise de similaridade sintática e semântica de descrições de interfaces de programação de aplicações (API, do inglês *application programming interface*) para identificar potenciais pontos de conexão entre CS. Esse método é constituído por três etapas interligadas: (i) definição de requisitos, onde são delineadas as motivações e justificativas para a criação do método; (ii) *design* do núcleo do método, onde são detalhadas todas as fases e componentes necessários para obter os pontos de conexão de API; e (iii) implementação do núcleo do método, onde são apresentadas os detalhes tecnológicos da ferramenta desenvolvida para suportar o método. Através dessa ferramenta, foram realizadas duas avaliações. Na primeira, foi executado um experimento controlado para avaliar o desempenho da ferramenta e definir empiricamente os melhores algoritmos de similaridade e limiares para a tarefa de identificação de pontos de conexão entre duas API bem conhecidas. Na segunda, a ferramenta foi empregada em um estudo de caso aplicado a um SoIS real de um fabricante global de computadores. O estudo abrangeu três cenários, envolvendo sete API de CS, e considerou as perspectivas de análise sintática, e análise sintático-semântica de API. Em seguida, foram conduzidas entrevistas semiestruturadas com cinco desenvolvedores que trabalharam no SoIS para avaliar a eficiência da ferramenta. Os resultados demonstraram que a ferramenta é bem-sucedida, com a análise de similaridade sintática e semântica de API provando ser eficiente para identificar *links* de interoperabilidade entre CS. Do ponto de vista prático, os algoritmos sintáticos demonstraram desempenho notável em ambientes locais. Ao mesmo tempo, a análise semântica, que é mais robusta, requer infraestrutura avançada devido à sua alta complexidade computacional. Do ponto de vista da pesquisa, o uso de similaridade semântica de textos curtos (STSS, do inglês *short text semantic similarity*) é ainda um campo promissor, e o uso de inteligência artificial com modelos largos de linguagem (LLM, do inglês *Large Language Models*) pode gerar novos resultados e descobertas de *links* de interoperabilidade em SoIS.

Palavras-chave: sistemas de sistemas-de-informação, descoberta de pontos de conexão, API, análise sintática, análise semântica.

ABSTRACT

Establishing interoperability links is a significant challenge in systems-of-information systems (SoIS) engineering. Even with constituent systems (CS) interfaces documentation, achieving such links is a difficult, time-consuming, and error-prone task that requires attention from CS developers, especially if it is performed manually. To contribute to that task, this thesis presents a semi-automatic method using both syntactic and semantic similarity analysis of application programming interface (API) descriptions to identify potential connection points among CS. That method consists of three interconnected steps: (i) requirements definition, where the motivations and justifications for creating the method are outlined; (ii) core method design, where all the phases and components necessary to obtain the API connection points are detailed; and (iii) core method implementation, where the technological details of the tool developed to support the method are presented. Through that tool, two evaluations were performed. In the first, a controlled experiment was executed to evaluate the tool's performance and empirically define the best similarity algorithms and thresholds for the task of identifying connection points between two well-known API. In the second, the tool was used in a case study applied to a real-world SoIS of a global computer manufacturer. The study covered three scenarios, involving seven CS API, and considered the perspectives of syntactic analysis and syntactic-semantic analysis of API. Then, semi-structured interviews were conducted with five developers who worked on the SoIS to evaluate the tool's efficiency. The results demonstrated that the tool is successful, with the syntactic and semantic similarity analysis of API proving to be efficient in identifying interoperability links between CS. From a practical point of view, the syntactic algorithms demonstrated notable performance in local environments. At the same time, semantic analysis, which is more robust, requires advanced infrastructure due to its high computational complexity. From a research perspective, the use of short text semantic similarity (STSS) is still a promising field, and the use of artificial intelligence with Large Language Models (LLM) can generate new results and discoveries of interoperability links in SoIS.

Keywords: systems-of-information systems, connection points discovery, API, syntactic analysis, semantic analysis.

LIST OF FIGURES

Figure 1 – Example of the transportation systems-of-systems 18
Figure 2 – Example of interoperability links from an SoIS of a global computer manufacturer 22
Figure 3 – Thesis Research Method 25
Figure 4 – Smart city SoIS hierarchy composed of SoS and SoIS 30
Figure 5 – SoIS characteristics 30
Figure 6 – Example of Virtual SoS 33
Figure 7 – Example of Collaborative SoS 34
Figure 8 – Example of Acknowledged SoS 34
Figure 9 – Example of Directed SoS 35
Figure 10 – Different types of SoS based on management policy 36
Figure 11 – Guidelines for the design of interoperability links in SoIS 37
Figure 12 – Address Book Service in OpenAPI (yaml) in Swagger UI 43
Figure 13 – Address Book Service (yaml) - Expanded GET method “/contacts” 44
Figure 14 – Address Book Service (yaml) - Expanded GET method “/contacts/contacId” 44
Figure 15 – Address Book Service (yaml) - Expanded DELETE method “/contacts/contacId” 45
Figure 16 – Knowledge-based representation example 54
Figure 17 – Feature-based representation example 54
Figure 18 – Deep Neural Network example 58
Figure 19 – General flow and main parts of our method 66
Figure 20 – Component diagram implemented in our core method 76
Figure 21 – Registration of components in API Gateway 77
Figure 22 – Tool: login 81
Figure 23 – Tool: Sign Up 82
Figure 24 – Tool: User created successfully 82
Figure 25 – Tool: Home 83
Figure 26 – Tool: API Insert 83
Figure 27 – Tool: API List 84
Figure 28 – Tool: Syntactic Analysis - input parameters 84

Figure 29 – Tool: Syntactic Analysis - Syntactic API connection points and evaluation metrics 85
Figure 30 – Tool: Syntactic-Semantic Analysis - Syntactic API connection points 85
Figure 31 – Tool: Syntactic-Semantic Analysis - Semantic API points 85
Figure 32 – Tool: Syntactic-Semantic Analysis - Syntactic-semantic API points and evaluation metrics 86
Figure 33 – Performance of the 6 string similarity algorithms based on Jira→Github ranking (*avg_rank*) 90
Figure 34 – Correlation chart among the 6 string similarity algorithms based on Jira→Github ranking (*avg_rank*) 90
Figure 35 – Performance of the 6 string similarity algorithms based on Jira→Github ranking (*major_rank*) 91
Figure 36 – Correlation chart among the 6 string similarity algorithms based on Jira→Github ranking (*major_rank*) 91
Figure 37 – CS and API of the DSoIS. 93

LIST OF TABLES

Table 1 – OpenAPI Properties 41
Table 2 – Comparison of related work with our method 46
Table 3 – Source articles for extraction of syntactic and semantic similarity analysis methods 49
Table 4 – Example of linguistic corpus 56
Table 5 – Catalog of the main methods of textual similarity analysis 64
Table 6 – API Connection Points 68
Table 7 – Key parameters adapted from Jira and Github based on their documentations 88
Table 8 – API list=(Jira 7.6.1 API (‘api/2/search’)→Git Hub 1.1.4 API (‘search/commits’)) | k = 10 | rank=(avg_rank) / Thereshold = 0.5 / Precision (0.75) - Recall (1.0) - Accuracy (0.9) 89
Table 9 – API list=(Jira 7.6.1 API (‘api/2/search’)→Git Hub 1.1.4 API (‘search/commits’)) | k = 10 | rank=(major_rank) / Thereshold = 0.8 / Precision (1.0) - Recall (1.0) - Accuracy (1.0) 89
Table 10 – API list = (CFSysAPI v1.0→NFSysAPI v1.0) | k = 10 | rank=(avg_rank) / Thereshold = 0.5 / Precision (0.3) - Recall (1.0) - Accuracy (0.3) 95
Table 11 – API list = (CFSysAPI v1.0→NFSysAPI v1.0) | k = 10 | rank=(major_rank) / Threshold (0.8) / Precision (1.0) Recall - (0.33) - Accuracy (0.8) 95
Table 12 – API list = (NFSysAPI v1.0→CFSysAPI v1.0) | k=10 | rank=(avg_rank) / Threshold (0.5) / Precision (0.143) - Recall - (1.0) - Accuracy (0.4) 96
Table 13 – API list = (NFSysAPI v1.0→CFSysAPI v1.0) | k=10 | rank=major_rank / Threshold (0.8) / Precision (1.0) - Recall - (1.0) - Accuracy (1.0) 96
Table 14 – API list = (DASys V1.0→LATSysAPI V2.3) | k = 10 | rank=avg_rank / Threshold (0.5) / Precision (0.6) - Recall (1.0) - Accuracy (0.6) 97
Table 15 – API list = (DASys V1.0→LATSysAPI V2.3) | k = 10 | rank=major_rank / Threshold (0.5) / Precision (0.6) - Recall (1.0) - Accuracy (0.6) 97
Table 16 – API list = (LATSysAPI V2.3→DASys V1.0) | k = 10 | rank=avg_rank / Threshold =0.5 / Precision (0.3) - Recall (1.0) - Accuracy (0.3) 98
Table 17 – API list = (LATSysAPI V2.3→DASys V1.0) | k = 10 | rank=major_rank / Threshold =0.8 / Precision (0.25) - Recall (1.0) - Accuracy (0.4) 98

Table 18 – Scenario#3.1 - Schemas and Circular references of the QUSysAPI and ITSys-API API 103
Table 19 – API list = (QUSysAPI→ITSysAPI) | k = 10 | rank=avg_rank (avg_syn_rank) 104
Table 20 – API list = (QUSysAPI→ITSysAPI) | k = 10 | rank=avg_rank (avg_sem_rank) 104
Table 21 – API list = (QUSysAPI→ITSysAPI) | k = 10 | rank=avg_rank / Threshold=0.7 / Precision (0.5) - Recall (0.5) - Accuracy (0.8) 104
Table 22 – API list = (QUSysAPI→ITSysAPI) | k = 10 | rank=major_rank (major_syn_rank) 105
Table 23 – API list = (QUSysAPI→ITSysAPI) | k = 10 | rank=major_rank (major_sem_rank) 105
Table 24 – API list = (QUSysAPI→ITSysAPI) | k = 10 | rank=major_rank / Threshold=0.8 / Precision (0.33) - Recall (0.5) - Accuracy (0.7) 105
Table 25 – Scenario#3.2 - Schemas and Circular references of the ITSysAPI and FDSys-API API 106
Table 26 – API list = (ITSysAPI →FDSysAPI) | k = 10 | rank=avg_rank(avg_syn_rank) 107
Table 27 – API list = (ITSysAPI →FDSysAPI) | k = 10 | rank=avg_rank(avg_sem_rank) 107
Table 28 – API list = (ITSysAPI →FDSysAPI) | k = 10 | rank=avg_rank / Threshold=0.7 / Precision (0.4) - Recall (1) - Accuracy (0.7) 107
Table 29 – API list = (ITSysAPI →FDSysAPI) | k = 10 | rank=major_rank(major_syn_rank) 108
Table 30 – API list = (ITSysAPI →FDSysAPI) | k = 10 | rank=major_rank(major_sem_rank) 108
Table 31 – API list = (ITSysAPI →FDSysAPI) | k = 10 | rank=major_rank / Threshold=0.8 / Precision (0.2) - Recall (1) - Accuracy (0.6) 108
Table 32 – Challenges for professionals 111
Table 33 – Challenges for researchers 113

LIST OF ABBREVIATIONS AND ACRONYMS

ADG	API Dataset Generator
AI	artificial intelligence
API	application programming interface
ASE	API Syntactic Extractor
ASSA	API Syntactic Similarity Analyzer
ASSESA	API Syntactic-Semantic Similarity Analyzer
BERT	Bidirectional Encoder Representations from Transformers
Bi-LSTM	Bidirectional Long Short-Term Memory
BPMN	Business Process Modeling Notation
CBSoft	Brazilian Conference on Software: Practice and Theory
CNN	Convolutional Neural Network
CS	constituent systems
CWE	Collaborative Working Environment
DSR	Design Science Research
EAI	Enterprise Application Integration
ESA	Explicit Semantic Analysis
ESM	Enterprise Semantic Model
FN	false negatives
FP	false positives
GPT	Generative Pre-Trained Transformer
HAL	Hyperspace Analogue to Language
HTML	hypertext markup language
HTTP	Hypertext Transfer Protocol
IC	information content
IDF	inverse document frequency
IETF	Internet Engineering Task Force
IS	information systems
ISJ	IEEE System Journal
IT	Information Technology
JQL	JIRA Query Language

JSON	JavaScript Object Notation
JSS	Journal of Systems and Software
LATAM	LATAM School in Software Engineering
LCS	longest common substring
LLM	Large Language Models
LSA	Latent Semantic Analysis
LSTM	Long Short-Term Memory
ML	machine learning
MLM	masked language modeling
NGD	Normalized Google Distance
NLP	natural language processing
NLU	natural language understanding
NSP	next sentence prediction
OAS	OpenAPI Specification
PLM	Product Lifecycle Management
RAG	Retrieval-Augmented Generation
RAML	RESTful API Modeling Language
REST	Representational State Transfer
RNN	Recurrent Neural Network
RoBERTa	Robustly Optimized BERT pre-training Approach
SEBoK	Systems Engineering Body of Knowledge
SESoS	International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems
SoIS	systems-of-information systems
SoS	systems-of-systems
STSS	short text semantic similarity
SVD	singular value decomposition
T5	text-to-text transfer transformer
TF-IDF	term frequency–inverse document frequency
TN	true negatives
TP	true positives
UECE	State University of Ceará

UI	user interface
UNIRIO	Federal University of the State of Rio de Janeiro
URL	Uniform Resource Locator
USP	University of São Paulo
WADL	Web Application Description Language
WSDL	Web Services Description Language
XML	Extensible Markup Language

CONTENTS

1 INTRODUCTION 17
1.1 General Context 17
1.2 Research Context 19
1.3 Motivation 21
1.4 Problem Statement 23
1.5 Research Question and Objectives 24
1.6 Research Method 24
1.7 Thesis Outline 27
2 BACKGROUND 29
2.1 System-of-Information Systems 29
2.1.1 Characteristics 29
2.1.2 Classification 32
2.1.3 Interoperability links 35
2.2 API Description 38
2.3 Related Work 45
2.4 Final Remarks 48
3 CATALOG OF TEXTUAL SIMILARITY ANALYSIS METHODS . . . 49
3.1 Syntactic Similarity 49
3.2 Semantic Similarity 52
3.2.1 Knowledge-based methods 53
3.2.2 Corpus-based methods 55
3.2.3 Deep Neural Network-based methods 58
3.3 Catalog 63
3.4 Final Remarks 63
4 METHOD 65
4.1 Requirements definition 65
4.2 Core method design 67
4.2.1 API Syntactic Extractor 69
4.2.2 API Dataset Generator 70
4.2.3 API Syntactic Similarity Analyzer 71
4.2.4 API Syntactic-Semantic Similarity Analyzer 73

4.3 **Core method implementation** . . . 75
4.4 **Final Remarks** . . . 78
5 **EVALUATION** . . . 79
5.1 **Tool - Overview** . . . 79
5.2 **Controlled Experiment** . . . 86
5.3 **Case Study** . . . 92
5.3.1 ***Data from SoIS*** . . . 92
5.3.2 ***Syntactic Analysis Perspective*** . . . 94
5.3.2.1 *Results* . . . *94*
5.3.2.2 *Interview with Project Developers* . . . *98*
5.3.3 ***Syntactic-Semantic Analysis Perspective*** . . . 101
5.3.3.1 *Results* . . . *101*
5.3.3.2 *Interview with Project Developers* . . . *108*
5.4 **Implications** . . . 110
5.4.1 ***Developers*** . . . 110
5.4.2 ***Researchers*** . . . 111
5.5 **Limitations** . . . 113
5.6 **Threats to Validity** . . . 114
5.6.1 ***Threats to internal validity*** . . . 114
5.6.2 ***Threats to external validity*** . . . 114
5.6.3 ***Threats to construct validity*** . . . 115
5.7 **Final Remarks** . . . 115
6 **CONCLUSION** . . . 117
6.1 **Thesis Summary** . . . 117
6.2 **Thesis contributions** . . . 118
6.3 **Published Scientific Artifacts** . . . 119
6.4 **Future Work** . . . 119
REFERENCES . . . 122

1 INTRODUCTION

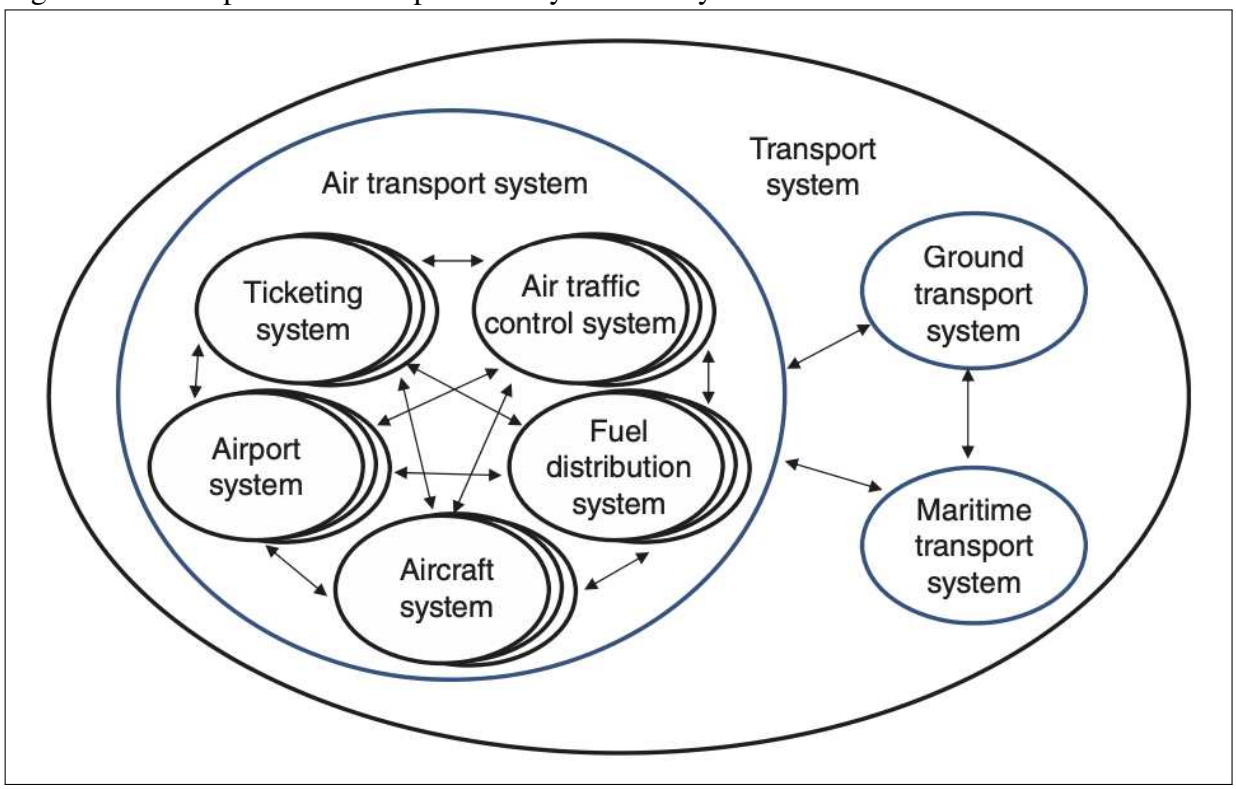
1.1 General Context

Software-intensive systems are becoming increasingly ubiquitous, larger, complex, and with wide application in several domains due to, among other factors, the evolution of integration and communication technologies (BATISTA, 2013; NAKAGAWA *et al.*, 2013). Although software reuse techniques have been effectively used to build and evolve large-scale systems (SELBY, 2005; BASSO *et al.*, 2013), the addition of new complex requirements (such as data-intensive computing, distributed execution, scalability, and technology heterogeneity) make the development of such systems more difficult and error-prone, which may lead the system to behave differently from expected (WALTON; MAIDEN, 2019). An interesting way to minimize such problems is to (re)use features from existing systems, which are generally well-validated and tested. Thus, these features can serve as a basis for the vertical or horizontal development of new capabilities in new systems (REBOVICH, 2014; OSÓRIO *et al.*, 2017). The former development strategy refers to the use of the components of a single system of an organization to produce the desired new functionality, while the latter strategy, more complex, occurs when components of different systems from different organizations, with different technologies and platforms, are used to create new functionalities. This last strategy characterizes systems-of-systems (SoS).

The definition of SoS can vary depending on the context (MANTHORPE, 1996; SAGE; CUPPAN, 2001; MAYK; MADNI, 2006). However, it can be described and summarized as a collection of independent and useful systems, commonly called constituent systems (CS), that cooperate and collaborate to fulfill a goal or to provide new capabilities with complex functions, which individual CS cannot independently achieve (MAIER, 1998; JAMSHIDI, 2017; ISO, 2019; OLIVERO *et al.*, 2022). An example of SoS can be seen in Figure 1, which presents comprehensive transportation SoS, encompassing three main SoS: air transport, ground transport, and maritime transport (ISO, 2019). Each of these SoS is designed to optimize passenger transport in its respective field of activity. In the air transport system, it is possible to identify several CS, such as ticketing, air traffic control, and fuel distribution. Although these CS have autonomy, independence, or different programming languages, they collaborate through the exchange of information to effectively serve passengers in the air sector.

Historically, SoS started to gain popularity in the military domain, with systems developed for defense, warfare, and global earth observations (DOMERÇANT; MAVRIS, 2011;

Figure 1 – Example of the transportation systems-of-systems



Source: ISO (2019, p. 236).

SMITH *et al.*, 2011; NJOKU, 2014; MAHMOOD, 2016). Over the years, new application scenarios especially those essential for our society (e.g., automotive, avionics, emergency management, smart systems, and information systems (IS)) have emerged (LANE; EPSTEIN, 2013; NIELSEN *et al.*, 2015; JAMSHIDI, 2017; VILLELA *et al.*, 2018; FERNANDES *et al.*, 2019). These last ones, IS, generally come from alliances between companies in the business sector that need to establish relationships with other companies to complement their mutual limitations, increase their influence, and provide more attractive offers to customers through interoperation between their systems (FERNANDES *et al.*, 2019; TEIXEIRA *et al.*, 2019). Such interoperability[1] is fundamental for integrating business processes and facilitating, for example, the sale, delivery, and payment processing of products provided by an associated company.

That increasing demand for interoperability in IS has given rise to this new type of SoS called systems-of-information systems (SoIS), which can be defined as a specific class of SoS oriented towards business processes, in which CS include IS to interoperate with each other to achieve objectives that none of them could deliver if they were not part of such a consortium (MAJD; MARIE-HÉLÈNE, 2017; TEIXEIRA *et al.*, 2019). The SoIS inherits

[1] In this thesis we adopt the following definition for interoperability: "Is the ability of multiple systems or components to share and utilize information effectively" (IEEE, 1990).

SoS characteristics (BOARDMAN; SAUSER, 2006; NIELSEN *et al.*, 2015; FERNANDES *et al.*, 2020), such as: (i) autonomy, which is related to how a system is governed by its own rules despite external influences; (ii) evolutionary development, meaning the SoIS evolves as constituent IS evolve as well; (iii) emergent behavior, which results from the synergistic collaboration among constituent IS and from the combination of the capabilities offered to form global behaviors; (iv) connectivity, since constituent IS are dispersed and depend on connectivity mechanisms for communication and information sharing; (v) interdependence, i.e., the mutual dependence between constituent IS to accomplish goals; (vi) diversity, once SoIS involves different technologies, data formats, people, processes, protocols, and even interpretations and purposes; (vii) dynamicity, which is characterized by the ability of an SoIS to modify its own structure and composition; (viii) belonging, that represents the IS perception of mutual benefit among the parts; and (ix) interoperability, which refers to the ability of the SoIS to incorporate a range of heterogeneous IS.

The SoIS area represents a recently planned field of study within SoS, needing the IS community to develop new processes, methods, techniques, approaches, and tools to support the development of SoIS (NETO *et al.*, 2016). Additionally, Neto *et al.* (2016) identified challenges in the field, which include: (i) how to conceive trustworthy SoIS, focusing on the SoIS' ability to maintain its activity to fulfill missions even with failures of some of its components, in addition to offering reliable services to its users; (ii) how to promote sustainability for and from SoIS, seeking to make it more sustainable, minimizing the use of natural resources and supporting sustainable missions, such as the development of SoIS to monitor rivers or bays; (iii) how to support the conception and elaboration of SoIS, focusing on the research of new theories and technologies to efficiently support the development of new SoIS, dealing with general aspects of development and emerging behaviors, which may arise from the adoption of IS based on its functionalities. This thesis focuses on the latter challenge, addressing issues regarding SoIS development (interoperability).

1.2 Research Context

The conception and elaboration of SoIS is essential in today's highly competitive business market, where companies need to establish agreements and commercial relationships with other companies to provide more attractive offers. That can be seen very commonly in: (i) co-marketing partnerships, where companies sell each other's products or services, aiming to

expand each company's audience and generate more sales and brand recognition; (ii) co-selling partnerships, in which companies refer each other's products or services to customers, aiming to win customers, reduce sales costs and generate greater revenue (YEPEZ, 2024). However, when a group of companies proposes to establish partnerships or commercial agreements, the challenge of making their IS interoperable arises (FERNANDES *et al.*, 2023). It happens mainly because each CS of an SoIS must not only meet the goals of the SoIS, providing its functionalities through voluntary interoperability but also must preserve the operational and management independence and fulfill its original purposes (FERNANDES *et al.*, 2019; BORGES *et al.*, 2022).

Establishing interoperability among IS is not a trivial activity involving several human, organizational, and technical factors that can affect the decision-making process of companies, and this can lead alliances to become temporary or longer (FERNANDES *et al.*, 2023). The human factors are related to collaboration between operators who interact in technological environments, mainly associated with the responsibilities of individuals, their human perceptions, and their relationships with companies, partners, and customers. Organizational factors are related to inter or intra-organizational connections, including aspects of business processes that provide new services. Technical factors are associated with hardware and software components, computer networks, and devices that make possible machine-to-machine communication.

To implement that interoperability based on an alliance between companies, Fernandes *et al.* (2021) established the concept of interoperability links, which represent connections between two or more CS through mutual communication between their IS through some form of sending and/or receiving messages. An example of an interoperability link, under the technical factors, is the sharing of data with another IS, through the export/import of files with metadata or by sending data through a Uniform Resource Locator (URL), using Hypertext Transfer Protocol (HTTP) request methods such as GET, POST (FERNANDES *et al.*, 2020). Figure 2 illustrates a real example of these interoperability links, extracted from a request flow of an SoIS from a global computer company that contains several IS and belong to different CS. Their interoperability was established to achieve a specific objective of SoIS, which is to obtain all the details of a specific quote through a certain quote. The process begins with a request via the GET method to the IS endpoint "*Quote(quoteId)*", which forwards it to IS2, responsible for all the basic quote information. This system returns data such as prices, items, availability, contacts, and payment methods. Based on these results, other requests are sent to specific IS, which hold complete data for each item. IS3 handles pricing and features, IS4 handles the individual items of a feature,

IS5 handles individual configuration items, IS6 handles comprehensive availability, and IS7 handles the customer details. After collecting all the data, IS1 can create a complete and detailed quote model and return it to the requester. It is possible to note that each one is autonomous and independent, maintaining its own information while providing some functionalities so that SoIS can achieve its objective. Finally, it is important to highlight that this chain of interoperability links was designed based on the activities carried out by the CS developers of that SoIS, based on detailed studies and the integrated operation of each IS. If the SoIS defined a new objective, and the information necessary to achieve it was in a new IS, called IS8, the developers would need to analyze the documentation and communication details of that IS. That process would include the exploration of specific endpoints and attributes to obtain the complementary information provided by IS8, thus ensuring compliance with the new objective of the SoIS.

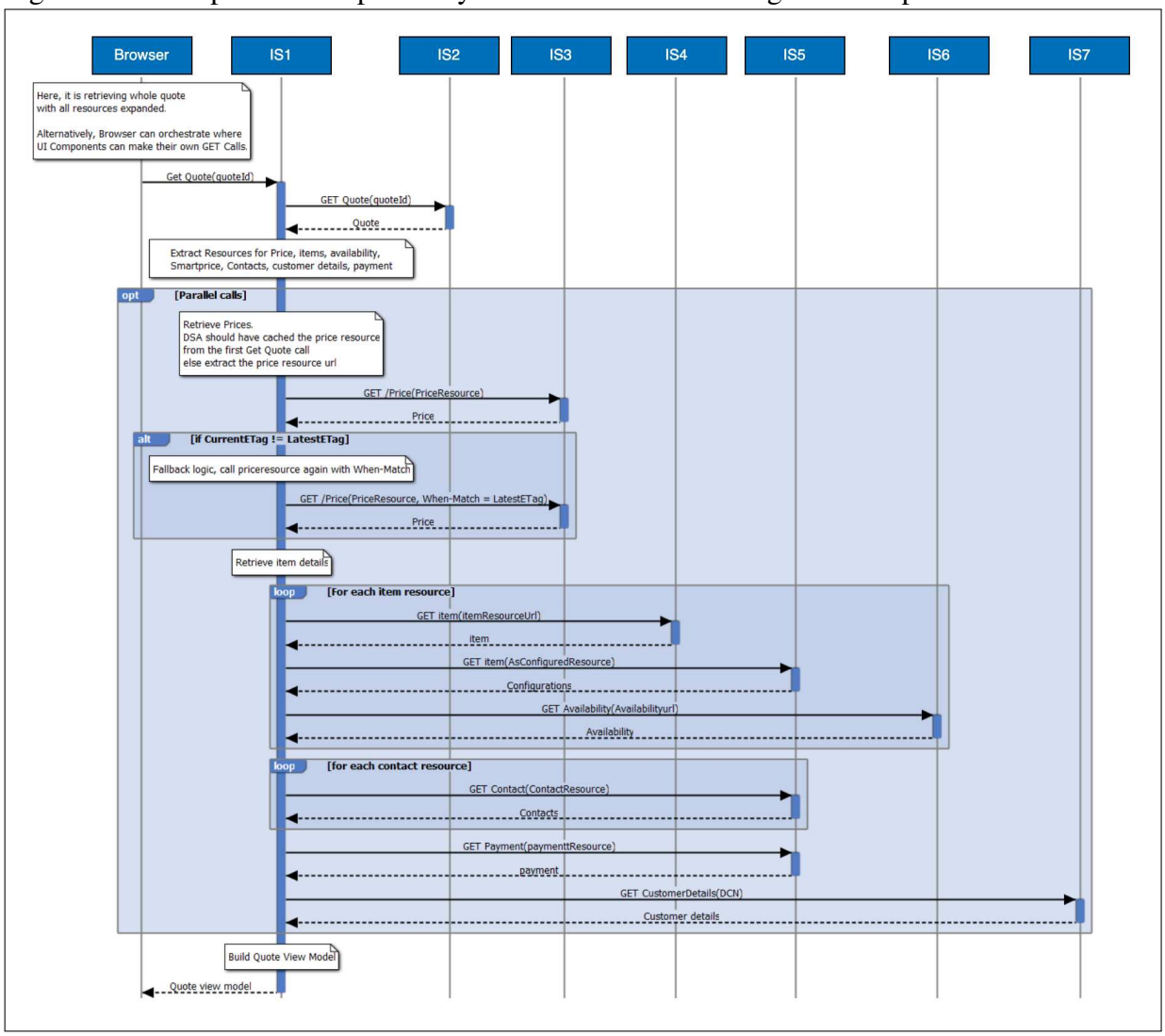
Regarding the research context, this thesis is located on the technical factors of interoperability, specifically in the effort to support interoperability links in SoIS.

1.3 Motivation

Establishing interoperability links is a significant challenge due to multiple aspects, among which the following stand out: (i) planning and designing of the system arrangement, which must be conducted in an orchestrated manner so that the CS input and output interfaces are connected to exchange data or reconfigured to maintain the services offered; (ii) providing technical solutions to overcome the heterogeneity between systems and effectively mediate their interactions (FERNANDES *et al.*, 2022); (iii) enabling data exchange between systems to support collaborative data collection and integrated services; (iv) promoting common standards for service management to improve interoperability; and (iv) development of efficient and reliable discovery process essential for service-oriented computing and data-driven systems while addressing data heterogeneity challenges (SADEGHI *et al.*, 2024).

To increase the chances of achieving these interoperability links, Fernandes *et al.* (2023) established guidelines focused on their design, highlighting the management of interoperability links. The main goal of these guidelines is to facilitate efficient interaction among the CS within the SoIS by providing documentation of the interface types for each CS. This documentation includes details on technical interoperability such as functions and procedures. Ensuring each CS can effectively contribute its specific capabilities to the overall objectives of the SoIS while maintaining its operational autonomy is essential.

Figure 2 – Example of interoperability links from an SoIS of a global computer manufacturer



Source: Author (adapted from SoIS manufacturer internal document).

Moreover, it is not easy to achieve these interoperability links even with such documentation available. This phenomenon was observed in a real SoIS maintained within a global computer manufacturing company. Over 30 months, the author of this thesis composed the research team for a research project in partnership with this company. During this time, CS developers were monitored in activities intended to identify interoperability links through the interface documentation of various CS. In summary, four recurring activities can be highlighted as outlined in (BORGES *et al.*, 2022). First, developers must secure permission credentials to access services of the target CS. Second, they need to understand the technology stack of each CS, which includes its execution environment, architecture, programming languages, and standards. Third, they must gather communication details of the target CS services, such as connection methods, authentication strategies, and data formats. That information can be obtained either

through direct discussions with the CS maintainers or by reviewing the CS service application programming interface (API) documentation. Lastly, developers must continually monitor API changes, as even minor modifications can disrupt integration interoperability and SoIS objective. Manually undertaking these tasks is often difficult, time-consuming, and error-prone.

There are several techniques for establishing CS interoperability in SoIS. Some of them include identifying interoperability links, using reference architectures, carrying out Business Process Modeling Notation (BPMN) studies in companies, and using other Product Lifecycle Management (PLM) systems (BORGES *et al.*, 2024). However, none of these techniques are applied in industrial cases; most are linked to either educational environments (AFOUTNI *et al.*, 2017; BATISTA *et al.*, 2022; FERNANDES *et al.*, 2022; OLIVEIRA *et al.*, 2022) or smart cities (MENDES *et al.*, 2018) in SoIS (BORGES *et al.*, 2024). Furthermore, to the best of our knowledge, none of these recent techniques offer real support for achieving interoperability links based on documentation analysis, especially via API (BORGES *et al.*, 2024). This limitation makes it challenging to identify potential interoperability links through API connection points efficiently and semi-automatically. It is important to highlight that all these issues (the points raised by (FERNANDES *et al.*, 2022) and (SADEGHI *et al.*, 2024), the on-site observation, the gap in the literature on efficient methods for establishing interoperability links, especially via API) motivated the studies conducted in this thesis.

1.4 Problem Statement

Considering the context and motivation previously discussed, the problem statement of this thesis can be formulated as follows:

> "*The difficulty faced by developer teams in establishing interoperability links among CS in SoIS, where they need to deeply inspect API documentation to gain a better understanding of their different data, attributes, and methods to establish suitable API connection points to support the rise of new behaviors in SoIS*."

From this perspective, the next section describes the research question and objectives of this thesis, which aim to address and mitigate the aforementioned problem.

1.5 Research Question and Objectives

According to Wohlin e Aurum (2015), the formulation of Research Questions (RQ) is key, as it defines the process to be followed during the investigation. A well-formulated RQ should direct the study to address relevant problems. Therefore, based on the problem statement stated in Section 1.4, this thesis aims to answer the following research question:

> **Thesis' RQ**. *How to systematically and efficiently support CS development teams in establishing interoperability links among CS in SoIS* ?

Based on this RQ, the main research objective of this thesis is **to develop a reliable semi-automatic method to support CS development teams in identifying API connection points to achieve interoperability links among CS in SoIS**. To accomplish this objective, the following goals have been defined:

G01. Identify the main textual similarity analysis techniques (syntactic and semantic) for API descriptions;

G02. Create an API analysis method, based on textual similarity analysis techniques, to identify API connection points among CS;

G03. Develop a solution, as tool, to support the proposed method;

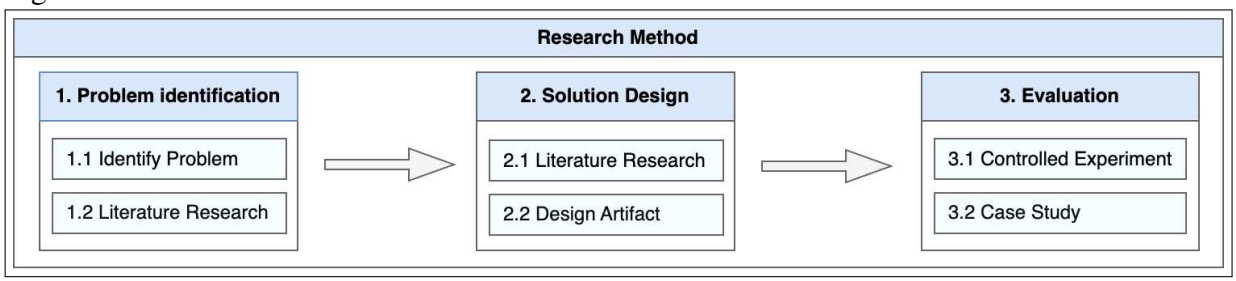
G04. Evaluate the feasibility and efficiency of the solution in SoIS case for identifying interoperability links, using both quantitative and qualitative metrics.

1.6 Research Method

The appropriate choice of a research method is essential to the success of any scientific investigation, offering a guide for systematic exploration and the creation of reliable knowledge. In this sense, we use as base the Design Science Research (DSR) process, proposed by Offermann *et al.* (2009), which is based on Design Science[2] to develop and evaluate artifacts such as frameworks, models and methods, with the aim of facing specific and complex challenges. Thus, the research method used in this thesis is illustrated in Figure 3, and is composed of three main steps: 1 - Problem Identification; 2 - Solution Design; and 3 - Evaluation. These steps are detailed below:

[2] Science that is distinguished from conventional science by its focus on generating knowledge applied to solving problems, having as one of its main objectives, the prescription of solutions (DRESCH *et al.*, 2015).

Figure 3 – Thesis Research Method



Source: Author (inspired by Offermann *et al.* (2009)).

1. **Problem Identification:**
   1.1 Identify Problem: the first step was to identify the problem. This was accomplished through direct observation of CS developers over 30 months, from June 2021 to January 2024, during a project partnership with an SoIS of a global computer manufacturer. During this period, difficulties were observed among developers in achieving interoperability links among the CS of that SoIS already in the first six months of the project. The details of this observation are described in the motivation and problem statement sections.
   1.2 Literature Research (problem): immediately after identifying the problem, we conducted a literature review to understand more about the issue and to verify the main approaches for interoperating the CS. We identified some existing gaps, one of the main ones being the lack of an efficient way to achieve interoperability links between the CS. Based on the results of the problem identification, the literature review, and the application, as well as a manual solution, we published our findings at International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems (SESoS) (BORGES *et al.*, 2022).
2. **Solution Design:**
   2.1 Literature Research (solution): once the research direction was evidenced by the SESoS award, we began to discuss possible solutions to improve the identification of API attributes, which we had been doing manually until then. We realized that API description comparisons could be made using text similarity analyses. To this end, we conducted literature research on the main techniques of both syntactic and semantic similarity, and summarized them into a comprehensive catalog, which is detailed in Chapter 3. From the catalog, we noted that syntactic similarity is ideal for API attributes, which have variable names defined by programmers, making semantic

extraction difficult. The semantic similarity, on the other hand, could be used in the textual descriptions of the API attributes. As a result, We presented the main techniques of syntactic and semantic text similarity analyses (BORGES, 2023a) at the *GESAD Talks* event in State University of Ceará (UECE).

2.2 Design Artifact: after identifying the main text similarity techniques applicable to API descriptions, we selected the method from among the types of artifacts available in DSR. According to March e Smith (1995), the artifact method consists of a structured set of steps, encapsulated by specific algorithms, designed to perform predefined tasks. Based on this understanding, we developed a method aimed at identifying API connection points, based on syntactic and semantic similarity analyses. That method was carefully developed based on previously defined requirements, which guided its design and implementation. Finally, we implemented the method and consolidated it as a functional solution in form of tool. All these aspects can be seen in more detail in Chapter 4.

3. **Evaluation:** Using the tool, we evaluated the method through a controlled experiment (3.1) and a case study (3.2) involving three distinct scenarios. In the controlled experiment, we applied the tool to the Jira and GitHub API to identify a known composition point and determine their API connection points, which could potentially be extended to API interoperability links. With promising results from that analysis, we conducted a case study with two scenarios, using the tool to identify API connection points in four CS within the SoIS. The goal was to verify if the tool could replicate results already known by the CS developers of the SoIS. Semi-structured interviews were also conducted to gather developer feedback on the results. The initial findings were presented at the 3rd LATAM School in Software Engineering (LATAM) Brazilian Conference on Software: Practice and Theory (CBSoft) (BORGES, 2023b), where we received positive feedback from multiple reviewers. Building on this, and through collaboration with researchers from Federal University of the State of Rio de Janeiro (UNIRIO), we expanded the study and submitted a paper with more comprehensive results, focusing on the syntactic part of the method, which was accepted for publication in the IEEE System Journal (ISJ) (BORGES *et al.*, 2024). Subsequently, we extended the case study by adding another scenario and incorporating semantic analysis into the tool to achieve more accurate results. Additional semi-structured interviews were conducted to gather developer perspectives

on this enhanced version of the tool. With these extended results, integrating both the syntactic and semantic components of the method, we plan to submit the findings to the Journal of Systems and Software (JSS). All these aspects can be seen in more detail in Chapter 5.

In addition to the evaluation in the controlled experiment and the case study, we were able to evaluate the reproducibility of our method in other SoS scenarios. This emerged during my trip to CBSoft 2023, when I had the opportunity to talk with several researchers, including some from University of São Paulo (USP), who identified the possibility of collaboration with our work. At their invitation, I visited USP to develop this research partnership. In some meetings, we identified that we could apply the syntactic analysis part of the method of this thesis in a Brazilian Space SoS, which includes several CS such as satellites, ground stations, mission centers, among others. Our method proved to be suitable for generating architectural configurations based on readings of the properties of the dynamic behavior of the CS both for sending and receiving messages, which can also be considered to achieve interoperability links in SoS, since when there is a failure in a CS, others can establish other alternative communications, links, so that the SoS does not fail. As a result of this partnership, we produced and approved an article, in Borges *et al.* (2024), at SESoS.

1.7 Thesis Outline

The thesis is structured in six chapters, including the Introduction, which are described below.

Chapter 2, named Background, provides the fundamental basis for understanding this thesis. It presents the main concepts of SoIS, API description, and related work, which are fundamental for the discussions and analyses in the subsequent chapters.

Chapter 3, Catalog of textual similarity analysis techniques, which aimed to obtain the main textual (syntactic and semantic) similarity analysis techniques in the literature for API descriptions in order to support our API identifying connection points method.

Chapter 4 describes in detail the method of this thesis. These details demonstrate its overview and main steps: (i) requirements definition, where the reasons for creating the method are outlined; (ii) core method design, where they are all the phases and components necessary to obtain the API connection points are detailed; and (iii) core method implementation, which

presents the technological details involved in the tool developed for the method.

Chapter 5 evaluates the method, through the tool, in two contexts. The first is a controlled experiment, which aim to determine the best syntactic similarity algorithms and thresholds for identifying connection points between two known API. The second is a case study, where the tool is applied into three scenarios, involving seven real CS API from a real-world SoIS of a global computer manufacturer. The results were obtained through interviews with five experienced developers, revealing relevant implications for both developers and researchers, as well as identifying existing limitations, and threats to validity.

Finally, Chapter 6 presents the conclusions of this thesis, providing a summary, the contributions, the published scientific artifacts, and the future work.

2 BACKGROUND

This chapter provides the essential background necessary for a comprehensive understanding of this thesis. Section 2.1 presents the fundamental concepts of SoIS, with emphasis on their main characteristics, classifications, and aspects related to interoperability links. Section 2.2 delves into API description, addressing its concept, distinctive features, and key description languages. Section 2.3 are discussed the main related work. Finally, Section 2.4 presents the final remarks.

2.1 System-of-Information Systems

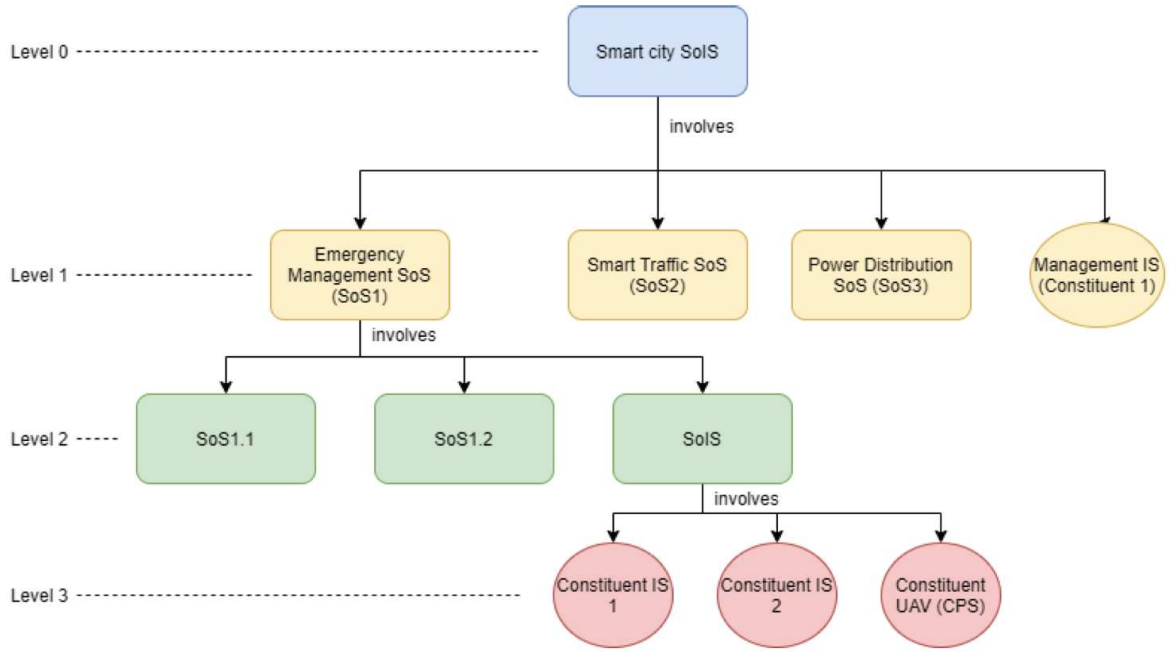
SoIS is a specialized category of SoS focused on business processes, which combine software-intensive IS with various other systems from different organizations (TEIXEIRA *et al.*, 2019). SoIS have emerged from the modernization of business processes and the globalization of a world where these IS interoperate to provide services, form commercial partnerships with suppliers, comply with regulations, and deliver products in a more agile and optimized manner (NETO *et al.*, 2017).

It is essential to highlight that, in order to be considered an SoIS, the presence of an IS is fundamental. This means that advanced techniques for SoS are not enough by themselves, especially concerning the specific methods for designing, implementing, and developing these systems, which include the close relationship between the technical and business levels (NETO *et al.*, 2021). Figure 4 presents a smart city SoIS composed of some SoS: Emergency Management SoS (SoS1), Smart Traffic SoS (SoS2), Power Distribution SoS (SoS3), and a Management IS (Constituent 1). The smart city is considered an SoIS due to the presence of Constituent 1. On the other hand, SoS1 is not an SoIS, although one of its constituents is an SoIS, since, if the IS is not at the direct hierarchical level, it will not require the specific engineering needs of SoIS nor will it involve a business process to interoperate its constituents. By the same logic, the SoIS (level 2) is an SoIS because it contains at least an IS in its hierarchy.

2.1.1 Characteristics

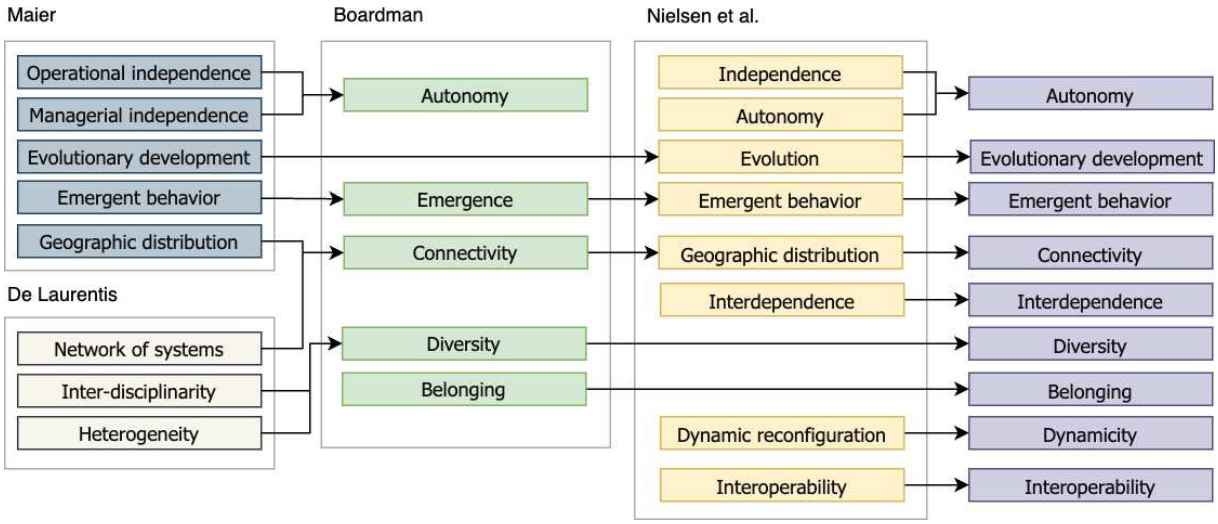
The SoIS characteristics have been extracted and refined over time from SoS (LU *et al.*, 2010; FERNANDES *et al.*, 2020). Figure 5 illustrates these characteristics discussed by SoS authors, such as Maier, De Laurentis, Boardman, and Nielsen, showing their correlation,

Figure 4 – Smart city SoIS hierarchy composed of SoS and SoIS



Source: Neto *et al.* (2021).

Figure 5 – SoIS characteristics



Source: Fernandes *et al.* (2020).

traceability, and evolution to the concept of SoIS (FERNANDES *et al.*, 2020). Among the main characteristics, which will be detailed below, are: autonomy, evolutionary development, emergent behavior, connectivity, interdependence, diversity, belonging, dynamism, and, finally, interoperability.

Autonomy refers to the ability of a system to be governed by its own rules, despite external influences. That implies operational independence (dissociation) of the components, meaning that IS maintain their own operations and are not exclusively dedicated to the purposes of the SoIS (FERNANDES *et al.*, 2020). Moreover, that autonomy is related to managerial independence, as different companies own and manage the involved IS. For example, the constituents perform their functions according to the rules of the organization to which they belong, even while addressing a global objective of the arrangement (MAIER, 1998). In other words, a constituent also operates in favor of an individual objective outside the arrangement (BOARDMAN; SAUSER, 2006).

Evolutionary development manifests when the entire arrangement evolves in response to the evolution of its constituents and presents constantly evolving purposes (MAIER, 1998). That evolution of the constituent IS occurs in response to the individual needs of organizations, bringing benefits to the SoIS (FERNANDES *et al.*, 2020). For example, a new functionality may be created in an IS, generating a new capability that can be leveraged by the SoIS. However, some IS functionalities may be discontinued. The SoIS must adapt to these changes to maximize gains or minimize undesirable effects. That characteristic makes an SoIS not appear to be fully formed.

Emergent behavior is a holistic phenomenon, intentional or not, that arises as a result of interoperability between constituent IS to achieve objectives. This behavior is the result of synergistic collaboration among constituents and is attributed to the arrangement as a whole. It may be desired or undesired, because, despite the benefits provided, the autonomy of IS causes the teams that manage them to tend to prioritize the individual objectives of these systems (GOROD *et al.*, 2008).

Connectivity refers to the need for communication between dispersed constituents, requiring them to be interconnected (MAIER, 1998; FERNANDES *et al.*, 2020). This characteristic is also related to the distribution of CS in SoS (MAIER, 1998). SoIS is considered a complex system because, according to the Systems Engineering Body of Knowledge (SEBoK), when several independent IS work together to achieve a common goal, a complex system is

formed (FERNANDES *et al.*, 2020). A complex system is composed of multiple independent and interdependent parts that interact in a non-linear manner, meaning that its behavior cannot be precisely expressed as the exclusive result of the sum of the activities and/or individual parts (ALAMPALLI; PARDO, 2014).

Interdependence, as a characteristic of SoIS, is explained by the necessary interaction between the CS. Similar to complex systems, interdependencies in SoIS are difficult to describe, predict, and design (ALAMPALLI; PARDO, 2014). This difficulty can be attributed to the diversity of the CS.

Diversity refers to the variety of elements present in an SoIS. Technological heterogeneity (e.g., programming languages, development paradigms, data formats), organizational heterogeneity (e.g., data semantics, missions, own human resources), human heterogeneity (e.g., diverse professional experiences, communication aspects, relationships between stakeholders), or a combination of these, illustrates this characteristic of diversity (FERNANDES *et al.*, 2020).

Belonging refers to the perception that a constituent has about the mutual benefit between the parties involved (BOARDMAN; SAUSER, 2006). Thus, the belonging of an IS to an arrangement such as the SoIS is less evident than that of a subsystem in a system composition. In a system composition, the subsystems are available to serve the purpose of the arrangement, often being merely components of a larger system, which do not exist decoupled from this system. Therefore, it is important to highlight that the SoIS must consider that the constituents can abandon the arrangement at any time due to the autonomy of the CS itself (SALADO, 2015).

The dynamicity of an SoIS architecture is explained by the possibility that CS can leave the arrangement at any time, while new constituents can join to achieve the formation goal. In that context, the mutation of the arrangement and the termination of interoperability links occur in the SoIS (FERNANDES *et al.*, 2020).

Finally, interoperability is the ability of an SoIS to communicate with a diversity of heterogeneous IS (FERNANDES *et al.*, 2020). This involves adapting interfaces, protocols, and standards to effectively connect legacy systems and newly designed systems, while ensuring the consistency needed to meet verification requirements and ensure compliance.

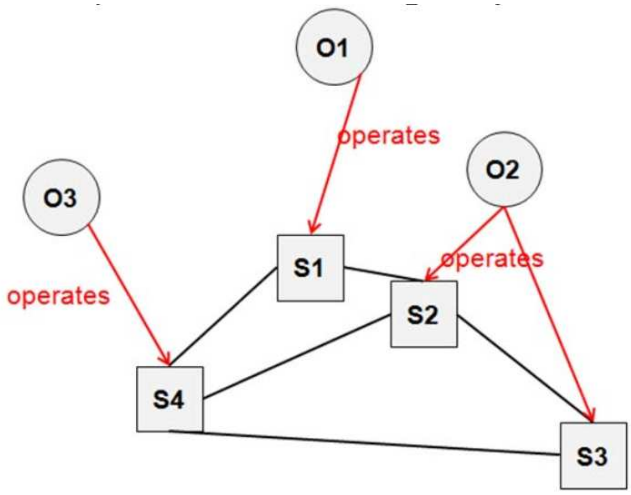
2.1.2 Classification

According to Neves *et al.* (2020), the SoIS classification follows the same structure as SoS. That classification indicates the level of management control that influences an its

architecture, determining how adaptable and cooperative each constituent system will be in relation to requirements, interfaces, data formats, and technologies (NIELSEN *et al.*, 2015). SoS (and SoIS), detailed below, can be categorized as virtual, collaborative, acknowledge, and directed (MAIER, 1998; NEVES *et al.*, 2020).

A virtual SoS has no central management authority or centrally agreed purpose (MAIER, 1998). Large-scale emergent behavior may occur and may even be desirable, but this type of SoS relies on relatively invisible mechanisms to maintain itself (NCUBE; LIM, 2018). An example of a virtual SoS is shown in Figure 6. It can be seen that the owners (O1, O2, O3) use their own systems to access other systems, seeking to achieve individual benefits, but there is no common goal or centralized management (NCUBE; LIM, 2018). Interoperation between them is achieved through recognized protocols or standards, rather than through specific agreements between pairs of systems (NCUBE; LIM, 2018).

Figure 6 – Example of Virtual SoS

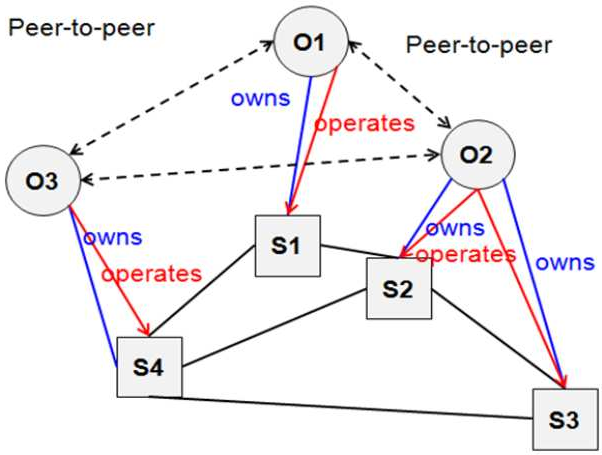


Source: Ncube e Lim (2018).

In a collaborative SoS, the CS interact voluntarily to achieve agreed-upon central purposes but do not have a central management Tekinerdogan (2019). Maier (1998) highlights that an example of this is the Internet, where the Internet Engineering Task Force (IETF) develops standards but does not have the power to enforce them. In that context, the main participants collectively decide how to provide or deny services, which, in practice, imposes and maintains certain standards. Figure 7 illustrates a collaborative SoS, where the constituents operate under a mutual agreement of cooperation, usually formalized by contracts or specific agreements (NCUBE; LIM, 2018). Despite this, there is no central management entity that defines the SoS requirements. The system owners (O1, O2, O3) retain control over their own systems and

collaborate with each other to achieve a shared benefit (NCUBE; LIM, 2018).

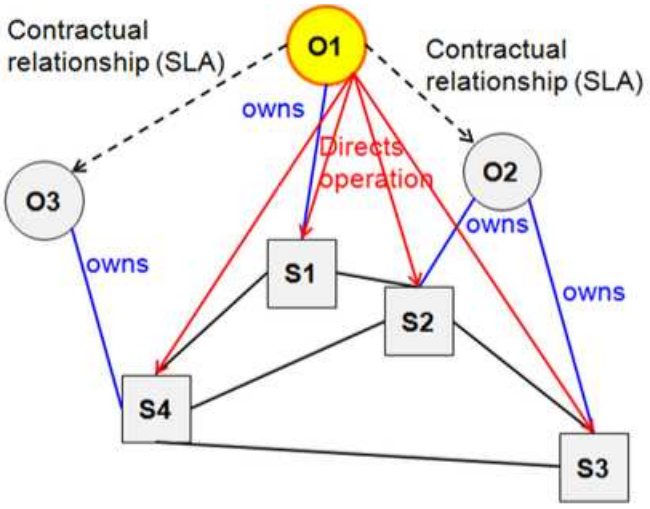
Figure 7 – Example of Collaborative SoS



Source: Ncube e Lim (2018).

Acknowledge SoS have shared objectives, a designated management authority, and dedicated resources (FERREIRA, 2023). However, the CS maintain their independence in terms of ownership, goals, funding, development, and sustainment (NCUBE; LIM, 2018). Any changes in the systems are carried out based on collaboration between the SoS and its CS. In Figure 8, an Acknowledge SoS is illustrated, where O1 directs the selection of systems and operations, while O2 and O3 maintain a contractual relationship with O1. In that case, the central management entity (O1) defines the requirements at the SoS level but exerts less control over the CS, which are owned by other stakeholders, such as O2 and O3 (S2, S3, S4) (NCUBE; LIM, 2018).

Figure 8 – Example of Acknowledged SoS

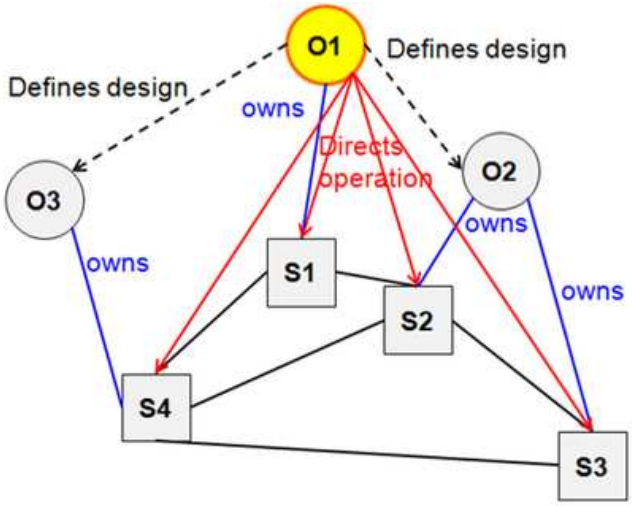


Source: Ncube e Lim (2018).

Finally, Directed SoS are those in which the set of CS are built and managed with

the objective of fulfilling specific purposes (TEKINERDOGAN, 2019). That type of SoS is centrally managed throughout its operation, ensuring that it continues to meet these purposes as well as any new purposes that may emerge (FERREIRA, 2023). Although the CS may operate independently, their normal operation is subordinate to the centrally managed purpose (MAIER, 1998). In Figure 9, a Directed SoS is illustrated, where O2 and O3 follow the direction of O1 in terms of requirements, specifications, and operation of the systems they manage (O2 is responsible for systems S2 and S3; O3 for S4) (NCUBE; LIM, 2018). That type of SoS is highly controlled by the central management entity (O1) (NCUBE; LIM, 2018).

Figure 9 – Example of Directed SoS



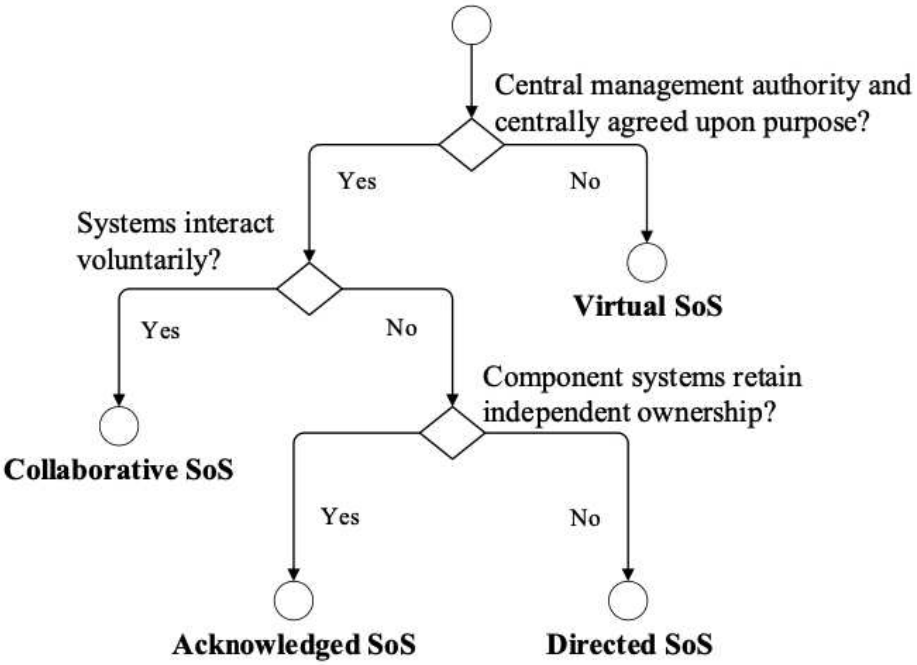
Source: Ncube e Lim (2018).

Figure 10 clearly summarizes how the classification of SoS can be understood based on their management policies. According to Tekinerdogan (2019), the first step is to determine whether there is a central management authority and centrally agreed upon purpose. If not, the SoS is classified as a Virtual SoS. If such authority exists, the next step is to check whether the systems interact voluntarily. If the answer is yes, the SoS is classified as a Collaborative SoS. If not, the final question is to check whether the component systems retain independent ownership. If they do, the SoS is classified as an Acknowledged SoS; otherwise, it is categorized as a Directed SoS.

2.1.3 Interoperability links

The concept of interoperability links has been extensively explored in Fernandes *et al.* (2020). She describes these links as connections established between two or more CS

Figure 10 – Different types of SoS based on management policy



Source: Tekinerdogan (2019).

to enable the exchange of messages, such as requests, and the interpretation of results via URL, using HTTP methods such as GET or POST. Fernandes *et al.* (2020) also highlights that managing these links and adjusting the level of interoperability between organizations is particularly challenging. One of the main challenges lies in the independence of IS, which allows them to join or leave an alliance according to the specific needs of the responsible organization. This flexibility can result in ephemeral interoperability links.

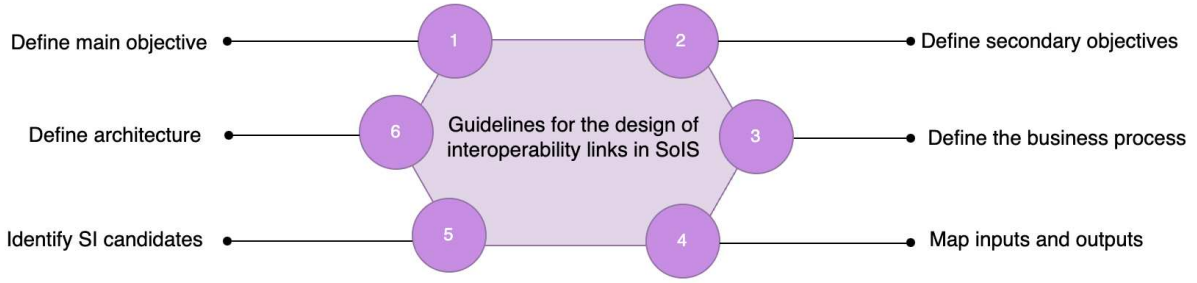
In her master's dissertation, she proposed comprehensive guidelines for the design of interoperability links in SoIS (FERNANDES *et al.*, 2020). The main objective of these guidelines is to support managers in creating and managing effective SoIS arrangements. These guidelines, which total six, are illustrated in Figure 11 and are detailed below:

- Defining the main objective is the first step in the SoIS requirements elicitation process, as it involves identifying stakeholders and their needs, and facilitates the identification of IS. Furthermore, defining objectives can be crucial for recognizing potential conflicts, especially important in SoIS, where constituents tend to prioritize their own individual objectives.
- Defining secondary objectives is the second step, and results from refining the main SoIS objective. That step is essential to identify the capabilities of the IS and for establishing the business process that will guide the operation of the arrangement. Secondary objectives

are related to functional and quality expectations, such as reliability, availability, and performance. Clarity in defining these objectives provides insights into the capabilities required to form the arrangement and helps identify the IS that can provide these resources.

– Defining the business process is the third step and involves the logical sequence of activities required to achieve a specific objective. The process details the responsibility of each activity and the interoperation between the involved actors, showing how they interact to achieve the desired result. Identifying these interactions helps to better understand the SoIS, the interdependencies between IS, the responsibilities, and the risks. The interactions can be between machines, machines and humans, or humans, depending on the level of interoperability desired.
– Mapping input and output formats is the fourth step, and aims to ensure that the SoIS operates with data as homogeneous as possible. Since SoIS are composed of loosely coupled and independent IS, each with its own data model, this step is essential to identify and resolve possible differences in data types and formats between IS.
– Identifying IS candidates is the fifth step to fulfill the SoIS primary goal, directly depending on the capabilities of the CS. The combination of these capabilities generates an emergent behavior. Therefore, that step involves identifying the IS that have the necessary capabilities to achieve the SoIS goals established in the previous steps.
– Defining the SoIS architecture is the sixth and final step, which involves developing a high-level description of the design. That step includes representing the software components, such as objects, processes, and data repositories, as well as their externally visible properties and the relationships between them. In SoIS context, these components are the CS.

Figure 11 – Guidelines for the design of interoperability links in SoIS



Source: Adapted from Fernandes *et al.* (2020).

Although the mentioned guidelines are primarily aimed at supporting managers, it

is important to highlight that the items related to the mapping of input and output formats, as well as the identification of IS candidates, can be used to support CS developers in establishing interoperability links. That can be done by analyzing the IS documentation, in order to provide SoIS emerging behaviors, taking into account their independence and autonomy.

2.2 API Description

An API is a set of definitions and protocols that facilitate communication between different software applications (OFOEDA *et al.*, 2019). API allow such software to share and use resources in a clear and efficient way, often using web technologies, such as HTTP to exchange data (SURWASE, 2016). Unlike a graphical user interface, API operate behind the scenes, being accessed primarily by developers to create new applications or services (ESPINHA *et al.*, 2014). They are generally designed with a focus on simplicity and efficiency, and are fundamental in building distributed systems, where they facilitate the selective sharing of information by exposing only the necessary data while keeping other internal details of the system hidden, contributing to overall security (WETTINGER *et al.*, 2015).

API documentation serves as a technical manual, providing detailed instructions on how to use the API and access its services. When well designed, that documentation not only improves the user experience, but also plays a crucial role in the success of the API (GOODWIN, 2024). The documentation should provide an explanation for each API request and examples of error messages. It is also important that the API documentation is actively maintained and always up to date (WAGNER, 2024).

For API documentation, API descriptions are essential, allowing developers to clearly understand the API purposes and functionalities (CREMASCHI; PAOLI, 2017). Additionally, they provide an overview of the available operations, specify the required parameters, and describe the response formats, facilitating integration with other systems and the development of robust applications (LUCKY *et al.*, 2016). Clear and detailed descriptions help to avoid misunderstandings and reduce the time it takes for new users to start using the API effectively (LUCKY *et al.*, 2016).

To simplify the construction of API descriptions and their behaviors, API description languages are used, which are domain-specific languages designed to describe API clearly and precisely, being readable by both humans and machines (BIEHL, 2016). Like programming languages, they are intuitive and can be easily written, read, and understood by API developers

and designers (BIEHL, 2016). In addition, they have a well-defined syntax, allowing them to be automatically processed by software, minimizing ambiguity and making them expressive and powerful (BIEHL, 2015). Any API description language must have essential properties, as it functions as a clear and unambiguous technical contract between the API provider and the consumer. According to Biehl (2015), the main characteristics of these languages can be cited:

- Compactness: the contract should be as concise as possible, including only what is necessary and relevant. Repetitions should be avoided through appropriate language abstractions.
- Precision: the API description language used to specify the contract must be precise and unambiguous.
- Relevance: language features should be directly relevant to the API design, without including unnecessary or superfluous information.
- Agility: It is easier to iterate on a design document than on source code. Furthermore, iterating on a dedicated description is even simpler than on a detailed API description.
- Clarity and structure: A detailed API description cannot be as clear, unambiguous, and well-structured as a dedicated language.
- Communication: although API description languages are technical, they should be understandable to non-programmers, such as architects or stakeholders. HTML documentation generated from them can reach an even broader audience.
- Quick validation: it should be possible to validate an API description quickly and easily, preferably in an automated manner.
- Quick Iteration: the API description language should support an agile and iterative development approach. Based on an initial draft, it should be possible to create a more refined second iteration, adjusting the level of detail as needed.
- Intuitive: the API description language should be intuitive and easy to use, without requiring extensive prior training.

In addition to the essential properties mentioned, it is important to highlight that several API description languages play a fundamental role in the definition, documentation and communication of API in different contexts. Each of them offers a unique set of functionalities and features, adapting to different needs and scenarios. Among these languages, the following can be highlighted:

- OpenAPI (formerly known as Swagger): it is an API description language, specifically

for Representational State Transfer (REST) API (BIEHL, 2015). It allows developers to specify the API structure and behavior in a format that is both human-readable and machine-readable, facilitating the automatic generation of documentation, input and output parameters for each operation, authentication methods, contact information, license, terms of use, and other information (SWAGGER, 2024). API specifications can be written in either YAML or JSON language, which will be described below (OPENAPI, 2020).

- RESTful API Modeling Language (RAML): it is a language designed to simplify the definition of REST API, recognized for its simplicity and ease of use (BIEHL, 2015). It allows developers to describe API resources, methods, and parameters in a clear and structured manner (BIEHL, 2015). Also based on YAML, the language facilitates the generation of both client and server source code, as well as the creation of comprehensive documentation for API users (RAML, 2024).
- API Blueprint: this API description language is known for its documentation-oriented approach (BIEHL, 2015). API Blueprint is designed to be easy to read and write, using logically-based markdown syntax, where each has its own distinct meaning, content, and position in the document, making it easy to create detailed and interactive documentation (APIBLUEPRINT, 2024). In addition to regular markdown syntax, API Blueprint adheres to GitHub flavored markdown syntax, which is a markdown dialect used for user content on GitHub and GitHub Enterprise (APIBLUEPRINT, 2024; GITHUB, 2024).
- Web Services Description Language (WSDL): it is a description language used primarily for SOAP-based web services (BIEHL, 2015). WSDL allows you to separate the description of the abstract functionality offered by a service from the concrete details of a service description, such as “how” and “where” that functionality is offered (W3C, 2024). It is important noting that WSDL is not suitable for describing REST services, as it does not support JSON data structures or API security schemes, such as OAuth. Its syntax is based on Extensible Markup Language (XML) (IBM, 2023).

In this thesis, we will give special emphasis to the API description language OpenAPI, chosen as the basis for our method’s tool due to: (i) its wide recognition and adoption in the definition of API; (ii) its robust and flexible structure, which allows developers to specify in detail the behavior and architecture of API; and (iii) its use as a API description language in the CS used in the evaluations of this thesis.

OpenAPI has eight essential properties that facilitate API specification, as shown

Table 1 – OpenAPI Properties

Property	Description
openapi	OpenAPI version
info	Provides metadata about the API, such as title, description, terms of service, contact, license and version.
servers	An array of server objects, which provide connectivity information to a target server.
security	A declaration of which security mechanisms can be used across the API, mainly authentication and authorization options
paths	The available paths and operations for the API, prefixed by server URL.
tags	groups paths into categories
components	An element to hold various schemas for the specification.
externalDocs	Additional external documentation.

Source: Koren e Klamma (2018), OpenAPI (2020).

in Table 1 (OPENAPI, 2020). The *openapi* property indicates the version of the specification used, ensuring compatibility. *Info* provides crucial metadata such as title, description, and version that are fundamental to identifying and understanding the API purpose. *Servers* defines the servers that provide connectivity, while *security* specifies security mechanisms, such as authentication , and authorization. *Paths* describes the paths and operations available in the API, organized by *tags* that group these paths into categories. *Components* stores reusable schemas, and *externalDocs* provides links to additional external documentation.

As previously reported, the OpenAPI specification can be expressed either YAML or JSON language (KOREN; KLAMMA, 2018). YAML is known for its readability, using indentation to represent the hierarchical structure of properties, which are realized as key-value pairs and objects (BIEHL, 2015). JSON, in the other hand, is a JavaScript-derived language widely used in API for its compatibility with several programming languages and ease of use in data transmission (SMITH, 2015). Such language is structured by keys and values, and the hierarchy is represented by nested square brackets, {}.

Listing 1 exemplifies the structure of an OpenAPI specification for an Address Book Service, using YAML syntax. It demonstrates how OpenAPI properties are employed to define the API key elements, including the version used (*openapi*, line 1), the available servers (*servers*, lines 2-4), general API informations (*info*, lines 5-8), the categorization of API operations (*tags*, lines 9-11), the different paths along with details of endpoints, methods, attributes, attribute descriptions, responses (*paths*, lines 12-66), and data schemas that can be reused across different API parts (*components*, lines 67-80). The output of Listing 1 can be visualized in Figure 12 using Swagger user interface (UI), which is an open source software that automatically generates an HTML5-based visualization and interaction interface from an OpenAPI file.

In that same figure, we can see three encapsulated paths. By expanding the “GET */contacts*” (Figure 13), we can see that this request has no parameters, and returns as a response, an array of *Contact* objects defined in components. By expanding the “GET */contacts/contactId*”

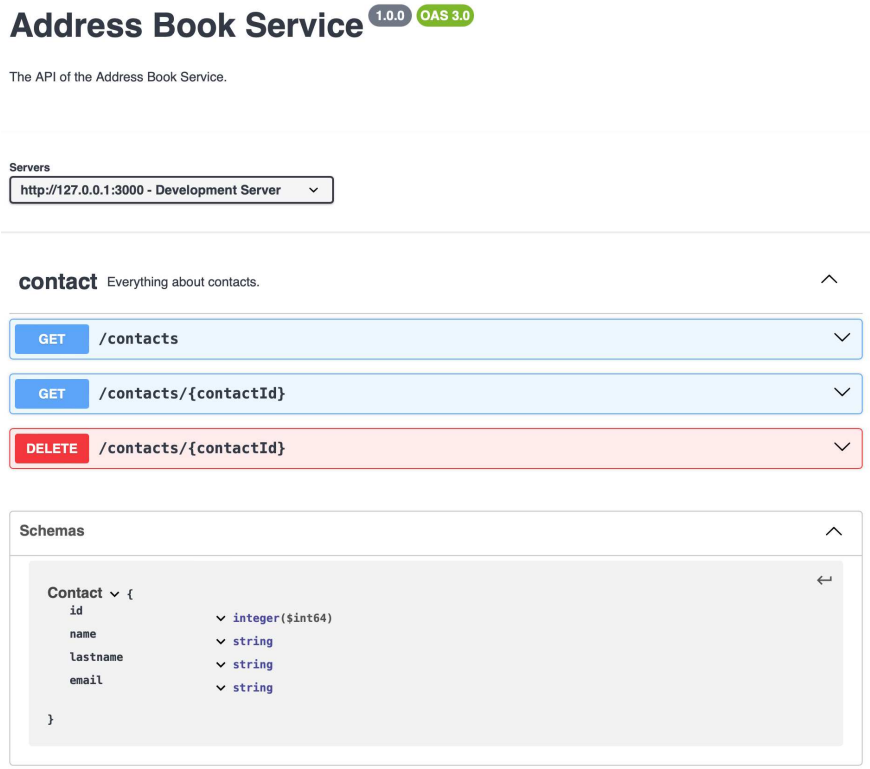
```yaml
openapi: 3.0.0
servers:
- description: Development Server
  url: http://127.0.0.1:3000
info:
  version: 1.0.0
  title: Address Book Service
  description: The API of the Address Book Service.
tags:
- name: contact
  description: Everything about contacts.
paths:
  "/contacts":
    get:
      tags:
      - contact
      description: Returns all contacts.
      operationId: getContacts
      responses:
        '200':
          description: All the contacts.
          content:
            application/json:
              schema:
                type: array
                items:
                  "$ref": "#/components/schemas/Contact"
  "/contacts/{contactId}":
    get:
      tags:
      - contact
      description: Returns a particular contact.
      operationId: getContactById
      parameters:
      - in: path
        name: contactId
        description: ID of a contact.
        required: true
        schema:
          type: integer
          format: int64
      responses:
        '200':
          description: A specific contact.
          content:
            application/json:
              schema:
                "$ref": "#/components/schemas/Contact"
    delete:
      tags:
      - contact
      description: Deletes a contact.
      operationId: deleteContactById
      parameters:
      - in: path
        name: contactId
        description: ID of a contact.
        required: true
        schema:
          type: integer
          format: int64
```

```yaml
      responses:
        '200':
          description: Contact deleted.
        '404':
          description: Contact not found.
components:
  schemas:
    Contact:
      type: object
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
        lastname:
          type: string
        email:
          type: string
```

Listing 1 – Address book example in OpenAPI (yaml). Adapted from Koren e Klamma (2018).

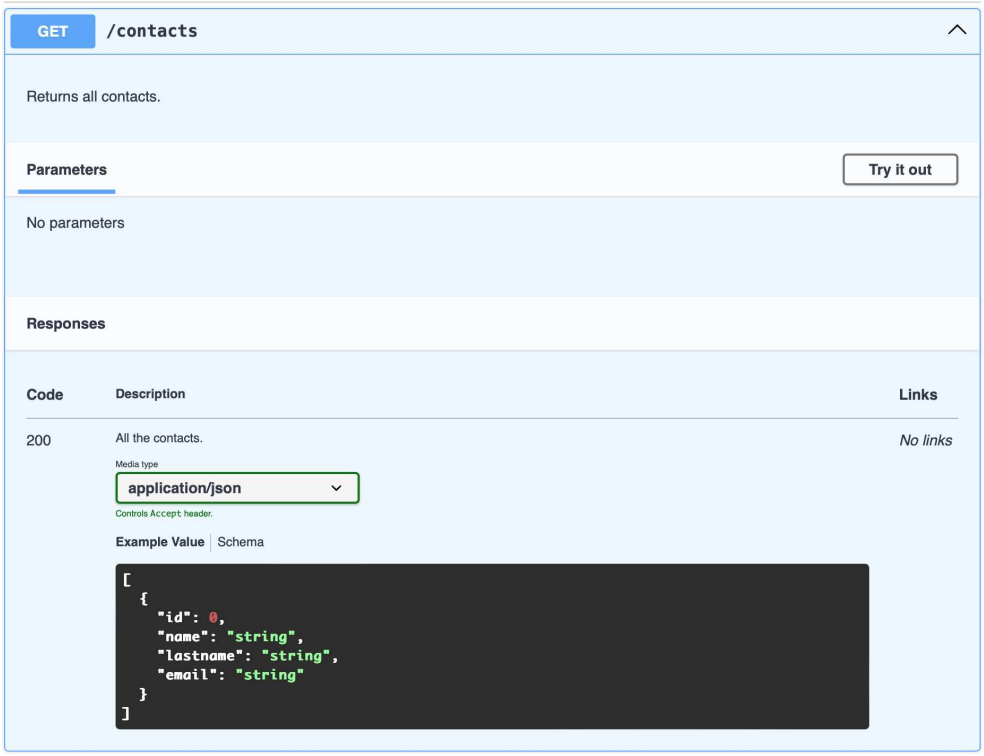
(Figure 14), we can see that it requires a *contactId* parameter and returns the corresponding *Contact* object. Finally, the "DELETE */contacts/contactId*" operation (Figure 15) deletes the contact identified by *contactId*. The response from this operation indicates whether the deletion was successful ("Contact deleted") or if the contact was not found ("Contact not found").

Figure 12 – Address Book Service in OpenAPI (yaml) in Swagger UI



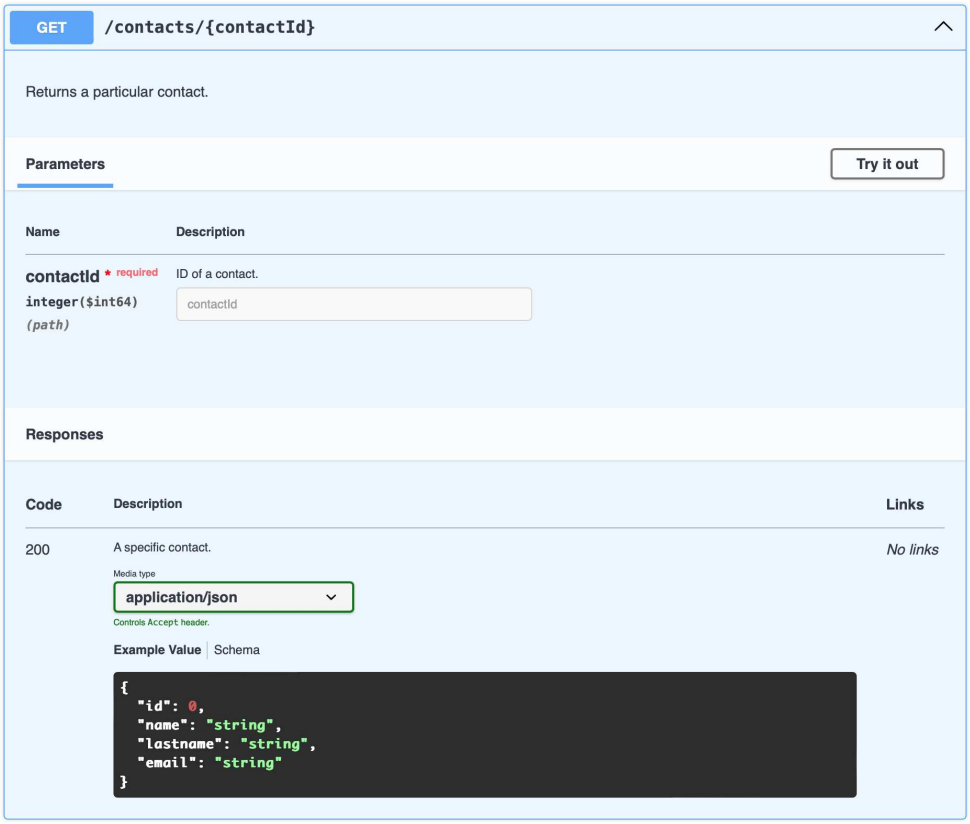
Source: Author.

Figure 13 – Address Book Service (yaml) - Expanded GET method “/contacts”



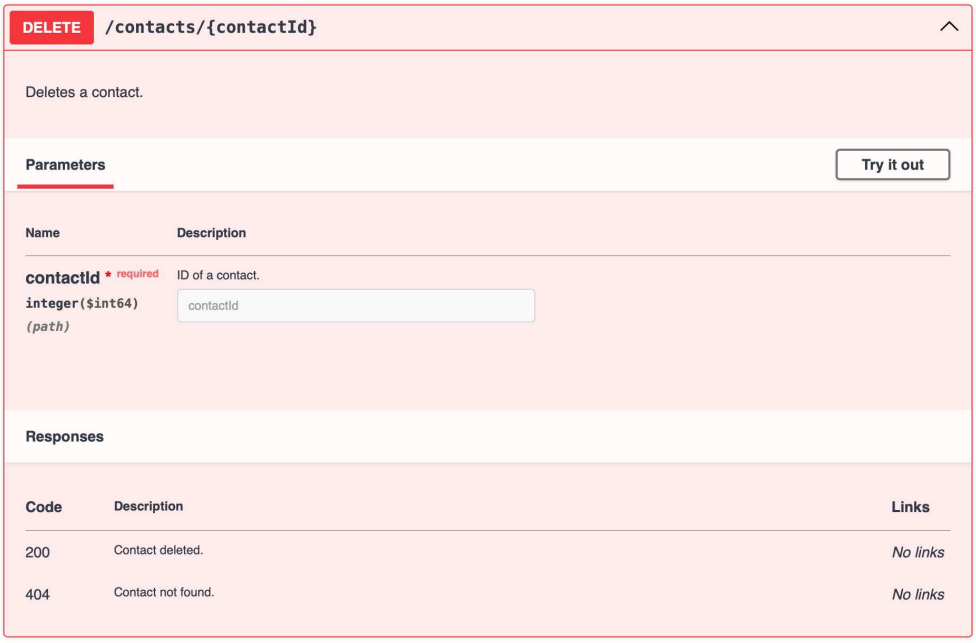
Source: Author.

Figure 14 – Address Book Service (yaml) - Expanded GET method “/contacts/contacId”



Source: Author.

Figure 15 – Address Book Service (yaml) - Expanded DELETE method “/contacts/contacId”



Source: Author.

2.3 Related Work

The related work discussed below follows adapted guidelines proposed by Kitchenham *et al.* (2015), where we used a manual search process in journals and conference proceedings focused on “methods, processes and tools” related to SoIS, and we complemented this approach with the forwards snowballing, which consists of identifying all articles that cite a previously known article or set of articles. Due to the difficulty of finding recent works focusing on SoIS, especially industrial cases, we included articles up to 8 years old (up to the current date of this thesis) involving case studies, action research and experiments in the eligibility criteria to ensure relevance (BORGES *et al.*, 2024). As a result, we selected nine articles (seven related to SoIS and two in a correlated area) and compared them to our method from five perspectives (P): P1 - capacity to support the emergence of SoIS; P2 - assistance in achieving interoperability links; P3 - possibility of validation by experts; P4 - applicability in industrial cases; and P5 - capacity to provide tools for reproducing results (BORGES *et al.*, 2024). It is important to highlight that these perspectives were carefully extracted from the most recurrent characteristics identified in the analyzed articles, in order to ensure traceability and alignment with the existing literature. For example: P1 was conceived from studies that highlighted the need for methods to support the emergence of new SoIS, considering some SoIS characteristics, such as emergent behavior, interdependence and interoperability, which were described in section 2.1.1. P2 was based on

articles that explored interoperability links, a recently debated topic in SoIS interoperability research. In turn, P3 and P4 were derived from works that emphasized the importance of qualitative validation conducted by experts and the practical application of methods in real industrial contexts. Finally, P5 was based on our recent study (BORGES *et al.*, 2024), that highlights reproducibility as essential for scientific advancement and the reliability of results. Table 2 highlights these perspectives (P1-P5) from related work and differentiates them in relation to our method.

Table 2 – Comparison of related work with our method

Related Work	**P1**	**P2**	**P3**	**P4**	**P5**
Fernandes et al. (FERNANDES *et al.*, 2022)	●	○	○	○	○
Batista et al. (BATISTA *et al.*, 2022)	●	○	●	○	○
Oliveira et al. (OLIVEIRA *et al.*, 2022)	●	○	○	○	○
Afoutni et al. (AFOUTNI *et al.*, 2017)	●	○	○	○	○
Mendes et al. (MENDES *et al.*, 2018)	●	●	○	○	○
Li (LI, 2021)	●	●	○	○	○
Saleh and Abel (SALEH; ABEL, 2018)	●	●	○	○	○
Yu et al. (YU *et al.*, 2023)	○	●	○	●	○
Dhoopati (DHOOPATI, 2023)	○	●	○	○	○
Our method	●	●	●	●	●

Source: Borges *et al.* (2024)

Fernandes *et al.* (2022) propose a six-phase method (i.e., objective setting, sub-objectives, identifying IS candidates, interaction flows, mapping inputs/outputs, and architecture) for identifying potential interoperability links, supporting the development of an SoIS. Applied in an educational environment SoIS at a school in Brazil, the method proved effective and yielded good results. In Batista et al. (BATISTA *et al.*, 2022), the authors present the ARC-SoISE, a reference architecture to assist system architects in deriving CS and evolving IS. The goal is to effectively integrate an educational SoIS. Validated by experts, the ARC-SoISE meets 92.2% of the essential criteria to achieve the objectives of an SoIS.

Oliveira *et al.* (2022) introduce a method based on Business Process Modeling Notation (BPMN) for identifying IS to form an SoIS through the analysis of organizational business processes. This method was successfully applied in the academic secretarial sector of a university campus, clearly identifying the integration needs among the IS. Afoutni *et al.* (2017) propose an SoS for Product Lifecycle Management (PLM), an approach aimed at managing a product throughout its lifecycle, using multi-agent systems to handle interoperability among

various involved IS, even though business cooperations may change. To do so, the authors developed a prototype using ARAS, a PLM platform, and MEMORAe, a web platform to manage heterogeneous knowledge.

Mendes *et al.* (2018) present a platform, called Mandala, that facilitates interoperability among distinct and autonomous IS in SoIS. It operates through a software layer, employing business process models and software agents to promote integration. The effectiveness of Mandala was proven in a case study with systems from a smart city, demonstrating its capacity to integrate systems in accordance with the demands of the SoIS. Li (2021) proposes a Collaborative Working Environment (CWE) as an ontology-based collaborative SoIS to apply algorithms that generate contextual recommendations for collaborators. From the proposed approach, the author developed a prototype that was applied, tested, and evaluated on a dataset of scientific collaborations.

Saleh e Abel (2018) present an SoIS approach to support students in managing educational resources to provide the ability to index, share, annotate and recommend important resources in the learning environment, through the integration of different information systems in the educational environment. The evaluation of the proposed approach was carried out in a case study at the University of Technology of Compiègne.

It is important to highlight that the concern with interoperability is also observed in the Enterprise Application Integration (EAI) field, which aims to facilitate the automated exchange of information between enterprise applications, mainly regarding the semantic model, which acts as meta-knowledge to improve interoperability between these applications (GORKHALI; XU, 2016). In fact, a recent tertiary study[1] Maciel *et al.* (2024) explores the topic of interoperability and highlights its various definitions, types, areas, and domains.

In this realm, Yu *et al.* (2023) propose a model for EAI based on the idea of Enterprise Semantic Model (ESM) with a focus on enterprise semantics. The model is based on some characteristics from which the three-layer architecture of the model can be highlighted (i.e., conceptual or semantic level, external or application level, internal or realization level). Such architecture can support interoperability in relation to software systems, including information and knowledge. In addition, the model was applied to a line of business steel making and manufacturing company, reaching a positive effect on interoperability. Dhoopati (2023) provides an overview of how artificial intelligence (AI) and machine learning (ML) can enhance EAI,

[1] A review whose selected publications are secondary studies (e.g., systematic literature reviews or systematic mapping studies) (KITCHENHAM *et al.*, 2015)

driving greater efficiency, better data quality to improve interoperability between systems. Some aspects related to this improvement include: (i) data mapping, which aims to identify patterns and map data between different systems; and (ii) data validation used to validate data flowing between applications, ensuring that it is accurate and consistent. However, our focus in the present work lies in the effort to establish interoperability links in SoIS through the CS API descriptions.

2.4 Final Remarks

This chapter introduced main concepts used in this thesis: SoIS, API description, and related work. Initially, we introduced the concept of SoIS, explaining its emergence and relevance in facilitating business partnerships to deliver new services and products. Next, we detailed the eight SoIS characteristics, discussed by some authors in the area over time. We also highlighted the four SoIS classifications, considering aspects such as political governance, interactions between systems and independent ownership. In addition, we addressed the interoperability links in SoIS, discussing their definitions and design guidelines, highlighting that the mapping of input and output formats and the identification of candidate IS are essential aspects to support developers in achieving these links, which is the focus of this thesis.

The second concept discussed was API description. We defined what an API is and highlighted the importance of proper documentation, highlighting that API description languages are essential to ensure accuracy and clarity in API documentation. We then cited their main characteristics, and presenting the most widely languages used in the market, with an emphasis on OpenAPI, due to its wide adoption, robustness, and ease of use. Finally, we illustrated an example of an OpenAPI in YAML, complemented by its visualization in a graphical interface, facilitating the understanding and use of API description.

The third concept corresponded to the main related work. Thus, we compared our method with nine articles, organizing them in a table with five distinct perspectives that help in understanding the current panorama and situate our method in relation to them. In addition, we highlighted the main points of each article, such as their contributions, methodologies and evaluations.

The next chapter will present the catalog proposed in this thesis, where the main syntactic and semantic textual similarity analysis methods for API descriptions will be presented.

3 CATALOG OF TEXTUAL SIMILARITY ANALYSIS METHODS

This chapter presents the catalog of textual similarity analysis methods, designed to compile the main syntactic and semantic similarity methods in the literature for API descriptions, supporting our method for identifying API connection points. Section 3.1 discusses Syntactic Similarity, detailing key methods for comparing the structural strings elements. Section 3.2 explores Semantic Similarity, focusing on methods that assess the meaning and relationship between texts. Section 3.3 introduces the catalog, integrating syntactic and semantic methods. Finally, Section 3.4 presents the final remarks.

It is worth noting that the articles used to compile the main methods in the catalog followed an approach based on the guidelines proposed by Kitchenham *et al.* (2015), incorporating a manual search complemented by the forward snowballing technique, focused on textual similarity analyses. That procedure enabled the identification of additional studies that referenced previously selected articles, expanding the scope of the research. The results of that approach are presented in Table 3, which lists the main source articles, along with their respective publication years, from which the syntactic and semantic methods presented in the catalog were extracted.

Table 3 – Source articles for extraction of syntactic and semantic similarity analysis methods

Article	**Year**
Short-Text Semantic Similarity (STSS): Techniques, Challenges and Future Perspectives	2023
Evolution of Semantic Similarity - A Survey	2021
Short text similarity measurement methods: a review	2021
Measurement of Text Similarity: A Survey	2020
A Survey on Semantic Similarity	2019
Framework for Syntactic String Similarity Measures	2019
Feature-based approaches to semantic similarity assessment of concepts using Wikipedia	2015
A Survey on Semantic Similarity Measure	2014
Description and Evaluation of Semantic similarity Measures Approaches	2013

Source: Author.

3.1 Syntactic Similarity

Syntactic or lexical analysis uses string similarity algorithms to examine the structure and sequence of characters without making any assumptions about the meaning of the language or content. That is especially important when it comes to API attributes, which often have different naming conventions, making it difficult to extract their semantic meaning. A clear

example is the attribute "*customer_id*", which can also be represented as "*ct_id*" or "*cstid*". Such variations arise from the fact that different development teams use their own naming conventions, creating similar attributes with different syntax. String similarity algorithms can be classified into several categories, including edit distance-based, token-based, and sequence-based, where it one has own particularities and applications. A detailed description of these categories is presented below.

Edit distance-based refers to a class of algorithms used to calculate the number of operations required to transform one string into another (RISTAD; YIANILOS, 1998). The greater the number of operations, the lower the similarity between the two strings. As a representative measure in this category, it is defined as the minimum number of primitive operations (i.e. insertions, deletions, and substitutions) necessary to convert one string into another (ZHANG *et al.*, 2010). Variations of edit distance have been proposed, differing based on the number, type, and cost of operations (GALI *et al.*, 2019). These include the hamming (HAMMING, 1950), jaro (JARO, 1989), and jaro-winkler (WINKLER, 1990), levenshtein distance (LEVENSHTEIN, 1966), damerau-levenshtein distance (DAMERAU, 1964), needleman-wunsch (NEEDLEMAN; WUNSCH, 1970), smith-waterman (SMITH *et al.*, 1981), and the gotoh (GOTOH, 1982).

Among the algorithms mentioned above, we will detail only those that will be used in this thesis, which are: (i) *hamming*, which calculates the distance by overlapping the strings involved (of the same size) and looking for the places where they diverge points (WAGGENER; WAGGENER, 1995), e.g. for the strings "*test*" and "*text*", the hamming is 1, and such a calculation can be seen in Equation 3.1 (STABILI *et al.*, 2017), where A and B are the strings, A_i and B_i are the characters in position i, and k is the size of string A or B, as they must be the same length; (ii) *levenshtein*, which calculates the distance using transformations from one string to another ($A \rightarrow B$), utilizing insertion, deletion, and replacement of characters (SUNILKUMAR; SHAJI, 2019), and such a calculation can be seen in Equation 3.2 (MULIADI *et al.*, 2023), where $|X|$ is the length of a string X, *tail(X)* is a string of all but the first character in X; and (iii) *jaro-winkler*, which uses a prefix scale that gives higher scores to strings that contain the same characters within a given distance and if they have the same character order, so the higher the character match from the start, the higher is the index(ADDURU *et al.*, 2018), and such a calculation can be seen in Equation 3.3 (MULIADI *et al.*, 2023), where l is the length of the common prefix at the beginning of the string up to 4 characters, p is a constant scale factor for adjustments to common prefixes, m is the matching number and t the transposition number of

the characters and $|X|$ is the length of a string X.

$$hamming = \sum_{i=1}^{k} |A_i - B_i| \qquad ,where$$
$$A_i = B_i \Rightarrow hamming_i = 0 \qquad A_i \neq B_i \Rightarrow hamming_i = 1 \tag{3.1}$$

$$lev(A,B) = \begin{cases} |A| & if|B| = 0, \\ |B| & if|A| = 0, \\ lev(tail(a), tail(b))) & if A[0] = B[0], \\ 1 + min \begin{cases} lev(tail(A), B)) \\ lev(A, tail(B)) \\ lev(tail(A), tail(B)) \end{cases} & otherwise, \end{cases} \tag{3.2}$$

$$jaro\_winkler = jaro + lp(1 - jaro) \qquad ,where$$
$$jaro = \begin{cases} 0 & if(m = 0), \\ \frac{1}{3}\left(\frac{m}{|A|} + \frac{m}{|B|} + \frac{m-t}{m}\right) & otherwise, \end{cases} \tag{3.3}$$

In token-based algorithms, inputs are represented as sets of tokens rather than complete strings (GALI *et al.*, 2019). The goal is to identify matching tokens between two sets, with similarity increasing proportionally to the number of matching tokens. A string can be converted into a set of tokens by splitting it using a delimiter. It is worth noting that in this class of algorithms, tokens of different lengths have equal importance. Among the algorithms in this class, the following can be mentioned: jaccard (SUNILKUMAR; SHAJI, 2019), sorensen (SORENSEN, 1948), dice (ADAMSON; BOREHAM, 1974), monge-elkan (MONGE; ELKAN, 1996), tanimoto (ROGERS; TANIMOTO, 1960).

Among the algorithms mentioned above and used in this thesis, the following stand out and will be discussed in more detail: (i) *jaccard*, which is a statistical method used to find the similarity and diversity between two defined sets (SUNILKUMAR; SHAJI, 2019), and such a calculation can be seen in Equation 3.4 (DHARAVATH; SINGH, 2016), which uses the intersection of the number of tokens A and B ($|A \cap B|$) divided by the union of the number of tokens ($|A| + |B| - |A \cap B|$); (ii) *sorensen* is another method that quantifies the similarity between two sets of tokens (SORENSEN, 1948), its formula can be seen in Equation 3.5 (SORENSEN,

1948), which uses 2 times the intersection number of tokens A and B divided by the sum of length of each token $|X|$, where X can be A or B.

$$jaccard = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{3.4}$$

$$soresen = \frac{2|A \cap B|}{|A| + |B|} \tag{3.5}$$

Sequence-based is a class of algorithms aimed at identifying similarity between common substrings in strings. These algorithms, in more detail, search for the longest sequences of substrings shared between both strings, where a greater number of theses sequences results in a higher similarity score (MAYANK, 2019). It is important to highlight that substrings of the same length have the same weight in the comparison (MAYANK, 2019). Among the main algorithms in this class include the longest common substring (LCS) (FRIEDMAN; SIDELI, 1992), and the ratcliff-obershelp (KALBALIYEV; RUSTAMOV, 2020). The latter algorithm is used in this thesis, and its formula is shown in Equation 3.6 (KALBALIYEV; RUSTAMOV, 2020). The equation calculates the similarity between two strings multiplying by 2 the number of common characters (K_m), and dividing the result by the sum of the lengths of both strings.

$$Ratcliff = \frac{2K_m}{|A| + |B|} \tag{3.6}$$

3.2 Semantic Similarity

Semantic analysis is a linguistic technique in natural language processing (NLP) that uses methods and techniques to measure the degree of semantic equivalence between two items, which can be concepts, sentences or documents (P.; SHAJI, 2019). That analysis can be applied in several areas, such as document classification, information retrieval, question answering, sentiment analysis, plagiarism detection and comparison, machine translation, and short textual similarity (ELAVARASI *et al.*, 2014; P.; SHAJI, 2019; AMUR *et al.*, 2023). An interesting use case of the latter is the comparison of API attribute descriptions, where the similarity between

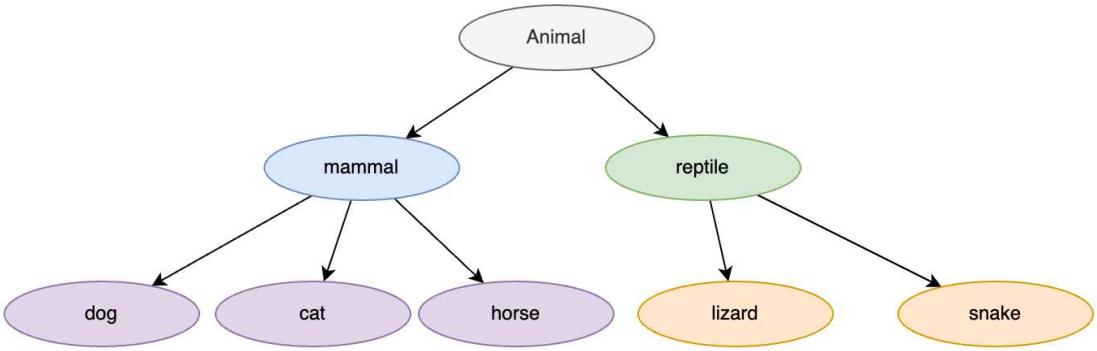
texts can be evaluated to identify semantic relationships (BORGES *et al.*, 2024). For example, the attribute description *client_id*, which means "a unique identifier of a customer in the system", and *customer_key*, which means "a unique key that identifies a customer", can be considered semantically similar, as they both represent the same concept in terms of customer identification. According to Chandrasekaran e Mago (2021) the main similarity approaches can be categorized into three types of methods: knowledge-based, corpus-based, and deep neural network-based.

3.2.1 Knowledge-based methods

Knowledge-based methods are approaches that calculate semantic similarity between terms using information from one or more structured knowledge sources, such as ontologies, lexical databases, thesauri, and dictionaries (CHANDRASEKARAN; MAGO, 2021). These sources provide an organized representation of terms or concepts, connected by semantic relationships, allowing these methods to more accurately capture the real meaning of terms and reduce ambiguities. Knowledge-based semantic similarity methods are classified into three types: edge counting-based, feature-based, and information content-based (CHANDRASEKARAN; MAGO, 2021).

The edge counting-based method consists of representing concepts in a hierarchically structured knowledge graph (CHANDRASEKARAN; MAGO, 2021). The similarity between these concepts is determined by counting the number of edges that separate them: the greater this distance, the lower the similarity between the terms (SLIMANI, 2013). An example of a knowledge graph is illustrated in Figure 16, which presents the relationship between some mammal and reptile animals. According to the *path* similarity proposed by Rada *et al.* (1989), the similarity between two terms can be calculated from the shortest path that connects them. Equation 3.7 demonstrates this relationship, where the similarity between two concepts c_1 and c_2 is inversely proportional to the shortest path length between them, *min_len*(c_1, c_2). In addition to *path* similarity, other methods in this class include those proposed by *wup* (WU; PALMER, 1994), *lch* (LEACOCK, 1994), *hso* (HIRST *et al.*, 1998), and *li* (LI *et al.*, 2003). More details about these methods can be found in (BORGES, 2023a).

Figure 16 – Knowledge-based representation example



Source: Adapted from (P.; SHAJI, 2019).

$$\text{Sim}_{\text{path}}(c_1, c_2) = \frac{1}{1 + \text{min\_len}(c_1, c_2)} \tag{3.7}$$

Feature-based methods attempt to address the limitations of path-based measures, since links in an ontology do not always represent uniform distances (SÁNCHEZ *et al.*, 2012). Such methods aim to calculate the similarity between words considering their properties, close concepts, and glosses (SÁNCHEZ *et al.*, 2012; SLIMANI, 2013). Common properties and concepts tend to increase similarity, while non-common ones reduce it (JIANG *et al.*, 2015). The gloss represents the meaning of a word in the dictionary, and a collection of glosses composes a glossary (CHANDRASEKARAN; MAGO, 2021). Figure 17 shows an example of a feature-based method, which includes the elements: computer, tablet, and mouse. By comparing the features of these elements through a feature-matching function, it is possible to observe that the "*computer/tablet*" pair will be considered more similar than the "*computer/mouse*" pair (HARISPE *et al.*, 2013). That occurs because the first pair shares more features in common (two attributes) compared to the latter, which shares only one. Among the main methods that can be cited are: *lesk* similarity (LESK, 1986), *tversky* (TVERSKY, 1977), *re* (RODRIGUEZ; EGENHOFER, 2003), *x-similarity* (PETRAKIS *et al.*, 2006). More details about these methods can be found in (BORGES, 2023a).

Figure 17 – Feature-based representation example

```
Computer = { Computer , Object, Thing}
Tablet = { Tablet, Object, Thing}
Mouse = { Mouse, Mammal, Animal, Thing}
```

Source: Harispe *et al.* (2013)

Finally, the information content (IC) methods are defined as the amount of information they convey in a given context (SÁNCHEZ; BATET, 2013). A high IC value indicates that the word is more specific and less ambiguous, while low IC values suggest that the word is more abstract in its meaning (ZHU; IGLESIAS, 2016). The word specificity is calculated using inverse document frequency (IDF), based on the Formula 3.8, where *N* is the total number of documents in a given corpus; and *df(t)* is the number of documents that contain the given term *t* (HAVRLANT; KREINOVICH, 2017). More specific words tend to occur less frequently in a document. For example, in a corpus with 1000 documents (*N* = 1000), if the term "*computer*" appears in 10 documents ($df(t) = 10$), the IDF value would be: $IDF(computer) = log(1000/10) = log(100) = 2$. On the other hand, if the term "*mouse*" appears in 500 documents ($df(t) = 500$), the IDF would be: $IDF(mouse) = log(1000/500) = log(2) \approx 0.301$. In that case, the term "*computer*" is more specific since it has a higher IDF value, while the term "*mouse*" is more common and less informative. IC methods measure the similarity between terms using the IC value assigned to each one or by exploiting the idea of a "common subsumer", which refers to concepts that share the same information. Among the main methods that can be cited are: *res* (RESNIK, 1995), *jcn* (JIANG; CONRATH, 1997), *lin* (LIN, 1998). More details about these methods can also be found in (BORGES, 2023a).

$$\mathrm{IDF}(t) = \ln\left(\frac{N}{\mathrm{DF}(t)}\right) \tag{3.8}$$

3.2.2 Corpus-based methods

Corpus-based methods assess the similarity between terms from information extracted from large corpora, which are sets of linguistic data, whether from various fields or specific areas, for example, information stored in a computer with a specific purpose of analysis (THANAKI, 2017; CHANDRASEKARAN; MAGO, 2021). An example of a corpus can be seen in Table 4, which contains the following fields: *textID*, representing the code of the text as a whole (for example, a paragraph), *ID*, an offset value for a word, and *wordID*, an integer that corresponds to the word (CORPORA, 2024). From these corpus, the methods use the so-called "distributional hypothesis", which notes that "words with similar meanings tend to occur frequently in similar contexts" (GORMAN; CURRAN, 2006). However, these methods do not consider the intrinsic meaning of words, focusing only on their co-occurrences (CHANDRASEKARAN; MAGO, 2021).

Table 4 – Example of linguistic corpus

textID	ID	wordID
2040250	110933753	848
2040250	110933754	3
2040250	110933755	560
2040250	110933756	459620
2040250	110933757	6891
2040250	110933758	10
2040251	110933779	7140251
2040251	110933780	11850
2040251	110933781	187678
2040251	110933782	26
2040251	110933783	19957
2040251	110933784	19
2040251	110933785	41
2040251	110933786	64
2040251	110933787	160
2040251	110933788	4
2040251	110933789	4155

Source: Corpora (2024)

According to Chandrasekaran e Mago (2021), the main corpus-based methods are:

– Latent Semantic Analysis (LSA) is a widely used technique for measuring semantic similarity in text. It constructs a word co-occurrence matrix where rows represent words, columns represent paragraphs, and cells contain word counts. Dimensionality reduction is achieved using singular value decomposition (SVD), which breaks the matrix into three matrices, preserving the structure of word similarities. Each word is then represented as a vector, and semantic similarity is calculated using the cosine similarity between angles of vectors (WANG; DONG, 2020).
– Hyperspace Analogue to Language (HAL) creates a word co-occurrence matrix where both rows and columns represent words, and the matrix elements are populated with association strength values. These values are calculated by sliding a variable-sized “window” over the corpus, with the association strength decreasing as the distance between words increases. Word vectors are formed by combining their row and column data. Dimensionality reduction is achieved by removing columns. Semantic similarity is calculated using Euclidean distance (straight line distance between two points) or Manhattan distance (sum of the absolute differences the word vectors) (WANG; DONG, 2020).
– Explicit Semantic Analysis (ESA) measures semantic similarity using Wikipedia concepts, making it adaptable across domains and languages. Each Wikipedia concept is represented as a vector of words, and an inverted index links words to associated concepts. The association strength is weighted using term frequency–inverse document frequency (TF-

IDF)[1], and weak associations are removed. The input text is represented as "interpretation vectors" of weighted concepts, and semantic similarity is calculated by determining the cosine similarity between these vectors.

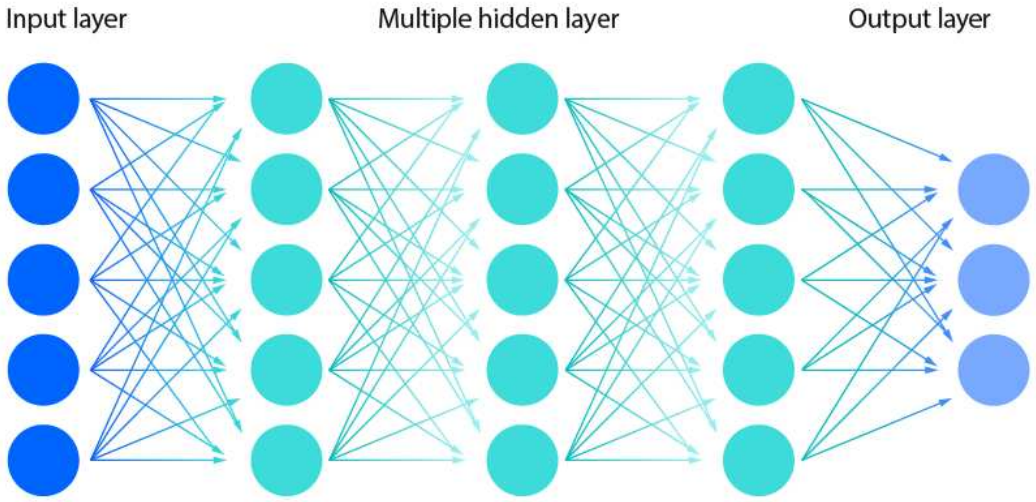
– Normalized Google Distance (NGD) measures the similarity between two terms based on the results returned when querying them in the Google search engine. It is based on the premise that two terms tend to appear together more frequently on web pages when they are more closely related. NGD is widely used to assess the semantic relatedness between terms, rather than semantic similarity, because related terms can often occur together even if they have opposite meanings.
– Dependency-based models analyze the meaning of a word or phrase from its neighbors in a context window. They use inductive dependency parsing to build a "syntactic context model" that considers the nodes that precede and follow the word in the parse tree. The vector representation of the word is created by summing the surrounding windows of words and considering their frequency in the corpus. The semantic similarity between words is then calculated using based on the generated vectors.
– Kernel-based methods have been widely used to identify patterns in textual data, allowing the detection of similarity between different text fragments. Three main types of kernels stand out: the string kernel (LODHI *et al.*, 2002), which calculates the similarity between documents based on the matching of non-consecutive subsequences of characters; the sequence kernel (CANCEDDA *et al.*, 2003), where documents are treated as sequences of words; and the tree kernel (MOSCHITTI *et al.*, 2008), which applies various kernel functions to model properties of parse trees on kernel-based machines.
– Word attention models capture the relevance of words in the underlying corpora before calculating semantic similarity. This is because human interpretation of similarity often relies on keywords within a given context. Techniques such as word frequency, alignment, and term association are used to determine attention weights, highlighting the most important words in the analyzed text.
– Word-Alignment models calculate the semantic similarity of sentences based on their alignment in a large corpus. The system calculates the semantic similarity between two sentences as a ratio of the aligned context words in the sentences to the total words in both sentences.

[1] A statistical measure indicating a word's significance in a document collection (ZHANG *et al.*, 2008).

3.2.3 Deep Neural Network-based methods

Deep Neural Network-based methods are a subset of machine learning that uses multi-layered neural networks, called Deep Neural Networks, to simulate complex decision-making power (SHINDE; SHAH, 2018). An example of deep neural network can be seen in Figure 18, where you can see the multiple hidden layers between the input and output layers (IBM, 2024). Unlike traditional approaches, which relied on manual rules or simple statistical models, deep neural networks allow models to learn from large volumes of data. Such methods have increasingly attracted researchers in various areas, including medicine, engineering, robotics, finance, and NLP (RAZAVI, 2021). Regarding the latter, the semantic representation of text can be highlighted, due to the ability of models to recognize superficial patterns, and also to capture complex relationships between words and sentences (MANSOOR *et al.*, 2020). When well trained, deep neural networks can map words and phrases to a high-dimensional vector space, where semantic relations are represented by the proximity and orientation of these vectors (SANBORN; SKRYZALIN, 2015). According to Amur *et al.* (2023), the main deep learning techniques applied to the semantic similarity analysis of short text (as in this thesis, which involves the analysis of API descriptions) include Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), Bidirectional Long Short-Term Memory (Bi-LSTM), and Transformers, the latter being the state of the art and will be used in this thesis.

Figure 18 – Deep Neural Network example



Source: IBM (2024).

Convolutional Neural Network are a type of artificial neural network capable of automatically learning to filter, cluster, and combine features of data, making them one of the

most representative architectures in the field of deep learning (LI *et al.*, 2021). Although they are most commonly applied in classification and computer vision tasks, CNN have also demonstrated excellent performance in several NLP tasks, such as sentence modeling, information retrieval in search queries, and syntactic analysis (ZHENG *et al.*, 2019). With their multi-layer convolutional mechanism, these networks are capable of capturing complex semantic patterns at different levels of abstraction, accurately representing the hierarchical structure of sentences (ZHENG *et al.*, 2019). In addition, they excel at identifying semantic similarities between texts, making them particularly effective in text matching tasks.

According to Chandrasekaran e Mago (2021), Long Short-Term Memory (LSTM) networks are an advanced type of Recurrent Neural Network (RNN)[2], designed to solve the problem of long-term dependence of RNN, which consists of the difficulty of using past information efficiently in current contexts. When processing text, LSTM use gate mechanisms that allow the network to decide which information should be retained or discarded, ensuring the preservation of the context necessary to predict the next word or phrase. For example, in the passage "Mary is from Finland. She is fluent in Finnish. She loves to travel", when processing the second sentence, it is important to remember the words "Mary" and "Finland". However, when reaching the third sentence, the network can ignore "Finland" without compromising comprehension. The LSTM architecture makes this selectivity possible.

Bidirectional Long Short-Term Memory combines the power of LSTM with bidirectional processing, allowing the model to simultaneously capture the previous and subsequent context of an input sequence, which processes sequential data in both directions: forward and backward (CHANDRASEKARAN; MAGO, 2021). That approach makes the model more effective in capturing long-term dependencies throughout the sequence. In the context of semantic similarity, the paper of He e Lin (2016) stands out, which uses Bidirectional Long Short-Term Memory as a model for interactions between pairs of words, with a similarity-focused layer, capable of more efficiently capturing the semantic information of texts.

The Transformer is a widely recognized deep neural model adopted in several fields, such as computer vision, speech processing, and especially in NLP (LIN *et al.*, 2022). Among its main applications in NLP are tasks such as text classification, machine translation, and question answering (KHAN *et al.*, 2022). Originally proposed by (VASWANI *et al.*, 2017), the

[2] A deep neural network trained on sequential or time-series data to create a neural network capable of performing specific or sequential instructions based on information retained in a "memory" of previous inputs (DAS *et al.*, 2023).

Transformer uses attention mechanisms to weight the influence of different parts of the input data, allowing to capture the semantic properties of words. Chandrasekaran e Mago (2021) report that the Transformer architecture is still composed of two main parts: encoder and decoder. The encoder contains several attention layers, followed by a fully connected feed-forward neural network, which is a multilayer structure where information flows in one direction, from input to output. The decoder, although similar to the encoder, has an additional attention layer that adjusts the attention weights based on the encoder outputs. According to Lin *et al.* (2022) the Transformer architecture can be applied in three different ways. The first is in encoder-decoder mode, used in sequence-to-sequence tasks, such as machine translation, where the complete architecture is implemented. The second approach uses only the encoder, with its outputs representing the input sequence, being common in natural language understanding (NLU) tasks, such as text classification and sequence labeling. Finally, there is the option of using only the decoder, eliminating the cross-attention module, which is typical in sequence generation tasks, such as language modeling. Among the main models based on Transformers, the following can be highlighted:

- Encoder-Decoder:
  - text-to-text transfer transformer (T5) is a transformer-based architecture that utilizes transfer learning, where the model is initially pre-trained on a data-rich task and then fine-tuned for a specific task (RAFFEL *et al.*, 2020). It adopts a unified framework that transforms all text-based language problems into a text input and output format, which allows the same model, functions, and parameters to be applied to a wide variety of tasks such as translation, classification, question generation (RAFFEL *et al.*, 2020; GROVER *et al.*, 2021).
  - BART is a denoising autoencoder for pretraining sequence-to-sequence models (LEWIS *et al.*, 2019). It uses an encoder-decoder architecture and is trained on documents corrupted by a noise function, learning to restore the original text from this corrupted version (LEWIS *et al.*, 2019). BART's encoder is bidirectional, similar to BERT, capturing contextual information in both directions to process the corrupted text, while its decoder is autoregressive, like GPT, generating text word by word from left to right based on previous predictions (PEARCE *et al.*, 2021). BART can be used to solve several tasks including machine translation, abstractive question

answering, and text summarization (QUATRA; CAGLIERO, 2023).

- Encoder-only:
  - Bidirectional Encoder Representations from Transformers (BERT): originally proposed by (DEVLIN *et al.*, 2019), it uses transformer encoders as a basis for pre-training models on various NLP tasks. Devlin *et al.* (2019) reports that BERT training process is divided into two main steps: pre-training and fine-tuning. In pre-training, the model is trained with unlabeled data on different tasks. In fine-tuning, BERT is initialized with the parameters acquired in pre-training and fine-tuned using labeled data for specific tasks. According to Khan *et al.* (2022) during pre-training, BERT uses two fundamental techniques. The first is masked language modeling (MLM), in which some words in a sentence are masked and the model must predict what these words are. This method allows the model to learn to capture the bidirectional context of the words. The second technique is next sentence prediction (NSP), which evaluates whether a pair of sentences in the original text appear in the correct sequence. These mechanisms allow BERT to develop a deeper understanding of both words and their meanings in the context in which they are inserted. Other variations of BERT models include DistilBERT (SANH *et al.*, 2019), SciBERT (BELTAGY *et al.*, 2019), TinyBERT (JIAO *et al.*, 2020), ALBERT (LAN *et al.*, 2020), DeBERTa (HE *et al.*, 2021), and RoBERTa (LIU *et al.*, 2020), which is detailed below.
  - Robustly Optimized BERT pre-training Approach (RoBERTa): as said in preview item, it is a variant of BERT that aims to optimize its performance by adjusting several methodological parameters of the original version. These adjustments allowed exploring the importance of hyperparameters, such as the impact of larger pre-training datasets (LIU *et al.*, 2020). Unlike BERT, RoBERTa was trained with dynamic masking instead of static masking, eliminated the NSP step, thus using full sentences, and employed large mini-batches and a more extensive byte-pair encoding (ACHEAMPONG *et al.*, 2021). RoBERTa outperforms BERT in various NLP tasks due to its more extensive training, but this comes at the cost of being more computationally expensive, requiring more time to execute (ACHEAMPONG *et al.*, 2021).
- Decoder-only:
  - XLNet: is an autoregressive pre-training method that uses permutation concepts to

enable bidirectional context learning (YANG *et al.*, 2019). According to Acheampong *et al.* (2021), although it is a variant of BERT, the main difference between them lies in the training objectives. While BERT uses token masking to predict masked words based on a bidirectional context, XLNet adopts a permutation approach, where the model is trained on all possible permutations of the words in a sentence. This allows XLNet to capture bidirectional dependencies more effectively while maintaining the advantages of autoregressive modeling. The authors of XLNET (YANG *et al.*, 2019), claim that it outperforms BERT on 20 tasks, often by a large margin, including question answering, natural language inference, sentiment analysis, and document classification.

– Generative Pre-Trained Transformer (GPT) are language models based on the transformer architecture, specifically the decoder part, and are pre-trained on large volumes of textual data (YENDURI *et al.*, 2024). They can perform a wide range of tasks, such as generating human-like text (ZHU; LUO, 2022). One of the factors that has driven the growth of GPT is the use of unlabeled data, which allows the model to produce appropriate responses based on the inputs, without the need for manual labeling of the data used for its training (YENDURI *et al.*, 2024). The evolution of GPT models reflects significant advances in NLP. OpenAI released GPT-1 in 2018, with 117 million parameters, marking the first model capable of interpreting text and responding efficiently (YENDURI *et al.*, 2024). In 2019, GPT-2 brought a leap to 1.5 billion parameters, using a dataset 10 times larger (RADFORD *et al.*, 2019). Then, in 2020, GPT-3, with 175 billion parameters and trained on a vast corpus of 500 billion words, substantially increased the model's capacity (YE *et al.*, 2023). In 2022, GPT-3.5 was released and maintained a similar or higher number of parameters than GPT-3, but with more accurate answers and better contextualization (AU2; KATZ, 2022). Finally, on March 14, 2023, GPT-4 was released, although the exact number of parameters has not been officially released, it is estimated that it could reach 1.8 trillion parameters (BUCHER; MARTINI, 2024). This latest version brings significant improvements through reinforcement learning with human and AI feedback (BUCHER; MARTINI, 2024).

3.3 Catalog

The main methods for syntactic and semantic similarity analysis discussed previously have been consolidated and organized into a catalog, as presented in Table 5. For syntactic similarity, the catalog includes: edit distance-based, token-based, and sequence-based. For semantic similarity, it includes the following: knowledge-based, corpus-based, and deep neural network-based. That organized structure provides a comprehensive and systematic overview of the available methods, facilitating both the understanding and application of them in textual analysis of different levels of complexity, from the simplest to the most advanced.

3.4 Final Remarks

This chapter presented a comprehensive catalog of textual similarity analysis methods, with the purpose of identifying and organizing the main syntactic and semantic analysis techniques available in the literature for use in API descriptions.

First, we discussed syntactic similarity methods, which include algorithms such as edit distance-based, token-based, and sequence-based. These techniques are widely applied to detect structural similarities between strings, serving as the basis for the evaluation of API attributes.

Next, we explored semantic similarity methods, which bring a richer and more contextual approach to the relationships between terms and concepts. These methods were organized into three main categories: (i) knowledge-based methods, which use structured sources such as ontologies and taxonomies to calculate similarities and encompass techniques such as edge counting, feature analysis, and IC measures; (ii) corpus-based methods, with approaches such as LSA, HAL, ESA, NGD, dependency methods, kernels and word attention techniques; and (iii) deep neural network-based methods, including models such as CNN, LSTM, Bi-LSTM and, most importantly, Transformers. Transformer models represent the state of the art in textual analysis, offering an enhanced ability to capture semantic nuances in complex contexts.

Finally, we presented a catalog of textual similarity methods providing a consolidated view of the available techniques, serving as a solid basis for establishing interoperability links in SoIS through the textual analysis of API descriptions.

Table 5 – Catalog of the main methods of textual similarity analysis

Syntactic Similarity	Edit distance-based	hamming jaro jaro-winker levenshtein Damerau-Levenshtein Needleman-Wunsch Smith-Waterman Gotoh	
	Token-based	jaccard sorensen dice monge-elkan tanimoto	
	Sequence-based	lcs ratcliff-obershelp	
Semantic Similarity	Knowledge-based	Edge counting-based	path wup lch hso
		Feature-based	lesk tversky re x-similarity
		Information content-based	res jcn lin
	Corpus-based	LSA HAL ESA NGD Dependency-based Kernel-based word-attention	
	Deep Neural network-based	CNN LSTM Bi-LSTM	
		Transformer-based	BERT RoBERTa DistilBERT SciBERT ALBERT TinyBERT ALBERT DeBERTa XLNet GPT T5 BART

Source: Author.

4 METHOD

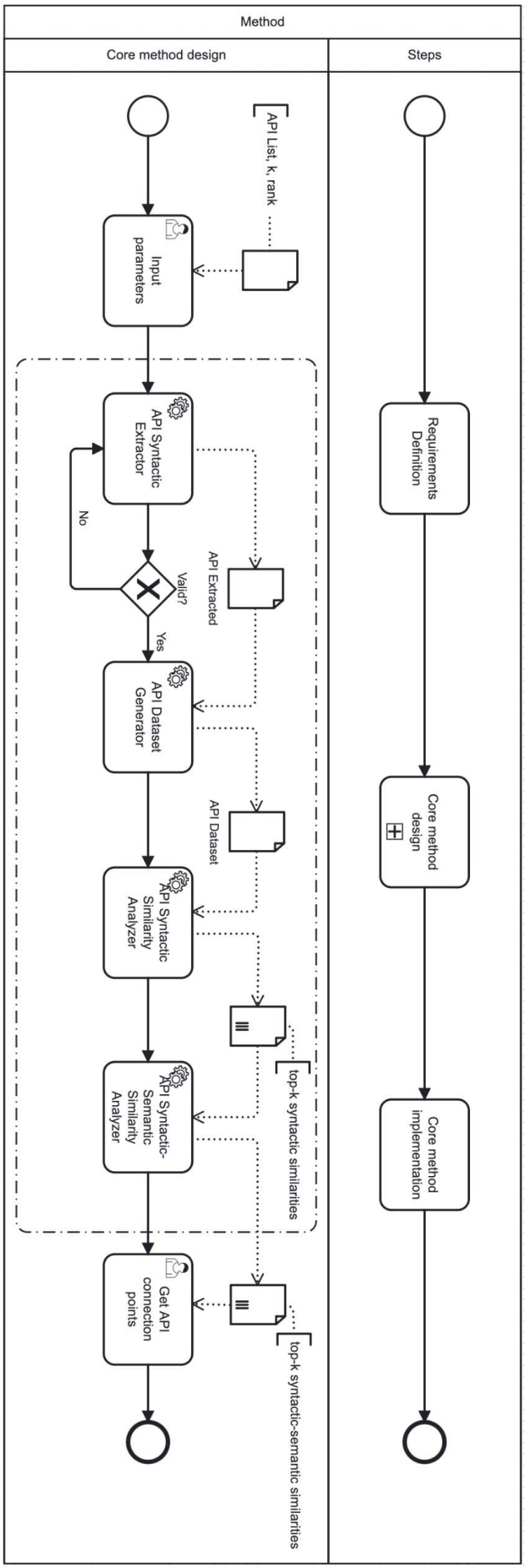
This chapter presents the artifact method developed in this thesis, based on our research method (section 1.6), which as written previously in that section, consists of a set of structured steps that can be implemented by algorithms to perform a specific task. The proposed method was created to help CS developers identify interoperability links in SoIS by identifying CS connection points, conducted through detailed analyses of their respective API (BORGES *et al.*, 2024). The method, which can be seen in Figure 19, consists of three interconnected steps: (i) requirements definition, where the motivations and justifications for the method's creation are outlined; (ii) core method design, where all the phases and components necessary to obtain the API connection points are detailed; and (iii) core method implementation, which introduces the technological details surrounding the solution (tool) developed to support the core method. These steps are explained in detail below.

4.1 Requirements definition

In this first stage, we defined the requirements to justify the proposition of our method, which were based on recent literature on SoIS and on the in-place observation of an industrial case study for establishing interoperability links. In the literature, we identified a gap in research on SoIS, particularly in the industrial sector, as studies predominantly focus on education. We also noticed a lack of methods for achieving interoperability links, especially in mapping input and output data for exchanging information or re-configuring the use of services offered, which is crucial for effective communication between the CS of SoIS while maintaining low coupling (FERNANDES *et al.*, 2022). Additionally, in (SADEGHI *et al.*, 2024), the authors suggest requirements for interoperability in large collaborative SoS. Among these, we can highlight "discoverability", which aims to establish processes for efficiently and reliably locating services and data. This is essential for service-oriented computing and data-driven systems, essential for overcoming barriers of data heterogeneity and the diversity of API and services.

Another requirement came from the direct observation of CS developers for 30 months in achieving interoperability links in SoIS. In our previous articles, (BORGES *et al.*, 2022; BORGES *et al.*, 2024), we highlighted some of them. First, developers must ensure permission credentials to access the services offered by target CS. Second, they need to understand the technology stack of each CS, which includes its execution environment, architecture,

Figure 19 – General flow and main parts of our method



Source: Author.

programming languages, and standards. Third, they must gather communication details of the target CS services, such as connection methods, authentication strategies, and data formats. This information can be obtained through direct discussions with the CS maintainers or by reviewing the service CS' API documentation. Lastly, developers need to continuously monitor API changes, as even minor modifications can disrupt interoperability and the objective of the SoIS.

4.2 Core method design

Figure 19, Core method design lane, presents the design workflow of our core method. A CS developer must submits *Input parameters*, which includes an *API list* with their respective description files, a number k representing the quantity of top results (*top-k*), and a *rank* parameter for ordering these top results. The main part of core method, comprising four interconnected activities analyzes this input. After the analysis, the core processes it and generates a *top-k* similarity matrix of API connection points based on syntactic-semantic similarities, which CS developers can get to their interoperability tasks.

Table 6 details the 33 API connection points, and their meanings. These points are segmented into eight categories: (i) General Information, highlighting API names, attributes, attributes descriptions, endpoints, and methods; (ii) Syntactic Similarities Metrics, showcasing the six string similarity algorithms from our catalog of syntactic text analysis methods, chosen for their adeptness in handling strings and, by extension, API attributes (BORGES *et al.*, 2024); (iii) Syntactic Rankings, ordering the connection points by two criteria (average and major) based on the execution of syntactic algorithms; (iv) Auxiliary Calculations, supplementing data for similarities metrics or rankings and includes the gini index, i.e., a statistical metric for value inequality in a distribution, commonly used in economics (FURMAN *et al.*, 2019); (v) Syntactic Evaluation, comparing predicted and correct values of the syntactic connection points to determine the metrics precision, recall, and accuracy; (vi) Semantic Similarities Metrics, showcasing the four semantic similarity algorithms from our catalog of semantic text analysis methods, were chosen not only for their ability to work with short texts and, by extension, attribute descriptions from API, but also based on their basis on state-of-the-art transformer architectures that utilize at least the encoder part (encoder and encoder-decoder models), which are excellent for text similarity and classification tasks; (vii) Semantic Rankings, ordering the connection points using also two criteria (average and major) based on the execution of semantic

Table 6 – API Connection Points

	#	Field	Meaning	Data Type (*Value)
General Information	1	Origin API (OA)	Origin API name	String
	2	Target API (TA)	Target API name	String
	3	OA Out Attr	Origin API response (output) attribute	String
	4	TA In Attr	Target API request (input) attribute	String
	5	OA Out Attr Descr	Origin API response (output) attribute description	String
	6	TA In Attr Descr	Target API request (input) attribute description	String
	7	OA Out Attr dt	Origin API response attribute data type	String
	8	TA In Attr dt	Target API request attribute data type Origin	String
	9	OA Out Attr parent	API response attribute parent	String
	10	TA In Attr parent	Target API request attribute parent	String
	11	OA Url	Attribute endpoint (URL) in Origin API	String
	12	OA Method	Attribute method in Origin API	String
	13	TA Url	Attribute endpoint in Target API	String
	14	TA Method	Attribute method in Target API	String
Syn. Metrics[1]	15	hamming	Hamming String distance algorithm	Float (from 0 to 1)
	16	levenshtein	Levenshtein String distance algorithm	Float (from 0 to 1)
	17	jaro_winkler	Jaro Winkler String distance algorithm	Float (from 0 to 1)
	18	jaccard	Jaccard String distance algorithm	Float (from 0 to 1)
	19	sorensen	Sorensen String distance algorithm	Float (from 0 to 1)
	20	ratcliffz_obershelp	Ratcliff Obershelp String distance algorithm	Float (from 0 to 1)
Ra[2]	21	avg_syn_rank	Average of syntactic similarity algorithms	Float (from 0 to 1)
	22	major_syn_rank	Major index among syntactic similarity algorithms	Float (from 0 to 1)
Aj[3]	23	gini	Gini index of similarities metrics	Float (from 0 to 1)
Eva[4]	24	prediction_syn	Predicted syntactic connection point	Boolean (false=0, true=1)
	25	correct_syn	Correct syntactic connection point	Boolean (false=0, true=1)
Sem. Met.[5]	26	bert	Bidirectional Encoder Representations from Transformers	Float (from 0 to 1)
	27	roberta	Robustly Optimized BERT Pre-training Approach	Float (from 0 to 1)
	28	T5	Text-to-Text Transfer Transformer	Float (from 0 to 1)
	29	BART	A denoising autoencoder for pretraining sequence-to-sequence models	Float (from 0 to 1)
Ra[6]	30	major_sem_rank	Major index among string similarity algorithms	Float (from 0 to 1)
	31	avg_sem_rank	Average of string similarity algorithms	Float (from 0 to 1)
Eva[7]	32	prediction_syn_sem	Predicted syntactic-semantic connection point	Boolean (false=0, true=1)
	33	correct_syn_sem	Correct syntactic-semantic connection point	Boolean (false=0, true=1)

Notes:
[1] Syntactic Similarity Metrics
[2] Syntactic Rankings
[3] Auxiliary Calculations
[4] Syntactic Evaluation
[5] Semantic Similarity Metrics
[6] Semantic Rankings
[7] Syntactic-Semantic Evaluation
Source: Author.

algorithms; and (viii) Evaluation, comparing predicted and correct values of the syntactic-semantic connection points to determine also the metrics precision, recall, and accuracy.

To generate these API connection points, our core method uses four interconnected activities. The first one, API Syntactic Extractor, validates the API contracts, extracting and storing API details. The second one, API Dataset Generator, leverages that data to produce a Cartesian product of all the response attributes of an API (*OA Out Attr*) with all the request attributes of each other API (*TA In Attr*) and relevant supporting information (see Table 6: General Information group) forming a dataset base for the syntactic and semantic similarity analyses. The third one, API Syntactic Similarity Analyzer, analyzes the dataset generated

by API Dataset Generator and calculate for each pair of API attributes (*OA Out Attr* $\rightarrow$ *TA In Attr*) the syntactic similarity metrics based on the 6 string similarity algorithms, as well such as ordering rankings, gini, and syntactic evaluation (see Table 6: Syntactic Similarity Metrics, Syntactic Rankings, Auxiliary Calculations group, and Syntactic Evaluation). The fourth one activity, ASSESA, complements the syntactic analysis results by calculating semantic similarity of the API attribute descriptions (*OA Out descr* $\rightarrow$ *TA In descr*), and generating the semantic rankings, and syntactic-semantic evaluation (see Table 6: Semantic Similarity Metrics, Semantic Rankings, and Syntactic-Semantic Evaluation).

In the following subsections, we will provide a detailed and in-depth exploration of how each of the four activities functions. We will also explain all the specifics involved in measuring the efficiency of our method.

4.2.1 API Syntactic Extractor

The API Syntactic Extractor (ASE) is the first activity belonging to the design of our core method (BORGES *et al.*, 2024). It is responsible for extracting the main information present in the description files based on OpenAPI Specification (OAS)[1], which was chosen due to its widespread adoption and comprehensive framework for defining and documenting REST API (SCHWICHTENBERG *et al.*, 2017). The ASE also validates the three API contracts: the first one ensures compatibility with the OAS versions 2.* and 3.*; the second addresses the file extensions (*yaml*, *yml*, *json*); and the third checks for the absence of circular structures or references that could trigger infinite loops. Following this validation, it extracts all essential information of each API (structures of paths, components, definitions, endpoints, methods, fields, input and output parameters) to a JSON structure organized by API names, requests and responses.

To do that the ASE follows the Algorithm 1, where it receives as input data an API list, where each API has a name and the directory path, and the output results is a list of information extracted from the API involved. In that algorithm, in detail, it can be seen that in line 2 an empty list (*exApiList*) is created that will receive the information extracted from the API. On line 3, all vector API must be iterated over. In lines 4 and 5, empty lists must be created to store all requests (*requests*) and all responses (*responses*) for each API. After reading the API (line 6), it is checked if it complies with the 3 API contracts (line 7). If yes, there are two more

[1] Availabe at: https://swagger.io/specification/

checks in the API. The first is to check if the API is of the OpenAPI type (line 8), and if so, there will have to be two functions for inputting information specific to OpenAPI (version 3.*), in this case "*extrOpRequests*" and "*extrOpResponses*" to feed the list of *requests* and *responses* (lines 9 and 10), respectively. The second is to check if the API is of type Swagger (2.*) (line 11). If yes, the lists of *requests* and *responses* should be fed by the functions "*extrSwRequests*" and "*extrSwResponses*" (lines 12 and 13). Finally, the list of information extracted from the API is fed, in JSON format (function *adaptToJson*), with information about the *API name, requests and responses* (line 15) and returned by the Algorithm on line 18.

Algorithm 1: API Syntactic Extractor Algorithm

```
   Data: apiList = [api] such that api has a name and a path
   Result: exApiList
 1 begin
 2     exApiList ← ∅
 3     for api ∈ apiList do
 4         requests ← ∅
 5         responses ← ∅
 6         Read api
 7         if api is validInContract then
 8             if api is OpenAPI then
 9                 requests.append(extrOpRequests(api))
10                 responses.append(extrOpResponses(api))
11             else if api is Swagger then
12                 requests.append(extrSwRequests(api))
13                 responses.append(extrSwResponses(api))
14             end
15             exApiList.append(adaptToJson(api[name], requests, responses))
16         end
17     end
18     return exApiList
19 end
```

Source: Borges *et al.* (2024).

4.2.2 API Dataset Generator

The API Dataset Generator (ADG) is the second activity belonging to our core method design (BORGES *et al.*, 2024). It is responsible for generating the dataset with all 14 general information data (Table 6) for subsequent similarity analyses. These information are constructed through a cartesian product of all response attributes (*OA Out Attr*) present in each endpoint of an API with all other request attributes (*Ta In Attr*) present in each other API. To generate such a dataset, the ADG must follow the Algorithm 2, which can be seen that it receives as input data a list of information extracted from the API (generated by the ASE) and the output result is a dataset with the cartesian product of the 14 general information data (see Table 6).

In Algorithm 2, in details, it can be seen that in line 2 an empty list is created (*dtProdCartesian*) that will receive the dataset of the cartesian product. In line 3, the list of information extracted from the API is iterated. In line 4, an empty auxiliary variable (*auxExpApi*) is created to help in the process of cleaning the list of extracted information. More precisely, this variable is assigned both the removal of duplicate items (line 5) and the removal of null endpoints (line 6). After that, the response information of each clean auxiliary API is iterated (line 7), and from it the response attributes are obtained with their respective parents (line 8). Subsequently, the request information is also iterated from each clean API (line 9), and within the loop the request attributes with their respective parents are obtained (line 10) and the tuple with the information extractions of the 14 general information data are obtained based on response attributes, request attributes, as well as general response and request information from the API involved (line 11). Finally, the tuple is added to the dataset (line 12), which after all iterations is returned by the algorithm (line 16).

Algorithm 2: API Dataset Generator Algorithm

```
   Data: exApiList
   Result: dtProdCartesian
 1 begin
 2     dtProdCartesian ⟵ ∅
 3     for exApi ∈ exApiList do
 4         auxExApi ← ∅
 5         auxExApi ← remDuplicatedEndpoints(exApi)
 6         auxExApi ← remNullEndpoints(auxExApi)
 7         for resOriginApi ∈ auxExApi[responses] do
 8             resAttr ← getAttrWithParents(resOriginApi)
 9             for reqTargetApi ∈ auxExApi[requests] do
10                 reqAttr ← getAttrWithParents(reqTargetApi)
11                 tuple ← extractInf(resAttr, reqAttr,
                     resOriginApi, reqTargetApi)
12                 dtProdCartesian.append(tuple)
13             end
14         end
15     end
16     return dtProdCartesian
17 end
```

Source: Borges *et al.* (2024).

4.2.3 API Syntactic Similarity Analyzer

The API Syntactic Similarity Analyzer (ASSA) is the third activity in the our core method design (BORGES *et al.*, 2024). It is responsible for analyzing the dataset generated by ADG, generating syntactic similarity metrics, syntactic rankings, auxiliary calculations, and

syntactic evaluation to obtain as result the syntactic connection points. The syntactic similarity metrics are based on the implementations, normalized from 0 to 1, of the string similarity algorithms: *Hamming, Levensthein, Jaro_Winkler, Jaccard, Sorensen, and Ratcliff-Obershelp*. The syntactic rankings are 2: (i) *avg_syn_rank*, which is generated based on the average of the metrics of the calculated string similarity algorithms; (ii) *major_syn_rank*, which is constituted by the highest index of the similarity algorithm based on a given tuple of the dataset. The auxiliary calculation is the *gini* index, which in this case aims to measure the degree of equality behavior of the majority of the population of the similarity algorithms. Syntactic evaluation include *prediction* (0 or 1) and *correct* (0 or 1) for each attribute pair (*Oa Out Attr* and *Ta In Attr*) in the tuple. In *prediction*, a pair is recommended (value 1) if its respective rank, *avg_syn_rank* (Eq 4.1) or *major_syn_rank* (Eq 4.2), exceeds a threshold defined manually by us; otherwise, that pair is not recommended (value 0). The *correct* values are the actual values (1 or 0), previously known or indicated by experts at the time of displaying the results for each tuple. From that, we can obtain: true positives (TP) when the *prediction* and *correct* ones are 1; false positives (FP) if the *prediction* is 1, but the *correct* one is 0; true negatives (TN) with both *prediction* and *correct* ones are 0; and false negatives (FN) when the *prediction* is 0, but the *correct* one is 1. These categories are essential for calculating *precision* (Eq. 4.3), *recall* (Eq. 4.4), and *accuracy* (Eq. 4.5). Visual representations (e.g., correlation charts) can also be produced.

$$prediction\_syn\_avg = \begin{cases} 1 & if\ avg\_syn\_rank > threshold \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

$$prediction\_syn\_major = \begin{cases} 1 & if\ major\_syn\_rank > threshold \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

$$precision = \frac{TP}{TP+FP} \tag{4.3}$$

$$recall = \frac{TP}{TP+FN} \tag{4.4}$$

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{4.5}$$

To generate similarity metrics, associated rankings, auxiliary calculation, and syntactic evaluation, the ASSA follows the Algorithm 3. In that algorithm, it can be seen that it receives 3 parameters as input data: *dtProdCartesian*, which is the dataset generated by the ADG; *k*, which is a number of tuples; *rank*, name of the sort syntactic rank. The output results are the top *k* tuples of syntactic connection points ordered by the *rank* parameter chosen in the input, and in descending order. In more detail, line 2 shows the creation of an empty list (*top_k_syntactic_connection_points*) that will receive the syntactic connection points. In line 3, every tuple in the cartesian product is iterated. After that, in line 4, all 6 string similarity metrics are calculated. From these metrics, the average and major syntactic rankings (line 5), the auxiliary information *gini* (line 6), prediction result *prediction* (line 7) can be calculated. The result of *correct* is fed manually with previous data or during the exposition of the results, being 0 for false and 1 for true. Soon after, the list with the *top-k* syntactic connection points is fed with each tuple added of their respective metrics of similarities, rankings, *gini*, *prediction*, and *correct* (line 9). Finally, that list is ordered by *k* and *rank*, and returned by the algorithm (line 11).

Algorithm 3: API Syntactic Similarity Algorithm

```
   Data: dtProdCartesian, k, rank
   Result: top_k_syntactic_connection_points
 1 begin
 2     top_k_syntactic_connection_points ← ∅
 3     for tuple ∈ dtProdCartesian do
 4         metrics ← calcMetrics(tuple) /* ham, lev, jar, jac, sor, rat          */
 5         ranks ← calcRanks(metrics) /* avg_syn_rank, major_syn_rank            */
 6         gini ← calcGini(metrics)
 7         prediction ← calcPred(ranks)
 8         correct ← manualData
 9         top_k_syntactic_connection_points.append(asOne(tuple,metrics,ranks,gini,prediction,
             correct))
10     end
11     return sort(top_k_syntactic_connection_points, rank, k)
12 end
```

Source: Borges *et al.* (2024).

4.2.4 API Syntactic-Semantic Similarity Analyzer

The API Syntactic-Semantic Similarity Analyzer (ASSESA) is the fourth and final activity activity in our core method design. The API Syntactic-Semantic Similarity Analyzer (ASSESA) is the responsible for analyzing the *top-k* syntactic connection points generated by ASSA, and generating semantic similarity metrics for API attribute descriptions, semantic

rankings, and syntactic-semantic evaluations to obtain as result the syntactic-semantic connection points. The semantic similarity metrics are based on the implementations, also normalized from 0 to 1, of the short text similarity algorithms applied on API attributes descriptions, which are: bert, roberta, t5, and bart. The semantic rankings are also 2: (i) *avg_sem_rank*, which is generated based on the average of the metrics of the calculated short text similarity algorithms; (ii) *major_syn_rank*, which is constituted by the highest index of the similarity algorithm based on a given tuple. Syntactic-semantic evaluation involves the fields *prediction* (0 or 1) and *correct* (0 or 1), considering both the syntactic analysis of the attributes (*Oa Out Attr* and *Ta In Attr*) and semantic analysis of the descriptions of the API attributes (*Oa Out Attr Descr* and *Ta In Attr Descr*) in the corresponding tuple, with the aim of improving the classification of API connection points. In the *prediction* field, a connection point is recommended (value 1) if the weighted average of the syntactic rank of the API attributes (*avg_syn_rank* or *major_syn_rank*) and the semantic ranks of the API attributes descriptions (*avg_sem_rank* or *major_sem_rank*) exceeds a manually defined limit; otherwise, the pair is not recommended (value 0). The weights w_1 and w_2 represent the relative importance of syntactic and semantic rankings, respectively. When $w_1 > w_2$, syntactic analysis has greater relevance; otherwise, semantic analysis is prioritized. The corresponding formulas for these predictions can be seen in equations 4.6 and 4.7, respectively. The *correct* values are the actual values (1 or 0), previously known or indicated by experts at the time of displaying the results for each tuple. Based on this, we can also obtain the true positives, false positives, true negatives, and false negatives to calculate the precision, recall and accuracy metrics, according to Equations 4.3, 4.4 and 4.5 mentioned in the previous section. Additionally, visual representations can also be produced.

$$prediction\_syn\_sem\_avg = \begin{cases} 1 & if \ \frac{w_1 \cdot \text{avg\_syn\_rank} + w_2 \cdot \text{avg\_sem\_rank}}{w_1 + w_2} > threshold \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

$$prediction\_syn\_sem\_major = \begin{cases} 1 & if \ \frac{w_1 \cdot \text{major\_syn\_rank} + w_2 \cdot \text{major\_sem\_rank}}{w_1 + w_2} > threshold \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$

To generate semantic similarity metrics, semantic rankings, and syntactic-semantic evaluation to obtain the syntactic-semantic connection points, the ASSESA follows the Algorithm

4. It can be seen that it accepts three inputs: *top_k_syntactic_connection_points*, which is the *top k* syntactic connection points generated by ASSA; *k*, which is a number of tuples; *rank*, name of the sort semantic rank. The output results are *k* tuples of API connection points ordered by the *rank* parameter chosen in the input, and in descending order. In that algorithm, in detail, line 2 shows the creation of an empty list (*top_k_syntactic_semantic_connection_points*) that will receive the connection points. After that, for each tuple in the *top k* syntactic connection points (line 3), it calculates four short text similarity metrics (line 4), calculates rankings (line 5), and generates a predictions (line 6) based on the average of the syntactic and semantic ranks. The *correct* value is manually set either beforehand or during results presentation (line 7). The list of API connection points with syntactic and semantic details (*top_k_syntactic_semantic_connection_points*), enriched with metrics, semantic rankings, predictions, and correct are compiled (line 8) and then sorted by *rank* and *k* and returned (line 10).

Algorithm 4: API Syntactic-Semantic Similarity Algorithm

```
   Data: top_k_syntactic_connection_points, k, rank
   Result: top_k_syntactic_semantic_connection_points
1  begin
2      top_k_syntactic_semantic_connection_points ← ∅
3      for tuple ∈ top_k_syntactic_connection_points do
4          metrics ← calcMetrics(tuple)
           /* bert, roberta, t5, bart                                   */
5          ranks ← calcRanks(metrics)
           /* avg_sem_rank, major_sem_rank                              */
6          prediction ← calcPred(tuple[syn_rank], ranks)
7          correct ← manualData
8          top_k_syntactic_semantic_connection_points.append(asOne(tuple, metrics, ranks,
             prediction, correct))
9      end
10     return sort(top_k_syntactic_semantic_connection_points, k, rank)
11 end
```

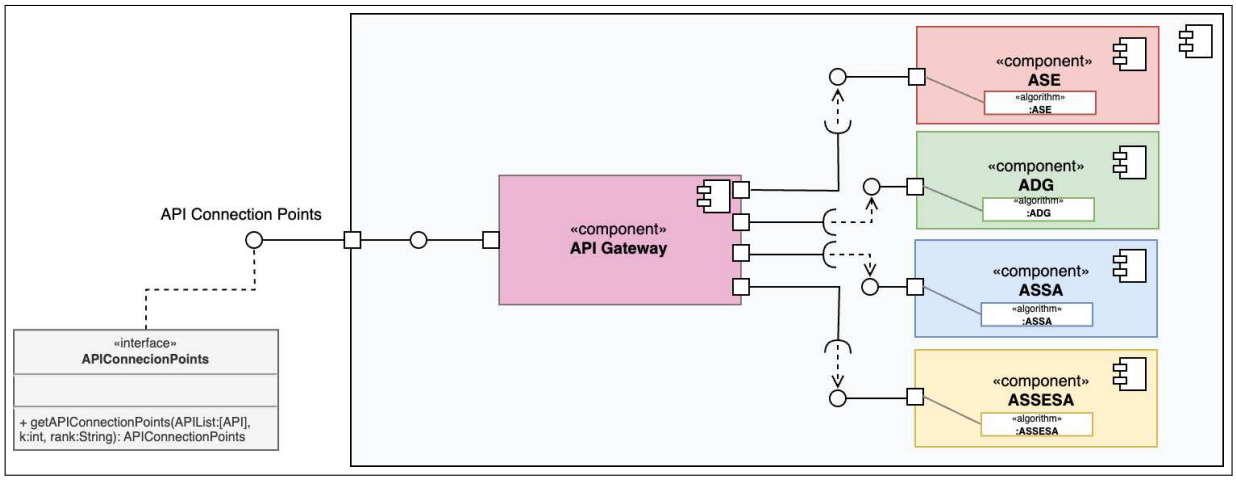
Source: Author.

4.3 Core method implementation

The core method design was implemented using a microservices architecture, as illustrated in the component diagram in Figure 20. The crucial part of the diagram is the *APIConnectionPoints* interface, which defines the API connection points as the returning data essentials. In that figure, it can still be seen that a gateway was added to receives this interface and provides a single access point to the four components of core method, which one following its own algorithm. The API gateway function is to transparently route and manage all request traffic

to specific destination components that hold the respective information. To do so, it will need to register the components, which are information from the schema of four components, as well as the endpoints to obtain the information wherever they are. Thus, it will have a comprehensive super schema that covers all the functionalities available in the registered components, as can be seen in Figure 21.

Figure 20 – Component diagram implemented in our core method



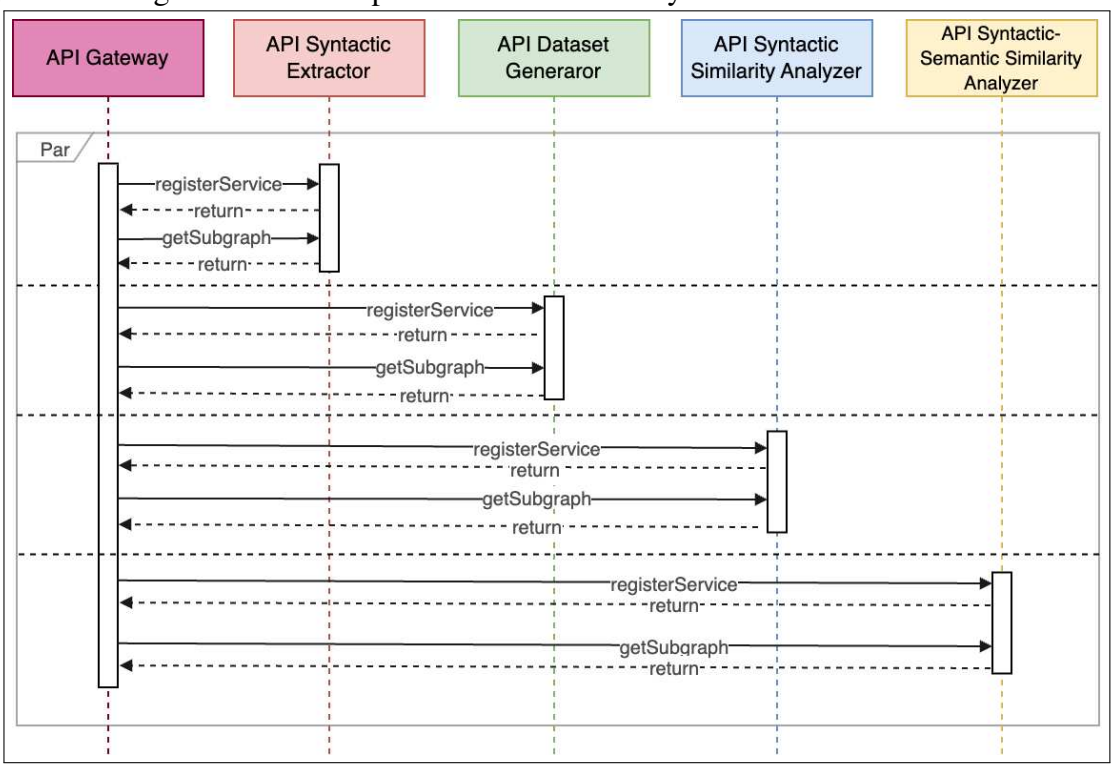
Source: Author.

From that component diagram, we generate the tool (our solution to support the method), which was implemented in terms of code based on two languages: Javascript and Python. They were chosen because they are the most used languages in the world in 2022, according to the Github report (GITHUB, 2022). With Javascript, the API Gateway[2] and the API Syntactic Extractor[3] were implemented. In the API Gateway, GraphQL[4] was used, which is a query language for API (PUUSTINEN, 2020), and is excellent to be used as a Gateway in microservices (TOURONEN *et al.*, 2019; TREBICHAVSKỲ, 2021). In more detail, Apollo Gateway was used, which is part of the Apollo GraphQL platform[5], which registers a list of microservices based on their endpoints, and which has an interesting feature, which is to obtain a super graph from the sub graphs ("schema") of the registered microservices, and makes them available in a single interface for centralized and on-demand access to microservices. In the API Syntactic Extractor, in addition to the codes that implemented the Algorithm 1, some auxiliary

[2] Available at: https://github.com/marcosborges1/api-gateway
[3] Available at: https://github.com/marcosborges1/api-syntactic-extractor-service
[4] Available at: https://graphql.org
[5] Available at: https://www.apollographql.com/

libraries can be highlighted: Apollo Server[6], Apollo Subgraph[7], and SwaggerParser[8]. The first two were used to allow the API Gateway to visualize its sub graphs. The latter was used to carry out validations and extract key information.

Figure 21 – Registration of components in API Gateway



Source: Author.

With the Python language, the API Dataset Generator[9], the API Syntactic Similarity Analyzer[10], and the API Syntactic-Semantic Similarity Analyzer[11] were implemented. For the API Dataset Generator, most of the implementation of the respective Algorithm 2 was done by the standard mechanisms of the language itself, and the rest was done with the help of libraries: TextDistance[12], for the calculation of algorithms of string similarities; Pandas[13] for generating the dataset in *.csv* format. For the API Syntactic Similarity Analyzer, most of the implementation of the Algorithm 3 (metrics, rankings, gini and predictions) was also done in pure language, and the rest with the help of libraries: MatPlotLib[14], for the generation of graphs, and Pandas,

6 Available at: https://www.apollographql.com/docs/apollo-server/
7 Available at: https://www.npmjs.com/package/@apollo/subgraph
8 Available at: https://www.npmjs.com/package/swagger-parser
9 Available at: https://github.com/marcosborges1/api-dataset-generator-service
10 Available at: https://github.com/marcosborges1/api-syntactic-similarity-analyzer-service
11 Available at: https://github.com/marcosborges1/api-syntactic-semantic-similarity-analyzer-service
12 Available at: https://pypi.org/project/textdistance/
13 Available at: https://pandas.pydata.org
14 Available at: https://matplotlib.org

for sampling data in table format. Finally, API Syntactic-Semantic Similarity Analyzer also implemented Algorithm 5 using the Python language itself, and to calculate the semantic part of the deep learning algorithms the following libraries and models provided by: Transformers[15], Sentence-Transformers[16], and Torch[17].

4.4 Final Remarks

In this chapter, we presented the method developed to support CS developers in achieving interoperability links in SoIS. The method identifies connection points within CS through detailed analyses of their respective API and is described in three interconnected steps: requirements definition, core method design, and core method implementation.

In the requirements definition, we explored the main motivations that underpinned the creation of the method, based both on recent literature on SoIS and on a 30-month direct observation of CS developers' activities in achieving interoperability links in SoIS. In the core method design, we detailed the systematic activities to identify API connection points, which are fundamental elements for CS developers to implement interoperability links. These activities, a total of four, follow specific and complementary algorithms which generate 33 API connection points, including general information about the API, such as attribute names, attribute descriptions, endpoints, syntactic methods, and semantic methods. In the core method implementation, we presented the technical details, including the programming languages and libraries used in the construction of the tool that supports the method.

In the next chapter, we will address the evaluation of the developed tool in two different contexts: a controlled experiment, and a case study.

[15] Available at: https://pypi.org/project/transformers/
[16] Available at: https://pypi.org/project/sentence-transformers/
[17] Available at: https://pypi.org/project/torch/

5 EVALUATION

This chapter presents our evaluation, which covers the application of our tool (solution to support the method) in two different contexts: a controlled experiment and a case study. First of all, in Section 5.1, we provide an overview of the tool, highlighting its modules and the process of obtaining the results. Then, in section 5.2, we perform the first evaluation, which involves the controlled experiment using Jira and Github API. Section 5.3 is the second evaluation and involves case study, which is subdivided three scenarios involving five CS from a real-world SoIS of a global computer manufacturer. Section 5.4 implications for both developers and researchers are presented. Section 5.5 discusses the limitation, while Section 5.6 discusses the threats to validity. Finally, section 5.7 presents the final remarks.

5.1 Tool - Overview

The tool we developed includes a front-end to interact with the back-end of the components implemented and made available in Section 4.3. For confidentiality reasons, the full source code, covering both the front-end and back-end, will not be made publicly available, as it was part of the delivery in the technology transfer process to a global computer manufacturing company, with which we maintained a research partnership for 30 months. Once the tool configured correctly, a the user will see the login screen, which has the “username” and “password” fields, as shown in Figure 22. If the user is not yet registered, he or she can register by following the instructions in Figure 23, filling in the fields: “username”, “email”, and “password”. Once registered, the user will receive a confirmation message “User created successfully.”, as shown in Figure 24. After that, the user can return to the login screen using the “username” and “password” data to access the tool’s internal features.

After login is successfully completed, the user will see the message “*Welcome to the AgapeTool: The initial information and modules are now available in the left-side menu*”, as shown in Figure 25. In that figure, it is possible to note that the user will have access to various important modules for navigating the tool. In addition to the “Home” link, which contains the initial welcome message, the following modules will be available: (i) API: this module deals with API management; (ii) Syntactic: this module deals with the API syntactic analysis, showing as results the indications of interoperability links among them; (iii) Syntactic-semantic: this module deals with the addition of semantic analysis to the syntactic analysis, aiming to improve

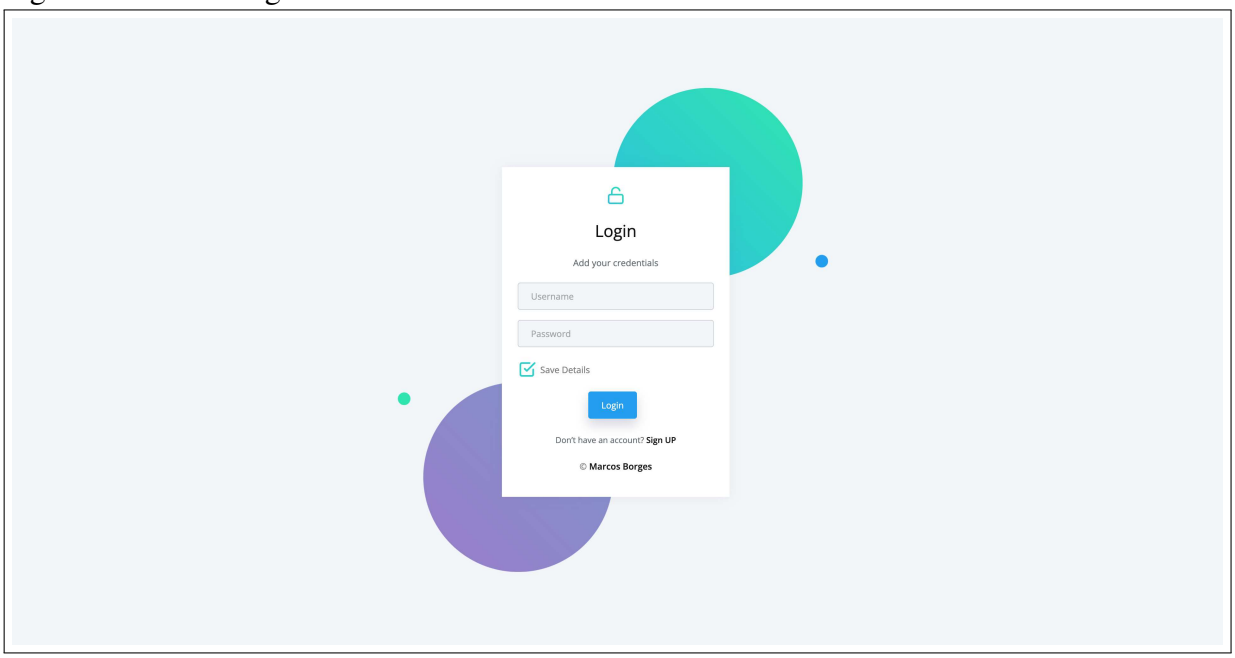
the indication of interoperability links.

The API module is responsible for API management. It is composed of two main web pages: Insert and List. On the Insert page (Figure 26), the user has the ability to add new API to the tool. This page features two important fields: “file” and “name”. The “file” field allows the user to upload the API file. An interesting detail that makes it easier to use is that, when uploading the file, the “name” field is automatically filled in with the name of the API, extracted from the uploaded file. This not only speeds up the insertion process, but also reduces the possibility of manual errors when inserting the API name. The List page (Figure 27) provides an overview of all API that have been inserted into the tool. On this page, users can view the added API, with options to change or delete as needed. The change functionality allows users to make adjustments to already inserted API, while deletion allows users to remove API that are no longer needed or that have been replaced by more up-to-date versions. The API managed through this module will serve as the basis for the next modules of the tool, which deal with API similarity analysis: syntactic and syntactic-semantic.

The API syntactic similarity analysis module is responsible for evaluating and identifying connection points between different API. In Figure 28, the user can analyze several API to determine these connection points. That can be done in two ways: directly typing in the desired API or selecting them manually by clicking on them. The chosen *API list* are displayed in the “API Selected” section on the right of the screen. In addition to selecting the *API list*, the user also needs to define the number k and the *rank*, according to the input parameters of our method, as shown in Figure 19. With all settings defined, the user can click the “Analyze” button to start the calculations and identify the API connection points. The results of this analysis are shown in Figure 29 as “Output - API Syntactic Connection Points”, which are tuples detailing each attribute connection point between the API. These tuples include information on the syntactic similarity metrics, syntactic rankings, auxiliary calculations, and syntactic evaluation, as described in Table 6. An important aspect of the module is the ability to indicate the parameter “*correct_syn*”, where 0 means that the attributes are unrelated and 1 means that the attributes are related, as also described in Table 6. That is indicated by experts who know the API information.

With the information provided by experts, the user can then click on the “Generate Evaluation Metrics” button to calculate evaluation metrics such as true positives, false positives, true negatives, and false negatives, which can be seen in Figure 29. These metrics are essential

Figure 22 – Tool: login



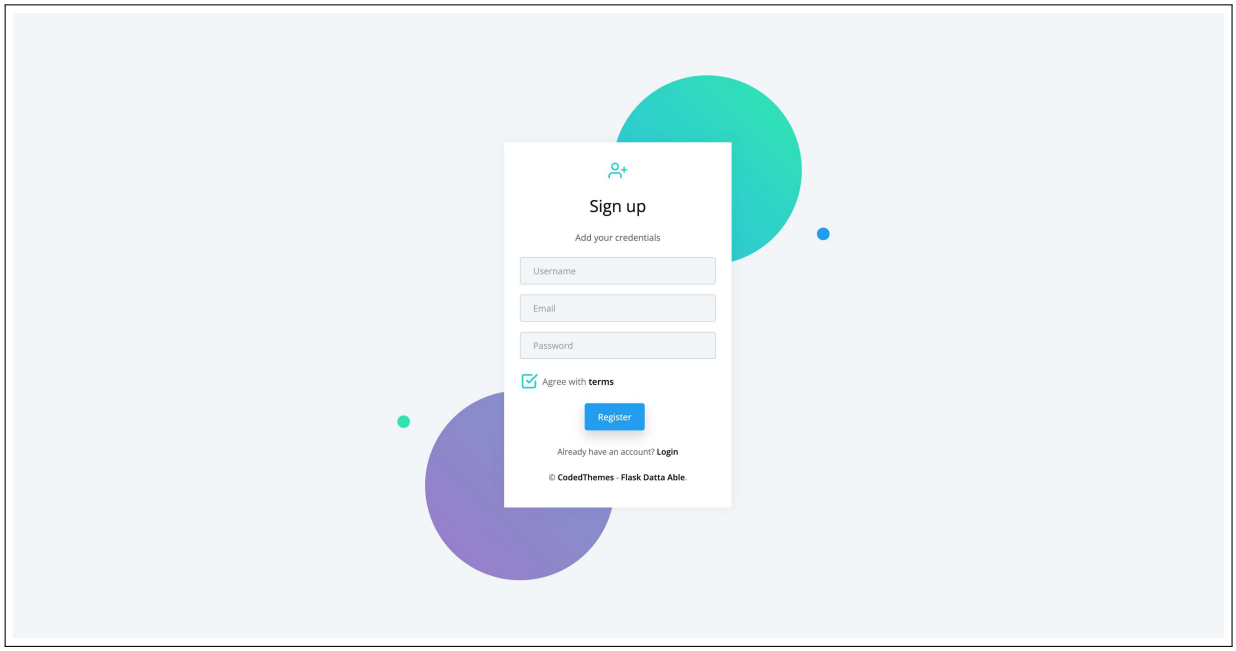
Source: Author.

for calculating precision, recall, and accuracy, all described in detail in section 4.2.3. With these API connection points, CS developers can implement them in their preferred programming language to achieve interoperability links among API.

Similarly to the API syntactic similarity analysis module, the API syntactic-semantic similarity analysis module also identifies connection points. However, it considers both the syntactic part of the API attributes and the semantic part of the descriptions of these attributes. As with the syntactic similarity module, the user can select API in two different ways, in addition to configuring parameters for analysis. However, the process generates three main results: (i) syntactic analysis of API attributes (Figure 30), (ii) semantic analysis of API attribute descriptions (Figure 31), and (iii) combined analysis of syntactic and semantic dimensions, in addition to evaluation metrics, in the same way as in the API syntactic similarity module (API Syntactic Similarity Analysis Module), as illustrated in Figure 32.

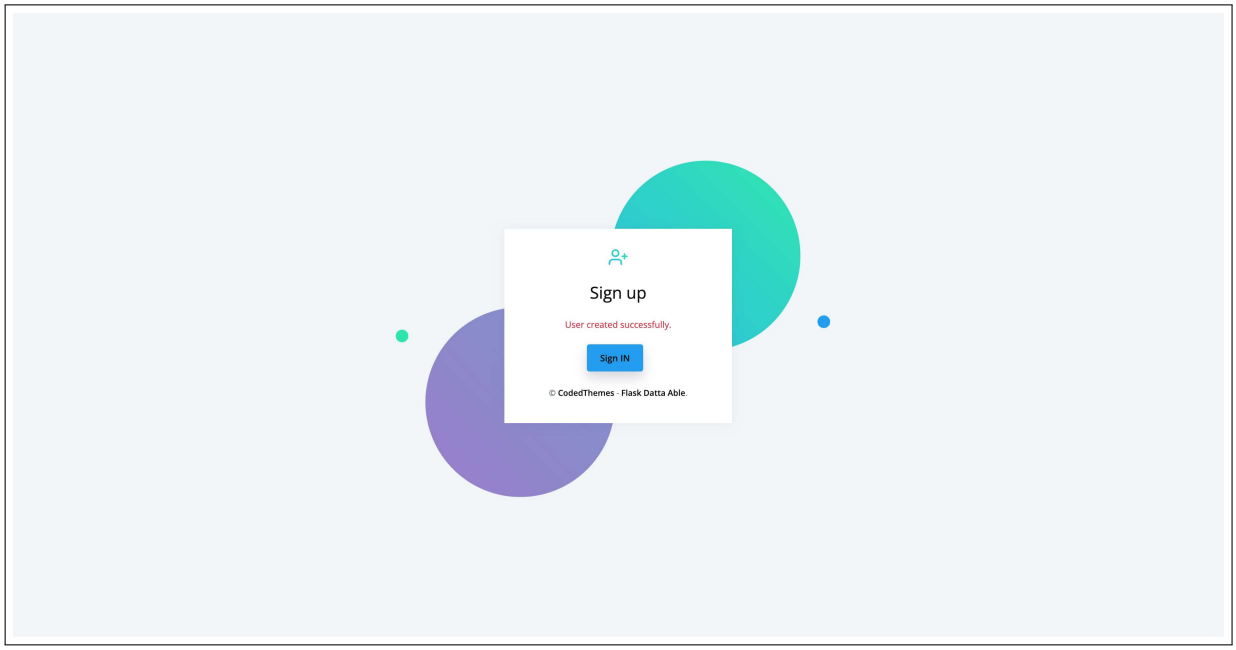
It is worth noting that the following results, which will be used in the controlled experiment and case study sections, are derived from the execution of the processes described in this tool overview section.

Figure 23 – Tool: Sign Up



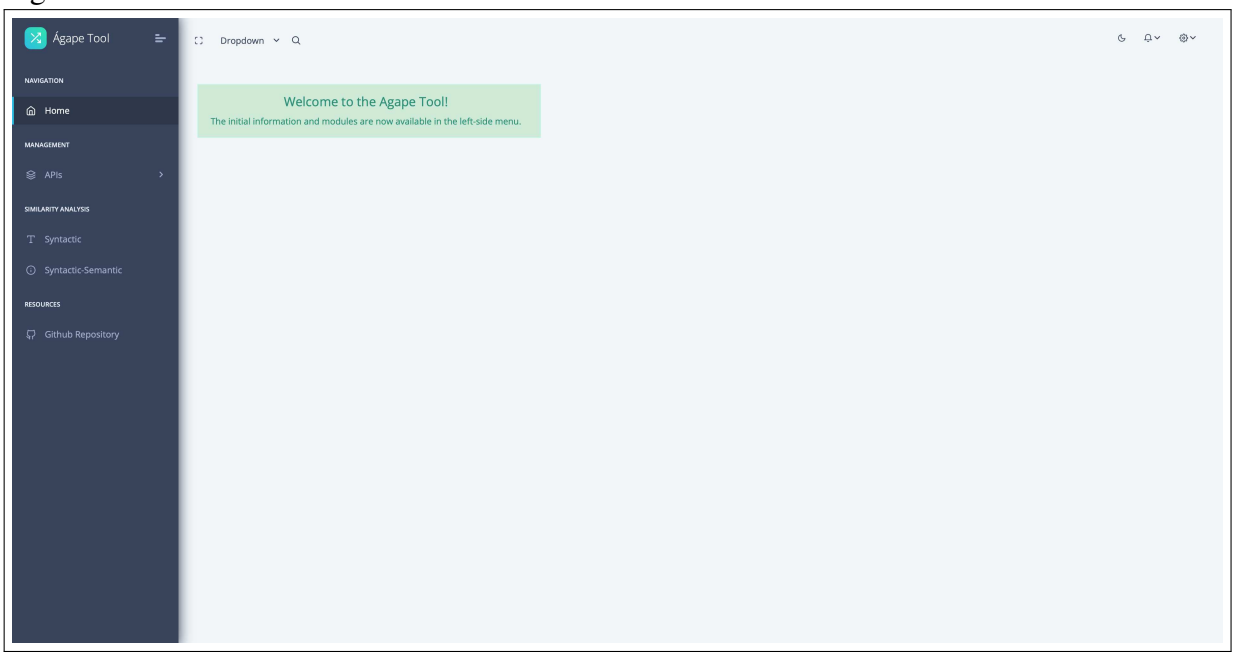
Source: Author.

Figure 24 – Tool: User created successfully



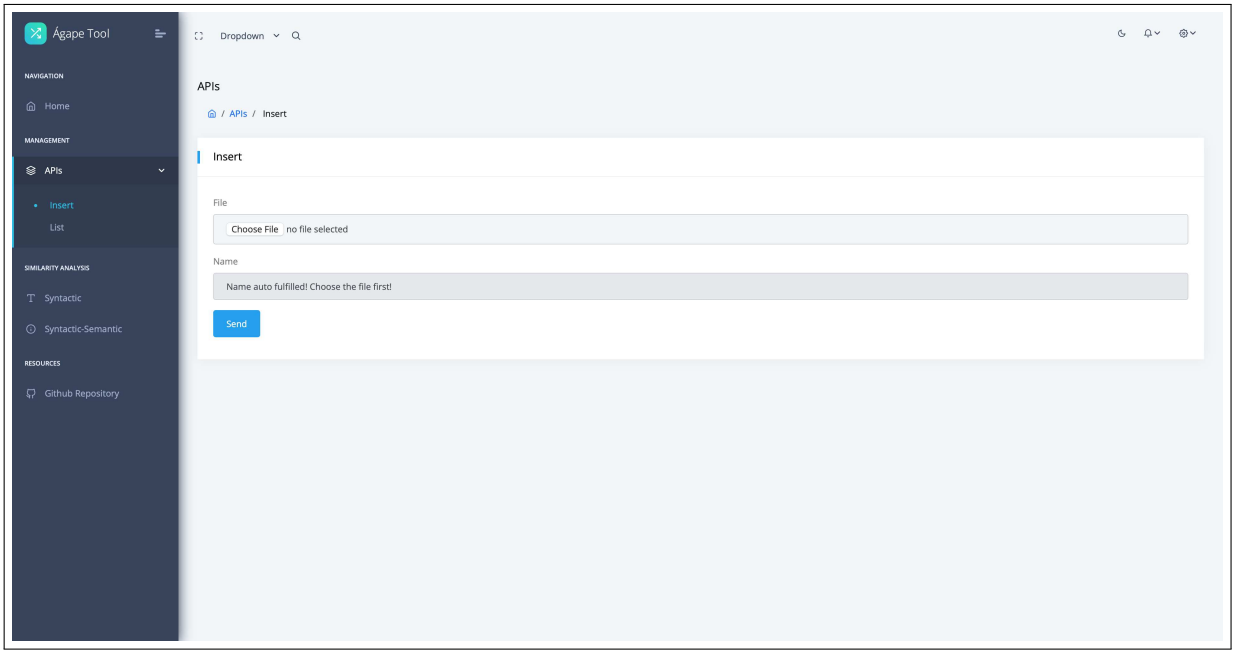
Source: Author.

Figure 25 – Tool: Home



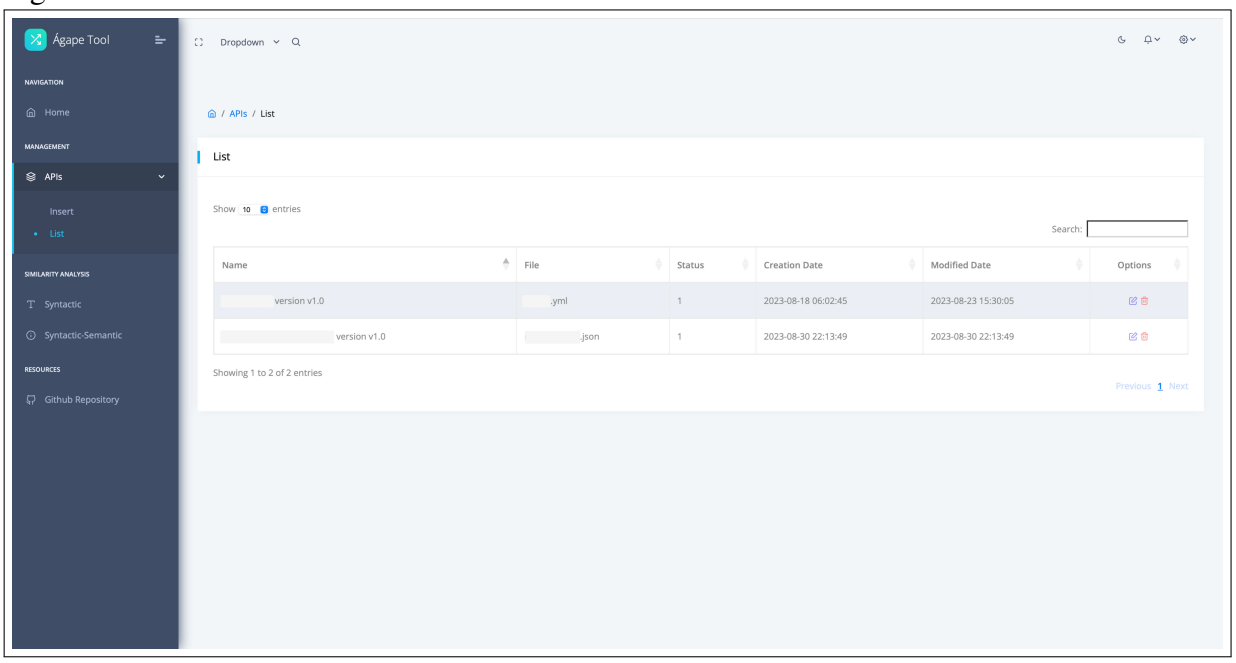
Source: Author.

Figure 26 – Tool: API Insert



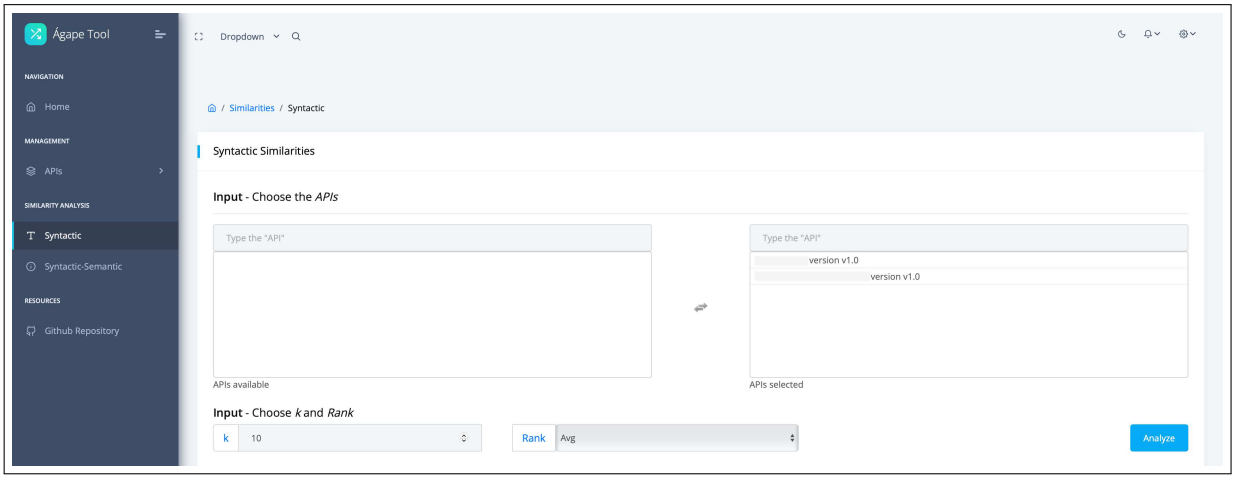
Source: Author.

Figure 27 – Tool: API List



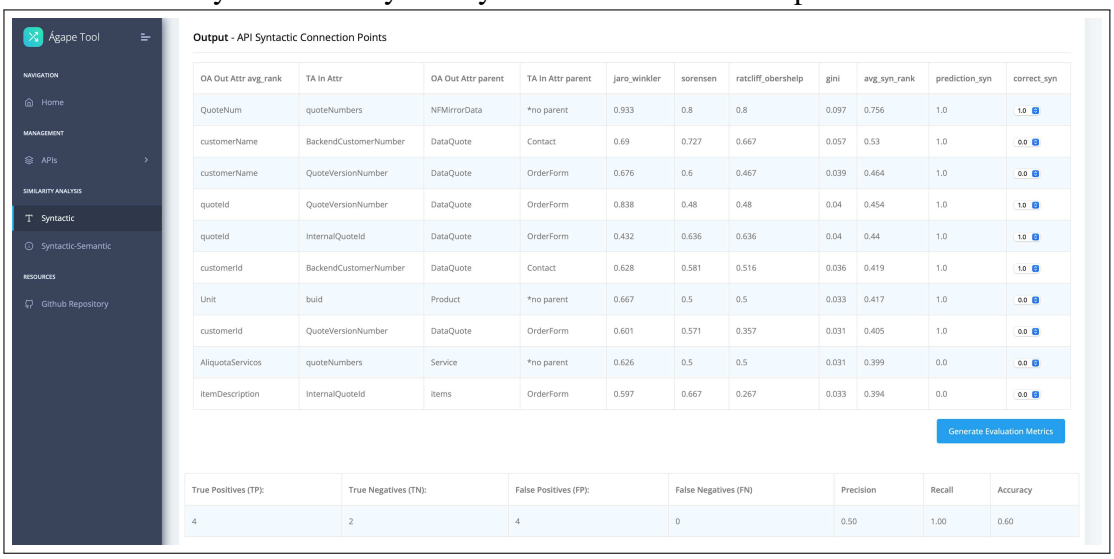
Source: Author.

Figure 28 – Tool: Syntactic Analysis - input parameters



Source: Author.

Figure 29 – Tool: Syntactic Analysis - Syntactic API connection points and evaluation metrics



Source: Author.

Figure 30 – Tool: Syntactic-Semantic Analysis - Syntactic API connection points

Source: Author.

Figure 31 – Tool: Syntactic-Semantic Analysis - Semantic API points

Source: Author.

Figure 32 – Tool: Syntactic-Semantic Analysis - Syntactic-semantic API points and evaluation metrics

Source: Author.

5.2 Controlled Experiment

As a controlled experiment, we aim to identify the best string similarity algorithms and optimal thresholds for detecting API connection points using our tool, using API syntactic similarity analysis module (BORGES *et al.*, 2024). We used the tool on two real API with known composition points. Using the top-performing similarity algorithms and thresholds, we will fine-tune our tool for use in section 5.3 with more complex and real CS API.

We employed the tool to examine the Jira[1] and GitHub[2] API (BORGES *et al.*, 2024). Our main objective was to obtain the best string similarity metrics from six algorithms (Table 6) for a known connection point between these API, as indicated by Vieira *et al.* (2019). Our focus was on determining top-performing algorithms from average and major rankings for mapping Github commits to Jira issues. Building on insights from Vieira *et al.* (2019), we found that Jira's endpoint "*api/2/search*" is pivotal for composing GitHub's "*search/commits*" request. Prior to analysis, we adjusted both API to align with our method's guidelines.

To meet the API contracts of the our method, we had to make two adaptations to the Jira API. The first was the transformation of the API from Web Application Description Language (WADL), available in its documentation, to Open API, in this situation we chose

[1] Available at: https://docs.atlassian.com/software/jira/docs/api/REST/7.6.1/jira-rest-plugin.wadl

[2] Available at: https://raw.githubusercontent.com/github/rest-api-description/main/descriptions/api.github.com/api.github.com.json

version 3.0, which is one of the formats accepted by the tool. The second adaptation was made in Jira to remove a circular reference in the schema "*link-groups*", which had a circular reference to itself in its "*groups*".

We even made some optimizations in the two API, to make them more complete. That completeness was based on the documentation of each API. In this thesis we will present the main ones. For example, in the JIRA endpoint "*api/2/search*" there is a JIRA Query Language (JQL) parameter of type string, but this parameter is not just a simple string, it is actually a well-defined query and it needs to be standardized with some reserved words inherent to the language. An important aspect is noticed in the JQL description: "The query contains one or more search keywords and qualifiers". Checking the documentation and such data, the *jql* parameter can be abstracted to the following possibilities of extended parameters, which can be seen in Table 7. Also in this same table can be seen the Github API abstraction in the "*search/commits*" endpoint for the Query (*q*) parameter of type string, which can be seen in its documentation that "The query contains one or more search keywords and qualifiers", and from the documentation can be abstracted to the extended *q* parameter.

By respecting and adapting the API contracts, we were able to obtain the main results of our study based on two ordered rankings, in descending order, of the API connection points. For the first ordered ranking, we ran the tool with the following parameters: *API list* = [Jira API, Github API], *k*=10 (10 tuples), *rank=avg_rank*. For the second ordered ranking we just changed the ranking attribute to *major_rank*. It is important to highlight that the correct values coming from the templates based on Vieira *et al.* (2019) study were updated from the dataset generated by the tool.

The API connection points ordered by *avg_rank* can be seen in Table 8, where it is noticed that the tool obtained good results for metric evaluations setting true predictions (*prediction=1*) of the API connection points from the 0.5 threshold of the *avg_rank* attribute, which we adjusted manually from some executions with different values of thresholds. With that, the ranking obtained a precision of 0.75, recall of 1.0, and accuracy of 0.9. That table shows the exact connection point of Jira, which are the attributes *key* of *statusCategory*, *key* of *issues*, *key* of *author* with the Github *keyword* of *keywords* are exactly the connection points. From Table 8 and Figure 33 it can be seen that the similarity algorithms *jaro_winkler*, *sorensen* and *ratcliff_obershelp* are the best considering the average of the algorithms. Figure 34 a high degree of positive correlation can be seen between *soresen-jaccard* (1) and *jaro_winkler-*

Table 7 – Key parameters adapted from Jira and Github based on their documentations

API	Endpoint	Parameter	Extended parameter based on documentation
Jira (7.6.1)	api/2/search	jql:string	{"jql:" {"affectedVersion": "string", "approval": "string", "assignee": "string", "attachments": "string", "category": "string", "comment": "string", "component": "string or int", "created": "string", "creator": "string", "cf[]": "string", "Customer Request Type":"string", "description": "string", "due": "string", "environment": "string", "epic link": "string", "filter": "string or int", "fixVersion": "string or int", "issueKey": "string", "labels": "string", "lastViewed": "string", "level": "string or int", "originalEstimate": "string", "parent": "string", "priority": "string or int", "project": "string or int", "remainingEstimate": "string", "reporter": "string", "request-channel-type": "string", "request-last-activity-time": "string", "resolution": "string or int", "resolved": "string", "sprint": "string or int", "status": "string or int", "summary": "string", "text": "string", "timeSpent": "string", "Type": { "type": "string","issueType": "string or int"}, "updated": "string","voter": "string", "votes": "int", "watcher": "string", "watchers": "int", "worklogAuthor": "string", "Work log comment": { "comment": "string","summary": "string"}, "worklogDate": "string", "workRatio":"int"}}
Git Hub (1.1.4)	/search/commits	q:string	{ "q": [{"keywords":[{"keyword": "string"}]}, {"qualifiers": [{"author": "string"},{"committer": "string"},{"author-name": "string"}, {"committer-name": "string"}, {"author-email": "string"},{"committer-email": "string"}, {"author-date": "string"},{"merge": true},{"hash": "string"},{"parent": "string"}, {"tree": "string"}, {"user": "string"}, {"org": "string"}, {"repo": "string"},{"is": "string"}]}] }

Source: Author.

Table 8 – API list=(Jira 7.6.1 API (‘api/2/search’)→Git Hub 1.1.4 API (‘search/commits’)) | k = 10 | rank=(avg_rank) / Thereshold = 0.5 / Precision (0.75) - Recall (1.0) - Accuracy (0.9)

Index	OA Out Attr	TA In Attr	OA Out Attr parent	TA In Attr parent	hamming	levenshtein	jaro_winkler	jaccard	sorensen	ratcliff_obershelp	prediction	correct	avg_rank
1	key	keyword	statusCategory	keywords	0.429	0.429	0.867	0.429	0.600	0.600	1	1	0.559000
2	key	keyword	issues	keywords	0.429	0.429	0.867	0.429	0.600	0.600	1	1	0.559000
3	key	keyword	author	keywords	0.429	0.429	0.867	0.429	0.600	0.600	1	1	0.559000
4	created	tree	histories	qualifiers	0.286	0.429	0.726	0.571	0.727	0.545	1	0	0.547333
5	href	tree	header	qualifiers	0.500	0.500	0.667	0.333	0.500	0.500	0	0	0.500000
6	href	tree	links	qualifiers	0.500	0.500	0.667	0.333	0.500	0.500	0	0	0.500000
7	type	tree	historyMetadata	qualifiers	0.500	0.500	0.667	0.333	0.500	0.500	0	0	0.500000
8	type	tree	actor	qualifiers	0.500	0.500	0.667	0.333	0.500	0.500	0	0	0.500000
9	type	tree	cause	qualifiers	0.500	0.500	0.667	0.333	0.500	0.500	0	0	0.500000
10	type	tree	generator	qualifiers	0.500	0.500	0.667	0.333	0.500	0.500	0	0	0.500000

Source: Borges *et al.* (2024).

Table 9 – API list=(Jira 7.6.1 API (‘api/2/search’)→Git Hub 1.1.4 API (‘search/commits’)) | k = 10 | rank=(major_rank) / Thereshold = 0.8 / Precision (1.0) - Recall (1.0) - Accuracy (1.0)

Index	OA Out Attr	TA In Attr	OA Out Attr parent	TA In Attr parent	hamming	levenshtein	jaro_winkler	jaccard	sorensen	ratcliff_obershelp	prediction	correct	major_rank
1	key	keyword	statusCategory	keywords	0.429	0.429	0.867	0.429	0.600	0.600	1	1	0.867
2	key	keyword	issues	keywords	0.429	0.429	0.867	0.429	0.600	0.600	1	1	0.867
3	key	keyword	author	keywords	0.429	0.429	0.867	0.429	0.600	0.600	1	1	0.867
4	colorName	author-name	statusCategory	qualifiers	0.000	0.545	0.737	0.429	0.600	0.600	0	0	0.737
5	created	tree	histories	qualifiers	0.286	0.429	0.726	0.571	0.727	0.545	0	0	0.727
6	emailDescription	committer-email	historyMetadata	qualifiers	0.062	0.188	0.539	0.550	0.710	0.323	0	0	0.710
7	timeZone	committer-name	author	qualifiers	0.143	0.286	0.696	0.467	0.636	0.273	0	0	0.696
8	descriptionKey	repo	historyMetadata	qualifiers	0.071	0.214	0.679	0.286	0.444	0.333	0	0	0.679
9	self	user	statusCategory	qualifiers	0.000	0.250	0.667	0.333	0.500	0.500	0	0	0.667
10	avatarUrl	author-email	generator	qualifiers	0.167	0.250	0.667	0.400	0.571	0.286	0	0	0.667

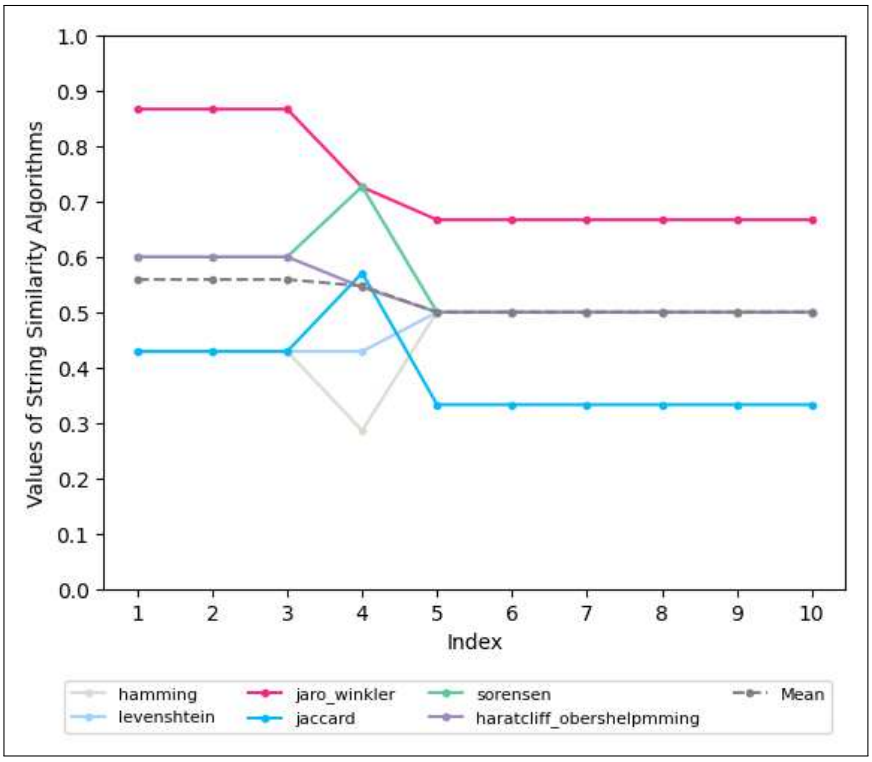
Source: Borges *et al.* (2024).

ratcliff_obershelp (0.99). In the negative correlation, the *hamming-jaccard* (-1) and *sorensen-hamming* (-0.99) relationship can be highlighted.

The API connection points ordered by *major_rank* can be seen in Table 9, which shows excellent results considering the threshold above 0.8 based on the *major_rank* attribute to make predictions of API connection points. For that example, all evaluation metrics were assertive, resulting in precision of 1.0, recall of 1.0, and accuracy of 1.0. In Figure 35 shows the best similarity algorithms (*jaro_winkler*, *sorensen*, *ratcliff_obershelp*) considering the average of *major_rank*. In Figure 36 a high degree of positive correlation can be seen between soresen-jaccard (1) and levenshtein-ratcliff_obershelp (0.9). However, in the negative correlation, highlights are not noticed.

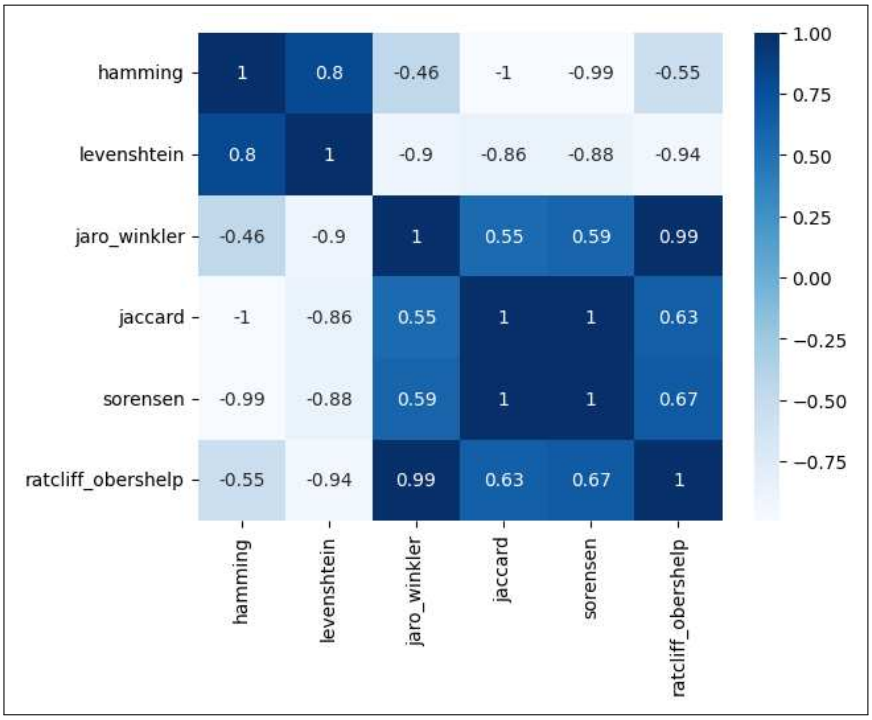
So, based on the two ordering of API connection points presented above, it was evident that the best string similarity algorithms for the predictions are the following: *jaro_winkler*, *sorensen*, and *ratcliff_obershelp*. Their ideal thresholds are 0.5 for *avg_rank* and 0.8 for *major_rank* for predictions of API connection points. Such algorithms and thresholds will be considered as a basis for next evaluations.

Figure 33 – Performance of the 6 string similarity algorithms based on Jira→Github ranking (*avg_rank*)



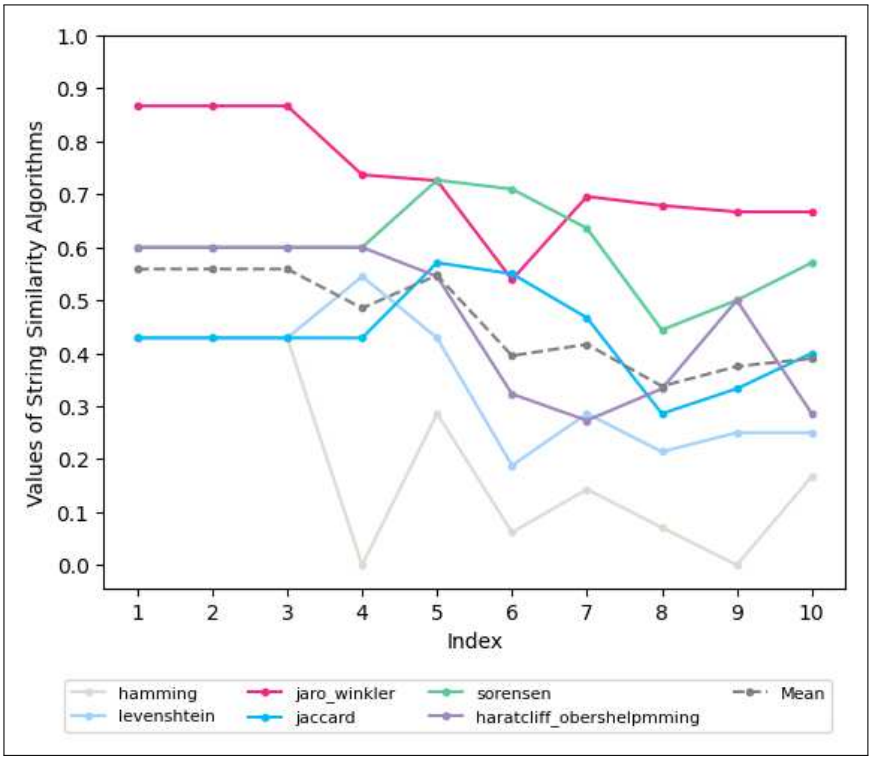
Source: Author.

Figure 34 – Correlation chart among the 6 string similarity algorithms based on Jira→Github ranking (*avg_rank*)



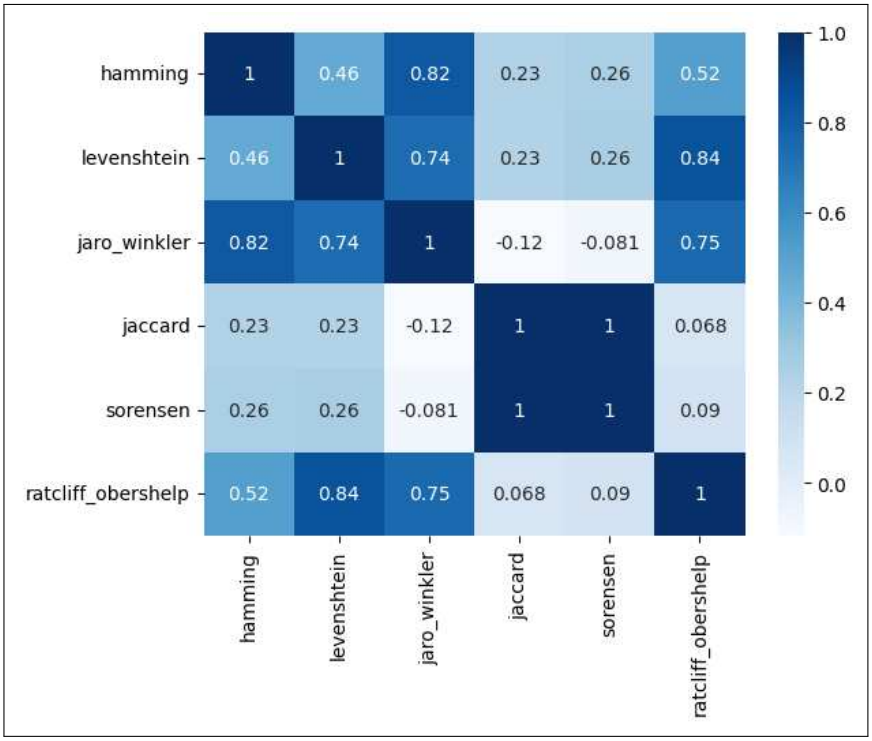
Source: Author.

Figure 35 – Performance of the 6 string similarity algorithms based on Jira→Github ranking (*major_rank*)



Source: Author.

Figure 36 – Correlation chart among the 6 string similarity algorithms based on Jira→Github ranking (*major_rank*)



Source: Author.

5.3 Case Study

This section presents the evaluation of our tool on a real-world SoIS sales system from a global computer manufacture, in which the composition points of five CS are known to the developers of another CS, with the goal of establishing interoperability links. For clarity and organization, we have structured this real case study into two main analysis perspectives: (i) syntactic, focused on the analysis of the syntactic API attributes; and (ii) syntactic-semantic, focused on the analysis of API syntactic attributes and the semantic analysis of the textual descriptions of these attributes.

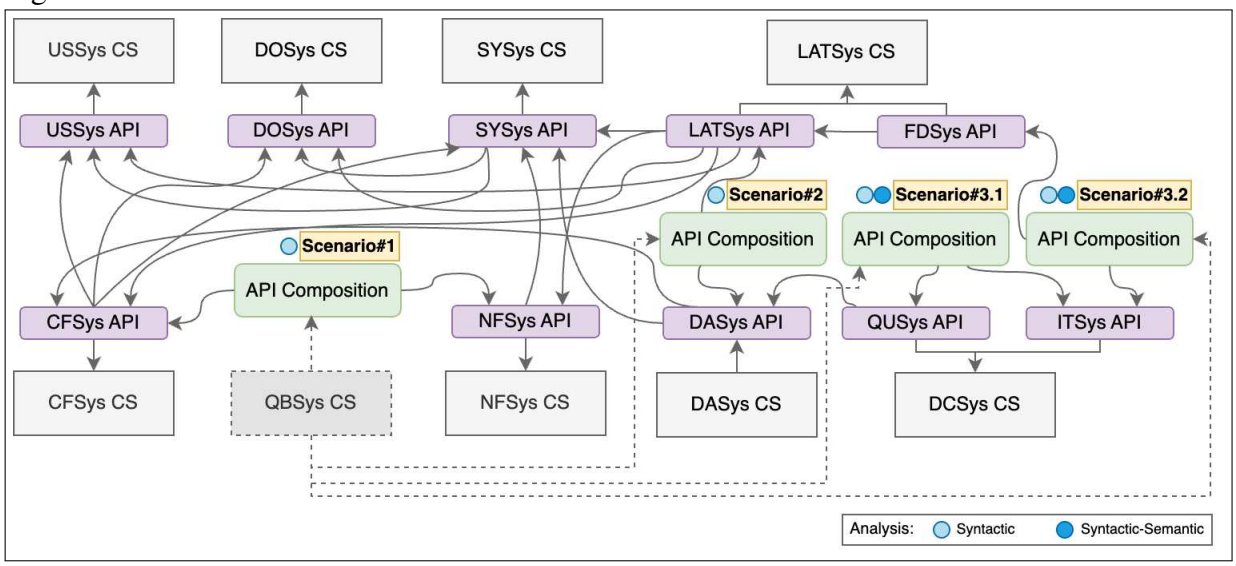
It is important to highlight that the definition of case study used in this thesis is that of Wohlin (2021): "A case study is an empirical investigation of a case, using multiple data collection methods, to study a contemporary phenomenon in its real-life context, and with the investigator(s) not taking an active role in the case investigated.". That definition is contemplated in this thesis, since we conducted an empirical investigation using several data collection methods, such as observation of the CS developers activities, analysis of API based on documentation and interviews with developers. Furthermore, the role of the researcher was strictly limited to verifying the results obtained, with the objective of evaluating the viability of the tool, without any direct interference in the case study investigated.

Thus, we will describe below the SoIS data (subsection 5.3.1), the syntactic analysis perspective (subsection 5.3.2), and the syntactic-semantic analysis perspective (5.3.3).

5.3.1 Data from SoIS

In the Information Technology (IT) sector, we evaluated a large real-world SoIS sales system from a global computer manufacturer. For confidentiality reasons, the company will be identified as DSoIS. DSoIS has approximately 120.000 employees, a market cap of $89.33 billion, 237 API, and 91 CS (BORGES *et al.*, 2024). In the scope of this evaluation, we use some CS, as shown in Figure 37, including: (i) USSys for for selling personal computers and accessories; (ii) DOSys for selling professional equipment, such as network infrastructure and data centers; (iii) SYSys for calculating taxes and fees on products in Brazil; (iv) CFSys for generating product quotes; (v) NFSys for controlling invoice issuance; and (vi) QSys, for extracting and calculating of invoice data; (vii) DASys for global sales and order management; (viii) LATSys for fiscal document processes; and (ix) DCSys, for managing quote and order

Figure 37 – CS and API of the DSoIS.



Source: Adapted from Borges *et al.* (2024).

transactions (BORGES *et al.*, 2024).

As shown in Figure 37, the communication layout among these CS reveals distinct API (based on either REST or SOAP) for each CS to ensure fluid interactions. USSys, DOSys, SYSys, CFSys, NFSys, LATSys, DASys, and DQSys provide one or more own API, however we highlight only those that are used or fundamental to understanding the context. SYSys communicates with both USSys and DOSys, and CFSys connects to USSys, DOSys, and SYSys. On the other hand, NFSys operates through SYSys. LATSys interconnects with USSys, DOSys, CFSys, SYSys and NFSys. DASys connects to CFSys, and SYSys. DCSys interconnects with DASys. However, with QBSys still under development and without an own API.

Achieving interoperability links proved to be a significant and time-consuming challenge for the QBSys CS developers in all three scenarios in the case study. In **Scenario#1**, the integration involved the CFSys and NFSys CS, using the CFSysAPI and NFSysAPI API. In **Scenario#2**, the process included the DASys and LATSys CS, using the DASysAPI and LATSysAPI API. Finally, in **Scenario#3**, interoperability was achieved between the DCSys and LATSys CS, using the QUSysAPI, ITSysAPI, and FDSysAPI API. Theses links were crucial to allow communication between theses respective CS. By making a single request for data, theses CS can collaborate and leverage their API to enrich data from multiple sources and enhance their overall capabilities. However, despite having access to the API and their documentation, QBSys CS developers found it difficult to identify the correct API connection points, as it was crucial to accurately determine the specific data to be exchanged, ensure compatibility between interfaces (including methods, parameters and data types) and ensure a seamless connection process.

5.3.2 Syntactic Analysis Perspective

In this subsection, we report the evaluation of our tool applied to the API of DSoIS scenarios #1 and #2, with emphasis on the syntactic analysis perspective, which analyzes API attributes. In this context, the interoperability links between the four CS involved in these scenarios were identified by the QB CS developers, allowing a comparative analysis of the results generated by the tool. Therefore, our aim was to investigate whether we could replicate the results obtained by those CS developers by configuring the tool with the optimal algorithms (*jaro_winkler*, *sorensen* and *ratcliff_obershelp*) and thresholds (0.5 for *avg_rank* and 0.8 for *major_rank*) obtained from the controlled experiment.

To do so, we conducted semi-structured interviews with them and their feedback was crucial for our method and tool evolution. Subsequent subsections present the results from applying the tool on four CS API from the SoIS (5.3.2.1), and the interviews with five CS developers regarding the tool's feasibility (5.3.2.2).

5.3.2.1 Results

- Scenario#1:

After an intensive development phase, QBSys CS developers achieved and implemented the interoperability links of CFQSys and NFSys through their respective API. A set of five developers had hands-on experience with interoperability links tasks, which was crucial in deploying and interpreting results from our tool. When we executed the tool on the CFSysAPI (Open API version 2.0) and NFSysAPI (Open API version 3.0.1) in their presence, it ran seamlessly with no alert. This flawless execution showed that the API conformed to the standards of the three API contracts, and that it was not necessary to adapt them.

For a comprehensive evaluation of potential API connection points for achieving interoperability links, we used four distinct ordered rankings to obtain all connection possibilities. Two compared CFSysAPI and NFSysAPI based on *avg_rank* and *major_rank*, and other two did the reverse sense, i.e., NFSysAPI to CFSysAPI. The five developers meticulously reviewed these rankings, which facilitated the derivation of precision, recall, and accuracy evaluation metrics.

Table 10 contains the first ranking generated by our tool with the following parameters: *API List*=[CFSysAPI, NFSysAPI], *k*=10, *rank=avg_rank*. After analysis by the developers, it was verified that the tool had a precision of 0.3, recall of 1.0 and accuracy of 0.3, with a

gini index below 0.17, which characterizes a certain equality of distribution for the connection points. Table 11 shows the second ranking generated from our tool with the parameters: *API List*=[CFSysAPI, NFSysAPI], *k*=10, *rank=major_rank*. After analysis by the developers again, we obatined a precision of 1.0, recall of 0.33, and accuracy of 0.8, with a gini index below 0.13. It can still be noticed that in both tables 8 of the 10 crossed pairs are identical (*customerName / BackendCustomerNumber, quoteId / QuoteVersionNumber, customerName / QuoteVersion-Number, customerId / BackendCustomerNumber, quoteId / InternalQuoteID, itemDescription / QuoteVersionNumber, customerId / InternalQuoteId, customerId / QuoteVersionNumber*), the which characterizes a good degree of homogeneity between the two rankings.

Table 10 – API list = (CFSysAPI v1.0→NFSysAPI v1.0) | k = 10 | rank=(avg_rank) / Thereshold = 0.5 / Precision (0.3) - Recall (1.0) - Accuracy (0.3)

Index	OA Out Attr	TA In Attr	OA Out Attr parent	TA In Attr parent	jaro_winkler	sorensen	ratcliff_obershelp	prediction	correct	gini	avg_rank
1	customerName	BackendCustomerNumber	DataQuote	Contact	0.690	0.727	0.667	1	0	0.161058	0.694667
2	quoteId	QuoteVersionNumber	DataQuote	OrderForm	0.838	0.480	0.480	1	1	0.129227	0.599333
3	customerName	QuoteVersionNumber	DataQuote	OrderForm	0.676	0.600	0.467	1	0	0.115007	0.581000
4	customerId	BackendCustomerNumber	DataQuote	Contact	0.628	0.581	0.516	1	1	0.110911	0.575000
5	quoteId	InternalQuoteId	DataQuote	OrderForm	0.432	0.636	0.636	1	1	0.110624	0.568000
6	customerName	InternalQuoteId	DataQuote	OrderForm	0.588	0.593	0.444	1	0	0.099392	0.541667
7	itemDescription	QuoteVersionNumber	items	OrderForm	0.618	0.545	0.424	1	0	0.095414	0.529000
8	customerId	InternalQuoteId	DataQuote	OrderForm	0.611	0.560	0.400	1	0	0.094102	0.523667
9	itemDescription	InternalQuoteId	items	OrderForm	0.597	0.667	0.267	1	0	0.096954	0.510333
10	customerId	QuoteVersionNumber	DataQuote	OrderForm	0.601	0.571	0.357	1	0	0.090521	0.509667

Source: Author.

Table 11 – API list = (CFSysAPI v1.0→NFSysAPI v1.0) | k = 10 | rank=(major_rank) / Threshold (0.8) / Precision (1.0) Recall - (0.33) - Accuracy (0.8)

Index	OA Out Attr	TA In Attr	OA Out Attr parent	TA In Attr parent	jaro_winkler	sorensen	ratcliff_obershelp	prediction	correct	gini	major_rank
1	quoteId	QuoteVersionNumber	DataQuote	OrderForm	0.838	0.480	0.480	1	1	0.129227	0.838
2	customerName	BackendCustomerNumber	DataQuote	Contact	0.690	0.727	0.667	0	0	0.161058	0.727
3	customerName	QuoteVersionNumber	DataQuote	OrderForm	0.676	0.600	0.467	0	0	0.115007	0.676
4	itemDescription	InternalQuoteId	items	OrderForm	0.597	0.667	0.267	0	0	0.096954	0.667
5	totalPrice	InternalQuoteId	DataQuote	OrderForm	0.556	0.640	0.320	0	0	0.091237	0.640
6	quoteId	InternalQuoteId	DataQuote	OrderForm	0.432	0.636	0.636	0	1	0.110624	0.636
7	customerId	BackendCustomerNumber	DataQuote	Contact	0.628	0.581	0.516	0	1	0.110911	0.628
8	itemDescription	QuoteVersionNumber	items	OrderForm	0.618	0.545	0.424	0	0	0.095414	0.618
9	customerId	InternalQuoteId	DataQuote	OrderForm	0.611	0.560	0.400	0	0	0.094102	0.611
10	customerId	QuoteVersionNumber	DataQuote	OrderForm	0.601	0.571	0.357	0	0	0.090521	0.601

Source: Author.

Table 12 demonstrates the third ranking generated by the tool, with the following parameters: *API List*=[NFSysAPI, CFSysAPI], *k*=10, *rank=avg_rank*. From the developers' verification, it can be seen that the tool had a precision of approximately 0.143, a recall of 1.0 and an accuracy of 0.4, with a *gini* index below 0.24. Finally, the Table 13 displays the fourth ranking generated by the tool with the parameters: *API list*=[NFSysAPI, CFSysAPI], *k*=10, *rank=major_rank*. After the developers indicated the correct API connection points, it was possible to observe that the tool had precision, recall and accuracy equal to 1, with a *gini* index below 0.24. A good degree of homogeneity can also be seen in both tables, 7 of the 10 crossed

pairs are identical (*QuoteNum / quoteNumbers, AliquotaINSS / quoteNumbers, Unit / buid, AliquotaServicos / quoteNumbers, AliquotaIR / quoteNumbers, AliquotaPIS / quoteNumbers, AliquotaISS / quoteNumbers*), which demonstrates a good degree of relationship between the rankings.

Table 12 – API list = (NFSysAPI v1.0→CFSysAPI v1.0) | k=10 | rank=(avg_rank) / Threshold (0.5) / Precision (0.143) - Recall - (1.0) - Accuracy (0.4)

Index	OA Out Attr	TA In Attr	OA Out Attr parent	TA In Attr parent	jaro_winkler	sorensen	ratcliff_obershelp	prediction	correct	gini	avg_rank
1	QuoteNum	quoteNumbers	NFMirrorData	*no parent	0.933	0.800	0.800	1	1	0.238943	0.844333
2	AliquotaINSS	quoteNumbers	Service	*no parent	0.667	0.500	0.500	1	0	0.104988	0.555667
3	Unit	buid	Product	*no parent	0.667	0.500	0.500	1	0	0.104988	0.555667
4	AliquotaServicos	quoteNumbers	Service	*no parent	0.626	0.500	0.500	1	0	0.099097	0.542000
5	DofSequence	quoteNumbers	NFMirrorData	*no parent	0.515	0.609	0.435	1	0	0.091703	0.519667
6	AliquotaIR	quoteNumbers	Service	*no parent	0.639	0.455	0.455	1	0	0.091375	0.516333
7	AliquotaCOFINS	quoteNumbers	Service	*no parent	0.591	0.462	0.462	1	0	0.086241	0.505000
8	AliquotaPIS	quoteNumbers	Service	*no parent	0.624	0.435	0.435	0	0	0.085314	0.498000
9	AliquotaISS	quoteNumbers	Service	*no parent	0.624	0.435	0.435	0	0	0.085314	0.498000
10	AliquotaCSLL	quoteNumbers	Service	*no parent	0.611	0.417	0.417	0	0	0.080122	0.481667

Source: Author.

Table 13 – API list = (NFSysAPI v1.0→CFSysAPI v1.0) | k=10 | rank=major_rank / Threshold (0.8) / Precision (1.0) - Recall - (1.0) - Accuracy (1.0)

Index	OA Out Attr	TA In Attr	OA Out Attr parent	TA In Attr parent	jaro_winkler	sorensen	ratcliff_obershelp	prediction	correct	gini	major_rank
1	QuoteNum	quoteNumbers	NFMirrorData	*no parent	0.933	0.800	0.800	1	1	0.238943	0.933
2	Unit	buid	Product	*no parent	0.667	0.500	0.500	0	0	0.104988	0.667
3	AliquotaINSS	quoteNumbers	Service	*no parent	0.667	0.500	0.500	0	0	0.104988	0.667
4	TributadoNoMunicipio	buid	Header	*no parent	0.650	0.333	0.250	0	0	0.066210	0.650
5	AliquotaIR	quoteNumbers	Service	*no parent	0.639	0.455	0.455	0	0	0.091375	0.639
6	AliquotaServicos	quoteNumbers	Service	*no parent	0.626	0.500	0.500	0	0	0.099097	0.626
7	AliquotaPIS	quoteNumbers	Service	*no parent	0.624	0.435	0.435	0	0	0.085314	0.624
8	AliquotaISS	quoteNumbers	Service	*no parent	0.624	0.435	0.435	0	0	0.085314	0.624
9	MunicioDePrestacaoDoServico	buid	Header	*no parent	0.620	0.194	0.194	0	0	0.051075	0.620
10	Bairro	buid	Tomador	*no parent	0.611	0.400	0.400	0	0	0.077036	0.611

Source: Author.

- Scenario#2:

In the next phase, the same CS developers needed to achieve a new goal defined by DSoIS, i.e., ensuring interoperability of QBSys with the LATSys and DASyS API to meet newly established internal requirements. After an period of analysis and development, the team successfully composed these API, through their documentation. That composition allowed us to conduct the analysis of the results, examining the efficiency of our tool in a different scenario.

Therefore, we ran our tool on the LATSysAPI (Open API version 3.0.1) and DASyS-API (Open API version 3.1) in their presence and no problems were detected with the API contracts. Moreover, we also generated four ordered rankings. Two of them were used to compare LATSysAPI and DASySAPI based on the *avg_rank* and *major_rank* metrics, and other two performed the opposite, i.e., from DASySAPI to LATSysAPI. Those five CS developers also reviewed the ratings, which allowed us to record the evaluation metrics.

Table 14 contains the first ranking generated by our tool with the following parame ters: *API List*=[DASysAPI, LATSysAPI], *k*=10, and *rank=avg_rank*, and we found a precision of 0.6, recall of 1.0, and accuracy of 0.6. The gini index was below 0.34. Table 15 demonstrates the second ranking generated with the following parameters: *API List*=[DASysAPI, LATSysAPI], *k*=10, *rank=major_rank*. After analysis by the developers again, the results indicate a precision of 0.6, recall of 1, and accuracy of 0.6, with a gini index below 0.34 too. It can be seen in these two rankings that the precision, recall and accuracy values were identical. In addition, there was a high similarity between the pairs, with 9 out of 10 cross-pairs present in both rankings, namely: *unitPrice / unitPrice* (2 times), *quoteId / quoteId*, *itemTypeCode / itemType* (3 times) and *serviceLevelCode / serviceCode* (3 times).

Table 14 – API list = (DASys V1.0→LATSysAPI V2.3) | k = 10 | rank=avg_rank / Threshold (0.5) / Precision (0.6) - Recall (1.0) - Accuracy (0.6)

Index	OA Out Attr	TA In Attr	OA Out Attr Parent	TA In Attr Parent	jaro_winkler	sorensen	ratcliff_obershelp	prediction	correct	gini	avg_rank
1	unitPrice	unitPrice	associatedTiedItems	mirrorItems	1.000	1.000	1.000	1	1	0.333333	1.000000
2	unitPrice	unitPrice	Item.Domain.OpenBasketLite.Product	mirrorItems	1.000	1.000	1.000	1	1	0.333333	1.000000
3	quoteId	quoteId	smartPrice	CreateMirrorRequestDto	1.000	1.000	1.000	1	1	0.333333	1.000000
4	itemTypeCode	itemType	skus	mirrorItems	0.933	0.800	0.800	1	1	0.238943	0.844333
5	itemTypeCode	itemType	Item.Domain.OpenBasketLite.Product	mirrorItems	0.933	0.800	0.800	1	1	0.238943	0.844333
6	itemTypeCode	itemType	associatedTiedItems	mirrorItems	0.933	0.800	0.800	1	1	0.238943	0.844333
7	serviceLevelCode	serviceCode	aposSkus	mirrorItems	0.901	0.815	0.815	1	0	0.237806	0.843667
8	serviceLevelCode	serviceCode	contractSummary	mirrorItems	0.901	0.815	0.815	1	0	0.237806	0.843667
9	serviceLevelCode	serviceCode	skus	mirrorItems	0.901	0.815	0.815	1	0	0.237806	0.843667
10	expirationDate	creationDate	serviceTags	CreateMirrorRequestDto	0.840	0.846	0.769	1	0	0.223631	0.818333

Source: Author.

Table 15 – API list = (DASys V1.0→LATSysAPI V2.3) | k = 10 | rank=major_rank / Threshold (0.5) / Precision (0.6) - Recall (1.0) - Accuracy (0.6)

Index	OA Out Attr	TA In Attr	OA Out Attr Parent	TA In Attr Parent	jaro_winkler	sorensen	ratcliff_obershelp	prediction	correct	gini	major_rank
1	unitPrice	unitPrice	Item.Domain.OpenBasketLite.Product	mirrorItems	1.000	1.000	1.000	1	1	0.333333	1.000
2	quoteId	quoteId	smartPrice	CreateMirrorRequestDto	1.000	1.000	1.000	1	1	0.333333	1.000
3	unitPrice	unitPrice	associatedTiedItems	mirrorItems	1.000	1.000	1.000	1	1	0.333333	1.000
4	itemTypeCode	itemType	Item.Domain.OpenBasketLite.Product	mirrorItems	0.933	0.800	0.800	1	1	0.238943	0.933
5	itemTypeCode	itemType	skus	mirrorItems	0.933	0.800	0.800	1	1	0.238943	0.933
6	itemTypeCode	itemType	associatedTiedItems	mirrorItems	0.933	0.800	0.800	1	1	0.238943	0.933
7	serviceLevelCode	serviceCode	skus	mirrorItems	0.901	0.815	0.815	1	0	0.237806	0.901
8	serviceLevelCode	serviceCode	aposSkus	mirrorItems	0.901	0.815	0.815	1	0	0.237806	0.901
9	serviceLevelCode	serviceCode	contractSummary	mirrorItems	0.901	0.815	0.815	1	0	0.237806	0.901
10	serviceType	serviceCode	skus	mirrorItems	0.891	0.727	0.727	1	0	0.205660	0.891

Source: Author.

Table 16 presents the third ranking generated by our tool, configured with the following parameters: *API List*=[LATSysAPI, DASysAPI], *k*=10, *rank=avg_rank*. Based on verification from the developers, the tool achieved a precision of approximately 0.3, a recall of 1.0 and an accuracy of 0.3, with a gini index below 0.34. Finally, the Table 17 displays the fourth ranking generated by the tool with the parameters: *API list*=[LATSysAPI, DASysAPI], *k*=10, *rank=major_rank*. After developers re-identified the correct API connection points, it can be seen that the tool had a precision of approximately 0.25, recall of 1.0, and accuracy of 0.4, with

a gini index below 0.34. We also observed a good degree of homogeneity in 7 out of 10 matched pairs (*quoteId / quoteId*, *serviceType / severity*, *serviceTag / severity*, *quoteVersion / quoteId*, *isVariablePriced / isValidationFromServiceCall* two times, and *moduleId / quoteId*), indicating a strong alignment betwwen the third and fourth rankings.

Table 16 – API list = (LATSysAPI V2.3→DASys V1.0) | k = 10 | rank=avg_rank / Threshold =0.5 / Precision (0.3) - Recall (1.0) - Accuracy (0.3)

Index	OA Out Attr	TA In Attr	OA Out Attr Parent	TA In Attr Parent	jaro_winkler	sorensen	ratcliff_obershelp	prediction	correct	gini	avg_rank
1	quoteId	quoteId	smartPrice	*no parent	1.000	1.000	1.000	1	1	0.333333	1.000000
2	serviceType	severity	skus	*no parent	0.861	0.842	0.632	1	0	0.205523	0.778333
3	serviceTag	severity	associatedTiedItems	*no parent	0.810	0.778	0.556	1	0	0.174502	0.714667
4	serviceTag	severity	Item.Domain.OpenBasketLite.Product	*no parent	0.810	0.778	0.556	1	0	0.174502	0.714667
5	quoteVersion	quoteId	smartPrice	*no parent	0.871	0.632	0.632	1	1	0.173054	0.711667
6	isValid	validate	validationResult	*no parent	0.646	0.667	0.667	1	1	0.145233	0.660000
7	isVariablePriced	isValidationFromServiceCall	Item.Domain.OpenBasketLite.Product	*no parent	0.822	0.651	0.465	1	0	0.146190	0.646000
8	isVariablePriced	isValidationFromServiceCall	skus	*no parent	0.822	0.651	0.465	1	0	0.146190	0.646000
9	isVariablePriced	isValidationFromServiceCall	associatedTiedItems	*no parent	0.822	0.651	0.465	1	0	0.146190	0.646000
10	moduleId	quoteId	parentInfo	*no parent	0.713	0.667	0.533	1	0	0.137483	0.637667

Source: Author.

Table 17 – API list = (LATSysAPI V2.3→DASys V1.0) | k = 10 | rank=major_rank / Threshold =0.8 / Precision (0.25) - Recall (1.0) - Accuracy (0.4)

Index	OA Out Attr	TA In Attr	OA Out Attr Parent	TA In Attr Parent	jaro_winkler	sorensen	ratcliff_obershelp	prediction	correct	gini	major_rank
1	quoteId	quoteId	smartPrice	*no parent	1.000	1.000	1.000	1	1	0.333333	1.000
2	quoteVersion	quoteId	smartPrice	*no parent	0.871	0.632	0.632	1	1	0.173054	0.871
3	serviceType	severity	skus	*no parent	0.861	0.842	0.632	1	0	0.205523	0.861
4	isValid	isValidationFromServiceCall	validationResult	*no parent	0.852	0.412	0.412	1	0	0.118377	0.852
5	isVariable	isValidationFromServiceCall	skus	*no parent	0.824	0.486	0.432	1	0	0.122422	0.824
6	isVariablePriced	isValidationFromServiceCall	skus	*no parent	0.822	0.651	0.465	1	0	0.146190	0.822
7	isVariablePriced	isValidationFromServiceCall	Item.Domain.OpenBasketLite.Product	*no parent	0.822	0.651	0.465	1	0	0.146190	0.822
8	serviceTag	severity	Item.Domain.OpenBasketLite.Product	*no parent	0.810	0.778	0.556	1	0	0.174502	0.810
9	serviceLifeDurationDays	isValidationFromServiceCall	aposSkus	*no parent	0.668	0.760	0.320	0	0	0.125136	0.760
10	moduleId	quoteId	parentInfo	*no parent	0.713	0.667	0.533	0	0	0.137483	0.713

Source: Author.

5.3.2.2 Interview with Project Developers

At the end of the developers' interactions with the ordered rankings, questions were asked to assess the efficiency of the method in identifying API connection points and in assessing the viability of the tool in other projects. Among the available qualitative methods, which are questionnaires, focus groups, and interviews, we chose the latter, more precisely semi-structured interviews, a format that allows the researcher to prepare a list of questions related to one or more topics, with flexibility in conducting the interview (KITCHENHAM *et al.*, 2015). Thus, during the interview, we explained in detail to the developers all method steps, from the inputs (*API list*, *k*, *rank*) to the outputs (*top k* API connection points ordered by ranks). Then, specific questions (Q) were asked in an interactive and indirect way (BORGES *et al.*, 2024):

– Q1: Did the tool accurately identify connection points for API based on the rankings? If so, what percentage of accuracy would you rate?

- Q2: Did the tool identify any connection points you had not manually noticed?
- Q3: Would you use the tool as a CS API connection starting point?
- Q4: Would you suggest this tool to other development teams within a company working on CS interoperability links tasks?
- Q5: What enhancements would make the tool better suited to your needs?

- Scenario#1:

The interview lasted about 57 minutes and 4 seconds, it occurred on December 7, 2022 via Google Meet platform (BORGES *et al.*, 2024). It was observed through the developers' responses to each question that the tool in general is not only very efficient and promising, but also helps with new API composition tasks in other projects, identifying API connection points among the involved API. For Q1, everyone agreed that the tool was able to provide API connection points, with an arithmetic mean of 100% of hits. Although the agreement was 100%, it was noticed by the developers that sometimes the tool recommends correct API connection points in one ranking and not in another, and vice versa. One of the programmers, nicknamed A, reported the following: "*for me the tool indicated well some API connection points in a certain ranking, but not in another. I believe that these indications must be cohesive and consider only one of the rankings or one mixture of them to not have this kind of divergence*". They were told that one of the possible reasons why there were differences in the indications was due to the setting of the thresholds used. Based on these findings, we will be researching new ways to implement variable thresholds to precisely improve API connection points.

Regarding Q2, developers reported that the tool correctly identified the points already known, with the addition of the point from *QuoteNum* (NFSySAPI) to the *quoteNumber* (CFSySAPI), which they had not noticed. Indeed, for this reason, they liked the result very much, but reported that there are different scenarios, even if they are few, in which endpoint attributes have similar or equal meanings but are syntactically different, and that the tool did not achieve or improve the results of such scenarios. This can be evident in developer B's response, who reported that: "*of course the tool has merit, but we want to see cases of different attributes with the same meaning being identified and such a feature added to the tool*". It was explained to them that the focus of the tool was syntactic because the results, in addition to being processed quickly, could cover more scenarios than the semantic way. However, this issue will be considered for future work.

For Q3, developers unanimously considered the tool not only as a good start but also want to use it in the next connection project. This can be seen in the consideration of developer A who reported that: "*the team can already use the tool in the next new API called DASys, which will replace the current form of sales communication*". Another developer, C, corroborated with A and even said that: "*the tool could avoid the time spent studying this new API, as the analysis of its description could be minimized, which would allow them to focus on attributes and endpoints at first found by the tool*".

For Q4, they also categorically reported that they would easily recommend the tool to other developers, and that different groups could give different feedback and gradually improve the tool, which would also serve them, the ones most interested in these connection tasks of CS. Finally, for Q5, they said they would like to see the addition of functionality that involves extracting the meaning of endpoint attributes. In addition, more considerations could be sent to me by email when they are using and testing the tool in their activities.

- Scenario#2:

The interview was performed on June 10, 2023, also via Google Meet platform, and lasted 42 minutes (BORGES *et al.*, 2024). For Q1, developers estimated that the tool accurately matched about 90% of API connection points. This high success rate reinforced their confidence in the tool's ability to effectively capture syntactic connections. However, they pointed out one limitation: attributes that are syntactically distinct but semantically similar, such as "item" and "product", were not recognized as connection points. That limitation highlighted the need for future improvements so that the tool can include semantic matching capabilities and expand its reach in more contexts.

For Q2, developers reported that while the tool identified most of the connection points already mapped to the LATSys and DASys API, this time it did not reveal additional points that had not yet been identified. Regarding Q3, developers stated that they would use the tool in their work activities. Since the last interview, they have noted that the tool provides a solid basis for the initial identification process, making this task less time-consuming and more comprehensive. In addition, they highlighted that the tool reduces manual work and minimizes the risk of oversight errors during the early stages of the analysis.

For Q4, developers would recommend the use of this tool to other development teams working on CS interoperability links, as they see its value as an essential support for

identifying API connection points. They also reported that while the tool does not replace manual verification, it supports the process efficiently, adding reliability and speed to the analysis, mainly by cross-referencing potential connection points, making the process not only faster but also more accurate.

Finally, for Q5, developers unanimously recommended integrating a semantic similarity layer into the tool, aiming to complement the syntactic similarity results. According to them, that integration would allow overcoming limitations in scenarios where syntactically distinct but semantically equivalent terms, such as “item” and “product”. Such integration would also provide more robust support for their analytical decisions, offering a more comprehensive view of the API connection points.

5.3.3 Syntactic-Semantic Analysis Perspective

In this subsection, we report the evaluation of our tool applied to the DSoIS scenario#3, focusing on syntactic-semantic analysis. That analysis covers both the API attributes and their textual descriptions. In this context, the interoperability links between two CS involved in the scenario#3 were previously identified by the QB CS developers, which allowed a comparative analysis of the results generated by the tool. Our goal was to investigate whether it would be possible to replicate these results using the tool configured not only with syntactic algorithms, but also with deep learning algorithms for textual comparison, such as BERT, RoBERTa, T5 and Bart.

To this end, we conducted new semi-structured interviews with the developers, whose feedback was fundamental to our findings. The following subsections detail the results of applying the tool on three CS API from the SoIS (5.3.3.1) and the interviews conducted with five CS developers about the tool’s results (5.3.3.2).

5.3.3.1 Results

- Scenario#3:

Subsequently, CS developers were tasked with meeting a new DSoIS objective to meet newly established internal requirements (Scenario#3): obtaining order quote data from DQSys and using LATSys to record the fiscal data of items and products in these quotes. To achieve that objective, they conducted a detailed study of the API of the CS involved and

ensured interoperability by composing three API: two from DQSys and one from LATSys. That allowed not only the analysis of the results, but also their evaluation through our tool, which now considers both the syntactic attributes and the descriptions of these attributes from a semantic perspective.

Scenario#3 was subdivided into two sub scenarios. In the first sub scenario (called Scenario#3.1), the objective was to generate global quotations and obtain the associated data through the QUSysAPI API, belonging to CS DQSys. Then, all items and products from these quotations were analyzed, serving as a basis for the next step. In the second sub scenario (Scenario#3.2), the details of the extracted products were processed in the FDSysAPI API, from CS SYSys, with the purpose of recording fiscal data for these products, facilitating the generation of tax documents. The results were stored in a database, facilitating future queries based on quotations already made by them.

- Scenario#3.1:

In the presence of CS developers again, we ran the tool on the following DQSyS API: QUSysAPI (OpenAPI version 3.0.1) and ITSysAPI (OpenAPI version 3.1). Among the three API contracts analyzed (see section 4.2.1), the first two – compatibility with OAS versions and file extension – were met. However, the third contract, referring to the validation of circular references to avoid infinite loops, presented problems identified during execution. The API schemas, together with their respective attributes with circular references, are detailed in Table 18. These problems were solved through internal adaptations, preserving the structural and functional integrity of the API.

After the adaptations made to the API, the tool stopped presenting errors, allowing the desired results to be obtained: the syntactic-semantic rankings. To facilitate understanding of the process of generating these rankings, the syntactic ranking will be presented first, followed by the semantic ranking, and finally the combination of both, which results in the syntactic-semantic ranking.

Since the developers already had prior knowledge about the interoperability links between the API, they requested our analysis of the methods, attributes, and endpoints that presented these known connections. Among the scenarios indicated, they highlighted that the response from the QUSysAPI "*/v3/quotes/quoteId*" endpoint was composed with the ITSysAPI "*/v1/items/itemID*" endpoint. Based on this indication, we filtered the relevant endpoints and

Table 18 – Scenario#3.1 - Schemas and Circular references of the QUSysAPI and ITSysAPI API

API	Schema	Circular reference
QUSysAPI	Quote.Domain.ItemPricingSummary	associatedItemPriceSummary
	Quote.Domain.Product	associatedItems
	PostQuote.PostProductModel	associatedTiedItems
	Quote.Domain.AssociatedItemProduct	associatedTiedItems
	PostQuote.PostProductModel	associatedItems
	AssociatedItemProduct	associatedTiedItems
	PostQuote.Hierarchy	child
	UpsertPriceAdjustment.Hierarchy	child
	Quote.Domain.Hierarchy	child
	UpsertSkuPriceAdjustments.Hierarchy	child
	Quote.Domain.Node	child
	PostQuote.PostItemPricingSummaryModel	child
	PostQuote.UpsertQuoteChildConnectionModel	childItems
	Quote.Domain.ChildConnection	childItems
	Quote.Domain.QuoteItem	childItems
	PostQuote.PostChildConnectionModel	childItems
ITSysAPI	Item	descendants
	ItemModel	descendants
	SelectionsItemResponseModel	descendants
	Product	associatedItems
	Item.Attribute	extendedProperties

Source: Author.

generated two syntactic-semantic rankings in the context of the communication between the QUSysAPI and the ITSysAPI: one considering the *avg_rank* and the other the *major_rank*.

The first ranking was configured with the following parameters: *API List* = [QUSysAPI, ITSysAPI], k = 10 and *rank* = *avg_rank*. The process of generating the syntactic-semantic ranking was carried out in detailed steps. First, the syntactic ranking was generated based on the adjusted algorithms (*jaro_winkler*, *sorensen* and *ratcliff_obershelp*), the results of which can be seen in Table 19. Then, based on the tuples of this syntactic ranking, the semantic rankings were calculated based on the attribute descriptions, using the BERT, RoBERTa, T5 and Bart methods, the results of which are available in Table 20. Finally, the syntactic-semantic predictions were obtained based on Equations 4.6 and 4.7. The weights defined for these predictions were 0.3 for w_1 (syntactic) and 0.7 for w_2 (semantic), justified by the greater relevance of semantic analysis in understanding the context, as highlighted by Jackson e Moulinier (2007). We also adjusted the thresholds, setting 0.7 for *avg_rank* and 0.8 for *major_rank*. The prediction results are presented in Table 21, organized based on the attribute **calc_syn_sem*, an auto calculated auxiliary field used to facilitate the indications of the *prediction_syn_sem* field. After analysis by the developers, it was found that the tool presented a precision of 0.5, a recall of 0.5 and an accuracy of 0.8. Furthermore, the *gini* index below 0.17 indicated a relative equality in the distribution of the

identified connection points.

Table 19 – API list = (QUSysAPI→ITSysAPI) | k = 10 | rank=avg_rank (avg_syn_rank)

Index	OA Out Attr	TA In Attr	OA Out Attr parent	TA In Attr parent	jaro_winkler	sorensen	ratcliff_obershelp	avg_syn_rank
1	itemUrl	itemId	items	*no parent	0.848	0.615	0.615	0.693
2	parentId	descendantId	items	*no parent	0.614	0.600	0.500	0.571
3	linkId	itemId	Quote.Domain.Quote	*no parent	0.667	0.500	0.500	0.556
4	parentId	itemId	items	*no parent	0.625	0.571	0.429	0.542
5	openBasketItemId	itemId	items	*no parent	0.486	0.545	0.545	0.525
6	id	itemId	Quote.Domain.Quote	*no parent	0.556	0.500	0.500	0.519
7	id	itemId	items	*no parent	0.556	0.500	0.500	0.519
8	catalogId	descendantId	items	*no parent	0.509	0.476	0.476	0.487
9	quotedDate	descendantId	Quote.Domain.Quote	*no parent	0.533	0.545	0.364	0.481
10	clearPriceEnabled	descendantId	items	*no parent	0.522	0.483	0.414	0.473

Source: Author.

Table 20 – API list = (QUSysAPI→ITSysAPI) | k = 10 | rank=avg_rank (avg_sem_rank)

Index	OA Out Attr	OA Out Attr descr	TA In Attr	TA In Attr descr	bert	roberta	t5	bart	avg_sem_rank
7	id	The unique ID for the item, used to reference or retrieve a specific item	itemId	The unique ID for the item	0.810	0.869	0.796	0.821	0.824
2	parentId	The ID of the item's parent, establishing a hierarchical relationship with an item	descendantId	(Optional) The unique ID for a related descendant item	0.784	0.797	0.747	0.755	0.771
3	linkId	The unique link ID associated with the quote	itemId	The unique ID for the item	0.869	0.529	0.824	0.770	0.748
6	id	The unique ID for the quote, used to reference or retrieve a specific quote	itemId	The unique ID for the item	0.783	0.601	0.741	0.764	0.722
4	parentId	The ID of the item's parent, establishing a hierarchical relationship with an item	itemId	The unique ID for the item	0.759	0.561	0.665	0.716	0.675
8	catalogId	The unique ID for the catalog	descendantId	(Optional) The unique ID for a related descendant item	0.733	0.570	0.622	0.725	0.662
5	openBasketItemId	The open basket item ID associated with an item, used to track or manage items in an open basket context	itemId	The unique ID for the item	0.704	0.607	0.654	0.599	0.641
1	itemUrl	The URL associated with the item, typically used to retrieve or access item details	itemId	The unique ID for the item	0.702	0.488	0.729	0.643	0.640
10	clearPriceEnabled	Indicates whether clear pricing is enabled for an item	descendantId	(Optional) The unique ID for a related descendant item	0.751	0.143	0.603	0.693	0.547
9	quotedDate	The date when the quote was created or issued	descendantId	(Optional) The unique ID for a related descendant item	0.680	0.078	0.505	0.630	0.473

Source: Author.

Table 21 – API list = (QUSysAPI→ITSysAPI) | k = 10 | rank=avg_rank / Threshold=0.7 / Precision (0.5) - Recall (0.5) - Accuracy (0.8)

Index	OA Out Attr	OA Out Attr descr	TA In Attr	TA In Attr descr	avg_rank	avg_sem_rank	gini	*calc_syn_sem	prediction_syn_sem	correct_syn_sem
7	id	The unique ID for the item, used to reference or retrieve a specific item	itemId	The unique ID for the item	0.519	0.824	0.090	0.732	1	1
2	parentId	The ID of the item's parent, establishing a hierarchical relationship with an item	descendantId	(Optional) The unique ID for a related descendant item	0.571	0.771	0.110	0.711	1	0
3	linkId	The unique link ID associated with the quote	itemId	The unique ID for the item	0.556	0.748	0.105	0.690	0	0
6	id	The unique ID for the quote, used to reference or retrieve a specific quote	itemId	The unique ID for the item	0.519	0.722	0.090	0.661	0	0
1	itemUrl	The URL associated with the item, typically used to retrieve or access item details	itemId	The unique ID for the item	0.693	0.640	0.164	0.656	0	0
4	parentId	The ID of the item's parent, establishing a hierarchical relationship with an item	itemId	The unique ID for the item	0.542	0.675	0.100	0.635	0	1
8	catalogId	The unique ID for the catalog	descendantId	(Optional) The unique ID for a related descendant item	0.487	0.662	0.079	0.609	0	0
5	openBasketItemId	The open basket item ID associated with an item, used to track or manage items in an open basket context	itemId	The unique ID for the item	0.525	0.641	0.092	0.606	0	0
10	clearPriceEnabled	Indicates whether clear pricing is enabled for an item	descendantId	(Optional) The unique ID for a related descendant item	0.473	0.547	0.075	0.525	0	0
9	quotedDate	The date when the quote was created or issued	descendantId	(Optional) The unique ID for a related descendant item	0.481	0.473	0.079	0.475	0	0

Source: Author.

Similarly, the second syntactic-semantic ranking was generated with the following parameters: *API List* = [QUSysAPI, ITSysAPI], k = 10, *rank* = *major_rank*, applying the same endpoint filters indicated by the QB developers, as described previously. Thus, the syntactic ranking is presented in Table 22, while the semantic ranking is in Table 23. Finally, the syntactic-

semantic ranking, ordered by the auxiliary field *calc_syn_sem*, can be viewed in Table 24. After a new analysis performed by the developers, the following results were obtained: precision of 0.33, recall of 0.5 and accuracy of 0.7, with a *gini* index also lower than 0.17. It can still be noticed that in both tables (21 and 23) 6 of the 10 crossed pairs are identical (*itemUrl / itemId, linkId / itemId, parentId / itemId, parentId / descendantId, and id / itemId* twice), which characterizes also a good degree of homogeneity also between these two rankings.

Table 22 – API list = (QUSysAPI→ITSysAPI) | k = 10 | rank=major_rank (major_syn_rank)

Index	OA Out Attr	TA In Attr	OA Out Attr parent	TA In Attr parent	jaro_winkler	sorensen	ratcliff_obershelp	major_rank
1	itemUrl	itemId	items	*no parent	0.848	0.615	0.615	0.848
2	linkId	itemId	Quote.Domain.Quote	*no parent	0.667	0.500	0.500	0.667
3	parentId	itemId	items	*no parent	0.625	0.571	0.429	0.625
4	parentId	descendantId	items	*no parent	0.614	0.600	0.500	0.614
5	catalogId	itemId	items	*no parent	0.611	0.400	0.400	0.611
6	quotedDate	itemId	Quote.Domain.Quote	*no parent	0.600	0.375	0.375	0.600
7	id	itemId	Quote.Domain.Quote	*no parent	0.556	0.500	0.500	0.556
8	id	itemId	items	*no parent	0.556	0.500	0.500	0.556
9	$schema	descendantId	Quote.Domain.Quote	*no parent	0.552	0.421	0.421	0.552
10	priceShippingViewUrl	itemId	Quote.Domain.Quote	*no parent	0.550	0.231	0.231	0.550

Source: Author.

Table 23 – API list = (QUSysAPI→ITSysAPI) | k = 10 | rank=major_rank (major_sem_rank)

Index	OA Out Attr	OA Out Attr descr	TA In Attr	TA In Attr descr	bert	roberta	t5	bart	major_sem_rank
8	id	Identifies the item ID	itemId	Identifies the ID of the item.	0.890	0.917	0.889	0.893	0.917
5	catalogId	Represents the catalog ID of the item.	itemId	Identifies the ID of the item.	0.907	0.759	0.883	0.817	0.907
3	parentId	Indicates the parent ID of the item Id	itemId	Identifies the ID of the item.	0.846	0.628	0.832	0.767	0.846
7	id	Identifies the quote ID	itemId	Identifies the ID of the item.	0.785	0.562	0.843	0.821	0.843
4	parentId	Indicates the parent ID of the item Id	descendantId	Represents the descendant ID.	0.813	0.617	0.738	0.665	0.813
1	itemUrl	Specifies the URL of a specific item.	itemId	Identifies the ID of the item.	0.787	0.459	0.810	0.741	0.810
2	linkId	Identifies the quote link ID	itemId	Identifies the ID of the item.	0.788	0.489	0.809	0.802	0.809
9	$schema	Specifies the schema of the quote.	descendantId	Represents the descendant ID.	0.747	0.266	0.699	0.658	0.747
6	quotedDate	The date of the quotation.	itemId	Identifies the ID of the item.	0.734	0.126	0.649	0.655	0.734
10	priceShippingViewUrl	URL to view the shipping price.	itemId	Identifies the ID of the item.	0.726	0.144	0.528	0.628	0.726

Source: Author.

Table 24 – API list = (QUSysAPI→ITSysAPI) | k = 10 | rank=major_rank / Threshold=0.8 / Precision (0.33) - Recall (0.5) - Accuracy (0.7)

Index	OA Out Attr	OA Out Attr descr	TA In Attr	TA In Attr descr	major_rank	major_sem_rank	gini	*calc_syn_sem	prediction_syn_sem	correct_syn_sem
1	itemUrl	Specifies the URL of a specific item.	itemId	Identifies the ID of the item.	0.848	0.810	0.164	0.821	1	0
5	catalogId	Represents the catalog ID of the item.	itemId	Identifies the ID of the item.	0.611	0.907	0.077	0.818	1	0
8	id	Identifies the item ID	itemId	Identifies the ID of the item.	0.556	0.917	0.090	0.809	1	1
3	parentId	Indicates the parent ID of the item Id	itemId	Identifies the ID of the item.	0.625	0.846	0.100	0.780	0	1
2	linkId	Identifies the quote link ID	itemId	Identifies the ID of the item.	0.667	0.809	0.105	0.766	0	0
7	id	Identifies the quote ID	itemId	Identifies the ID of the item.	0.556	0.843	0.090	0.757	0	0
4	parentId	Indicates the parent ID of the item Id	descendantId	Represents the descendant ID.	0.614	0.813	0.110	0.753	0	0
6	quotedDate	The date of the quotation.	itemId	Identifies the ID of the item.	0.600	0.734	0.071	0.694	0	0
9	$schema	Specifies the schema of the quote.	descendantId	Represents the descendant ID.	0.552	0.747	0.073	0.688	0	0
10	priceShippingViewUrl	URL to view the shipping price.	itemId	Identifies the ID of the item.	0.550	0.726	0.045	0.673	0	0

Source: Author.

- Scenario#3.2:

With the participation of CS developers in the last stage of scenario 3, we ran the

tool on the ITSysAPI (OpenAPI version 3.0.1) and FDSysAPI (OpenAPI version 3.0.1) API. Among the three API contracts analyzed, as in sub scenario#3.1, only the third contract, referring to the validation of circular references to avoid infinite loops, also presented problems. The API schema, along with their respective attributes with circular references, are detailed in Table 25. Such problems were also overcome through internal adaptations, maintaining the functional structure of the API.

Table 25 – Scenario#3.2 - Schemas and Circular references of the ITSysAPI and FDSysAPI API

API	**Schema**	**Circular reference**
ITSysAPI	Item	descendants
	ItemModel	descendants
	SelectionsItemResponseModel	descendants
	Product	associatedItems
	Item.Attribute	extendedProperties
FDSysAPI	Type	declaringType
	Type	customAttributes
	Type	attributeType
	Type	reflectedType
	Type	underlyingSystemType
	Type	genericTypeArguments
	Type	baseType
	Type	declaredNestedTypes
	Type	exportedTypes
	Assembly	customAttributes
	ConstructorInfo	customAttributes
	EvenInfo	customAttributes
	FieldInfo	customAttributes
	MemberInfo	customAttributes
	MethodBase	customAttributes
	MethodInfo	customAttributes
	Module	customAttributes

Source: Author.

After the API adaptations, we were able to run the tool without errors, which allowed us to obtain the results and extract the metrics. As in sub scenario#3.1, the developers also asked us to generate the syntactic-semantic rankings at the API connection points they already knew, where the response from the "*/v1/items/itemID*" endpoint of the ITSysAPI was composed with the "*/api/v1/Mirrors*" endpoint of the FDSysAPI, which also caused us to generate 2 syntactic-semantic rankings, one considering the *avg_rank* and the other the *major_rank*.

The first ranking was configured with the following parameters: *API List* = [ITSyS-API, FDSySAPI], k = 10 and *rank* = *avg_rank*. The syntactic ranking is available in Table 26, while the semantic ranking can be viewed in Table 27. Finally, the syntactic-semantic ranking, ordered by the auxiliary field **calc_syn_sem*, is presented in Table 28. After analysis carried out

by the developers, it was found that the tool obtained a precision of 0.4, a recall of 1.0 and an accuracy of 0.7, with a *gini* index also lower than 0.22, which indicates a certain equality in the distribution of the connection points.

Table 26 – API list = (ITSysAPI →FDSysAPI) | k = 10 | rank=avg_rank(avg_syn_rank)

Index	OA Out Attr	TA In Attr	OA Out Attr parent	TA In Attr parent	jaro_winkler	sorensen	ratcliff_obershelp	avg_rank
1	quantity	quantity	ItemModel	mirrorItems	1.000	1.000	1.000	1.000
2	unitQuantity	quantity	ItemModel	mirrorItems	0.764	0.800	0.800	0.788
3	itemUrl	itemNumber	ItemModel	mirrorItems	0.891	0.706	0.706	0.768
4	priceItems	pricesIncludeIcms	ItemModel	mirrorItems	0.841	0.667	0.667	0.725
5	priceItems	unitPrice	ItemModel	mirrorItems	0.683	0.737	0.526	0.649
6	itemSkus	itemNumber	ItemModel	mirrorItems	0.825	0.556	0.556	0.646
7	type	itemType	ItemModel	mirrorItems	0.583	0.667	0.667	0.639
8	description	creationDate	ItemModel	MirrorRequest	0.645	0.696	0.522	0.621
9	priceItemsUrl	unitPrice	properties	mirrorItems	0.629	0.727	0.455	0.604
10	openBasketItemId	operationType	ItemModel	MirrorRequest	0.640	0.621	0.552	0.604

Source: Author.

Table 27 – API list = (ITSysAPI →FDSysAPI) | k = 10 | rank=avg_rank(avg_sem_rank)

Index	OA Out Attr	OA Out Attr descr	TA In Attr	TA In Attr descr	bert	roberta	t5	bart	avg_rank(avg_sem_rank)
7	type	The type of the item	itemType	The type of the item in the mirror	0.850	0.636	0.823	0.889	0.800
5	priceItems	The price of the items	unitPrice	The unit price of the item in the mirror	0.813	0.663	0.765	0.840	0.770
3	itemUrl	The URL associated with the item	itemNumber	The unique number associated with the mirror item	0.809	0.422	0.847	0.820	0.725
2	unitQuantity	The quantity per unit of the item	quantity	The quantity of items in the mirror	0.781	0.423	0.698	0.810	0.678
1	quantity	The quantity of items	quantity	The quantity of items in the mirror	0.838	0.555	0.441	0.858	0.673
9	priceItemsUrl	The URL to retrieve pricing of the items	unitPrice	The unit price of the item in the mirror	0.711	0.485	0.654	0.720	0.642
4	priceItems	The price of the items	pricesIncludeIcms	Indicates whether the item's prices include ICMS	0.790	0.436	0.565	0.687	0.619
6	itemSkus	A list of SKUs (Stock Keeping Units) associated with the item	itemNumber	The unique number associated with the mirror item	0.740	0.275	0.652	0.715	0.596
8	description	The description of the item	creationDate	The date the mirror is created	0.701	0.273	0.554	0.817	0.586
10	openBasketItemId	The open basket item ID associated with an item	operationType	The type of operation associated with the mirror	0.714	0.061	0.608	0.657	0.510

Source: Author.

Table 28 – API list = (ITSysAPI →FDSysAPI) | k = 10 | rank=avg_rank / Threshold=0.7 / Precision (0.4) - Recall (1) - Accuracy (0.7)

Index	OA Out Attr	OA Out Attr descr	TA In Attr	TA In Attr descr	avg_rank	avg_sem_rank	gini	*calc_syn_sem	prediction_syn_sem	correct_syn_sem
1	quantity	The quantity of items	quantity	The quantity of items in the mirror	1.000	0.673	0.210	0.771	1	1
7	type	The type of the item	itemType	The type of the item in the mirror	0.639	0.800	0.137	0.752	1	1
3	itemUrl	The URL associated with the item	itemNumber	The unique number associated with the mirror item	0.768	0.725	0.199	0.738	1	0
5	priceItems	The price of the items	unitPrice	The unit price of the item in the mirror	0.649	0.770	0.143	0.734	1	0
2	unitQuantity	The quantity per unit of the item	quantity	The quantity of items in the mirror	0.788	0.678	0.207	0.711	1	0
4	priceItems	The price of the items	pricesIncludeIcms	Indicates whether the item's prices include ICMS	0.725	0.619	0.177	0.651	0	0
9	priceItemsUrl	The URL to retrieve pricing of the items	unitPrice	The unit price of the item in the mirror	0.604	0.642	0.126	0.631	0	0
6	itemSkus	A list of SKUs (Stock Keeping Units) associated with the item	itemNumber	The unique number associated with the mirror item	0.646	0.596	0.144	0.611	0	0
8	description	The description of the item	creationDate	The date the mirror is created	0.621	0.586	0.130	0.596	0	0
10	openBasketItemId	The open basket item ID associated with an item	operationType	The type of operation associated with the mirror	0.604	0.510	0.122	0.538	0	0

Source: Author.

The second ranking was generated with the following parameters *API List* = [ITSyS-API, FDSySAPI], *k* = 10, *rank* = *major_rank*, applying also the same endpoint filters indicated by the QB developers. The syntactic ranking can be seen in Table 29, the semantic ranking can be seen in Table 30, and the syntactic-semantic ranking, ordered by the auxiliary field **calc_syn_sem*, can be seen in Table 31. After analysis by the developers again, we obtained a precision of 0.2, recall of 1, and accuracy of 0.6, with a gini index below 0.22. It can also be noticed that in both tables 6 of the 10 crossed pairs are identical (*quantity / quantity, itemUrl*

/ itemNumber, priceItems / pricesIncludeIcms, itemSkus / itemNumber, unitQuantity / quantity, priceItems / unitPrice), which characterizes a good degree of homogeneity between the two rankings.

Table 29 – API list = (ITSysAPI →FDSysAPI) | k = 10 | rank=major_rank(major_syn_rank)

Index	OA Out Attr	TA In Attr	OA Out Attr parent	TA In Attr parent	jaro_winkler	sorensen	ratcliff_obershelp	major_rank
1	quantity	quantity	ItemModel	mirrorItems	1.000	1.000	1.000	1.000
2	itemUrl	itemNumber	ItemModel	mirrorItems	0.891	0.706	0.706	0.891
3	priceItemsUrl	pricesIncludeIcms	properties	mirrorItems	0.844	0.733	0.600	0.844
4	priceItems	pricesIncludeIcms	ItemModel	mirrorItems	0.841	0.667	0.667	0.841
5	itemSkus	itemNumber	ItemModel	mirrorItems	0.825	0.556	0.556	0.825
6	unitQuantity	quantity	ItemModel	mirrorItems	0.764	0.800	0.800	0.800
7	description	unitPrice	ItemModel	mirrorItems	0.571	0.800	0.200	0.800
8	configuredSelectionsUrl	issuerLocationCode	properties	MirrorRequest	0.673	0.780	0.244	0.780
9	description	siteCode	ItemModel	MirrorRequest	0.560	0.737	0.211	0.737
10	priceItems	unitPrice	ItemModel	mirrorItems	0.683	0.737	0.526	0.737

Source: Author.

Table 30 – API list = (ITSysAPI →FDSysAPI) | k = 10 | rank=major_rank(major_sem_rank)

Index	OA Out Attr	OA Out Attr descr	TA In Attr	TA In Attr descr	bert	roberta	t5	bart	major_sem_rank
1	quantity	The quantity of items	quantity	The quantity of items in the mirror	0.838	0.555	0.441	0.858	0.858
10	priceItems	The price of the items	unitPrice	The unit price of the item in the mirror	0.813	0.663	0.765	0.840	0.840
6	unitQuantity	The quantity per unit of the item	quantity	The quantity of items in the mirror	0.781	0.423	0.698	0.810	0.810
4	priceItems	The price of the items	pricesIncludeIcms	Indicates whether the item's prices include ICMS	0.790	0.436	0.565	0.687	0.790
2	itemUrl	The URL to retrieve the item	itemNumber	The unique number associated with the mirror item	0.784	0.336	0.654	0.705	0.784
7	description	The description of the item	unitPrice	The unit price of the item in the mirror	0.778	0.533	0.696	0.724	0.778
3	priceItemsUrl	The URL to retrieve pricing of the items	pricesIncludeIcms	Indicates whether the item's prices include ICMS	0.759	0.437	0.598	0.728	0.759
5	itemSkus	A list of SKUs (Stock Keeping Units) associated with the item	itemNumber	The unique number associated with the mirror item	0.740	0.275	0.652	0.715	0.740
9	description	The description of the item	siteCode	The side code associated with the mirror	0.735	0.204	0.617	0.611	0.735
8	configuredSelectionsUrl	The URL to retrieve the configured selections for the item	issuerLocationCode	The location code of the issuer associated with the mirror	0.708	0.153	0.709	0.674	0.709

Source: Author.

Table 31 – API list = (ITSysAPI →FDSysAPI) | k = 10 | rank=major_rank / Threshold=0.8 / Precision (0.2) - Recall (1) - Accuracy (0.6)

Index	OA Out Attr	OA Out Attr descr	TA In Attr	TA In Attr descr	major_rank	major_sem_rank	gini	*calc_syn_sem	prediction_syn_sem	correct_syn_sem
1	quantity	The quantity of items	quantity	The quantity of items in the mirror	1.000	0.858	0.210	0.901	1	1
2	itemUrl	The URL to retrieve the item	itemNumber	The unique number associated with the mirror item	0.891	0.784	0.199	0.816	1	0
10	priceItems	The price of the items	unitPrice	The unit price of the item in the mirror	0.737	0.840	0.143	0.809	1	0
6	unitQuantity	The quantity per unit of the item	quantity	The quantity of items in the mirror	0.800	0.810	0.207	0.807	1	0
4	priceItems	The price of the items	pricesIncludeIcms	Indicates whether the item's prices include ICMS	0.841	0.790	0.177	0.805	1	0
7	description	The description of the item	unitPrice	The unit price of the item in the mirror	0.800	0.778	0.112	0.785	0	0
3	priceItemsUrl	The URL to retrieve pricing of the items	pricesIncludeIcms	Indicates whether the item's prices include ICMS	0.844	0.759	0.179	0.784	0	0
5	itemSkus	A list of SKUs (Stock Keeping Units) associated with the item	itemNumber	The unique number associated with the mirror item	0.825	0.740	0.144	0.766	0	0
9	description	The description of the item	siteCode	The side code associated with the mirror	0.737	0.735	0.100	0.736	0	0
8	configuredSelectionsUrl	The URL to retrieve the configured selections for the item	issuerLocationCode	The location code of the issuer associated with the mirror	0.780	0.709	0.125	0.730	0	0

Source: Author.

5.3.3.2 Interview with Project Developers

Scenario#3:

The third semi-structured interview was conducted on December 20, 2024, using the Google Meet platform, lasting 39 minutes. It was observed through the developers' answers that the semantic similarity layer increased confidence in the results, as the focus on the meaning of the information made the responses clearer and more secure than those based solely on the API attributes. The interview was conducted after the developers' interactions with the ordered

rankings, and aimed to evaluate the tool in identifying connection points between API and their viability for future use in the participants' activities. In addition, the interview revisited the steps of the method, promoting discussions about the improvements suggested in previous interviews, especially the addition of a semantic similarity layer in the descriptions of the API attributes. Based on these discussions, new questions were developed, seeking to deepen the analysis of the proposed functionalities and explore their relevance in the practical context.

- Q1: Does the semantic analysis layer on API attribute descriptions help reduce ambiguity or identify relevant connections between syntactic attributes? If so, how does this impact your confidence in the results?
- Q2: Do you consider that the semantic layer integrated into the tool facilitates its application in more complex scenarios of establishing interoperability links via API?
- Q3: Does the combination of syntactic and semantic analysis make the tool more accessible and effective for teams with less experience in CS interoperability tasks?
- Q4: What improvements or new features do you suggest to better align the tool with your daily needs?

For Q1, the developers agreed that the addition of the semantic layer contributes significantly to highlighting the essence of the attributes by presenting their semantic meaning. One of the developers, identified as A, gave an example: "*This was evident in the first ranking, where the composition between QUSysAPI and ITSySAPI was shown, with the Id attribute belonging to the item connecting to the itemID. It can also be seen that there is another ID attribute, although it is syntactically the same, it refers to Quote, which is different*". This emphasis by developer A was corroborated by the other developers as a factor that increases their confidence in the tool's results, as it allows them to check the connection points through the attribute descriptions. Developer B reinforced this idea, explaining: "*The discovery of the points needs to be precise. In addition to identifying the corresponding attributes, it is essential to verify whether their descriptions make sense in the composition of another CS API, otherwise there will be inconsistencies and errors right in the testing environment.*"

For Q2, all participants agreed that the semantic layer broadens the applicability of the tool to more complex interoperability link scenarios. In general, they said that it is useful for checking semantic equivalence, since incorrect compositions compromise the delivery of emerging functionality required by new SoIS objectives.

For Q3, developers responded that the combination of syntactic and semantic analysis

makes the tool more accessible, especially for less experienced teams. Developer B explained: "*For teams with new members, this can help them navigate API descriptions by acting as a shortcut, providing initial suggestions for connection points.*" Another developer, C, added: "*This saves time and reduces the learning curve in identifying connection points between API.*"

For question Q4, the developers suggested two significant improvements. The first would be to include a feature that would provide detailed explanations of the identified connections, using artificial intelligence to generate summaries and additional information. The second suggestion was to implement an AI-based conversational chatbot, allowing users to textually describe a question. The chatbot would analyze the available API, identifying and returning the endpoints with the most relevant attributes and compatible with that question. These proposals arose because, in the last evaluation, the focus was on specific endpoints, and now they want to work with more textual explorations involving multiple API. So much so that developer B highlighted this need by stating: "*It would be interesting to have a text input field where we could textually describe the attributes and endpoints of a given API X that could be equivalent to those of API Y. This functionality, combined with checking API attributes and their descriptions, would make the tool more robust for API analysis.*"

5.4 Implications

In light of the use of the tool and subsequent interviews with CS developers, some notable implications for both developers and researchers emerged. They are outlined below.

5.4.1 Developers

We can infer that the tool stands out as a feasible instrument point for identifying connection points among CS in SoIS. It exhaustively cross-references the attributes of each endpoint present in the API description files, evaluating their syntactic and semantic similarities and highlighting the most promising matches considering the attributes and their respective descriptions. A testament to its proficiency was the ability to "discover" an connection point that even experienced CS developers had not checked, as indicated in scenario#1 Q2. That achievement highlights the immense potential of the tool as an asset in daily operations involving CS connections.

While its design and function are attuned to the SoIS context, it is plausible to suggest

to developers the tool's potential scalability. It could be aptly tailored to cater to various contexts wherein systems offer API. This spans a wide array of domains, including but not limited to web services, microservices, and Mashups[3]. Additionally, the datasets generated by tool present a valuable resource for developers, facilitating both experimentation and learning. That helps them achieve more extensive and complex API connections efficiently. Whether analyzing these datasets partially or completely, developers can gain insights to predict potential challenges, thus helping to formulate more effective strategies.

Furthermore, a relevant practical implication for developers is related to the execution time of the different types of calculations, whether syntactic or semantic. Tests performed on a Mac M1 machine, with a 3.2 GHz processor and 8 GB of RAM, demonstrated that syntactic algorithms process 10 tuples in just 0.0030 seconds. In contrast, semantic calculations for the same set require 16.3892 seconds. That significant difference in execution time makes semantic calculations infeasible to execute locally, especially as the number of tuples analyzed grows exponentially. For example, when considering the dataset generated for ITSysAPI and FDSysAPI, which contains more than 500,000 tuples, executing semantic calculations on a local machine with these specifications is impractical due to time and computational resource constraints. This scenario highlights the need to plan the application of semantic algorithms in more robust infrastructures, such as cloud servers or distributed processing clusters. On the other hand, syntactic algorithms, due to their efficiency, are suitable for rapid analysis in local environments.

Based on this, some questions were developed as challenges for developers, with the aim of exploring new paths and perspectives. These questions are presented in Table 32:

Table 32 – Challenges for professionals

#	Challenges
1	How to optimize semantic algorithms to reduce execution time on machines with limited resources?
2	What are the impacts of using different hardware architectures (e.g. GPUs, TPUs) on the execution of semantic algorithms?
3	How to adapt semantic models to operate incrementally, processing large volumes of data in smaller parts?

Source: Author.

5.4.2 Researchers

In the context of the research, we can conclude that we have reached a good level of maturity in the syntactic dimension, considering some of the main string similarity algorithms

[3] Web application that integrates data, logic, and user interface components from different applications (JAYATHILAKA *et al.*, 2014)

available in the literature. We used six of these algorithms, classified in three different domains, with calculations that employ unique techniques, such as substitution, position change, sub string intersection, weight assignment, among others. This diversity of approaches strengthens the robustness of the method in different scenarios.

Regarding the maturity of the method in the semantic dimension, there is still room for significant advances, especially in the NLP field. We conducted a comprehensive survey in the domain of words and attribute descriptions. In the context of words, we explored NLP techniques that use dictionaries and thesauri (knowledge-based methods), as well as methods that rely on corpora (corpus-based). These approaches have shown high potential to address the complex challenge of measuring polysemy and semantic nuances in words, as indicated by recent studies (YANG *et al.*, 2021). As for attribute descriptions, the area of short text semantic similarity (STSS) is still an emerging and promising field of research (YANG *et al.*, 2021; AMUR *et al.*, 2023). That area has attracted increasing interest from both academia and industry, especially due to the limitations of traditional ML techniques to discern semantics in short texts. These texts generally present weak signals and high ambiguity, becoming a significant challenge for conventional methods (AMUR *et al.*, 2023). Thus, the development of more robust and effective approaches in the semantic analysis of short descriptions remains an essential goal for the advancement of the method.

A particularly interesting field for research is the use of Large Language Models (LLM), which are AI models trained on large volumes of textual data, capable of interpreting and generating content in a contextualized way and adapted to the specific needs of the application (JOHNSEN, 2024). LLM have revolutionized text analysis (YANG *et al.*, 2023), and a promising path may be their embeddings, which are high-dimensional vector representations capable of calculating semantic similarities efficiently through numerical calculations. In addition, LLM models can be fine-tuned using our datasets generated by our tool or API in general, in order to allow a better understanding of technical and contextual terms. This can significantly improve performance in tasks such as synonym identification, term disambiguation, and contextual similarity recognition in API descriptions. For example, models such as Llama[4], Claude[5], and Jamba[6], can provide alternatives to the models presented in our catalog. This exploration can open up even more promising avenues, taking advantage of both the speed of syntactic algorithms

[4] Available at: https://www.llama.com
[5] Available at: https://www.anthropic.com/claude
[6] Available at: https://www.ai21.com/jamba

and the semantic depth provided by LLM.

Based on this, some questions were developed as challenges for researchers, with the aim of exploring new researches. These questions are presented in Table 33:

Table 33 – Challenges for researchers

#	Challenges
1	How can semantic models extract and interpret dynamic changes in business processes in SoIS and automatically indicate the most suitable API to establish interoperability links?
2	How to identify, from a descriptive text, the most suitable possible API connections from a specific endpoint, facilitating the achievement of interoperability links in SoIS?
3	How to reduce the computational cost of large-scale semantic computations without compromising the quality of the analysis?

Source: Author.

5.5 Limitations

The results, although interesting, present certain limitations that must be addressed. Firstly, for the tool to work effectively, it is essential that developers specify return attributes in API for endpoints described with status 200. This explicit detailing is critical because, in this thesis, they are the response attributes of an API that correlate with input attributes of another. A lack of clarity in this aspect can prevent connection from happening. That explicit breakdown was performed for both CFSysAPI and NFSysAPI, where responses were meticulously reviewed and subsequently incorporated into the descriptions of the respective API.

Secondly, the presentation of the tool results between the avg and major rankings highlights another important limitation. The inconsistency in the display of results means that a connection point recommendation from API may be highlighted in one ranking but absent in another, or vice versa. This may be due to the influence of thresholds or to variations in the similarity of the analyzed pairs. An example related to thresholds is the situation of QUSysAPI $\rightarrow$ ITSysAPI, in which the pair *parentId* and *descendantId* is recommended in the *avg_rank* ranking (Table 21), but is not recommended in the *major_rank* ranking (Table 24). On the other hand, an example related to pair similarity can be observed in ITSysAPI $\rightarrow$ FDSysAPI, where the attributes *type* and *itemType* were correctly recommended in the *avg_rank* ranking (Table 28), but were not even indicated in the *major_rank* ranking (Table 31). Given these inconsistencies, a more comprehensive approach, such as merging or intersecting classification results, can provide a more consistent and accurate view of the API connection points.

5.6 Threats to Validity

In empirical research, ensuring the authenticity and reliability of results is of paramount importance. As we deepened our investigation, based on our methodologies and data choices, we discerned a number of threats regarding internal, external, and construct validity. These identified threats are further detailed in subsequent subsections.

5.6.1 Threats to internal validity

In this thesis, two key threats to internal validity are discernible: the choice of Jira and Github API endpoints and the reliance on a fixed threshold parameter. The first threat emanates from our decision to choose specific endpoints (1 from GitHub and 1 from Jira) for attribute similarity comparisons leads to notable concerns. Chief among them is the potential exclusion of invaluable data, especially when considering GitHub's 562 endpoints and Jira's 205 endpoints at the time of writing. Although our cartesian product encompasses over 100.000 tuples, this only scratches the surface, suggesting a richer dataset with a wider range of metrics that could be extracted.

The second threat arises from our reliance on a fixed threshold parameter, anchored in the selection of specific endpoints. Static parameter like this run the risk of ignoring the dynamic nuances present in the data. Ideally, this threshold should be adaptive in response to data, using advanced methodologies such as machine learning. Such refinements could increase accuracy in identifying API connection points.

5.6.2 Threats to external validity

A vulnerability in threats to external validity lies in its reliance on broad parameters, which are based on documentation examined during the controlled experiment (section 5.2). While these extensions undoubtedly expand the ability to identify connection points, they may not be universally relevant in all scenarios. Because that approach requires a significant investment of time and intensive documentation studies, it may not be a suitable strategy for all API scenarios. Furthermore, there is a latent risk that conclusions may be biased towards parameters that take advantage of these extensions. Such bias could restrict the scope of the study, impeding its ability to extend its conclusions to a more diverse range of API contexts.

5.6.3 Threats to construct validity

The potential threat to construct validity arises from the correctness of the metrics employed in our evaluation. We specifically selected precision, recall, and accuracy as our evaluation metrics, holding a consistent k-value of 10 for overall ratings and API connection points suggestions. To bolster the credibility of our recommendations, we lean on seasoned developers, proficient in the realm of CS interoperability tasks, to validate our results. That involvement serves as a strong safeguard, guaranteeing the integrity of our evaluation and protecting against potential inconsistencies.

5.7 Final Remarks

In this chapter, we present a summary of the evaluation of our method, applying our tool in two different contexts: a controlled experiment, and a case study.

Initially, we provide an overview of our tool, highlighting its structure, main modules, and the process for obtaining results and metrics. Next, with this basic understanding of the tool, we performed the first evaluation in a controlled experiment, whose objective was to identify the most effective string similarity algorithms and the optimal thresholds for detecting API connection points. We used the API syntactic similarity analysis module of the tool to examine the Jira and GitHub API. The analysis of the results showed empirically that three of the six algorithms (*jaro_winkler*, *sorensen* and *ratcliff_obershelp*) stood out as the most effective for this purpose. In addition, the optimal thresholds were identified as 0.5 for *avg_rank*, and 0.8 for *major_rank*.

The second evaluation was a case study involving API from five CS in a real-world SoIS of a global computer manufacturer. The case study was conducted from two distinct perspectives: syntactic analysis and syntactic-semantic analysis. From the syntactic analysis perspective, we used the optimal thresholds defined in the controlled experiment, with values of 0.5 for *avg_rank* and 0.8 for *major_rank*, applied to two scenarios (#1 and #2) of the SoIS. From the syntactic-semantic analysis perspective, we assigned weights of 0.3 for w_1, 0.7 for w_2, prioritizing the semantic dimension inspired on Jackson e Moulinier (2007). For that scenario, we adjusted the thresholds to 0.7 for *avg_rank* and 0.8 for *major_rank*, which was applied to the third scenario (sub scenarios #3.1 and #3.2). The results of the three scenarios were evaluated by five CS developers, through semi-structured interviews. Their feedback allowed us to extract

relevant implications for both developers and researchers. In addition, we highlighted the main limitations and identified threats to validity.

The next chapter shows the conclusion of this thesis providing its summary, the contributions, the published scientific artifacts, and the future work.

6 CONCLUSION

This chapter presents the conclusion of the doctoral thesis, consolidating the main aspects of the research carried out. Section 6.1 provides the thesis summary, highlighting its main points. Section 6.2 describes the main thesis contributions. Then, Section 6.3 details the main results, together with the scientific published artifacts resulting from the research. Finally, Section 6.4 addresses the next steps and future work.

6.1 Thesis Summary

This thesis investigated the problem of establishing interoperability links between CS in SoIS, as introduced in Chapter 1. That problem is a common challenge faced by CS developers, who need to analyze and study extensive API documentation to verify diverse data, attributes, and methods to identify API connection points and enable new emergent behaviors in SoIS. On hand, all the support information related to the main concepts such as SoIS, API descriptions, and the main related work, were described in Chapter 2. On the other hand, we conducted literature research based on on the main techniques of API descriptions to be applied in the our method (solution), which generated the catalog of textual similarity analysis techniques, as highlighted in Chapter 3.

So based on the catalog and looking for the problem, we introduced a semi-automatic method, presented in Chapter 4, designed to assist CS developers in SoIS interoperability links tasks, by discovering connection points among CS through similarity analysis of their API. The method encompasses three interconnected steps aiming to produce rankings of potential API connection points, which are syntactic and semantic similarity algorithms. From these rankings, CS developers can interact and identify the correct connection points, potentially reducing the time spent on analyzing and studying the exact CS connection points, a task that is difficult, time-consuming and error-prone.

We also generated a tool to support the method, as shown in chapter 5. Its overview is presented in detail in section 5.1, where we also explore each step of interaction with the tool and the process to generate the results, which are the API connection points. Furthermore, the tool was evaluated in two different contexts. In the first evaluation, described in section 5.2, we conducted a controlled experiment using the tool on the Jira and GitHub API. During this experiment, we empirically identified the best thresholds and syntactic algorithms. In the second

evaluation, presented in section 5.3, we conducted a case study with a five CS API on a real SoIS of a global computer manufacturer, involving 3 different scenarios. That evaluation included the participation of five CS developers, who evaluated the efficiency of the tool, and this helped us to draw implications for developers and researchers, section 5.4. Finally, limitations are presented in the section 5.5, and threats to validity in the section 5.6. Currently, the tool is in its first stable version, with continuous improvements being implemented. Our goal is to evolve it into a robust version, towards a commercial tool to be released by us.

6.2 Thesis contributions

The main objective of this thesis was to develop a reliable semi-automatic method to support CS development teams in identifying API connection points, facilitating the achievements of interoperability links in SoIS through CS API. Throughout the research, contributions were generated during the construction and results acquisition stages, which can be summarized as follows:

- Comprehensive catalog of techniques for textual similarity analysis: we have analyzed and organized a detailed and structured catalog that brings together 48 of the main syntactic and semantic approaches available in the literature for analyzing API descriptions. This catalog provides a robust foundation, serving as a starting point for the development of new approaches aimed at establishing interoperability links in SoIS through the analysis of textual descriptions of CS API.
- Analysis Method for automatic discovery of API connection points in SoIS: we propose an innovative method to support CS development teams in efficiently identifying connection points between API, aiming to support the achievement of interoperability links in SoIS. This method simplifies and accelerates the establishment of new emerging behaviors in SoIS, based on the functionalities already existing in CS.
- Availability of the back-end codes of the implemented tool: as part of the contribution, we make available the back-end codes of the tool that implements the proposed method. That tool automates textual analysis and identification of connection points, providing a practical and accessible solution to support CS developers in their activities of achieving interoperability links in the context of SoIS.

6.3 Published Scientific Artifacts

To date, this research has resulted in the following scientific artifacts, including this thesis, listed below:

- Papers:
  - 🏆 BORGES, M. V. D. F.; ROCHA, L. S.; MAIA, P. H. M. MicroGraphQL: a unified communication approach for systems of systems using microservices and GraphQL. *In*: IEEE/ACM INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING FOR SYSTEMS-OF-SYSTEMS AND SOFTWARE ECOSYSTEMS, 10., 2022, [s. l.]. **Proceedings** [...]. [S. l.]: [s. n.], 2022. p. 33–40. *One of the best papers*.
  - BORGES, M. V. D. F.; MANZANO, W.; ROCHA, L. S.; MAIA, P. H. M.; NAKAGAWA, E. Y. Towards automatic generation of systems-of-systems architectural configurations. *In*: IEEE/ACM INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING FOR SYSTEMS-OF-SYSTEMS AND SOFTWARE ECOSYSTEMS, 12., 2024, [s. l.]. **Proceedings** [...]. [S. l.]: [s. n.], 2024. p. 29-36.
  - BORGES, M.; ROCHA, L.; MAIA, P. H. M.; SANTOS, R. P. d. Agape: A syntactic-based analysis method for automatic api connection points discovery in systems-of-information systems. **IEEE Systems Journal**, [s. l.], v. 18, n. 4, p. 2052–2061, 2024.
- Other artifacts:
  - BORGES, M. V. d. F. Main Methods and Techniques of Syntactic and Semantic Similarities. In: GESAD TALKS at State University of Ceara (UECE). **Zenodo**, 2023a. Available at: https://doi.org/10.5281/zenodo.11328557.
  - BORGES, M. V. d. F. INTENSIFY: A SYNTACTIC-BASED ANALYSIS APPROACH FOR AUTOMATIC APIS INTEGRATION POINTS DISCOVERY IN SYSTEMS- OF-SYSTEMS. In: 3rd LATAM School in Software Engineering (LATAM), Brazilian Conference on Software: Practice and Theory (CBSoft). **Zenodo**, 2023b. Available at: https://doi.org/10.5281/zenodo.10790888.

6.4 Future Work

Future work in this research can be organized into four main perspectives: (1) writing a scientific paper for the Journal of Systems and Software (JSS); (2) refining the method and

tool; (3) planning the implementation of AI-based incremental fine-tuning; and (4) exploring the use of AI agents. Each of these directions seeks to consolidate and expand the results obtained in this thesis, aligning them with practical and scientific demands. Below, we detail each one of them:

1. Writing a scientific paper for the Journal of Systems and Software (JSS): we plan to write a paper detailing the syntactic-semantic analysis method developed in this thesis, together with the results obtained. This paper will be submitted to the JSS, in calls that address topics directly or indirectly related to systems-of-systems. With this, we seek to contribute to the advancement of the field, promoting the dissemination of our findings, as well as obtaining feedback on our proposed method.
2. Refining the method and tool: based on developers feedback and identified areas for improvement, we plan to improve the method and tool by addressing the following key aspects:
   - Adding detailed explanations or summaries using AI: we will incorporate LLM, such as LLaMA 3 (3.3 70B text-only), to generate more detailed explanations of connections between API. We are planning to combine that model with the Retrieval-Augmented Generation (RAG) technique, which retrieves relevant information and uses context from API to produce clearer and more reliable summaries. This functionality will further increase developers' confidence in the results presented.
   - Creating an AI-based conversational chatbot: we will add a question-answering functionality to the API connections dataset, allowing for advanced textual searches. The chatbot will be developed with specialized and accurate LLM models for the task, expanding the scope of the tool and offering a more flexible and intuitive interaction for users.
   - Expanding analytics beyond the REST API: we will plan to extend the scope of the tool to include gRPC and GraphQL, aiming to support new ways of communicating among systems, ensuring greater comprehensiveness and flexibility in adapting to modern needs.
3. Planning the implementation of AI-based incremental fine-tuning: we plan to implement a incremental learning system that tracks how developers use the tool over time, considering filter settings or natural language questions. That approach will allow us to continuously fine-tune the model, adapting it to specific contexts and refining its accuracy and usefulness

based on the actual behavior of CS developers.

4. Exploring the use of AI agents: finally, we will explore the integration of intelligent AI agents based on LLM to manage more complex workflows, automate analysis, and provide real-time suggestions. These agents could, for example, guide developers through the analysis process or anticipate common needs, make decisions and take actions to help identify possible equivalences between API. This functionality would increase the efficiency of the system and promote a more interactive and dynamic user experience.

REFERENCES

ACHEAMPONG, F. A.; NUNOO-MENSAH, H.; CHEN, W. Transformer models for text-based emotion detection: a review of BERT-based approaches. **Artificial Intelligence Review**, [s. l.], v. 54, n. 8, p. 5789–5829, 2021.

ADAMSON, G. W.; BOREHAM, J. The use of an association measure based on character structure to identify semantically related pairs of words and document titles. **Information Storage and Retrieval**, [s. l.], v. 10, n. 7, p. 253–260, 1974.

ADDURU, V.; HASAN, S. A.; LIU, J.; LING, Y.; DATLA, V. V.; QADIR, A.; FARRI, O. Towards dataset creation and establishing baselines for sentence-level neural clinical paraphrase generation and simplification. *In:* WORKSHOP ON KNOWLEDGE DISCOVERY IN HEALTHCARE, KHD@IJCAI, 2018, [s. l.]. **Proceedings** [...]. [s. l.]: WS, 2018. p. 45–52.

AFOUTNI, Z.; LE-DUIGOU, J.; ABEL, M.-H.; EYNARD, B. Towards a proactive interoperability solution in systems of information systems: a PLM perspective. *In*: INTERNATIONAL CONFERENCE ON PRODUCT LIFECYCLE MANAGEMENT, 14., 2017, Seville. **Proceedings** [...]. Seville: Springer, 2017. p. 580–589.

ALAMPALLI, S.; PARDO, T. A study of complex systems developed through public-private partnerships. *In:* INTERNATIONAL CONFERENCE ON THEORY AND PRACTICE OF ELECTRONIC GOVERNANCE, 8., 2014, [s. l.]. **Proceedings** [...], [s. l.]: [s. n.], 2014. p. 442–445.

AMUR, Z. H.; HOOI, Y. K.; BHANBHRO, H.; DAHRI, K.; SOOMRO, G. M. Short-text semantic similarity (stss): Techniques, challenges and future perspectives. **Applied Sciences**, [s. l.], v. 13, n. 6, p. 3911, 2023.

APIBLUEPRINT. **API Blueprint Specification**. 2024. Available at: https://apiblueprint.org/documentation/specification.html. Accessed on: Aug. 22, 2024.

AU2, M. B. I.; KATZ, D. M. **GPT Takes the Bar Exam**. 2022. Available at: https://arxiv.org/abs/2212.14402. Accessed on: Aug. 21, 2024.

BASSO, F. P.; PILLAT, R. M.; OLIVEIRA, T. C.; BECKER, L. B. Supporting large scale model transformation reuse. **SIGPLAN Notices**, New York, v. 49, n. 3, p. 169–178, oct. 2013.

BATISTA, P. E. P.; RODRIGUES, C. L.; NETO, V. V. G.; KASSAB, M. Arc-SOISE: Towards a reference architecture for constituents of educational systems-of-information systems. *In:* SYSTEM OF SYSTEMS ENGINEERING CONFERENCE, 17., 2022, [s. l.]. **Proceedings** [...], [s. l.]: [s. n.], 2022. p. 142–147.

BATISTA, T. Challenges for SoS architecture description. *In:* INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING FOR SYSTEMS-OF-SYSTEMS, 1., 2013, New York. **Proceedings** [...]. New York: Association for Computing Machinery, 2013, p. 35–37.

BELTAGY, I.; LO, K.; COHAN, A. SciBERT: A pretrained language model for scientific text. *In:* CONFERENCE ON EMPIRICAL METHODS IN NATURAL LANGUAGE PROCESSING AND INTERNATIONAL JOINT CONFERENCE ON NATURAL LANGUAGE PROCESSING, 9., 2019, Hong Kong. **Proceedings** [...]. Hong Kong: ACL, 2019. p. 3615–3620.

BIEHL, M. **API architecture**. 2. ed. [S. l.]: API-University Press, 2015.

BIEHL, M. **RESTful API Design**. 3. ed. [S. l.]: API-University Press, 2016.

BOARDMAN, J.; SAUSER, B. System of systems - the meaning of of. *In:* IEEE/SMC INTERNATIONAL CONFERENCE ON SYSTEM OF SYSTEMS ENGINEERING, 2006, [s. l.]. **Proceedings** [...]. [s. l.]: [s. n.], 2006. p. 118-123.

BORGES, M.; ROCHA, L.; MAIA, P. H. M.; SANTOS, R. P. d. Agape: a syntactic-based analysis method for automatic API connection points discovery in systems-of-information systems. **IEEE Systems Journal**, [s. l.], v. 18, n. 4, p. 2052–2061, 2024.

BORGES, M. V. d. F. Main Methods and Techniques of Syntactic and Semantic Similarities. *In*: GESAD TALKS at State University of Ceara (UECE). **Zenodo**, 2023a. Available at: https://doi.org/10.5281/zenodo.11328557. Accessed on: Aug. 21, 2024.

BORGES, M. V. d. F. INTENSIFY: A SYNTACTIC-BASED ANALYSIS APPROACH FOR AUTOMATIC APIS INTEGRATION POINTS DISCOVERY IN SYSTEMS-OF-SYSTEMS. *In*: 3rd LATAM School in Software Engineering (LATAM), Brazilian Conference on Software: Practice and Theory (CBSoft). **Zenodo**, 2023b. Available at: https://doi.org/10.5281/zenodo.10790888. Accessed on: Oct. 2, 2024.

BORGES, M. V. D. F.; MANZANO, W.; ROCHA, L. S.; MAIA, P. H. M.; NAKAGAWA, E. Y. Towards automatic generation of systems-of-systems architectural configurations. *In:* IEEE/ACM INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING FOR SYSTEMS-OF-SYSTEMS AND SOFTWARE ECOSYSTEMS, 12., 2024, [s. l.]. **Proceedings** [...]. [s. l.]: [s. n.], 2024. p. 29–36.

BORGES, M. V. D. F.; ROCHA, L. S.; MAIA, P. H. M. MicroGraphQL: a unified communication approach for systems of systems using microservices and GraphQL. *In:* IEEE/ACM INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING FOR SYSTEMS-OF-SYSTEMS AND SOFTWARE ECOSYSTEMS, 10., 2022, [s. l.]. **Proceedings** [...]. [s. l.]: [s. n.], 2022. p. 33–40.

BUCHER, M. J. J.; MARTINI, M. Fine-tuned'small'llms (still) significantly outperform zero-shot generative ai models in text classification. **ArXiv preprint**, 2024. Available at: https://arxiv.org/abs/2406.08660. Accessed on: Apr. 1, 2024.

CANCEDDA, N.; GAUSSIER, E.; GOUTTE, C.; RENDERS, J. M. Word sequence kernels. **The Journal of Machine Learning Research**, [s. l.], v. 3, p. 1059–1082, 2003.

CHANDRASEKARAN, D.; MAGO, V. Evolution of semantic similarity—a survey. **ACM Computing Surveys**, New York, v. 54, n. 2, p.1-37, 2021.

CORPORA. **Full-text data from English-Corpora.org: billions of words of downloadable data**. 2024. Available at: https://www.corpusdata.org/database.asp. Accessed on: Oct. 4, 2024.

CREMASCHI, M.; PAOLI, F. D. Toward automatic semantic API descriptions to support services composition. *In:* EUROPEAN CONFERENCE ON SERVICE-ORIENTED AND CLOUD COMPUTING, 6., 2017, [s. l.]. **Proceedings** [...]. [s. l.]: Springer, 2017. p. 159–167.

DAMERAU, F. J. A technique for computer detection and correction of spelling errors. **Communications of the ACM**, New York, v. 7, n. 3, p. 171–176, 1964.

DAS, S.; TARIQ, A.; SANTOS, T.; KANTAREDDY, S. S.; BANERJEE, I. Recurrent neural networks (rnns): architectures, training tricks, and introduction to influential research. **Machine Learning for Brain Disorders**, New York, v. 197, p. 117–138, 2023.

DEVLIN, J.; CHANG, M.-W.; LEE, K.; TOUTANOVA, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *In*: **North American Chapter of the Association for Computational Linguistics**, 2019, Minnesota. **Proceedings** [...]. Minnesota: ACL, 2019, p. 4171–4186.

DHARAVATH, R.; SINGH, A. K. Entity resolution-based Jaccard similarity coefficient for heterogeneous distributed databases. *In:* INTERNATIONAL CONFERENCE ON COMPUTER AND COMMUNICATION TECHNOLOGIES, 2., 2015, [s. l.]. **Proceedings** [...]. [s. l.]: Springer, 2016. p. 497–507.

DHOOPATI, P. K. Enhancing enterprise application integration through artificial intelligence and machine learning. **International Journal of Computer Trends and Technology**, [s. l.], v. 71, n. 2, p. 54–60, 2023.

DOMERÇANT, J. C.; MAVRIS, D. N. Measuring the architectural complexity of military systems-of-systems. *In:* AEROSPACE CONFERENCE, 2011, [s. l.]. **Proceedings** [...]. [s. l.]: [s. n.], 2011. p. 1–16.

DRESCH, A.; LACERDA, D. P.; JR, J. A. V. A.; DRESCH, A.; LACERDA, D. P.; ANTUNES, J. A. V. **Design Science Research**. 2015. Available at: https://ieeexplore.ieee.org/document/5747653. Accessed on: Oct. 3, 2024.

ELAVARASI, S. A.; AKILANDESWARI, J.; MENAGA, K. A survey on semantic similarity measure. **International Journal of Research in Advent Technology**, [s. l.], v. 2, n. 3, p. 389–398, 2014.

ESPINHA, T.; ZAIDMAN, A.; GROSS, H.-G. Web API growing pains: Stories from client developers and their code. *In:* SOFTWARE EVOLUTION WEEK - IEEE CONFERENCE ON SOFTWARE MAINTENANCE, REENGINEERING, AND REVERSE ENGINEERING, 2014, [s. l.]. **Proceedings** [...]. [S. l.]: IEEE, 2014. p. 84–93.

FERNANDES, J.; CORDEIRO, F.; FERREIRA, F.; NETO, V. V. G.; SANTOS, R. P. d. A method for identification of potential interoperability links between information systems towards system-of-information systems. **iSys - Brazilian Journal of Information Systems**, [s. l.], v. 15, n. 1, p. 1–26, 2022.

FERNANDES, J.; FERREIRA, F.; CORDEIRO, F.; NETO, V. V. G.; SANTOS, R. Pereira dos. A conceptual model for systems-of-information systems. *In:* IEEE INTERNATIONAL CONFERENCE ON INFORMATION REUSE AND INTEGRATION FOR DATA SCIENCE, 20., 2019, [s. l.]. **Proceedings** [...]. [S. l.]: [s. n.], 2019. p. 364–371.

FERNANDES, J.; FERREIRA, F.; CORDEIRO, F.; NETO, V. G.; SANTOS, R. How can interoperability approaches impact on systems-of-information systems characteristics? *In:* BRAZILIAN SYMPOSIUM ON INFORMATION SYSTEMS, 16., 2020, New York. **Proceedings** [...]. New York: ACM, 2020. p. 1–8.

FERNANDES, J.; NETO, V. V. G.; SANTOS, R. P. d. An approach based on conceptual modeling to understand factors that influence interoperability in systems-of-information systems. *In:* BRAZILIAN SYMPOSIUM ON SOFTWARE QUALITY, 20., 2021, New York. **Proceedings** [...]. New York: ACM, 2021. p. 1–8.

FERNANDES, J.; OLIVEIRA, L.; NETO, V. V. G.; SANTOS, R. P. d.; ANGARITA, R.; GUEHIS, S.; CARDINALE, Y. PIS: Interoperability and decision-making process—a review. *In:* SPRINGER INTERNATIONAL PUBLISHING. **The Evolution of Pervasive Information Systems**. Cham: Springer, 2023. p. 157–190.

FERNANDES, J.; SANTOS, R. P. dos; NETO, V. V. G. **Uma abordagem baseada em modelagem conceitual para compreender fatores que influenciam interoperabilidade em sistemas-de-sistemas de informação**. 2020. Dissertation (Master in Computer Science) – Federal University of the State of Rio de Janeiro, Rio de Janeiro, 2020.

FERREIRA, F. H. C. **A framework for supporting the design of fault-tolerant systems-of-systems**. 2023. Thesis (PhD in Computer Science) – Federal University of the State of Rio de Janeiro, Rio de Janeiro, 2023. Available at: https://www.researchgate.net/profile/Francisco-Henrique-Ferreira-2/publication/375085917_A_Framework_for_Supporting_the_Design_of_Fault_Tolerant_Systems-of-Systems/links/6571ce72ea5f7f02054cc2ab/A-Framework-for-Supporting-the-Design-of-Fault-Tolerant-Systems-of-Systems.pdf. Accessed on: Jul. 12, 2024.

FRIEDMAN, C.; SIDELI, R. Tolerating spelling errors during patient validation. **Computers and Biomedical Research**, [s. l.], v. 25, n. 5, p. 486–509, 1992.

FURMAN, E.; KYE, Y.; SU, J. Computing the gini index: A note. **Economics Letters**, [s. l.], v. 185, n. 7, p. 1–9, 2019.

GALI, N.; MARIESCU-ISTODOR, R.; HOSTETTLER, D.; FRäNTI, P. Framework for syntactic string similarity measures. **Expert Systems with Applications**, [s. l.], v. 129, p. 169–185, 2019.

GITHUB. **The Top Programming Languages — Octoverse**. 2022. Available at: https://octoverse.github.com/2022/top-programming-languages. Accessed on: May 23, 2023.

GITHUB. **GitHub Flavored Markdown Specification**. 2024. Available at: https://github.github.com/gfm/#what-is-github-flavored-markdown-. Accessed on: Aug. 22, 2024.

GOODWIN, M. **What Is an API (Application Programming Interface)?**. 2024. Available at: https://www.ibm.com/topics/api. Accessed on: Aug. 20, 2024.

GORKHALI, A.; XU, L. D. Enterprise application integration in industrial integration: A literature review. **Journal of Industrial Integration and Management**, [s. l.], v. 01, n. 04, p. 1–26, 2016.

GORMAN, J.; CURRAN, J. R. Scaling distributional similarity to large corpora. *In:* INTERNATIONAL CONFERENCE ON COMPUTATIONAL LINGUISTICS AND ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, 21., 2006, [s. l.]. **Proceedings** [...]. [S. l.]: [s. n.], 2006. p. 361–368.

GOROD, A.; SAUSER, B.; BOARDMAN, J. System-of-systems engineering management: A review of modern history and a path forward. **IEEE Systems Journal**, [s. l.], v. 2, n. 4, p. 484–499, 2008.

GOTOH, O. An improved algorithm for matching biological sequences. **Journal of molecular biology**, [s. l.], v. 162, n. 3, p. 705–708, 1982.

GROVER, K.; KAUR, K.; TIWARI, K.; RUPALI; KUMAR, P. Deep learning based question generation using T5 transformer. *In:* INTERNATIONAL CONFERENCE ON ADVANCED COMPUTING, 10., 2020, Panaji. **Proceedings** [...]. Panaji: Springer, 2021. p. 243–255.

HAMMING, R. W. Error detecting and error correcting codes. **The Bell system technical journal**, [s. l.], v. 29, n. 2, p. 147–160, 1950.

HARISPE, S.; RANWEZ, S.; JANAQI, S.; MONTMAIN, J. Semantic measures for the comparison of units of language, concepts or instances from text and knowledge base analysis. **ArXiv preprint**, 2013. Available at: https://arxiv.org/abs/1310.1285. Accessed on: Jan. 10, 2024.

HAVRLANT, L.; KREINOVICH, V. A simple probabilistic explanation of term frequency-inverse document frequency (TF-IDF) heuristic (and variations motivated by this explanation). **International Journal of General Systems**, [s. l.], v. 46, n. 1, p. 27–36, 2017.

HE, H.; LIN, J. Pairwise word interaction modeling with deep neural networks for semantic similarity measurement. *In:* CONFERENCE OF THE NORTH AMERICAN CHAPTER OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS: HUMAN LANGUAGE TECHNOLOGIES, 2016, [s. l.]. **Proceedings** [...]. [S. l.]: [s. n.], 2016. p. 937–948.

HE, P.; LIU, X.; GAO, J.; CHEN, W. DeBERTa: Decoding-enhanced BERT with Disentangled Attention. **ArXiv preprint**, 2021. Available at: https://arxiv.org/abs/2006.03654. Accessed on: Jan. 10, 2024.

HIRST, G.; ST-ONGE, D. *et al.* Lexical chains as representations of context for the detection and correction of malapropisms. **WordNet**, [s. l.], v. 305, p. 305–332, 1998.

IBM. **What is WSDL?**. 2023. Available at: https://www.ibm.com/docs/en/integration-bus/10.0?topic=services-what-is-wsdl. Accessed on: Aug. 22, 2024.

IBM. **What is a Neural Network?**. 2024. Available at: https://www.ibm.com/topics/neural-networks. Accessed on: Oct. 26, 2024.

IEEE. IEEE standard glossary of software engineering terminology. **IEEE Standard**, [s. l.], p. 1–84, 1990.

ISO. **ISO/IEC/IEEE 21839:2019 - Systems and Software Engineering – System of Systems (SoS) Considerations in Life Cycle Stages of a System**. 2019. Available at: https://www.iso.org/standard/71955.htm. Accessed on: Jun. 26, 2024

JACKSON, P.; MOULINIER, I. **Natural Language Processing for Online Applications**. 2007. Available at: http://digital.casalini.it/9789027292445. Accessed on: Oct. 26, 2024.

JAMSHIDI, M. **Systems of systems engineering: principles and applications**. 1. ed. [s. l.]: CRC press, 2017.

JARO, M. A. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. **Journal of the American Statistical Association**, [s. l.], v. 84, n. 406, p. 414–420, 1989.

JAYATHILAKA, H.; PUCHER, A.; KRINTZ, C.; WOLSKI, R. Using syntactic and semantic similarity of web apis to estimate porting effort. **International Journal of Services Computing**, [s. l.], v. 2, n. 4, p. 1–14, 2014.

JIANG, J. J.; CONRATH, D. W. Semantic similarity based on corpus statistics and lexical taxonomy. **ArXiv preprint**, 1997. Available at: https://arxiv.org/abs/cmp-lg/9709008. Accessed on: Feb. 10, 2024.

JIANG, Y.; ZHANG, X.; TANG, Y.; NIE, R. Feature-based approaches to semantic similarity assessment of concepts using wikipedia. **Information Processing & Management**, [s. l.], v. 51, n. 3, p. 215–234, 2015.

JIAO, X.; YIN, Y.; SHANG, L.; JIANG, X.; CHEN, X.; LI, L.; WANG, F.; LIU, Q. TinyBERT: Distilling BERT for natural language understanding. *In:* FINDINGS OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, 2020, [s. l.]. **Proceedings** [...]. [S. l.]: Association for Computational Linguistics, 2020. p. 4163–4174.

JOHNSEN, M. **Large Language Models (LLMs)**. [*S. l.*]: Maria Johnsen, 2024.

KALBALIYEV, E.; RUSTAMOV, S. Text similarity detection using machine learning algorithms with character-based similarity measures. *In:* CONFERENCE ON MULTIMEDIA, INTERACTION, DESIGN AND INNOVATION, 2020, [s. l.]. **Proceedings** [...]. [S. l.]: Springer, 2020. p. 11–19.

KHAN, S.; NASEER, M.; HAYAT, M.; ZAMIR, S. W.; KHAN, F. S.; SHAH, M. Transformers in vision: A survey. **ACM computing surveys**, New York, v. 54, n. 10, p. 1–41, 2022.

KITCHENHAM, B. A.; BUDGEN, D.; BRERETON, P. **Evidence-based software engineering and systematic reviews**. 4. ed. [*S. l.*]: CRC press, 2015.

KOREN, I.; KLAMMA, R. The exploitation of openapi documentation for the generation of web frontends. *In*: **Companion Proceedings of the The Web Conference 2018**. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018. (WWW '18), p. 781–787. ISBN 9781450356404. Available at: https://doi.org/10.1145/3184558.3188740.

LAN, Z.; CHEN, M.; GOODMAN, S.; GIMPEL, K.; SHARMA, P.; SORICUT, R. The exploitation of OpenAPI documentation for the generation of web frontends. *In:* THE WEB CONFERENCE, 2018, Republic and Canton of Geneva. **Proceedings** [...]. Republic and Canton of Geneva: International World Wide Web Conferences Steering Committee, 2018. p. 781–787.

LANE, J. A.; EPSTEIN, D. **What is a System of Systems and Why Should I Care?**. 2013. Available at: https://api.semanticscholar.org/CorpusID:61771797. Accessed on: May 2, 2024.

LEACOCK, C. **Filling in a sparse training space for word sense identification**. 1994. Thesis (PhD in Computer Science) – Macquarie University, Sydney, 1994.

LESK, M. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. *In:* ANNUAL INTERNATIONAL CONFERENCE ON SYSTEMS DOCUMENTATION, 5., 1986, [s. l.]. **Proceedings** [...]. [S. l.]: [s. n.], 1986. p. 24–26.

LEVENSHTEIN, V. Binary codes capable of correcting deletions, insertions, and reversals. **Soviet physics doklady**, [s.l.], v. 10, n. 2, p. 707–710, 1966.

LEWIS, M.; LIU, Y.; GOYAL, N.; GHAZVININEJAD, M.; MOHAMED, A.; LEVY, O.; STOYANOV, V.; ZETTLEMOYER, L. **BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension**. 2019. Available at: https://arxiv.org/abs/1910.13461.

LI, S. **Context-aware recommender system for system of information systems**. 2021. Thesis (PhD in Computer Science) – Université de Technologie de Compiègne, Compiègne, 2021. Available at: https://theses.hal.science/tel-03232938. Accessed on: Jun 2, 2024.

LI, Y.; BANDAR, Z.; MCLEAN, D. An approach for measuring semantic similarity between words using multiple information sources. **IEEE Transactions on Knowledge and Data Engineering**, [s. l.], v. 15, n. 4, p. 871–882, 2003.

LI, Z.; LIU, F.; YANG, W.; PENG, S.; ZHOU, J. A survey of convolutional neural networks: analysis, applications, and prospects. **IEEE transactions on neural networks and learning systems**, [s. l.], v. 33, n. 12, p. 6999–7019, 2021.

LIN, D. An information-theoretic definition of similarity. *In:* INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 15., 1998, San Francisco. **Proceedings** [...]. San Francisco: Morgan Kaufmann Publishers, 1998, p. 296–304.

LIN, T.; WANG, Y.; LIU, X.; QIU, X. A survey of transformers. **AI Open**, [s. l.], v. 3, p. 111–132, 2022.

LIU, Y.; OTT, M.; GOYAL, N.; DU, J.; JOSHI, M.; CHEN, D.; LEVY, O.; LEWIS, M.; ZETTLEMOYER, L.; STOYANOV, V. **Ro{BERT}a: A Robustly Optimized {BERT} Pretraining Approach**. 2020. Available at: https://openreview.net/forum?id=SyxS0T4tvS. Accessed on: Jun 2, 2024.

LODHI, H.; SAUNDERS, C.; SHAWE-TAYLOR, J.; CRISTIANINI, N.; WATKINS, C. Text classification using string kernels. **Journal of machine learning research**, [s. l.], v. 2, n. 2, p. 419–444, 2002.

LU, Y.; PANETTO, H.; GU, X. Ontology approach for the interoperability of networked enterprises in supply chain environment. *In:* ON THE MOVE TO MEANINGFUL INTERNET SYSTEMS: OTM WORKSHOPS, 2010, Hersonissos. **Proceedings** [...]. Hersonissos: Springer, 2010. p. 229–238.

LUCKY, M. N.; CREMASCHI, M.; LODIGIANI, B.; MENOLASCINA, A.; PAOLI, F. D. Enriching API descriptions by adding API profiles through semantic annotation. *In:* INTERNATIONAL CONFERENCE ON SERVICE-ORIENTED COMPUTING, 14., 2016, Banff. **Proceedings** [...]. Banff: Springer, 2016. p. 780–794.

MACIEL, R. S. P.; VALLE, P. H. D.; SANTOS, K. S.; NAKAGAWA, E. Y. Systems interoperability types: A tertiary study. **ACM Computing Surveys**, New York, v. 56, n. 10, jun 2024.

MAHMOOD, A. 'sos call'at the other edge of chaos. **Journal of Systems Science and Complexity**, [s. l.], v. 29, p. 133–150, 2016.

MAIER, M. W. Architecting principles for systems-of-systems. **Systems Engineering: The Journal of the International Council on Systems Engineering**, [s. l.], v. 1, n. 4, p. 267–284, 1998.

MAJD, S.; MARIE-HÉLÈNE, A. System of information systems as support for learning ecosystem. *In:* EMERGING TECHNOLOGIES FOR EDUCATION, 7., 2017, Cham. **Proceedings** [...]. Cham: Springer, 2017. p. 29–37.

MANSOOR, M.; REHMAN, Z. U.; SHAHEEN, M.; KHAN, M. A.; HABIB, M. Deep learning based semantic similarity detection using text data. **Information Technology and Control**, [s. l.], v. 49, n. 4, p. 495–510, 2020.

MANTHORPE, W. H. The emerging joint system of systems: A systems engineering challenge and opportunity for APL. **Johns Hopkins APL Technical Digest**, [s. l.], v. 17, n. 3, p. 305–313, 1996.

MARCH, S. T.; SMITH, G. F. Design and natural science research on information technology. **Decision support systems**, [s. l.], v. 15, n. 4, p. 251–266, 1995.

MAYANK, M. **String Similarity—The Basic Know Your Algorithms Guide**. 2019. Available at: https://itnext.io/string-similarity-the-basic-know-your-algorithms-guide-3de3d7346227. Accessed on: Sep. 8, 2024.

MAYK, I.; MADNI, A. M. **The role of ontology in system-of-systems acquisition**. 2006. Available at: https://www.academia.edu/109982828/The_Role_of_Ontology_in_System_of_Systems_Acquisition. Accessed on: Sep. 6, 2024.

MENDES, A.; LOSS, S.; CAVALCANTE, E.; LOPES, F.; BATISTA, T. Mandala: an agent-based platform to support interoperability in systems-of-systems. *In:* INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING FOR SYSTEMS-OF-SYSTEMS, 6., 2018, [s. l.]. **Proceedings** [...]. [s. l.]: [s. n.], 2018. p. 21–28.

MONGE, A. E.; ELKAN, C. P. The field matching problem: Algorithms and applications. *In:* INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, 2., 1996, [s. l.]. **Proceedings** [...]. [s. l.]: AAAI Press, 1996. p. 267–270.

MOSCHITTI, A.; PIGHIN, D.; BASILI, R. Tree Kernels for Semantic Role Labeling. **Computational Linguistics**, [s. l.], v. 34, n. 2, p. 193–224, 2008.

MULIADI, N.; NUGROHO, B.; MULJONO, M. *et al.* Comparison of string similarity algorithm in post-processing ocr. **Journal of Applied Intelligent System**, [s. l.], v. 8, n. 1, p. 25–32, 2023.

NAKAGAWA, E. Y.; GONÇALVES, M.; GUESSI, M.; OLIVEIRA, L. B. R.; OQUENDO, F. The state of the art and future perspectives in systems of systems software architectures. *In:* INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING FOR SYSTEMS-OF-SYSTEMS, 1., 2013, New York. **Proceedings** [...]. New York: ACM, 2013. p. 13–20.

NCUBE, C.; LIM, S. L. On systems of systems engineering: A requirements engineering perspective and research agenda. *In:* INTERNATIONAL REQUIREMENTS ENGINEERING CONFERENCE, 26., 2018, [s. l.]. **Proceedings** [...]. [s. l.]: [s. n.], 2018. p. 112–123.

NEEDLEMAN, S. B.; WUNSCH, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. **Journal of molecular biology**, [s. l.], v. 48, n. 3, p. 443–453, 1970.

NETO, V. G.; OQUENDO, F.; NAKAGAWA, E. Y. Systems-of-systems: Challenges for information systems research in the next 10 years. *In:*: I GRAND RESEARCH CHALLENGES IN INFORMATION SYSTEMS IN BRAZIL, SPECIAL COMMITTEE ON INFORMATION SYSTEMS. 1., 2016, [s. l.]. **Proceedings** [...]. [S. l.]:[s. n.], 2016. p. 1–3.

NETO, V. V. G.; CAVALCANTE, E.; HACHEM, J. E.; SANTOS, D. S. On the interplay of business process modeling and missions in systems-of-information systems. *In:* IEEE/ACM JOINT INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING FOR SYSTEMS-OF-SYSTEMS AND WORKSHOP ON DISTRIBUTED SOFTWARE DEVELOPMENT, SOFTWARE ECOSYSTEMS AND SYSTEMS-OF-SYSTEMS, 5., 2017, [s. l.]. **Proceedings** [...]. [S. l.]: [s. n.], 2017. p. 72–73.

NETO, V. V. G.; LEBTAG, B. G. A.; TEIXEIRA, P. G.; BATISTA, P.; LOPES, V. C.; EL-HACHEM, J.; BUISSON, J.; OQUENDO, F.; FERNANDES, J.; FERREIRA, F. *et al.* Expanding frontiers: Settling an understanding of systems-of-information systems. **ArXiv preprint**, 2021. Available at: https://arxiv.org/abs/2103.14100. Accessed on: Feb. 10, 2024.

NEVES, V. d. O.; GARCéS, L.; NETO, V. G. G. Towards educational systems-of-information systems: Reporting results of an exploratory study. *In:* BRAZILIAN SYMPOSIUM ON INFORMATION SYSTEMS, 16., 2020, New York. **Proceedings** [...]. New York: ACM, 2020. p. 1–8.

NIELSEN, C. B.; LARSEN, P. G.; FITZGERALD, J.; WOODCOCK, J.; PELESKA, J. Systems of systems engineering: basic concepts, model-based techniques, and research directions. **ACM Computing Surveys**, New York, v. 48, n. 2, p. 1–41, 2015.

NJOKU, E. G. **Encyclopedia of remote sensing**. 1. ed. New York: Springer, 2014.

OFFERMANN, P.; LEVINA, O.; SCHöNHERR, M.; BUB, U. Outline of a design science research process. *In:* INTERNATIONAL CONFERENCE ON DESIGN SCIENCE RESEARCH IN INFORMATION SYSTEMS AND TECHNOLOGY, 4., 2009, New York. **Proceedings** [...]. New York: ACM, 2009.

OFOEDA, J.; BOATENG, R.; EFFAH, J. Application programming interface (api) research: A review of the past to inform the future. **International Journal of Enterprise Information Systems**, [s. l.], v. 15, n. 3, p. 76–95, 2019.

OLIVEIRA, L. da S.; VASCONCELOS, A. P. V.; SILVA, S. V.; SANTOS, R. P. dos. A systems-of-information systems identification method based on business process models analysis. **Journal of Management and Technology**, [s. l.], v. 22, n. 4, p. 90–115, 2022.

OLIVERO, M. A.; BERTOLINO, A.; DOMINGUEZ-MAYO, F. J.; MATTEUCCI, I.; ESCALONA, M. J. A delphi study to recognize and assess systems of systems vulnerabilities. **Information and Software Technology**, [s. l.], v. 146, p. 106874, 2022.

OPENAPI. **OpenAPI Specification 3.0.3**. 2020. Available at: https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.3.md. Accessed on: Aug. 21, 2024.

OSÓRIO, A. L.; BELLOUM, A.; AFSARMANESH, H.; CAMARINHA-MATOS, L. M. Agnostic Informatics System of Systems: The Open ISOS Services Framework. *In*: CAMARINHA-MATOS, L. M.; AFSARMANESH, H.; FORNASIERO, R. (eds.). **Collaboration in a Data-Rich World**. Cham: Springer, 2017. p. 407–420.

P., S.; SHAJI, A. P. A Survey on Semantic Similarity. *In*: INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTING, COMMUNICATION AND CONTROL. 1., 2019, [s. l.]. **Proceedings** [...]. [S. l.]: [s. n.], 2019. p. 1–8.

PEARCE, K.; ZHAN, T.; KOMANDURI, A.; ZHAN, J. A Comparative Study of Transformer-Based Language Models on Extractive Question Answering. **ArXiv preprint**, 2021. Available at: https://arxiv.org/abs/2110.03142. Accessed on: Feb. 10, 2024.

PETRAKIS, E. G.; VARELAS, G.; HLIAOUTAKIS, A.; RAFTOPOULOU, P. X-similarity: Computing semantic similarity between concepts from different ontologies. **Journal of Digital Information Management**, Citeseer, v. 4, n. 4, 2006. p. 233–237.

PUUSTINEN, O. **GraphQL for building microservices**. 2020. Dissertation (Master Program in Computer Science) – Tampere University, [s. l.], 2020. Available at: https://core.ac.uk/download/pdf/337275159.pdf. Accessed on: Mar. 10, 2024.

QUATRA, M. L.; CAGLIERO, L. Bart-it: An efficient sequence-to-sequence model for italian text summarization. **Future Internet**, [s. l.], v. 15, n. 1, p. 1–13, 2023.

RADA, R.; MILI, H.; BICKNELL, E.; BLETTNER, M. Development and application of a metric on semantic nets. **IEEE Transactions on Systems, Man, and Cybernetics**, [s. l.], v. 19, n. 1, p. 17–30, 1989.

RADFORD, A.; WU, J.; CHILD, R.; LUAN, D.; AMODEI, D.; SUTSKEVER, I. **Language models are unsupervised multitask learners**. 2019. Available at: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf. Accessed on: Mar. 18, 2024.

RAFFEL, C.; SHAZEER, N.; ROBERTS, A.; LEE, K.; NARANG, S.; MATENA, M.; ZHOU, Y.; LI, W.; LIU, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. **The Journal of Machine Learning Research**, [s. l.], v. 21, n.1, p. 5485–5555, 2020.

RAML. **Developer Resources**. 2024. Available at: https://raml.org/developer. Accessed on: Aug. 21, 2024.

RAZAVI, S. Deep learning, explained: Fundamentals, explainability, and bridgeability to process-based modelling. **Environmental Modelling & Software**, [s. l.], v. 144, p. 1–21, 2021.

REBOVICH, G. **MITRE Systems Engineering Guide**. 2014. Available at: https://www.mitre.org/sites/default/files/publications/se-guide-book-interactive.pdf. Accessed on: Aug. 25, 2024.

RESNIK, P. Using information content to evaluate semantic similarity in a taxonomy. **ArXiv preprint**, 1995. Available at: https://arxiv.org/abs/cmp-lg/9511007f. Accessed on: Apr. 25, 2024.

RISTAD, E. S.; YIANILOS, P. N. Learning string-edit distance. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, [s. l.], v. 20, n. 5, p. 522–532, 1998.

RODRIGUEZ, M.; EGENHOFER, M. Determining semantic similarity among entity classes from different ontologies. **IEEE Transactions on Knowledge and Data Engineering**, [s. l.], v. 15, n. 2, p. 442–456, 2003.

ROGERS, D. J.; TANIMOTO, T. T. A computer program for classifying plants: The computer is programmed to simulate the taxonomic process of comparing each case with every other case. **Science**, American Association for the Advancement of Science, [s. l.], v. 132, n. 3434, p. 1115–1118, 1960.

SADEGHI, M.; CARENINI, A.; CORCHO, O.; ROSSI, M.; SANTORO, R.; VOGELSANG, A. Interoperability of heterogeneous systems of systems: from requirements to a reference architecture. **The Journal of Supercomputing**, [s. l.] v. 80, n. 7, p. 8954–8987, 2024.

SAGE, A. P.; CUPPAN, C. D. On the systems engineering and management of systems of systems and federations of systems. **Information knowledge systems management**, [s. l.], v. 2, n. 4, p. 325–345, 2001.

SALADO, A. Abandonment: A natural consequence of autonomy and belonging in systems-of-systems. *In:* SYSTEM OF SYSTEMS ENGINEERING CONFERENCE, 10., 2015, [s. l.]. **Proceedings** [...]. [s. l.]: IEEE, 2015. p. 352–357.

SALEH, M.; ABEL, M.-H. System of information systems to support learners (a case study at the Université de Technologie de Compiègne). **Behaviour & Information Technology**, [s. l.], v. 37, n. 10-11, p. 1097–1110, 2018.

SANBORN, A.; SKRYZALIN, J. **Deep learning for semantic similarity**. 2015. Available at: https://cs224d.stanford.edu/reports/SanbornAdrian.pdf. Accessed on: Apr. 25, 2024.

SÁNCHEZ, D.; BATET, M. A semantic similarity method based on information content exploiting multiple ontologies. **Expert Systems with Applications**, [s. l.], v. 40, n. 4, p. 1393–1399, 2013.

SÁNCHEZ, D.; BATET, M.; ISERN, D.; VALLS, A. Ontology-based semantic similarity: A new feature-based approach. **Expert Systems with Applications**, [s. l.], v. 39, n. 9, p. 7718–7728, 2012.

SANH, V.; DEBUT, L.; CHAUMOND, J.; WOLF, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. **ArXiv preprint**, 2019. Available at: https://api.semanticscholar.org/CorpusID:203626972. Accessed on: Apr. 25, 2024

SCHWICHTENBERG, S.; GERTH, C.; ENGELS, G. From OpenAPI to semantic specifications and code adapters. *In:* IEEE INTERNATIONAL CONFERENCE ON WEB SERVICES, 2017, [s. l.]. **Proceedings** [...]. [s. l.]: IEEE, 2017. p. 484–491.

SELBY, R. Enabling reuse-based software development of large-scale systems. **IEEE Transactions on Software Engineering**, [s. l.], v. 31, n. 6, p. 495–510, 2005.

SHINDE, P. P.; SHAH, S. A review of machine learning and deep learning applications. *In:* INTERNATIONAL CONFERENCE ON COMPUTING COMMUNICATION CONTROL AND AUTOMATION, 4., 2018, [s. l.]. **Proceedings** [...]. [s. l.]: [s. n.], 2018. p. 1–6.

SLIMANI, T. Description and evaluation of semantic similarity measures approaches. **International Journal of Computer Applications**, [s. l.], v. 80, n. 10, p. 25–33, 2013.

SMITH, B. **Beginning JSON**. [*S. l.*]: Apress, 2015.

SMITH, J. A.; RUTH, B. G.; HARIKUMAR, J. **An army-centric system of systems analysis (SOSA) definition**. [*S. l.*]: Army Research Laboratory, 2011.

SMITH, T. F.; WATERMAN, M. S. *et al.* Identification of common molecular subsequences. **Journal of molecular biology**, [s. l.], v. 147, n. 1, p. 195–197, 1981.

SORENSEN, T. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on danish commons. **Biologiske skrifter**, [s. l.], v. 5, p. 1–34, 1948.

STABILI, D.; MARCHETTI, M.; COLAJANNI, M. Detecting attacks to internal vehicle networks through Hamming distance. *In:* AEIT INTERNATIONAL ANNUAL CONFERENCE, 2017, [s. l.]. **Proceedings** [...]. [S. l.]: IEEE, 2017. p. 1–6.

SUNILKUMAR, P.; SHAJI, A. P. A survey on semantic similarity. *In:* INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTING, COMMUNICATION AND CONTROL, 6., 2019, [s. l.]. **Proceedings** [...]. [S. l.]: IEEE, 2019. p. 1–8.

SURWASE, V. Rest api modeling languages - a developer's perspective. **International Journal For Science Technology And Engineering**, [s. l.], v. 2, p. 634–637, 2016.

SWAGGER. **About Swagger Specification**. 2024. Available at: https://swagger.io/docs/specification/about/. Accessed on: Aug. 21, 2024.

TEIXEIRA, P. G.; LOPES, V. H. L.; SANTOS, R. Pereira dos; KASSAB, M.; NETO, V. V. G. The status quo of systems-of-information systems. *In:* IEEE/ACM INTERNATIONAL WORKSHOP ON SOFTWARE ENGINEERING FOR SYSTEMS-OF-SYSTEMS AND WORKSHOP ON DISTRIBUTED SOFTWARE DEVELOPMENT, SOFTWARE ECOSYSTEMS AND SYSTEMS-OF-SYSTEMS, 7., 2019, [s. l.]. **Proceedings** [...]. [S. l.]: [s. n.], 2019. p. 34–41.

TEKINERDOGAN, B. Multi-dimensional classification of system-of-systems. *In:* SYSTEM OF SYSTEMS ENGINEERING CONFERENCE, 14., 2019, [s. l.]. **Proceedings** [...]. [S. l.]: [s. n.], 2019. p. 278–283.

THANAKI, J. **Python natural language processing**. [S. l.]: Packt Publishing Ltd, 2017.

TOURONEN, V. *et al.* **Microservice architecture patterns with graphql**. 2019. Thesis (PhD in Computer Science) – UNIVERSITY OF HELSINKI, Helsinki, 2019. Available at: https://helda.helsinki.fi/server/api/core/bitstreams/3e9a8bc5-0c01-4c4c-8d76-a734a1955740/content. Accessed on: Aug. 14, 2024.

TREBICHAVSKỲ, R. **Api gateways and microservice architectures**. 2021. Dissertation (Master in Computer Science) – Aalborg University, Copenhagen, 2021. Available at: https://vbn.aau.dk/ws/files/440058300/master_thesis.pdf. Accessed on: Aug. 14, 2024.

TVERSKY, A. Features of similarity. **Psychological review**, [s. l.], v. 84, n. 4, p. 327, 1977.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, L.; POLOSUKHIN, I. Attention is all you need. *In:* INTERNATIONAL CONFERENCE ON NEURAL INFORMATION PROCESSING SYSTEMS, 31., 2017, Red Hook. **Proceedings** [...]. Red Hook: Curran Associates Inc., 2017. p. 6000–6010.

VIEIRA, R.; SILVA, A. da; ROCHA, L.; GOMES, J. P. From reports to bug-fix commits: A 10 years dataset of bug-fixing activity from 55 Apache's open source projects. *In:* INTERNATIONAL CONFERENCE ON PREDICTIVE MODELS AND DATA ANALYTICS IN SOFTWARE ENGINEERING, 15., 2019, [s. l.]. **Proceedings** [...]. [S. l.]: [s. n.], 2019. p. 80–89.

VILLELA, K.; NASS, C.; NOVAIS, R.; SIMÕES, P.; TRAINA, A.; RODRIGUES, J.; MENENDEZ, J. M.; KURANO, J.; FRANKE, T.; POXRUCKER, A. Reliable and smart decision support system for emergency management based on crowdsourcing information. *In:* SPRINGER (Ed.). **Exploring Intelligent Decision Support Systems: Current State and New Trends**. Cham: Springer, 2018. p. 177–198.

W3C. **Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language**. 2024. Available at: https://www.w3.org/TR/wsdl/. Accessed on: Aug. 22, 2024.

WAGGENER, B.; WAGGENER, W. N. **Pulse code modulation techniques**. [S. l.]: Springer Science & Business Media, 1995.

WAGNER, J. **Understanding the Differences Between API Documentation, Specifications, and Definitions**. 2024. Available at: https://swagger.io/resources/articles/difference-between-api-documentation-specification. Accessed on: Aug. 20, 2024.

WALTON, P.; MAIDEN, N. **Integrated software reuse**: management and techniques. [S. l.]: Routledge, 2019.

WANG, J.; DONG, Y. Measurement of text similarity: a survey. **Information**, [s. l.], v. 11, n. 9, p. 421, 2020.

WETTINGER, J.; BREITENBÜCHER, U.; LEYMANN, F. *et al.* Any2API-automated apification. *In:* INTERNATIONAL CONFERENCE ON CLOUD COMPUTING AND SERVICES SCIENCE, 5., 2015, [s. l.]. **Proceedings** [...]. [s. l.]: SciTePress, 2015. p. 475–486.

WINKLER, W. E. **String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage**. 1990. Available at: https://www.researchgate.net/publication/243772975_String_Comparator_Metrics_and_Enhanced_Decision_Rules_in_the_Fellegi-Sunter_Model_of_Record_Linkage. Accessed on: Mar. 18, 2024.

WOHLIN, C. Case study research in software engineering—it is a case, and it is a study, but is it a case study? **Information and Software Technology**, [s.l.], v. 133, p. 106514, 2021.

WOHLIN, C.; AURUM, A. Towards a decision-making structure for selecting a research design in empirical software engineering. **Empirical Software Engineering**, [s. l.], v. 20, n. 6, p. 1427–1455, Dec 2015.

WU, Z.; PALMER, M. Verb semantics and lexical selection. **ArXiv preprint**, 1994. Available at: https://arxiv.org/abs/cmp-lg/9406033. Accessed on: Apr. 18, 2024.

YANG, J.; JIN, H.; TANG, R.; HAN, X.; FENG, Q.; JIANG, H.; YIN, B.; HU, X. **Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond**. 2023. Available at: https://arxiv.org/abs/2304.13712. Accessed on: Apr. 22, 2024.

YANG, J.; LI, Y.; GAO, C.; ZHANG, Y. Measuring the short text similarity based on semantic and syntactic information. **Future Generation Computer Systems**, [s. l.], v. 114, p. 169–180, 2021.

YANG, Z.; DAI, Z.; YANG, Y.; CARBONELL, J.; SALAKHUTDINOV, R.; LE, Q. V. XLNet: Generalized autoregressive pretraining for language understanding. *In:* INTERNATIONAL CONFERENCE ON NEURAL INFORMATION PROCESSING SYSTEMS, 33., 2019, New York. **Proceedings** [...]. New York: Curran Associates Inc., 2019.

YE, J.; CHEN, X.; XU, N.; ZU, C.; SHAO, Z.; LIU, S.; CUI, Y.; ZHOU, Z.; GONG, C.; SHEN, Y. *et al.* A comprehensive capability analysis of gpt-3 and gpt-3.5 series models. **ArXiv preprint**, 2023. Available at: https://arxiv.org/abs/2303.10420. Accessed on: Feb. 12, 2024.

YENDURI, G.; RAMALINGAM, M.; SELVI, G. C.; SUPRIYA, Y.; SRIVASTAVA, G.; MADDIKUNTA, P. K. R.; RAJ, G. D.; JHAVERI, R. H.; PRABADEVI, B.; WANG, W.; VASILAKOS, A. V.; GADEKALLU, T. R. Gpt (generative pre-trained transformer)— a comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions. **IEEE Access**, [s. l.], v. 12, p. 54608–54649, 2024.

YEPEZ, A. **Council Post: Go-To-Market Partnerships—Driving Innovation Through Collaboration**. 2024. Available at: https://www.forbes.com/sites/forbestechcouncil/2024/02/21/go-to-market-partnerships-driving-innovation-through-collaboration/?sh=700eca467ff4. Accessed on: May 3, 2024.

YU, H.; OGBEYEMI, A.; LIN, W.; HE, J.; SUN, W.; ZHANG, W.-J. A semantic model for enterprise application integration in the era of data explosion and globalisation. **Enterprise Information Systems**, [s. l.], v. 17, n. 4, p. 1–23, 2023.

ZHANG, W.; YOSHIDA, T.; TANG, X. TF-IDF, LSI and multi-word in information retrieval and text categorization. *In:* IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS, 2008, [s. l.]. **Proceedings** [...]. [S. l.]: [s. n.], 2008. p. 108–113.

ZHANG, Z.; HADJIELEFTHERIOU, M.; OOI, B. C.; SRIVASTAVA, D. Bed-tree: an all-purpose index structure for string similarity search based on edit distance. *In:* ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 10., 2010, New York. **Proceedings** [...]. New York: ACM, 2010. p. 915–926.

ZHENG, T.; GAO, Y.; WANG, F.; FAN, C.; FU, X.; LI, M.; ZHANG, Y.; ZHANG, S.; MA, H. Detection of medical text semantic similarity based on convolutional neural network. **BMC medical informatics and decision making**, [s. l.], v. 19, p. 1–11, 2019.

ZHU, G.; IGLESIAS, C. A. Computing semantic similarity of concepts in knowledge graphs. **IEEE Transactions on Knowledge and Data Engineering**, [s. l.], v. 29, n. 1, p. 72–85, 2016.

ZHU, Q.; LUO, J. Generative pre-trained transformer for design concept generation: an exploration. *In:* DESIGN SOCIETY CONFERENCE, 2022, [s. l.]. **Proceedings** [...]. [S. l.]: Cambridge University Press, 2022. p. 1825–1834.