FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Generation and transformation of virtual choreographies for platform interoperability: Iliad use case

Vítor Manuel da Silva Cavaleiro



Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Fernando José Cassola Marques

Second Supervisor: Prof. Leonel Caseiro Morgado

July 16, 2025

© Vítor Manuel da Silva Cavaleiro, 2025

Generation and transformation of virtual choreographies for platform interoperability: Iliad use case

Vítor Manuel da Silva Cavaleiro

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

President: Prof. Mariana Curado Malta
Referee: Prof. Ricardo Alexandre Peixoto de Queirós
Referee: Prof. Fernando José Cassola Marques

July 16, 2025

Resumo

Os derrames de petróleo são fenómenos complexos e dinâmicos com impactos ambientais e sociais significativos. A sua monitorização e análise exigem ferramentas interoperáveis e semanticamente robustas, capazes de representar padrões comportamentais com variação espaço-temporal. Esta tese propõe as coreografias virtuais como um modelo genérico e independente de plataforma para representar esses padrões.

A primeira contribuição consiste numa análise crítica do estado da arte, identificando paralelismos conceptuais e técnicos entre coreografias virtuais e paradigmas como guiões, modelos de processos e coreografias de serviços, com o objectivo de estabelecer uma base conceptual comum que permita a sua aplicação em diferentes domínios.

A segunda contribuição aplica este modelo a um caso real no âmbito do projecto ILIAD: a monitorização de derrames de petróleo. Foi desenvolvida uma arquitectura baseada na construção dinâmica de grafos orientada por manifestos, para gerar coreografias virtuais a partir de conjuntos de dados científicos. Um mecanismo de correspondência entre manifestos permite a integração com plataformas heterogéneas, viabilizando a execução das coreografias em ambientes multiplataforma.

A solução proposta comprovou-se viável, demonstrando capacidade para representar e executar padrões comportamentais em contextos complexos e em múltiplas plataformas. Para além de articular com sucesso os conceitos de coreografias virtuais e derrames de petróleo, o trabalho permitiu, face às exigências específicas do problema, uma evolução conceptual das coreografias.

Keywords: Virtual Choreography; Interoperability; Information Visualization; Oil Spill; Digital Twin; ILIAD

Abstract

Oil spills are complex and dynamic phenomena with significant environmental and social impacts. Their monitoring and analysis require interoperable and semantically robust tools capable of representing behavioural patterns with spatiotemporal variation. This thesis proposes virtual choreographies as a generic, platform-independent model for representing such patterns.

The first contribution is a critical analysis of the state of the art, identifying conceptual and technical parallels between virtual choreographies and related paradigms such as scripts, process models, and service choreographies, with the goal of establishing a common conceptual foundation for cross-domain applicability.

The second contribution applies this model to a real-world case within the ILIAD project: the monitoring of oil spills. A graph-based architecture was developed, guided by semantic manifests, to generate virtual choreographies from scientific datasets. A manifest-matching mechanism enables integration with heterogeneous platforms, supporting the execution of choreographies in multiplatform environments.

The proposed solution proved to be viable, demonstrating the ability to represent and execute behavioural patterns in complex contexts and across multiple platforms. Beyond successfully bridging the concepts of virtual choreographies and oil spill scenarios, this work enabled, in response to the specific demands of the problem, a conceptual evolution of virtual choreographies.

Keywords: Virtual Choreography; Interoperability; Information Visualization; Oil Spill; Digital Twin; ILIAD

UN Sustainable Development Goals

The United Nations Sustainable Development Goals (SDGs) provide a global framework to achieve a better and more sustainable future for all. It includes 17 goals to address the world's most pressing challenges, including poverty, inequality, climate change, environmental degradation, peace, and justice.

This research contributes to SDG 13 and SDG 14 by enabling semantic integration of scientific data and improving the representation of complex oceanic phenomena—such as oil spills—through platform-independent virtual choreographies embedded in digital twins. These contributions foster better decision-making, faster response mechanisms, and increased awareness of marine environmental threats.

The specific Sustainable Development Goals mentioned have the following names:

SDG 13 Take urgent action to combat climate change and its impacts

SDG 14 Conserve and sustainably use the oceans, seas and marine resources for sustainable development

SDG	**Target**	**Contribution**	**Performance Indicators and Metrics**
13	13.1	Enables visualisation of the behaviour of marine hazards such as oil spills, improving climate resilience through enhanced preparedness and response.	Number of visualisation systems integrating spill behaviour modelling.
14	14.1	Provides interoperable representations of marine pollution events, enhancing monitoring, coordination and mitigation efforts in marine ecosystems.	Increase in cross-platform representation of pollution events.

Acknowledgements

Agradeço à minha família por todo o apoio ao longo deste percurso. Em especial, ao meu pai e à minha mãe, que viveram esta tese comigo, sofreram por mim e partilharam as minhas ansiedades como se fossem deles. À minha irmã, alguém a quem tento sempre servir de exemplo. À minha namorada, que também passou por este processo e com quem partilhei o apoio mútuo em cada fase. Ao meu cão, muito imporante agradecer ao meu cão, que ele bem sabe que também não conseguiu dormir enquanto eu não fosse.

Aos meus amigos da faculdade, por todos os momentos partilhados e pelas coisas que aprendi e que levarei comigo destes cinco anos.

Ao INESC TEC e aos seus colaboradores, pelo acolhimento e pelo feedback valioso que também contribuiu diretamente para o desenvolvimento desta dissertação.

Quero agradecer a Yaraslau Maksimchyk, autor da plataforma de monitorização de derrames de óleo implementada em Three.js, por ter disponibilizado a sua plataforma para fins de avaliação das abordagens e métodos desenvolvidos nesta dissertação. Quero também agradecer a Demetrius Lacet, autor das plataformas de monitorização de derrames de óleo implementadas em Cesium e Unity, por ter disponibilizado as suas plataformas para fins de avaliação das abordagens e métodos desenvolvidos nesta dissertação.

Gostaria ainda de agradecer ao projeto ILIAD - Digital Twins of the Ocean que está enquadrado no programa de investigação e inovação Horizonte 2020 da União Europeia, ao abrigo do acordo de subvenção n.º 101037643 e ao projeto BLUE-X - Accelerating Offshore Renewables " do programa de investigação e inovação Horizonte Europa da União Europeia, ao abrigo do acordo de subvenção n.º 101131527.

Agradeço especialmente aos meus supervisores, Professor Fernando Cassola e Professor Leonel Morgado, por terem sido a peça-chave para a realização desta dissertação. Sem o seu conhecimento, apoio e experiência, os resultados aqui apresentados não teriam sido possíveis.

Finalmente, agradeço à FEUP pelos últimos cinco anos. Uma instituição de excelência, onde tive o privilégio de crescer, tanto a nível académico como pessoal.

A todos, o meu sincero obrigado.

Vítor Manuel da Silva Cavaleiro

"A melhor forma de prever o futuro é inventá-lo."

Alan Kay

Contents

1 Introduction **1**
- 1.1 Context and Motivation . . . 1
- 1.2 Problem . . . 1
- 1.3 Objective . . . 2
- 1.4 Research Questions . . . 2
- 1.5 Field Case . . . 3
  - 1.5.1 Project description . . . 3
  - 1.5.2 Solution architecture . . . 3
  - 1.5.3 Dataset Description . . . 3
  - 1.5.4 Visualization and Interaction . . . 4
  - 1.5.5 Challlenges . . . 4
  - 1.5.6 Conclusion . . . 5
- 1.6 Dissertation Outline . . . 5

2 Literature Review **7**
- 2.1 Introduction . . . 7
- 2.2 Virtual Choreographies . . . 7
  - 2.2.1 Scope Definition . . . 8
  - 2.2.2 Locating the Evidence . . . 11
  - 2.2.3 State of the Art . . . 15
  - 2.2.4 Discussion . . . 17
  - 2.2.5 Conclusions . . . 18
- 2.3 Oil Spill Behaviour Pattern Modelling . . . 18
  - 2.3.1 Scope Definition . . . 19
  - 2.3.2 Research Questions . . . 19
  - 2.3.3 Inclusion/Exclusion Criteria . . . 19
  - 2.3.4 Locating the Evidence . . . 20
  - 2.3.5 State of the Art . . . 20
  - 2.3.6 Discussion . . . 27

3 Methodology **29**
- 3.1 Design Science Research . . . 29
- 3.2 Application of the Design Science Research Methodology . . . 31
- 3.3 Problem and Objectives . . . 31
- 3.4 Artefact Development and Evaluation . . . 31

4 Theoretical background **33**
4.1 Model Driven Architecture and Templates . . . 33
4.2 Model-Driven Architecture and Virtual Choreographies . . . 34
4.3 Dataflow Architecture . . . 35
4.4 Plugin Architecture . . . 36

5 Iteration 1 - VChor Framework **38**
5.1 Design . . . 38
5.1.1 Information Model of Virtual Choreographies . . . 38
5.1.2 Generator Architecture . . . 39
5.1.3 Transformer Architecutre . . . 40
5.2 Prototype . . . 42
5.2.1 Information Model Prototype . . . 42
5.2.2 Generator Prototype . . . 45
5.2.3 Transformer prototype . . . 46
5.3 Demonstration . . . 48
5.3.1 Choreography and recipe generation . . . 48
5.3.2 Cesium-based platform Integration . . . 49
5.3.3 Unity-based platform Integration . . . 52
5.4 Evaluation . . . 55
5.4.1 Something more dynamic . . . 55
5.4.2 Putting it all together . . . 56
5.4.3 Not the right tone . . . 57
5.4.4 Don't reinvent the wheel . . . 57
5.5 Communication . . . 57
5.6 Conclusions . . . 58
5.6.1 Limitations . . . 58
5.6.2 Planned Design Iteration . . . 58

6 Iteration 2 - VC Flow v1 **59**
6.1 Design . . . 59
6.1.1 Canvas . . . 59
6.1.2 Transfomer . . . 63
6.2 Prototype . . . 63
6.2.1 Information Model Prototype . . . 63
6.2.2 Canvas . . . 65
6.3 Demonstration . . . 72
6.3.1 Choreography generation . . . 72
6.3.2 Cesium-based platform transformation . . . 73
6.3.3 Unity-based platform transformation . . . 74
6.3.4 Transformations . . . 75
6.3.5 Platform Results . . . 77
6.4 Evaluation . . . 77
6.4.1 Almost perfect . . . 78
6.4.2 Where is the play button? . . . 78
6.5 Communication . . . 79
6.6 Conclusions . . . 79
6.6.1 Limitations . . . 79
6.6.2 Next iteration . . . 79

7 Iteration 3 - VC Flow v2 **80**
7.1 Reformulated Objectives . . . 80
7.2 Design . . . 80
7.2.1 Oil spill manifest . . . 80
7.2.2 Canvas enhancements . . . 82
7.2.3 Transformer enhancements . . . 82
7.3 Prototype . . . 83
7.3.1 Canvas enhancements . . . 83
7.3.2 Transformer enhancements . . . 84
7.4 Demonstration . . . 87
7.4.1 Choreography generation . . . 87
7.4.2 Cesium-based platform transformation . . . 88
7.4.3 Unity-based platform transformation . . . 89
7.4.4 ThreeJS-based platform transformation . . . 89
7.5 Evaluation . . . 92
7.5.1 It's just polygons . . . 93
7.5.2 Platform misconception . . . 94
7.5.3 Canvas limitations . . . 98
7.6 Communication . . . 99
7.7 Conclusions . . . 99
7.7.1 Limitations . . . 100

8 Conclusions **101**
8.1 Research Questions . . . 101
8.1.1 RQ1: How can virtual choreographies be employed in a domain and platform-independent manner to overcome the limitations of current domain-specific approaches? . . . 101
8.1.2 RQ2: What are the specific challenges and opportunities presented by the oil spill field case that can inform the broader application of virtual choreographies? . . . 102
8.2 Limitations . . . 103
8.3 Future Work . . . 103
8.4 Final Thoughts . . . 104

References **105**

A Code analysis for virtual choreographies literature review **111**

B Theme analysis for virtual choreographies literature review **119**

C Canvas plugin-nodes table **124**

List of Figures

2.1 Scoping Review Methodology [32] . . . 8
2.2 Oil spill interactions and behaviours [21] . . . 21
2.3 Human made interactions with oil spills [31] . . . 22
2.4 Revision window used to assess prediction results [28] . . . 26
2.5 Virtual choreography fit into Digital Twins . . . 28

3.1 Design Science Roadmap [37] . . . 30

4.1 Mapping between recipes [24] . . . 35
4.2 DFD example [60] . . . 36
4.3 Plugin Architecture [50] . . . 37

5.1 Generator architecture . . . 40
5.2 Transformer architecture . . . 41
5.3 Transformer architecture . . . 46
5.4 Generator Configuration . . . 49
5.5 Cesium-based Platform(static oil spill) . . . 51
5.6 Cesium-based Platform(dynamic oil spill) . . . 52
5.7 Unity-based Platform(static oil spill) . . . 54
5.8 Unity-based Platform(dynamic oil spill) . . . 55

6.1 DAG architecture . . . 60
6.2 DAG building interface(Website mock was built on top of template produced by @dhaval_godwani from excalidraw) . . . 62
6.3 Transformer adapted to the DAG core . . . 63
6.4 Canvas interface . . . 66
6.5 File Selector in DAG interface . . . 67
6.6 Simple generator-coroutine example . . . 67
6.7 Add one to the array operation in DAG . . . 68
6.8 Complex operation in DAG . . . 68
6.9 Template node in DAG . . . 69
6.10 Process entity node in DAG . . . 70
6.11 Process action node in DAG . . . 71
6.12 Fetch entity node in DAG . . . 71
6.13 Fetch actions node in DAG . . . 72
6.14 Generation pipeline . . . 73
6.15 Cesium transformation pipeline . . . 74
6.16 Unity transformation pipeline . . . 74
6.17 First phase of the transformation process . . . 75

6.18 Second phase of the transformation process 76
6.19 Third phase of the transformation process 77

7.1 Semantically annotated graph 82
7.2 Transformer architecture 83
7.3 Manifest injection 84
7.4 Transformer - Platform Configuration 85
7.5 Transformer - Transformation Configuration 85
7.6 Transformer - manifest comparison 86
7.7 Transformer - choreography input 87
7.8 Generation Pipeline 87
7.9 ThreeJS transformation Pipeline 89
7.10 Transformation setup 90
7.11 Complex Scenario virtual choregraphy input 91
7.12 ThreeJS-based Platform(static oil spill) 92
7.13 ThreeJS-based Platform(dynamic oil spill) 92
7.14 Unity patch for oil spills with holes 94
7.15 Embodiment Frame 96
7.16 Current approach 97
7.17 Proposed approach 98
7.18 Mock router in the canvas 99

List of Tables

2.1 Inclusion Criteria 10
2.2 Exclusion Criteria 11
2.3 Keywords and Synonyms 12
2.4 Resulting Articles 14

A.1 Codes for Research Question 1 111
A.2 Codes for Research Question 2 112
A.3 Codes for Research Question 3 113
A.4 Codes for Research Question 4 114
A.5 Codes for Research Question 5 118

B.1 Themes for Research Question 1 119
B.2 Themes for Research Question 2 120
B.3 Themes for Research Question 3 121
B.4 Codes for Research Question 4 121
B.5 Themes for Research Question 5 123

C.1 Canvas plug-ins 125

List of Acronyms

VC	Virtual Choreography
DSRM	Design Science Research Methodology
DAG	Directed Acyclic Graph
DFDs	DataFlow Diagrams
API	Application Programming Interface
MDA	Model-Driven Architecture
DSR	Design Science Research
TBCG	Template-Based Code Generation
CIM	Computation Independent Models
PIM	Platform Independent Models
PSM	Platform Specific Models

Chapter 1

Introduction

1.1 Context and Motivation

An oil spill is a pollution event where petroleum hydrocarbons are released into the environment [35]. This incident can occur on land or at sea and carries extremely detrimental effects on marine environments, economic activities, and human health [13, 33, 52]. Monitoring and managing such hazardous occurrences require advanced tools to help specialists study the spatiotemporal event [33].

Within the ILIAD project, INESC TEC is developing a new immersive environment to represent ocean behaviours in scenarios like oil spills as a powerful tool to aid research. Given the wide variety of virtual systems used to visualise information, the project aims to develop a new interoperable approach to represent and visualise ocean data. To reach this goal, this thesis proposes the concept of virtual choreographies as a solution.

In essence, virtual choreographies are an information model designed to represent behaviour patterns in a way that is detached from any specific form of implementation [5]. They aim for portable functionality across different systems, which allows virtual choreographies to describe various scenarios and use cases, surpassing limitations imposed by platform tools.

Virtual choreographies have been adopted for specific domains, such as virtual worlds, virtual reality learning systems, and interactive storytelling [5, 23, 47], and have pursued Model-Driven Engineering principles [47] in order to integrate choreographies into different platforms. According to recent research, the implementation of virtual choreographies has mostly been tied to the xAPI specification, and a technique based on translating well-defined choreography semantic spaces to platform semantic spaces has been proposed and has shown evidence of correct choreography-platform integration [24].

1.2 Problem

While a domain-independent representation has been proposed by Emanuel et al. [47], recent research in virtual choreography has abandoned this approach due to its complex nature [24],

favouring the xAPI specification instead.

However, the xAPI specification defines virtual choreographies in a way that goes beyond structuring or encoding data. It has purposely shifted its behaviour pattern-oriented logic toward the xAPI core aspects, which focus on eLearning monitoring [6], to represent interactive digital experiences, with an emphasis on describing scenarios rather than capturing pre-existing ones. While this vision is innovative, it limits the potential of virtual choreographies to interactive learning and authoring solutions, restricting their applicability to other domains.

An additional problem is that the connection between the platform and the virtual choreography must be successfully achieved to reproduce it in the application. However, platform APIs can be loosely structured and described in technological terms, as opposed to the well-defined semantic space scenarios reported in the literature. This mismatch creates a gap that complicates integration.

1.3 Objective

Two major scientific objectives have been outlined for this thesis:

1. **To map the concept of virtual choreographies within the current literature**, identifying similar or related concepts that have been applied in domains beyond those traditionally explored. This includes uncovering theoretical and practical overlaps with other fields, leading to an updated and more comprehensive definition of virtual choreographies.

2. **To investigate how virtual choreographies can be applied to solve the concrete problem presented by the oil spills field case**. This case represents a complex, spatially and temporally dynamic phenomenon with unique properties. By applying virtual choreographies to this scenario, the aim is not only to develop a meaningful domain-specific solution, but also to expose limitations, and gaps in current formulations within the virtual choreography knowledge space, thereby revealing new directions for conceptual development.

1.4 Research Questions

To address the research objectives, the following research questions have been established. These questions are designed to guide the investigation and ensure the research objectives are met.

- **RQ1**: How can virtual choreographies be employed in a domain and platform-independent manner to overcome the limitations of current domain-specific approaches?
  - **RQ1.1**: What other concepts exist in the literature that share similarities with virtual choreographies?
  - **RQ1.2**: What are the most appropriate specifications and modelling languages for describing behaviour patterns in virtual choreographies across different domains?

- **RQ2**: What are the specific challenges and opportunities presented by the oil spill field case that can inform the broader application of virtual choreographies?
  - **RQ2.1**: How can existing scientific datasets be used to create virtual choreographies?
  - **RQ2.2**: How can virtual choreographies be effectively integrated into oil spill monitoring platforms?

1.5 Field Case

This section includes the field case that was used to further explore the concept of virtual choreographies. Here the project will be described as well as the problems and challenges that were encountered.

1.5.1 Project description

This project is aligned with the ILIAD Initiative of building a *Federated European Digital Twin of the Ocean*, focusing on standardisation, data sharing, and reusability across distributed infrastructures.

Both the architecture and the visualisation platforms are completely independent of this work, meaning that no changes were made to either component in order to fit the use of virtual choreographies.

1.5.2 Solution architecture

At the core of the project is a modular, self-contained Application Package. This package combines a Common Workflow Language (CWL) document, which formally defines the inputs, outputs, processing steps, and metadata, with a pre-built Docker image. This setup ensures that the application behaves consistently, whether it is launched locally or in the cloud.

1.5.3 Dataset Description

The datasets are produced by the *application package*, which processes NetCDF files produced from oil spill simulators to extract spatial information. The polygons represent density distributions and are stored as `MultiPolygon` geometries in `GeoJSON` format.

Each entry includes:

- **Timestamp** – When the data was captured.
- **Density** – A scalar value representing the oil concentration.
- **MultiPolygon** – The geometry corresponding to the density area.

Multiple entries can share the same timestamp and refer to the same geometrical area. The data is essentially a raw spatial export without normalization.

1.5.4 Visualization and Interaction

To visualise the data, three distinct applications were adopted: CesiumJS for browser-based map visualisation, Unity3D for immersive simulation, and a Three.js-based platform for advanced visual analytics.

What's crucial to understand about these platforms is the type of data and structural information they require in order to function. Building upon this core information, each platform employs its own approaches to enhance, tailor, and display the content as required. This variability in presentation and customisation is not a concern for the virtual choreography component.

The Unity-based platform employs advanced wave simulation techniques, allowing for a detailed representation of oil spills with distinct densities. It also incorporates oil fragmentation effects, a minimap that signals the event location, and a compass to enhance navigation. All these visuals are layered on top of the choreography without being part of it, thereby enriching the user's understanding of the environmental context; for instance, the wave simulation operates independently of the main choreography.

In contrast, the cesium-based platform is the most basic option available. It primarily showcases a simple polygon representation with varying colours and densities on a global map, clearly indicating the exact location of events and allowing users to navigate around it.

The threeJS-based platform goes a step further by not only displaying spills on a world map but also providing multiple visualisation methods such as points, polygons, and smudges. Additionally, it gathers and presents valuable statistics for each oil spill, including metrics like area, perimeter, and dispersion distance, offering a more comprehensive analysis of the environmental impact. Furthermore, this platform doesn't just load a single choreography; it accommodates multiple choreographies that can all be viewed simultaneously without extensive details. Users can zoom in on a specific oil spill by selecting it on the map, allowing for a closer examination of that choreography. This capability further distinguishes this platform from others in the field.

One important point to note is that throughout this thesis, these platforms may be referred to by their underlying engines. For example, they may be mentioned as "Unity platforms" or "Cesium platforms" This terminology is not meant to imply that the platforms themselves are Unity or Cesium. It serves as a way to identify each platform by one of its distinguishing factors, in this case, its engine.

1.5.5 Challlenges

Each of the platforms operates under a fundamentally different technological model.

Unity uses a local 3D coordinate system (`XY`) tailored to the specific scene. It does not support geographic coordinates like latitude or longitude. Spill events are represented by a single timestamp and an array of polygons, each with a different density, but all sharing a single universal colour. This lack of per-actor visual differentiation, combined with the scene-bound coordinate system, makes data integration highly dependent on explicit conversion logic. Mapping geospatial

data into Unity requires transforming both the geometry and semantics of the data to fit Unity's constraints.

Cesium works with real-world coordinates (`latitude/longitude`) and uses a data structure built around `action lists`, where each action contains a list of `actors`. Each actor corresponds to a specific density. While it is possible to align multiple actors within an action, there is no explicit composition or topological awareness of the full spill event. This structure demands not only geographic alignment but also semantic interpretation to preserve the integrity of the original event model.

Three.js also supports georeferenced data (latitude/longitude) but is designed for greater flexibility and analytic depth. It treats each spill as a composite structure composed of multiple density layers, all tied to the same topological space. Additional metadata, such as dispersion radius, affected area, perimeter, and particle count, can be embedded per spill. This allows for high-fidelity representation of complex events and comparative analysis across multiple events. However, the data must be preprocessed and well-structured to be properly rendered.

1.5.6 Conclusion

The three platforms require fundamentally different data representations: Unity requires XY transformation, Cesium offers a flat actor structure tied to geolocation, and threeJS supports topological composition with analytical enrichment. This outlines that it is not only their visual aspects and functionalities that look different; their underlying technological demands are different.

1.6 Dissertation Outline

The dissertation is divided into eight chapters:

- **Chapter 1 – Introduction:** Sets out the research problem, objectives, and significance of the study. Describes the practical application of this dissertation within the Iliad project. Specifically, it presents key components of the project, the datasets, as well as the visualisation platforms, which directly shaped the artefacts designed during this thesis.
- **Chapter 2 – Literature Review:** Contains two focused reviews. The first explores the concept of virtual choreographies, mapping the theoretical and technological landscape. The second examines oil spill behaviour modelling, and their intersection with virtual choreography approaches.
- **Chapter 3 – Methodology:** Describes the Design Science Research approach used, including the iterative development cycles, design principles, and evaluation strategies.
- **Chapter 4 – Theoretical Background:** Clarifies key theoretical concepts and models that underpin the artefact's design, including foundational principles used across the iterations.
- **Chapters 5 to 7 – DSR Iterations:**

  - **Chapter 5 – Iteration 1 - Vchor Framework:** Introduces the initial artefact concept, informed by the theoretical foundations and within the field case restrictions, and reports on the first evaluation.
  - **Chapter 6 – Iteration 2 - VC Flow v1:** Describes refinements and adaptations based on the evaluation from the first prototype, including improved architectural design of the solution and of the prototype.
  - **Chapter 7 – Iteration 3 - VC Flow v2:** Presents the final iteration of the artefact, with reshaped objectives, a refined design and prototype, its evaluation results in relation to past solutions, and an assessment of its effectiveness across platforms.

- **Chapter 8 – Conclusion:** Summarises key contributions, discusses implications for research and practice, and outlines future research directions.

Chapter 2

Literature Review

2.1 Introduction

This chapter outlines the process followed to review current literature, building the scientific foundation of this work. It is composed of two parts, the first one focuses on building a theoretical knowledge base for the virtual choreographies. The second one focuses on exploring the work done in oil spill modelling in the context of virtual choreographies.

This approach establishes a fitting scope for this dissertation, preventing it from focusing on unimportant information, saving time, and providing a more complete overview of the current research gaps and challenges surrounding virtual choreographies.

2.2 Virtual Choreographies

The scoping review adheres to a unified protocol for scoping reviews [32], the definition of which was developed based on two primary methodologies for scoping reviews [25, 40].

Based on the previously mentioned protocol, the scoping review is described in three different phases: **"Scope Definition"**, **"Locating the Evidence"** and lastly **"Knowledge Retrieval"** as shown in figure 2.1. However, for this dissertation, only the first two phases were conducted, since the initial results were considered enough evidence to conduct the thematic analysis with the time that was available.

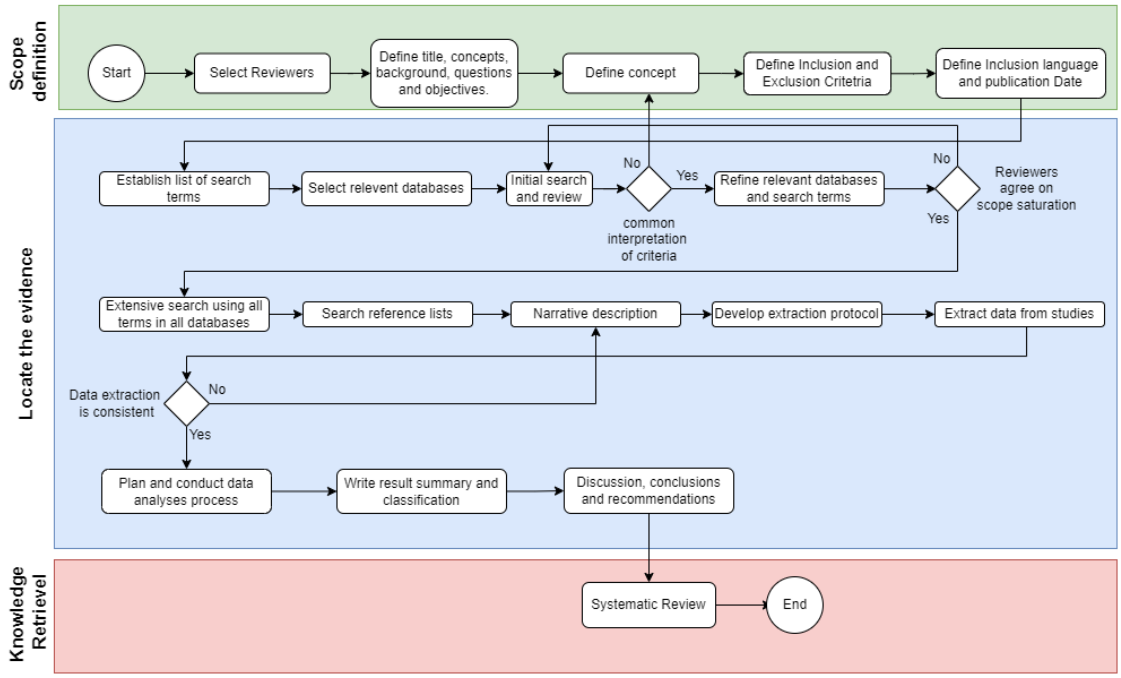


Figure 2.1: Scoping Review Methodology [32]

2.2.1 Scope Definition

The objective of this phase was to define the initial scope of the review. The scoping review was conducted in collaboration between the author and the supervisors of this dissertation. These actors correspond to the identified reviewers of the first step in the protocol 2.1.

2.2.1.1 Define title

In order to identify the title of the review, the following definitions were considered:

- Concept: Behaviour pattern modelling.
- Context: Use of Virtual choreographies to model behaviour patterns.
- Perspective: Expanding the current field of virtual choreographies with other concepts that share similarities.

From this analysis, the following title for the review emerged: Virtual choreographies- a mapping study of the literature.

2.2.1.2 Theoretical Background

Initial exploratory research on the topic of Virtual Choreography revealed various applications and terminologies.

In this article [5], the virtual reality authoring tool uses virtual choreographies to capture students' actions and industrial procedures, highlighting virtual choreography as a solution to model behaviours and sequences of tasks.

In this article [23], virtual choreographies are utilised as a solution for representing interactive and platform-independent stories (storyboarding).

In this article [47], virtual choreographies are defined in the sense of actions that a group of actors execute in a virtual world. The choreography is also classified as being multi-domain and is transformed from a platform-independent ontology to a platform-dependent ontology using a model-driven approach.

In this article [57], virtual choreographies are defined as scientific choreographies utilised to model the workflows of scientific experiments. The literature also indicates that the participants in these choreographies models can be represented through process models, service orchestrations, or workflow models.

The concept of virtual choreographies appears to be widely used, with various definitions and approaches to its application.

This scoping review aims to provide scientific value by establishing a foundational multidisciplinary study for virtual choreographies, creating a basis upon which future research can be built.

2.2.1.3 Concepts and Objective

Virtual choreography can be divided into two fundamental concepts. The first concept is "Virtual", in the sense that it can be interpreted as something that is tied to an abstract or logical meaning and is free from physical logic. It may be something theoretical or the representation of a thing in an environment that is not physically real. The second fundamental concept is "Choreography", which is a set of actions, behaviours, or events through time that follow a pattern. By joining these definitions we can classify "Virtual Choreography" as an abstract representation of a series of actions, behaviours, or events through time that follow a pattern.

The objectives of this scoping review are to explore current research on similar concepts across various disciplines, examine how these concepts are represented and applied, and compare them with virtual choreographies to develop a new formal definition.

2.2.1.4 Research Questions

Based on the concepts' definitions, the objectives, and the theoretical background, the following questions can be formalised for this review..

- **RQ1**: What terminology is used to refer to concepts similar to "virtual choreographies"?
- **RQ2**: What are those concepts similar to "virtual choreographies"?

- **RQ3**: How are those concepts used to replay/visualize/present a virtual choreography?
- **RQ4**: How are choreographies under those concepts identified/created?
- **RQ5**: What are the representational languages, technologies and formats?

2.2.1.5 Defining the Concept

This scoping review will survey the concept of Virtual Choreography in the current literature as defined in the concepts section 2.2.1.3 while taking into account the theoretical background. No restrictions are applied to the domain of the literature as it aims to be a multidisciplinary study. It will specifically target other reviews of literature to leverage the work carried out by researchers and therefore have a concise and summarised view of various research fields through the lenses of a short number of papers, which will also keep the review feasible with the time and resources available. The literature included must also have undergone peer review to ensure the quality of the work.

The outcome of this review should be aligned with the objectives and questions mentioned in the previous sections.

2.2.1.6 Inclusion and Exclusion Criteria

Based on the theoretical background, key concepts, and defined scope, the following tables for inclusion and exclusion criteria can be established.

Inclusion Criteria
Literature Survey or Literature Review
Must use or focus on concepts similar to "virtual choreographies" as defined in the foundational concepts

Table 2.1: Inclusion Criteria

Exclusion Criteria
Use of orchestration in a music or non-behaviour pattern context (e.g., educational orchestration)
Not a literature survey or review
Not within the scope of the study
Only addresses behaviours, not their patterns, sets, choreographies, or similar
Only available as an abstract or short annotation
Use of "model" or "pattern" as a synonym for algorithm or logic, not in the sense of a choreography
Use of "script" as a synonym for an interpreted or limited-scope programming language
Use of "script" in the sense of calligraphy or text
Use of "script" in the traditional theatre or storytelling context
Use of choreography in a dance or theatre context

Table 2.2: Exclusion Criteria

The literature must also be available in English, as it is the most widely used language. No restriction will be placed on the publication date to avoid excluding potentially meaningful results.

The inclusion and exclusion criteria are designed to filter out references to "virtual choreography" in contexts where the definition does not align with the foundational concept. For instance, it excludes uses of "choreography" in a physical manifestation or dance-related sense, "script" in a programming or linguistic context, and "orchestration" when it refers to music rather than sequences of events, behaviours, or actions. Additionally, studies that are not literature surveys or reviews are excluded; for example, population-based surveys rather than literature-focused reviews are not considered.

2.2.2 Locating the Evidence

The objective of this phase is to collect the necessary evidence for analysis.

2.2.2.1 Search Terms

With the identified terminology and foundational concepts outlined in the scope definition section 2.2.1, and with specialist guidance from the supervisors in formulating synonyms, the following table 2.3 of keywords was developed to address and identify virtual choreographies.

Keyword	**Synonyms**
behaviour patterns	behaviour chain, behaviour chains, behaviour of the simulation, behaviour orchestration, behaviour patterns, behaviour representation, behaviour chain, behaviour chains, behaviour of the simulation, behaviour orchestration, behaviour patterns, behaviour representation, chain of behaviours, chain of behaviours, chains of behaviour, chains of behaviour, simulation behaviour, simulation behaviour
Orchestration	agent orchestration, service orchestration
Process models	Process model, Process models
Scripts	agent script, multiagent script, story world plan, storyboard, storyboarding
Spectrum of tactics	Spectrum of tactics, tactics spectrum
Task sequences	Task sequences, sequence of tasks
Virtual choreography	choreographies, choreography
Workflow models	Workflow model, Workflow models

Table 2.3: Keywords and Synonyms

2.2.2.2 Selected Databases

To conduct the search, the selected database was SCOPUS, as it provides a direct interface to literature indexed in various libraries and encompasses a wide range of domains. The query string 2.1 formulated for the database consists of a sequence of OR operators between each identified keyword and synonyms, allowing the inclusion of the vast range of literature that is related to virtual choreographies. Additionally, it excludes literature not written in English and filters out material not in the form of conference papers, book chapters, or articles. It is also specified that keywords should be found in the title, abstract, or keywords section and that the title must include either "survey" or "review" to ensure the focus on literature reviews.

Listing 2.1: Query String

```
1 ( TITLE ( "survey" OR "review" ) AND TITLE-ABS-KEY ( "behaviour patterns" OR "
    behaviour chain" OR "behaviour chains" OR "behaviour of the simulation" OR "
    behaviour orchestration" OR "behaviour patterns" OR "behaviour representation"
    OR "behaviour chain" OR "behaviour chains" OR "behaviour of the simulation" OR
    "behaviour orchestration" OR "behaviour patterns" OR "behaviour representation"
     OR "chain of behaviours" OR "chain of behaviours" OR "chains of behaviour" OR
    "chains of behaviour" OR "simulation behaviour" OR "simulation behaviour" OR "
    Orchestration" OR "agent orchestration" OR "service orchestration" OR "Process
    models" OR "Process model" OR "Process models" OR "Scripts" OR "agent script"
    OR "multiagent script" OR "story world plan" OR "storyboard" OR "storyboarding"
```

```
 OR "Spectrum of tactics" OR "Spectrum of tactics" OR "tactics spectrum" OR "
Task sequences" OR "Task sequences" OR "sequence of tasks" OR "Virtual
choreography" OR "choreographies" OR "choreography" OR "Workflow models" OR "
Workflow model" OR "Workflow models" ) ) AND ( EXCLUDE ( DOCTYPE , "ch" ) OR
EXCLUDE ( DOCTYPE , "er" ) OR EXCLUDE ( DOCTYPE , "no" ) OR EXCLUDE ( DOCTYPE ,
 "dp" ) ) AND ( EXCLUDE ( LANGUAGE , "Chinese" ) OR EXCLUDE ( LANGUAGE , "
German" ) OR EXCLUDE ( LANGUAGE , "Italian" ) OR EXCLUDE ( LANGUAGE , "Russian"
 ) OR EXCLUDE ( LANGUAGE , "Japanese" ) OR EXCLUDE ( LANGUAGE , "Korean" ) OR
EXCLUDE ( LANGUAGE , "Turkish" ) OR EXCLUDE ( LANGUAGE , "Hungarian" ) OR
EXCLUDE ( LANGUAGE , "Undefined" ) OR EXCLUDE ( LANGUAGE , "Bosnian" ) OR
EXCLUDE ( LANGUAGE , "Malay" ) OR EXCLUDE ( LANGUAGE , "Persian" ) )
```

2.2.2.3 Narrative Description

The search on SCOPUS resulted in 2153 records, of which 5 were duplicates. Out of the remaining records, 751 were analysed by abstract, and 1402 were left unclassified. Out of the 751 classified records, 196 were accepted following the inclusion/exclusion criteria, while the rest were rejected.

From the accepted articles, an extraction based on diversified terminology was conducted to achieve a broader conceptual range. This was achieved by grouping the articles into different keyword families. Each keyword family corresponds to a row in the Search terms table 2.3. This work was automated using a script to identify which articles belonged to a certain family among accepted articles. Afterwards, an article would be sampled from a family pool, and the full text would be analysed for eligibility, according to the inclusion/exclusion criteria.

This selection process resulted in 9 sampled articles, 7 of which were eligible for analysis.

Title	**Keyword Family**
A Survey of Script Learning	Scripts
Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges	Workflow Models
Leveraging Machine Learning for Automatic Topic Discovery and Forecasting of Process Mining Research: A Literature Review	Process Models
Construction and Application of Event Logic Graph: A Survey	behaviour Patterns
Web Services Composition Mechanisms: A Review	Virtual Choreography, Workflow Models, and Orchestration
Representation primitives, process models and patient data in computer-interpretable clinical practice guidelines: A literature review of guideline representation models	Process models
Standardized languages and notations for graphical modelling of patient care processes: A systematic review	Process models, Task sequences

Table 2.4: Resulting Articles

2.2.2.4 Data analysis and Extraction

The extraction protocol aimed to exhaustively analyse and retrieve valuable information from the eligible articles. This process should be directly aligned with the research questions guiding the scoping review while prioritising the direct extraction of quotes from the text.

The data analysis follows the approach proposed by Vaismoradi et al. [53], which aims to provide direct answers to research questions 1, 2, 3, 4, and 5 as outlined in this section 1.4. The goal of this systematic analysis is to identify distinct themes connected with the concept of Virtual Choreography.

The first step of the thematic analysis involves generating codes through a cyclical process that transforms large information segments into smaller, more manageable units without losing meaningful insights. This process makes it easier to compare information in order to generate themes.

Using this approach, the codes were generated according to the information extracted from the studies. Each subset of codes, corresponding to a different research question, is presented in tables A.1, A.2, A.3, A.4, and A.5.

With the identified codes, the themes can be extracted through three main processes described by Vaismoradi et al. [53]: comparison, classification, and labelling. Comparison is a cyclical process in which similarities and differences are assessed between codes. Classification's role is to aggregate codes that follow similar features. Labelling involves giving a correct identification

to a group of codes that correspond to a unifying idea. Each assigned theme is also provided with a definition to represent what the theme aggregates and to justify why certain codes may fit better in certain themes.

Following this process, the themes were obtained and they are presented in Tables B.1, B.2, B.3, B.4, and B.5.

2.2.3 State of the Art

2.2.3.1 RQ1: What terminology is used to refer to concepts similar to "virtual choreographies"?

The scoping review identified multiple terms that refer to concepts similar to virtual choreographies, as shown in Table B.1.

The terminology includes choreography, orchestration, workflow [10], process model, behaviour/behavioural pattern [36], script, event segment, semantic frame [12], event logic graph [62] , and guideline representation [56]. Some terms are connected, such as event segment and script, where multiple event segments build the script. Some concepts appear in the same application, like workflow, choreography, and orchestration, in the context of web composition.

2.2.3.2 RQ2: What are those concepts similar to "virtual choreographies"?

While distinct in purpose, these concepts converge around the need to structure and interpret complex behavioural patterns. Scripts highlight a narrative dimension, functioning as semantically enriched representations of activities that capture and define behavioural patterns through the modelling of event evolution, including participants, actions, and multiple narrative sequences of events within a scenario [12, 62]. Choreography depicts the communication and collaboration among different services, emphasising the exchange of messages in a distributed ecosystem and capturing behavioural patterns through message sequences in multi-service scenarios [10]. Process Models concentrate on the systematic aspects of workflows and process logic, representing activities, states, and decision-making in a way that facilitates rigorous execution or simulation, with the ability to interact with services, progress in a flow across multiple entities, and represent actions, decisions, entities, and execution states in a machine-readable format [10, 36, 56].

2.2.3.3 RQ3: How are those concepts used to replay/visualise/present a virtual choreography?

Virtual choreographies can be visualised and presented through multiple technological approaches that range from basic graphical representations to sophisticated interactive environments. These include graphs [62] that present processes over time or depict relationships among entities, and

graphical simulations [34] that reproduce a system's behavioural logic in an interactive environment depicting system behaviour. Web-based approaches comprise sequences of web pages generated from pattern definitions to illustrate a process step by step, and automatic user interfaces generated from process definitions based on user interactions [16]. Real-time dashboards [16] provide feedback about processes and deliver metrics, while chord diagrams [36] analyse relationships and interactions between entities. Additional visualization methods include activity sequences that offer linear graphical representations, graphical languages [29] that serve as presentation means for mapping and communicating flows of actions and decisions, and temporal diagrams [29, 44] that function as solutions to show condition changes or represent actions between entities over time.

2.2.3.4 RQ4: Identification/Creation of Choreographies

Choreographies can be created or identified through diverse approaches that span from fully automated to completely manual methodologies. Dynamic adaptation occurs through service adaptation [4, 10], where choreographies can be updated by modifying composition upon context change or errors, or to adjust to certain needs. Automatic methods include automatic extraction or definition [10, 16] from semantically enriched descriptions, regressions, high-level abstract goals/conditions, automated planning, and machine learning extraction that identifies patterns using machine learning techniques, including specific deep learning techniques [62]. Semi-automated [10] approaches involve user-defined extraction with the aid of semantic guidance, blending user involvement with system support. Manual methods are based on human extraction [10, 16, 29] that is either visual or textual and often serve smaller processes or specific scenarios. Specialised techniques encompass pattern matching [16, 62] that identify choreographies by following strict rules, natural language processing (NLP) [12] for pattern identification in text, region-based discovery [36] methods, genetic algorithm discovery [36], and trace variant analysis [36] for activity sequence [44] extraction. Mathematical and optimisation approaches include process algebra/-calculus [10] with a mathematical foundation, linear integer programming [10] for optimal behavioural patterns, and model-driven architecture [11] for choreography creation. Process mining [16, 36] Strategies are employed for behavioural pattern identification.

2.2.3.5 RQ5: Representational Languages, Technologies, and Formats

The literature identifies a broad spectrum of specification and modelling approaches relevant to virtual choreographies.

Graphical modelling languages [10, 29, 36] stand out for describing processes at different levels of abstraction

Process modelling techniques include flowcharts, which provide procedural representations of workflows, and Petri nets, which define behavioural constraints and support formalisation of process models [16, 29, 36, 56].

Artifact-centric workflow modelling focuses on business artifacts, capturing the data lifecycle and its association with workflow logic from a data-centric perspective [16].

Event representation techniques include protagonist-based, multi-argument, and fine-grained approaches, such as the use of triple structures to express detailed event semantics [12].

Structural representations involve graphs, which capture event relationships and constraints, nested structures that support multi-level abstraction and relational models aligned with database logic [16, 56, 62].

Standard data formats include XML, RDF, and ontologies. These enable structured data representation, enriched semantics, and formal vocabulary constraints, supporting interoperability across systems [16, 56].

Behavioural modelling approaches comprise state machines for abstract behavioural model and constraints, declarative specifications based on condition-action rules, and procedural models that define ordered execution steps for combined processes [10, 16, 29, 56].

Finally, specialised representations include discrete and distributed event models, interaction constraint definitions, ad-hoc composition languages, runtime state representations, choreography protocol definitions (e.g., WSCI), action abstraction layers, and logic-based decision models for automatic choices [10, 16, 56].

2.2.4 Discussion

As a result of the state of the art and the analysis conducted, several conceptual clarifications emerge.

One of them is the definition of entities. Past definitions often refer to them as discrete *actors* or *objects*. Prior work does not propose a **hierarchical representation** where an entity could encapsulate sub-entities or operate at different levels of abstraction depending on the scenario.

In contrast, this work refines the concept: *entities* are abstract or physical components, capable of composition. It allows entities to be modelled at varying granularities, enabling more flexible and layered representations within a choreography.

Choreographies may also be treated not as a single sequence of events, but rather as multiple, potentially branching ones. A good example of the utility of such definitions is the concept of RPG games, where a decision system may be modelled as different sequences(branches) of events that may precede certain choices.

Based on these ideas, refinements, and considering past definitions, the following definition is proposed:

> **A Virtual Choreography** is a proposed pattern-oriented informational model describing how behaviours are coordinated among entities (physical or abstract) defined in layered abstractions in a platform-agnostic context. It places emphasis on the flow and interrelationships of events and actions, rather than on discrete data points. By focusing on recurring behaviour patterns, it aims to describe when and how entities interact, without prescribing a specific technical or physical infrastructure. The notion of “choreography” highlights the structured, sequenced, or branching nature of these

interactions, drawing from the analogy of choreographic coordination yet remaining detached from any single mode of implementation.

Some questions still remain for future reviews:

1. **How do events and actions distinguish themselves?** Do they overlap? Can they be used interchangeably, or are they completely different? Actions may be treated as components of an event [62] or as the necessary trigger to transition to a new event [12]. In information systems, such distinctions can be crucial from an architectural perspective. For example, events might be sporadic or triggered occurrences, while actions could represent more deliberate interactions. Clarifying this could lead to a stronger conceptual foundation.

2. **Must actions and events be tied to entities?** Can certain events(e.g., *raining*) exist without being tied to external entities? Loosening these connections may allow for more flexible modelling, depending on context and scenario granularity.

3. **Can choreographies be context-independent, beyond platform independence?** If behaviour patterns are transferable across domains, it may be possible to define choreographies that are fully abstract, reusable across both technical and semantic boundaries.

2.2.5 Conclusions

The identified terminology spans multiple domains and abstraction levels, reflecting how virtual choreographies adapt to different contexts, from high-level process management to detailed behavioural specifications.

The visualisation approaches range from basic static representations to interactive environments, providing options that scale with analysis needs (from simple process monitoring to complex behavioural simulation).

The identification and creation methods combine mathematical rigour with adaptive techniques, containing both formal verification, like process calculus, and flexible pattern discovery through machine learning approaches.

The representational technologies include different languages and formats, from standardised notation systems to specialised frameworks and data models, each optimised for specific choreography modelling requirements.

In conclusion, the mapping conducted in this review revealed conceptual gaps and overlaps, which led to new insights and the proposal of a revised formal definition of virtual choreographies.

2.3 Oil Spill Behaviour Pattern Modelling

This section outlines the literature review process undertaken for modelling oil spill behaviour patterns. It details the methodologies employed to define the scope, formulate research questions,

and identify relevant literature. By integrating structured search techniques and leveraging advanced AI tools, this review aims to provide a comprehensive overview of existing frameworks and practices in the field of oil spill modeling.

2.3.1 Scope Definition

The scope for the literature review was defined using the following dimensions:

- **Concept**: Behaviour pattern modelling.
- **Context**: Oil spill scenarios.
- **Perspective**: Virtual choreographies as the analytical approach for modelling behaviours.

This structure guided the literature selection and analysis process.

2.3.2 Research Questions

In order to guide this state-of-the-art review in the context of oil spills, the following research questions were formulated:

- **RQ1**: What are the common oil spill behaviour patterns?
- **RQ2**: How are oil spill behaviour patterns modelled?
- **RQ3**: How are oil spill behaviour patterns identified or created?
- **RQ4**: What are the representational languages, technologies, and formats for oil spill behaviour patterns?
- **RQ5**: How are behaviour patterns presented or visualised?

2.3.3 Inclusion/Exclusion Criteria

Since this literature review is focused on surveys and reviews on oil spill behaviour modeling, the following inclusion and exclusion criteria were established.

Articles were included if they:

- Were literature reviews.
- Addressed oil spill behaviour modelling.

Articles were excluded if they:

- Were not focused on oil spills.
- Were not peer-reviewed.
- Were not written in English.

- Were published before 2017.
- Were inaccessible in full text.
- Did not contribute to the defined research questions.

These criteria ensured the inclusion of high-quality, relevant, and up-to-date sources. The 2017 cut-off was chosen to prioritise recent scientific developments within the limited timeframe of this review.

2.3.4 Locating the Evidence

To identify relevant literature for this review, a structured search strategy was adopted, supported by preliminary exploration using AI tools.

Initially, AI-based systems such as *Undermind* and *Consensus* were used to gain a broad overview of the oil spill modelling field. These tools helped to familiarise with the domain, highlighting recurring terminology, modelling approaches, and behavioural concepts. This step served purely as orientation and to assert whether there were any missing points of view in the following search.

A structured search was then conducted using Google Scholar with the query: `allintitle: "oil spill" AND ("review" OR "survey")`. This yielded 122 results. After applying the inclusion/exclusion criteria, 13 articles were selected. One additional article was retrieved using the ResearchRabbit platform, and two more were manually included due to their relevance to geospatial data formats and standards for digital twins.

The selected articles served as the primary evidence base to answer the research questions.

2.3.5 State of the Art

2.3.5.1 Research Question 1: What Are the Behaviour Patterns in Oil Spills?

Oil spills interact with inanimate components like the sea, shoreline, atmosphere, and ice, as well as with living elements such as wildlife and habitats [46, 49, 58]. These interactions happen in diverse scenarios: offshore, onshore, in rivers, or on ice [58].

The behaviour patterns of oil spills are governed by a series of time-dependent physicochemical processes that transform oil and determine its fate including spreading [21, 49, 58], evaporation [21, 49, 58], entrainment and droplet-size formation [49], dispersion [21, 58], emulsification [21, 49, 58], dissolution [21, 49], photo-oxidation [21], biodegradation [21, 49, 58], sedimentation and sediment–oil interaction [21, 49], sheening [63], ice-related encapsulation and migration [58], transport at the surface and subsurface [49], turbulent mixing [21], and resurfacing [21]. These processes show that an oil spill is not a uniform or static entity, but a dynamic and heterogeneous actor shaped by continuous physicochemical transformations. Depending on the scale or objective of analysis, it can be observed at multiple levels of granularity—ranging from dispersed droplets to large surface slicks [49].

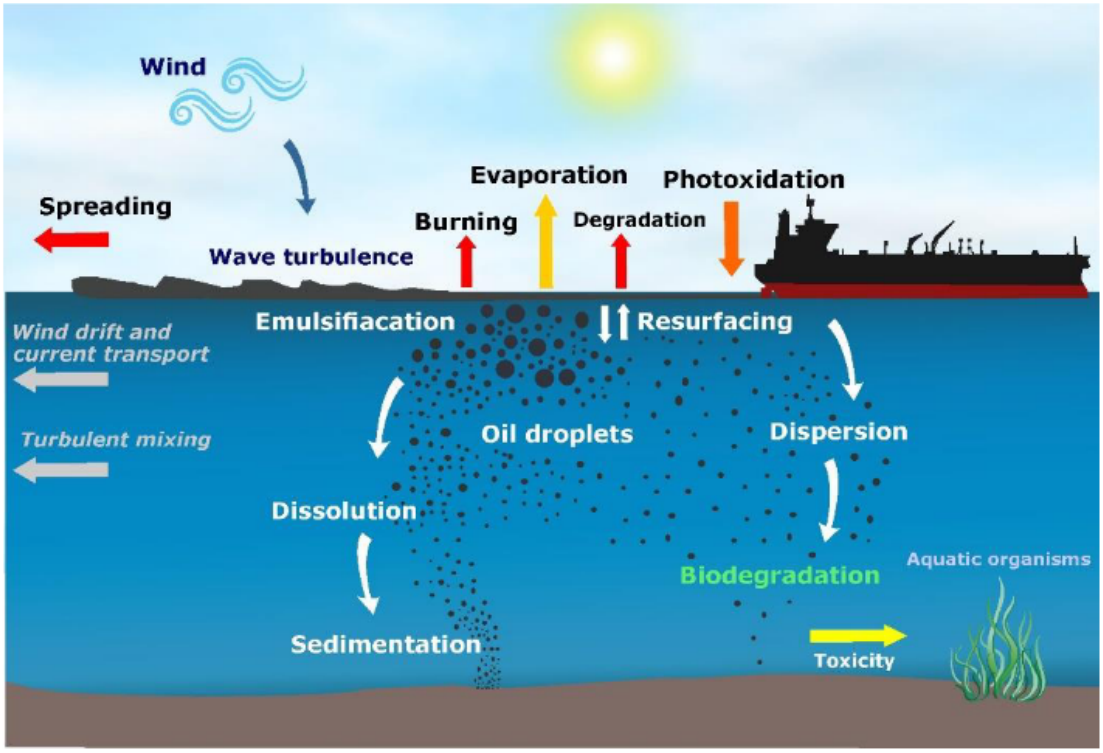


Figure 2.2: Oil spill interactions and behaviours [21]

In addition to these physicochemical processes, there are also behaviour patterns of a different nature, marked by the interaction of the oil with radiation and sound, which, in contrast to oil spill simulators, is particularly useful for remote sensing analysis. [9].

These behaviours are not fixed. The processes evolve over time, constituting patterns that progressively alter the properties and environmental interactions of the oil [58].

Finally, other types of behaviour patterns include human-made interactions meant to mitigate these events, such as in situ burning and mechanical containments.

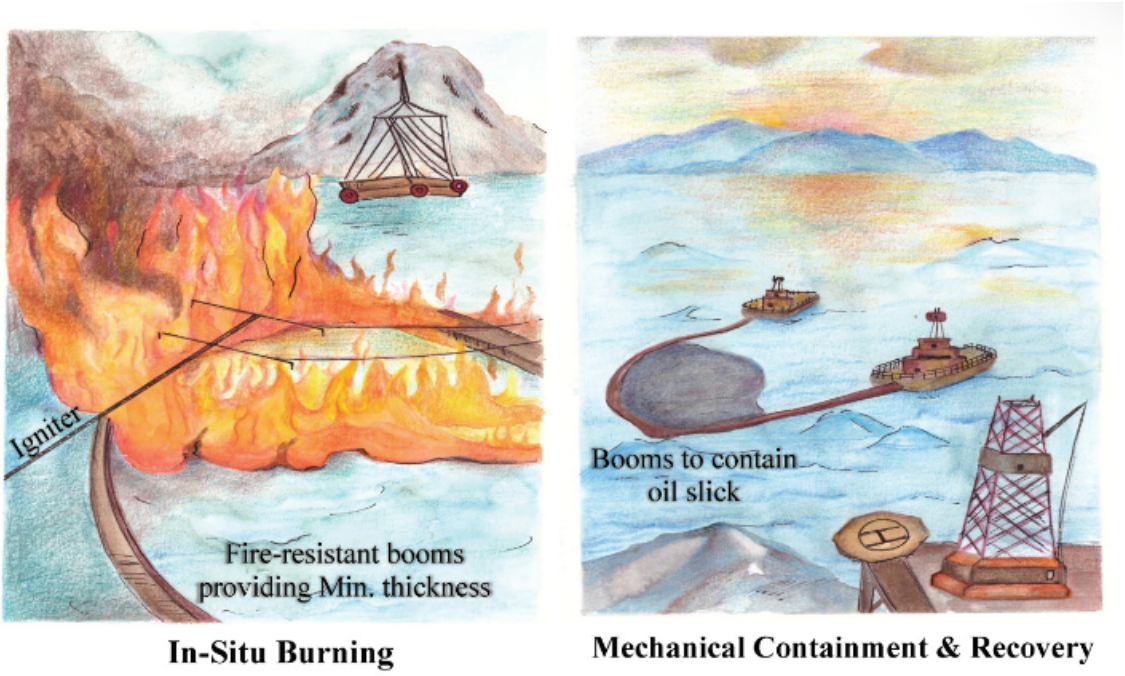


Figure 2.3: Human made interactions with oil spills [31]

2.3.5.2 RQ2: How are oil spill behaviour patterns modeled?

The modelling of oil spill behaviour patterns generally follows two broad and distinct approaches: one grounded in the physical simulation of the oil's dynamics in the marine environment, and another that relies on the extraction of behavioural patterns from observational data, especially through remote sensing.

Physical Modelling of Oil Spill Behaviour As synthesized in Keramea et al. (2021) [21], physical oil spill models are commonly categorised into two main types: Eulerian and Lagrangian approaches. Eulerian models typically rely on mathematical equations to model the oil slick as a continuous entity.

In contrast, Lagrangian models discretise the oil slick into numerous individual particles or elements, referred to as Lagrangian Elements (LEs), which are tracked through space and time. These particles interact with wind, waves, and ocean currents, simulating oil transport in a granular and temporally explicit way. The position and fate of each particle depend on the interactions with physical mechanisms. This approach has been particularly suitable for computationally efficient simulations and is more aligned with scenarios where quick forecasting is needed compared to Eulerian models. [21].

Lagrangian-based methods also enable three-dimensional modelling of oil transport and fate processes, applicable both at the ocean surface and subsurface levels [49]. Each LE represents a discrete mass of oil tracked over time, with physical properties such as radius, thickness, and composition subject to change due to environmental forcing and other LE. Importantly, this modelling

approach grants high granularity, in the sense that not only is the oil treated as a set of discrete entities, but these entities can themselves be decomposed into subcomponents, allowing simulation of transformation processes with high resolution [49].

Temporal interaction is central to these models. As Spaulding (2017) notes, the evolution of oil spill behaviour is inherently time-dependent, and many current implementations incorporate time-based functions to predict LE dynamics. Some recent developments apply discrete-particle simulation strategies, like Smooth Particle Hydrodynamics (SPH), to better handle interactions among elements and with complex free-surface flows. This shows a clear trend in oil spill modelling: moving towards more granular, discrete, and interaction-based paradigms [49].

Nonetheless, a critical limitation of many Lagrangian-based simulators is their limited capacity to simulate interaction between LEs—particularly in regions of thick oil near the source of the spill. To overcome this, newer models propose interaction-aware mechanisms between LEs that aim for convergence to physically consistent results as the number of elements increases [49].

Finally, hybrid Eulerian-Lagrangian approaches have also emerged, combining the strengths of both frameworks to simulate oil slick dynamics more comprehensively [31].

Feature based modelling from Observational Data Another modelling direction—emphasised in multiple recent reviews is the use of remote sensing and data-driven techniques to infer meaningful attributes from observational data, particularly Synthetic Aperture Radar (SAR) imagery.

As synthesised in [17, 31, 46], these techniques rely on categorising oil slick characteristics into well-defined feature groups:

- **Geometrical features**, such as area, perimeter, shape complexity, and ratios;
- **Physical/statistical features**, such as the mean, maximum, standard deviation, and contrast ratios of backscatter intensity values;
- **Contextual features**, such as proximity to ships or oil platforms and adjacency to other slicks;
- **Textural features**, usually derived from the gray-level co-occurrence matrix (GLCM), including entropy, energy, contrast, and correlation;
- **SAR polarimetric features**, such as the degree of polarisation, alpha angle, co-polarised phase difference, and various coherence and conformity coefficients.

These features are not merely descriptive but can also form the basis for defining higher-order indices and abstract behavioural parameters. For example, indices such as VV channel intensity, entropy, or ellipticity are used to support oil spill detection, characterisation, and classification tasks as stated in this review [31]. In this context, behaviour modelling consists of generating a profile of the oil slick based on indicators, rather than simulating its movement through physical equations.

Overall, this pattern-extraction approach represents a shift towards data-driven modelling, where oil spill behaviour is inferred from observed characteristics, rather than from physics principles. It complements the simulation-based strategies by offering concrete indicators of real-world spill conditions, which can be essential for rapid response and empirical validation of models.

2.3.5.3 RQ3: How are oil spill behaviour patterns identified/created?

The creation of oil spill behaviour patterns predominantly relies on simulation processes that integrate external environmental data into numerical or AI-driven models. These models are not self-contained; they are seeded with inputs from supporting systems such as hydrodynamic, wind, and wave models, as well as from measurement sources like HF radar or offshore buoys. This dependency on external data is fundamental to their predictive capacity. To produce realistic patterns, models are configured using a variety of variables, including oil type, initial spillage conditions, sea temperature, salinity, density, and bathymetry. Some systems even accept artificial or hypothetical inputs provided by users through interfaces, extending their capability beyond real-time or sensor-driven data, to support hypothetical scenarios. [21, 49, 58].

Generated oil spill behaviour patterns are not deterministic by default. Most models incorporate uncertainty evaluation mechanisms, either through built-in algorithms that perturb input fields like currents and winds or through ensemble runs that output multiple possible spill trajectories for the same initial conditions. This stochastic output is vital for decision-making, as it provides stakeholders with a range of plausible futures instead of a single deterministic trajectory [20, 21].

Additionally, oil spill simulations can run in both deterministic and stochastic modes. The stochastic mode, in particular, generates multiple parallel sequences of possible events. This is not just a technical feature, but a strategic necessity: it allows emergency response frameworks to prepare for multiple potential impacts [21].

Beyond classical numerical simulation, alternative methods using artificial intelligence are emerging. For instance, one line of work (Janati et al., 2020 [18], as cited by [31]) uses validated numerical models to generate synthetic datasets under a variety of environmental conditions. These datasets are then used to train artificial neural networks (ANNs), which can later predict oil spill spread patterns with less computational cost. Such approaches mark a shift from direct numerical computation to data-driven predictive modelling.

Similarly, ANNs have been employed to predict the probability of oil slick presence in specific regions, showing that behaviour pattern prediction can also be approached probabilistically using machine learning, not just through physical or empirical modelling [28].

Oil spill detection can also rely entirely on real-world data rather than simulations. This process starts with collecting information from multiple sensing platforms: Synthetic Aperture Radar (SAR), UAV imagery, passive microwave sensors, thermal infrared cameras, and laser fluorosensors [9, 31, 55]. SAR stands out as the dominant technology, effective even in ice waters [17, 58].

These sensing methods directly observe oil spill events, allowing detection and mapping without the need for simulation models. Techniques such as image segmentation, edge detection, and

thresholding are applied to extract oil slick boundaries from sensor data [9, 15]. Estimations of parameters like thickness or volume can be derived through passive microwave readings or thermal infrared imaging [9, 19].

Based on recent literature, machine learning, especially deep learning, has become increasingly prevalent in oil spill detection, outperforming traditional algorithms by effectively extracting complex features from SAR and optical imagery, enabling more accurate, automated, and scalable monitoring solutions [15, 17, 54].

These data-driven methods represent a fundamentally different logic than simulation: instead of modelling potential outcomes based on predefined inputs and physical laws, they infer oil presence and behaviour directly from sensor data.

In contrast, oil spill behaviour patterns extraction increasingly relies on hybrid approaches that combine image-based classification with numerical modelling to better understand and predict spill behaviour. For instance, Yang et al. [59] extract binary masks of oil slicks from Sentinel-1 SAR data using a deep learning approach, then feed these masks into numerical models to simulate spill trajectories. This approach shows a clear integration of detection and predictive simulation. As Spaulding [49] explains, numerical models predict oil dispersion by initialising particle flows using data from high-frequency radar and satellite feature tracking. These inputs are critical to accurately simulate the complex dynamics of oil spills. This combination enables real-time monitoring systems not only to detect spills but also to forecast their evolution, delivering actionable intelligence for mitigation and response efforts.

2.3.5.4 RQ4: What are the representational languages, technologies and formats for oil spill behaviour patterns?

Lagrangian oil spill behaviour models employ a representational construct that models the behaviour patterns of oil spills in spillets that are tracked through time:

Lagrangian Elements:

- Oil is discretized into *spillets* or Lagrangian Elements (LEs), each tracking a fixed mass or volume parcel through time [49].
- Surface and subsurface LEs carry properties such as radius, thickness, composition and optional subcomponent breakdowns [49].
- Fate models track LE-based metrics: floating oil dimensions, hydrocarbon concentrations in water, and fluxes to atmosphere [49].

2.3.5.5 RQ5: How are behaviour patterns presented and visualized?

Oil spill behaviour patterns are commonly presented through a range of geospatial visualization methods.

Trajectory outputs from predictive simulations are often plotted over maps to support the design of emergency reaction strategies [20]. These are frequently displayed within Geographic

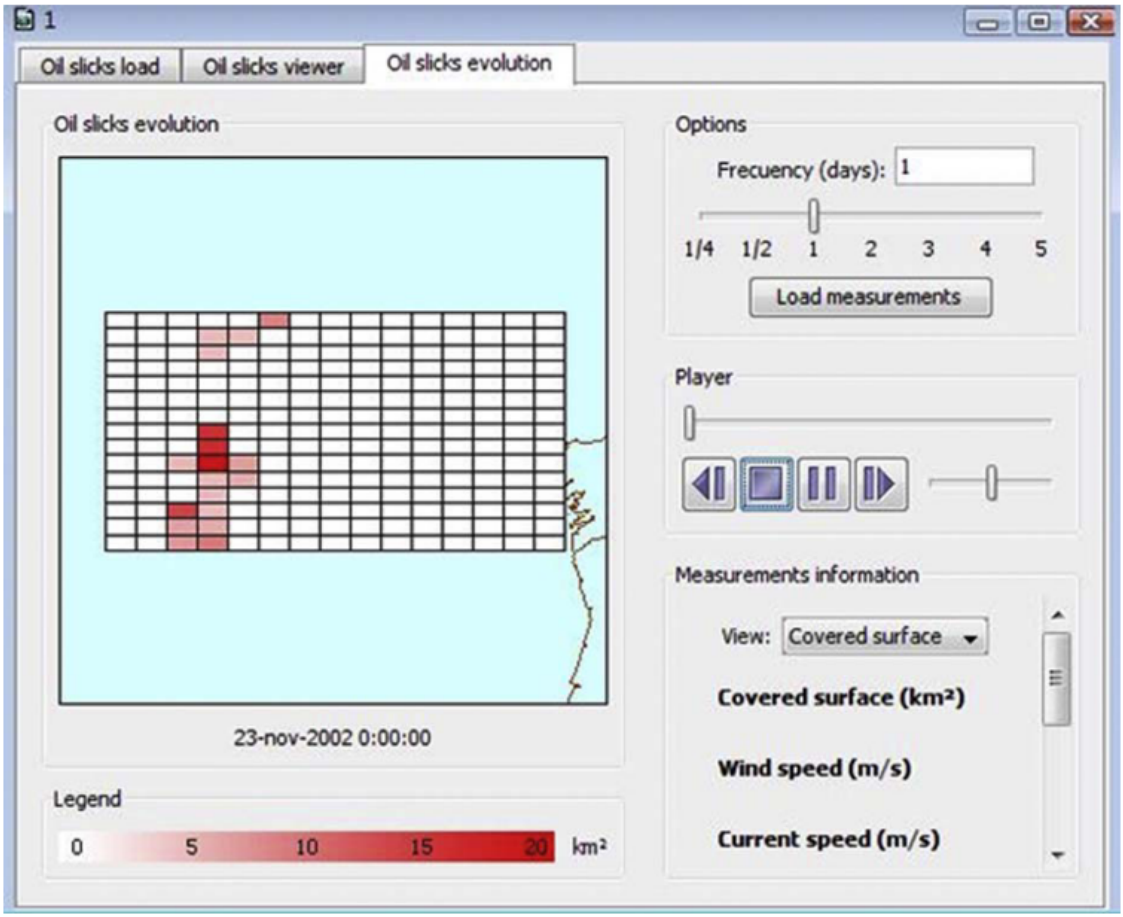


Figure 2.4: Revision window used to assess prediction results [28]

Information Systems (GIS), which serve as the primary platform for model output visualization [21]. GIS tools are also used to eliminate false detections in radar imagery, such as interference from land or near-shore areas [9].

Deep learning models such as convolutional neural networks (CNNs) enable various forms of visual annotation, including classification labels, bounding boxes, and probability maps, highlighting the location and extent of detected spills on image tiles [46].

Some operational systems offer real-time displays of sensor parameters and map outputs, as well as additional visualisations like slick thickness profiles derived from microwave sensors [9]. More advanced decision-support systems allowing expert users to assess prediction confidence [28].

Finally, tools like SMIS integrate features such as spill animations, concentration-versus-depth plots, and GIS-based risk analysis, enabling dynamic and multi-dimensional visualisation of spill behaviour [27].

2.3.6 Discussion

In light of the previously described state of the art and the established theoretical basis of virtual choreographies, we can examine how virtual choreographies align with and address certain gaps identified within the oil spill literature.

In relation to data standards and formats, the geospatial community has developed solutions to harmonize data according to FAIR principles.

- **STAC:** A catalog and API specification that makes spatiotemporal assets discoverable and accessible [45].
- **NetCDF:** A portable, platform-agnostic format for interoperable multidimensional scientific data [41].
- **GeoTIFF and Shapefile:** GeoTIFF stores georeferenced raster imagery [42], while Shapefile supports vector data—both natively integrable in GIS for visualization and analysis [8].
- **Parquet/GeoParquet and Zarr/GeoZarr:** Columnar and chunked data formats that enable scalable, cloud-native storage of raster and vector datasets [61].
- **Cloud Optimized GeoTIFF (COG):** A variant of GeoTIFF designed for efficient access in cloud environments [61].
- **GeoJSON, CovJSON, and KML:** Web-friendly formats for sharing vector and coverage data in browsers and mobile apps [2, 3, 26].
- **Ocean Information Model (OIM):** A data model and service framework ensuring interoperability among ocean science datasets and platforms [1].

While enabling interoperability and excelling at structured access to spatiotemporal data, the formats and standards mentioned above do not encode sequences of events that involve entities, relationships, or interactions.

The closest approach to the virtual choreography concept in this domain is found in Lagrangian simulations, where discrete entities (e.g., particles) evolve according to physical interactions. However, these models are physics-driven and non-semantic, not explicitly representing events, actions, or structured interactions between meaningful entities. These models are more closely aligned with simulation than structured information modelling, as they prioritise dynamic physical evolution over explicit semantic representation.

On the other hand, virtual choreographies bring the potential to represent behavioural aspects that are not captured by existing data formats and simulations.

The representation of oil spill behavioural patterns as virtual choreographies can significantly enhance monitoring and response systems. Choreographies explicitly model dangerous or critical events, transforming raw data into actionable insights. For instance, events like spill expansion or

the interaction of an oil spill with landmasses or ecosystems can be formally represented, triggering appropriate warnings or management actions.

Multiple parallel sequences can represent stochastic scenarios produced by oil spill simulations. This capability allows comparative analysis across model outputs or even against observed behaviours captured by remote sensing platforms.

Furthermore, virtual choreographies enable explicit modelling of oil spill fate and transport processes as interactions between distinct entities (e.g., oil, ocean currents, atmosphere, wind, and waves), similarly to how Lagrangian elements operate, but at the representation level. Such formalisation exposes the underlying behavioural logic within complex oil spill models and data algorithms.

As virtual choreographies are platform-independent, they can also serve as interoperable bridges among diverse data visualisation tools, ensuring consistent representation while preserving rigorous data integrity.

Virtual choreographies also support declarative enhancements, allowing hypothetical or non-existent elements to be interactively added, thereby facilitating scenario-based analyses. Furthermore, they permit significant flexibility in entity assignment, integrating heterogeneous elements such as physical objects (e.g., boats with specific attributes) and scientific geospatial entities (e.g., spill areas with hydrodynamic properties).

Interoperability in data provenance is another key advantage. Given the diverse methodologies present in the oil spill domain, including Lagrangian, Eulerian, and hybrid simulation models, as well as artificial intelligence-driven methods, the resulting heterogeneous datasets can be harmonised into consistent behavioural patterns through virtual choreographies.

Moreover, virtual choreographies can serve as an interoperability layer within digital twins, integrating diverse visualisation and analytical components through standardised behavioural patterns independent of specific data generation methods or tools.

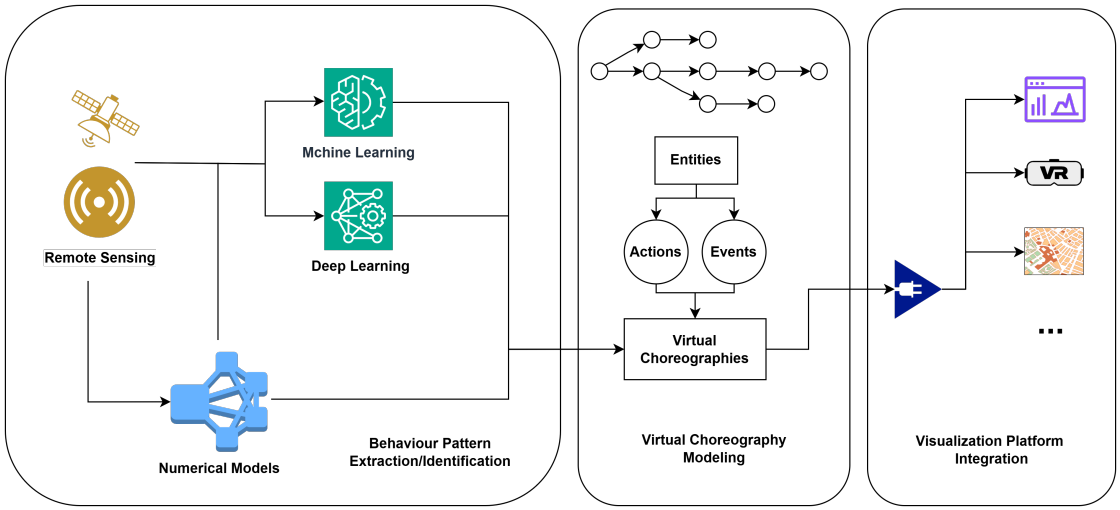


Figure 2.5: Virtual choreography fit into Digital Twins

Chapter 3

Methodology

This project follows a Design Science Research (DSR) methodology. The goal was not to passively observe but to build an artefact that solves a concrete problem in platform integration using virtual choreographies. The work was carried out within the scope of a real-world field case provided by the project stakeholders, anchoring the research in practical constraints and domain-specific challenges in the oil spill monitoring field. The process was iterative and artefact-centric, moving through cycles of design, implementation, evaluation, and reflection. Each iteration exposed flaws, led to refinements, and generated actionable knowledge.

3.1 Design Science Research

This research adopts the *Design Science Research* (DSR) paradigm, which is grounded in the principle that knowledge can be generated through the design, development, and evaluation of artefacts intended to solve real-world problems. As Hevner et al. articulated, DSR bridges scientific inquiry's rigour with practical application's relevance, positioning itself within a pragmatic perspective [14].

To structure this research, we follow the *Design Science Research Methodology* (DSRM) proposed by Peffers [37], which defines a coherent six-step process: (1) problem identification and motivation, (2) definition of solution objectives, (3) design and development, (4) demonstration, (5) evaluation, and (6) communication. This roadmap provides methodological guidance allowing for iterative refinement and responsiveness to evolving design needs. Figure 3.1 outlines this research process and its alignment with the phases of DSRM.

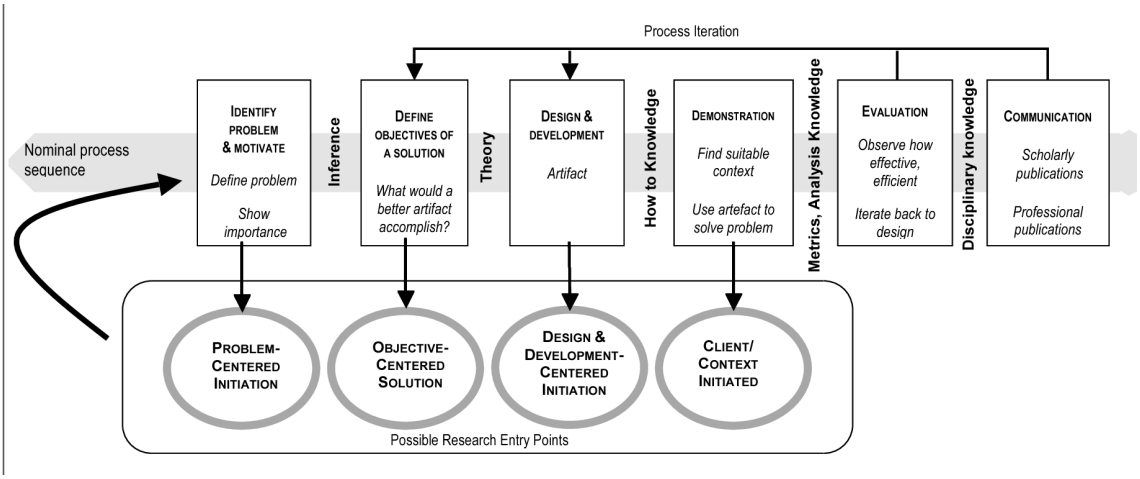


Figure 3.1: Design Science Roadmap [37]

1. Problem identification - This initial phase involves recognising and clearly articulating the research problem that needs to be addressed. It requires a thorough understanding of the current state of knowledge in the field, identification of gaps or limitations in existing solutions, and justification for why the problem is important and worth investigating. The problem identification phase establishes the foundation for the entire research endeavour by defining what needs to be solved and why it matters to both practitioners and the academic community. This phase often involves extensive literature review, stakeholder consultation, and analysis of real-world challenges to ensure the identified problem is both relevant and significant.

2. Define Objectives - This phase establishes specific, measurable goals that the research aims to achieve. These objectives serve as targets that guide the entire research process and provide clear criteria for evaluating success. The objectives must align with both the identified problem and the projected contributions to the field.

3. Design and Development - This phase encompasses the creation of artifacts that address the research objectives. This step requires careful consideration of theoretical foundations while focusing on practical implementation.

4. Demonstration - This phase involves testing the developed artifacts in real-world scenarios. Through practical application and testing, researchers gather evidence about the solution's effectiveness. The demonstration phase provides data about how the artifact performs under real conditions.

5. Evaluation - This critical phase analyses how well the demonstrated solution meets the established objectives. It involves systematic assessment of results against predefined criteria, comparison with existing solutions, and identification of both strengths and limitations. This evaluation determines whether the research has successfully addressed its initial goals or requires further iteration.

6. Communication - The final phase involves documenting and disseminating the research findings to the academic community.

3.2 Application of the Design Science Research Methodology

Applying the Design Science Research Methodology (DSRM) in this work follows the six-step process proposed by Peffers et al. [37]. The general problem and motivation were introduced in Chapter 1, while the problem identification emerged from the literature review conducted in Chapter 2. The research objectives were formulated to address the identified problem, constrained by the specific conditions of the field case, and are explicitly stated in the current chapter outlining the methodological framework.

The subsequent phases of the methodology, including design, implementation, evaluation, and communication, are depicted through the three iterative research cycles conducted in this work.

3.3 Problem and Objectives

The integration of virtual choreographies for the representation and monitoring of oil spills across multiple platforms remains an unresolved challenge. While the concept of virtual choreographies and similar applications in other domains is well documented, there is no clear reference or consolidated solution addressing complex environmental cases of oil spills. These events are characterised by their dynamic spatial and temporal behaviour, along with specific physical properties, demanding behavioural models capable of capturing this complexity [49].

In this context, this work aims to:

- Develop a conceptual model of virtual choreographies capable of representing oil spills' behaviour patterns, including their spatial, temporal, and physical dimensions.
- Apply the concept of virtual choreographies to generate and integrate oil spill behaviour patterns across multiple monitoring platforms.
- Validate the applicability of the general concept in a concrete case with specific constraints defined by the field case introduced in the first chapter 1.

3.4 Artefact Development and Evaluation

The artefact development proceeded through three iterative design cycles, each progressively refining the conceptual model and its technical implementation. The concept was tested using two distinct oil spill scenarios:

- A static oil spill scenario, characterised by a fixed geometry that remains unchanged over time.

- A dynamic oil spill scenario, composed of five distinct density layers, each with its own geometry evolving temporally.

The first two iterations were evaluated against two monitoring platforms: the Unity-based platform and the Cesium-based platform. During the third iteration, the ThreeJS-based platform was introduced, making the concept evaluation more robust by expanding the scope of integration testing.

Throughout the design cycles, evaluations focused on two key aspects: comparing the capabilities of the information models to address the essential behavioural patterns that can be obtained from the datasets, and assessing whether each monitoring platform could represent the information of the choreography within its limits. In the last iteration, the objectives shifted towards a verification relying on the choreography manifest and not just the choreography itself, as that was conceptually more aligned with established previous work.

Evaluation was conducted using a narrative research method [30]. This approach collected insights from multiple sources, including notes, meeting videos, and discussions, to chronologically and clearly structure the assessment process for each prototype in a narrative style. Given the dynamic and collaborative nature of the project, involving an external research team and two supervisors, this method enabled an adaptive and reflective evaluation throughout the design cycles.

Chapter 4

Theoretical background

This chapter outlines fundamental theoretical concepts from software engineering that will be utilised to create the artefacts in the design science research cycle. Key concepts include model-driven architecture (MDA) and specific elements within MDA, such as templates. Additionally, the chapter emphasises data flow and plugin architecture as foundational aspects of the prototype's architecture, along with essential concepts for multi-platform integration of virtual choreography.

4.1 Model Driven Architecture and Templates

Model Driven Architecture (MDA) provides a systematic approach to software development by introducing distinct abstraction levels: Computation Independent Models (CIM), Platform Independent Models (PIM), and Platform Specific Models (PSM) [43]. This separation enables platform-agnostic application definitions, improving portability through model transformations.

MDA supports a variety of transformations, such as PIM-to-PSM, PSM-to-PSM, and PSM-to-text, which allow derivation of platform-specific artefacts from abstract models [43]. The PSM-to-text transformation is particularly relevant, as it provides a direct path from models to executable code or other textual artefacts.

The specification of modelling languages within MDA relies on meta-modelling languages [43]. These technologies define the structure and semantics of modelling languages themselves, serving as a foundation for both model validation and transformation logic.

Template-Based Code Generation (TBCG) complements MDA by offering a synthesis mechanism to transform structured input into textual output using templates. According to Syriani et al. [51], TBCG consists of three primary components: the run-time input (data), the template (which includes both static and dynamic parts), and the textual output. However, the authors highlight a crucial fourth component, which is the meta-information of the run-time input, the template is expecting. A template is an abstract specification of the desired output. Its static part includes text fragments that appear unchanged in the result, while the dynamic part embeds meta-code that is interpreted by a template engine. This engine evaluates the meta-code against the run-time

input, guided by the expected run-time input meta-information. By combining structured templates, meta-information, and data, TBCG enables the generation of artefacts from models. It plays a central role in model-driven processes that require consistent and automated translation from high-level models to implementation-level outputs.

4.2 Model-Driven Architecture and Virtual Choreographies

A key limitation in the technological orchestration of interactive narratives lies in the coupling between information logic and platform-specific implementations. One promising strategy to bypass this constraint is the application of Model-Driven Architecture (MDA) to virtual choreographies, as proposed by Silva et al. (2014) [47]. Their approach leverages MDA principles to separate high-level conceptual models, described through ontologies, from the technical specifics of their deployment in virtual environments.

Building on this foundation, Lacet et al. (2020) [24] extend the MDA paradigm to address narrative choreographies not as descriptions of events that occurred, but as possible ingredients that prefigure narrative flows before their enactment. This perspective positions choreographies as narrative blueprints, capable of guiding system behaviour across heterogeneous platforms.

To achieve this concept, Lacet et al. [24] introduces a dual-recipe system grounded in the xAPI specification. A distinction is made between choreography recipes, which define the narrative's semantic structure, and platform recipes, which describe the semantic space of a specific implementation platform. A semantic mapping layer links these recipes bidirectionally: a single choreography recipe can be translated into multiple platform recipes and vice versa 4.1. This abstraction layer allows for automatic transformations between semantic spaces, enabling platform-agnostic deployment of interactive narratives. Each platform recipe contains its own semantic affordance. Therefore, mappings are constrained by the expressiveness and requirements of each platform's semantic space. In this architecture, interoperability is achieved not through standardisation of platforms, but through explicit, model-driven translation mechanisms.



Figure 4.1: Mapping between recipes [24]

4.3 Dataflow Architecture

Dataflow architecture centres on the representation of computations as graphs, where nodes are operations and arcs represent the flow of data between them. Unlike traditional sequential programming, dataflow models do not prescribe an explicit order of execution. Instead, the execution is governed by the availability of data [60], a quality that renders them particularly suitable for visual programming as shown in Figure 4.2.

These graphs are typically directed acyclic graphs (DAGs) and are commonly referred to as dataflow diagrams (DFDs). Each node in the DFD encapsulates a computation or transformation that is independent from all other nodes, while arcs represent the transfer of data. This fundamental shift decouples the logic of computation from any rigid sequential flow [60].

A key characteristic of the dataflow paradigm is that DFD designers operate directly on streams of data, manipulating them as they propagate through functions, diverge into branches, or recombine into more complex structures. Variable declarations, or naming conventions, become secondary; what matters is the evolution of data as it transforms through the network. This structure also allows DFDs to be composed dynamically on top of other DFDs or further refined and iterated in order to fit different solution spaces [60].

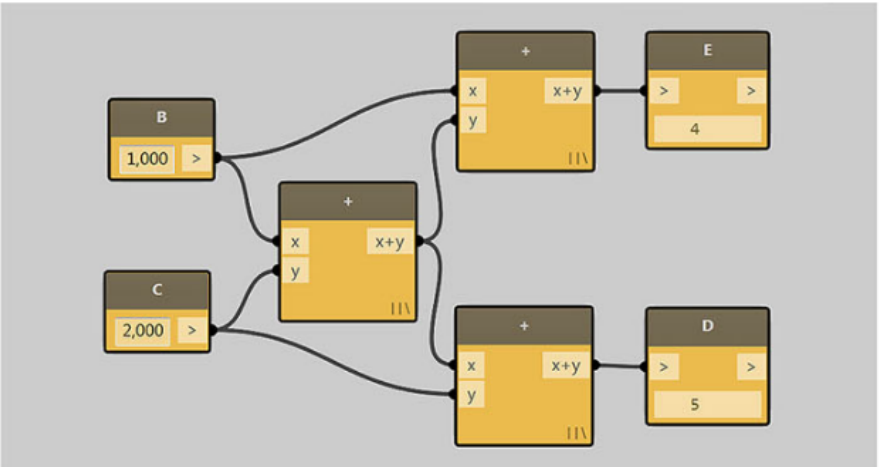


Figure 4.2: DFD example [60]

4.4 Plugin Architecture

Plugin-based design is founded on the idea that plugins provide additional functionality beyond a core product's features [7]. Expanding the system involves simply implementing new plugins and registering them with the application, where there is no need to understand or modify the core source code [50].

Plugins can vary in scale and are treated as essential units within a broader abstraction for extensible systems, referred to as Pluggable Systems [50]. This abstraction formalises a pattern where the core system remains separate from its extensions through the introduction of a plugin manager [50].

As described by Syeed et al., a pluggable system consists of three primary components: a domain-specific core, a plugin manager, and a set of plugins [50] as shown in Figure 4.3. The plugin manager handles registration, loading, and integration of plugins, acting as the intermediary between the core and the extensions.

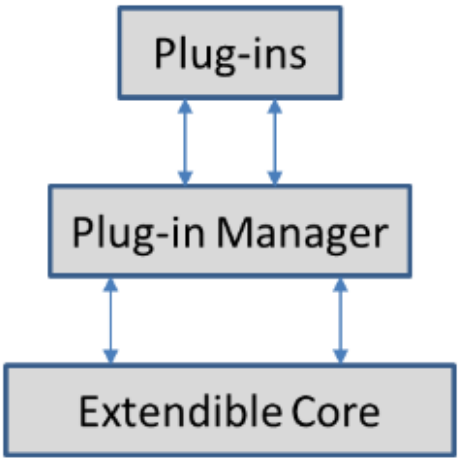


Figure 4.3: Plugin Architecture [50]

Chapter 5

Iteration 1 - VChor Framework

This chapter outlines the first approach and iteration of the design and prototype solution. The first solution is called the VChor Framework, as this iteration resulted in various artefacts that address different challenges, including choreography generation and validation, choreography transformation, and recipe matching.

5.1 Design

5.1.1 Information Model of Virtual Choreographies

The proposed information model for virtual choreographies establishes a structured and semantically rich description to represent dynamic interactions between entities over time. At its core, the model distinguishes between entities—namely actors and objects—and the actions they perform.

Entities are defined by their types, which determine the set of properties they may possess.

This distinction allows the model to express both static and time-varying aspects of entities involved in complex phenomena.

Inspired by the state of the art in virtual choreographies, the model also supports inter-entity references, allowing nested relationships that enable multi-layered abstractions and compositional structures.

Actions are modelled as statements that explicitly link an actor, an object, and a verb—optionally extended with contextual metadata. These statements form the narrative of the choreography, describing how entities interact, change state, or affect one another over time. The structure follows a triple pattern, commonly found in semantic web literature, and is also influenced by the xAPI specification for modelling discrete actions.

Finally, the choreography component provides a temporal sequence of behaviours, each referencing specific actions occurring at given timestamps. This component binds behaviours into a coherent, time-aware structure.

A key feature of the model is its integration with external semantic vocabularies and standards. These define types, properties, verbs, and extensions, ensuring clarity, consistency, and reusability across platforms.

Overall, the model ensures:

- Extensibility via custom or external vocabulary definitions.
- Platform-agnostic representation of time-evolving interactions.
- Support for compositional and nested entity structures.

5.1.1.1 Choreography Recipe

The recipe is essentially an imprint of the vocabularies, entity definitions, and behaviour statements that a given choreography supports, similar to how Demetrius et al. [24] describes the semantic space of both the choreography and the platform. It captures the essential elements and structure used in the choreography without adding additional abstraction or logic. This recipe is crucial for the platform integration step.

5.1.1.2 Additional tools

Based on the predefined entity definitions and behavioural statements, external tools can operate over the model to support integration and validation tasks.

One such tool is the *profiler*, which infers the underlying recipe from a choreography instance. The profiler constructs a meta-model that encapsulates the structure, interaction logic, and dependencies.

Another tool is the *validator*, which ensures that a choreography conforms to the structural, syntactic, and semantic constraints defined by the model. It should perform schema validation, cross-referencing, and consistency checks, ensuring that choreographies conform to the information model.

5.1.2 Generator Architecture

To address the challenge of choreography generation, the generator model employs a dataflow architecture based on a static directed acyclic graph (DAG), illustrated in Figure 5.1.

This architecture also uses a plugin system to extend its functionality, organised into three core groups:

- **Segmentation plugins**, responsible for dividing the dataset into meaningful segments.
- **Behaviour plugins**, which detect evidence of behaviours within each segment.
- **Handler plugins**, used to map data from each segment to specific properties or extensions in the choreography model.

This architecture expects that an end-user to manually declare the definitions(actors and scenes) and behaviors() that will produced, along with all the necessary data segmentation, behaviour evidence identification, and mapping mechanism to be configured, therefore it is a manual extraction

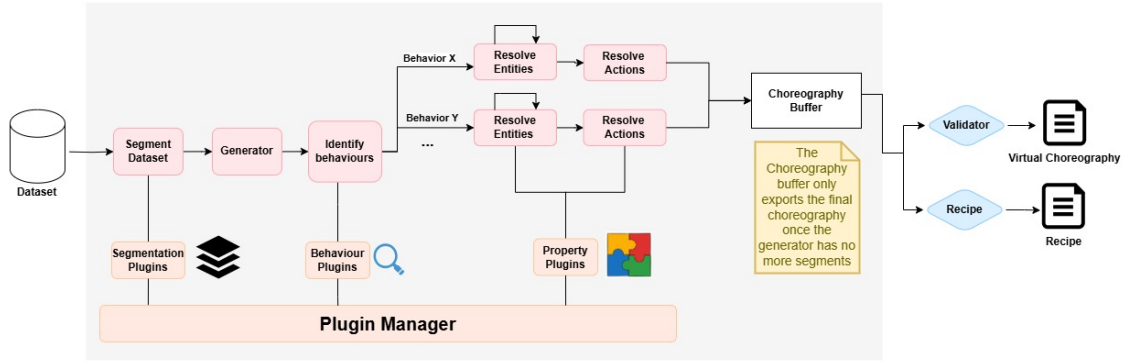


Figure 5.1: Generator architecture

of behaviour patterns, which is a common theme that was found in the literature, but that had never been previously applied to the specific context of virtual choregraphies.

The static dataflow 5.1 follows a clear operational logic tailored for generating choreographies that evolve over time. Since choreographies are inherently temporal, the first step segments the input dataset by a key variable (typically a timestamp) representing progression. Each segment is then passed to a generator that processes segments sequentially through the pipeline.

Next, within each segment, behaviour plugins identify relevant evidence to support specific behaviours. These behaviours correspond to actions and their associated verbs. Actors and objects are resolved by applying direct mappings or transformations between dataset fields and the choreography model. This mapping is handled by the handler plugins.

Additionally, because the data model supports composite entities, some entities may route data recursively to the entities that compose them, ensuring a complete resolution of the relationships.

Once actors and objects are resolved for an action, any extensions associated with that action are similarly processed via handler functions. Finally, the instantiated behaviour is stored in a choreography buffer, where actions are grouped by segment and emitted once the generator is empty.

The final choreography can then be deployed to the validator tool in order to confirm that it conforms to the information model. If it does, the profiler component infers a recipe from the choreography that can be used in the transformation steps.

5.1.3 Transformer Architecutre

The transformer architecture is based on Model-Driven Engineering (MDE), aiming to automatically and declaratively translate choreography concepts into platform concepts. This is achieved through templating techniques, which are responsible for building the output structure and injecting core information from the choreography. However, due to the inherent design of templating engines, their tools for data processing and transformation are often severely limited, as they are primarily built to display content rather than modify it.

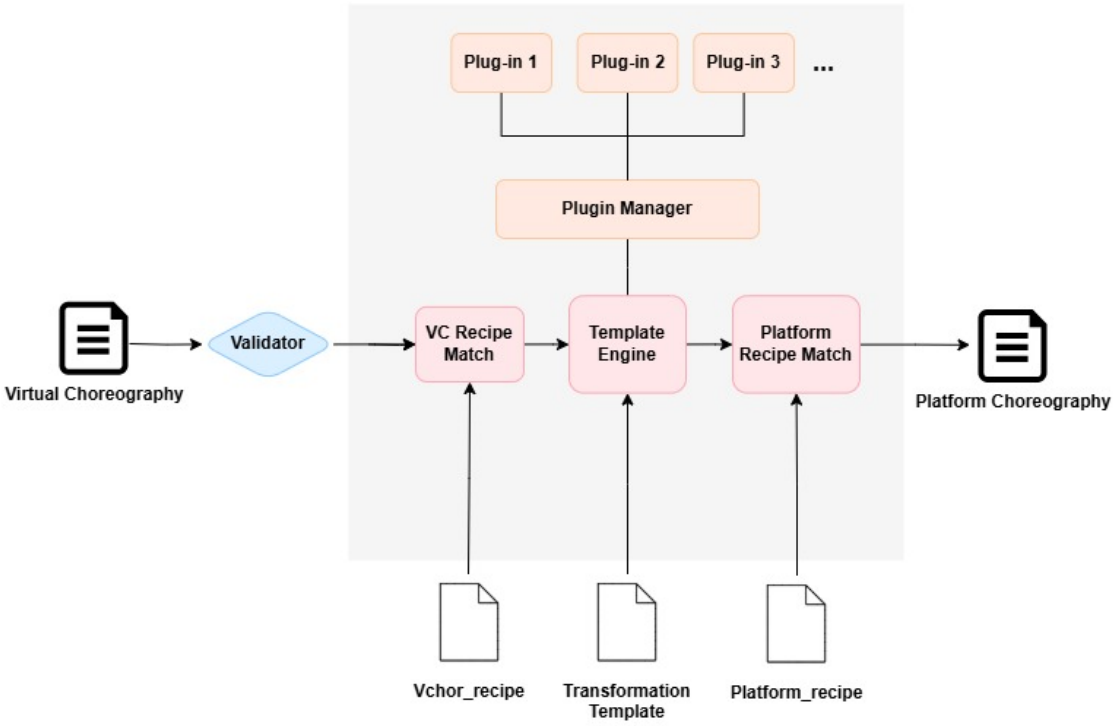


Figure 5.2: Transformer architecture

Therefore, this mechanism is coupled with a plugin architecture that provides core, customisable extensions responsible for additional processing steps that choreography elements may require before being injected into the template. One example is the Unity-based platform, which renders polygons on a technology-specific XY grid instead of using longitude and latitude coordinates. To correctly assign these coordinates—without hardcoding them into the choreography, which would undermine its platform-independent purpose—the transformation is performed within the templating engine using a plugin extension.

The templates introduced into the system to perform choreography transformations must conform both to the input choreography recipe and to the platform interface. As shown in Figure 5.2, before applying the transformation, a validation mechanism checks whether the input choreography matches the expected recipe. If validated, the transformation proceeds. Once the platform-specific choreography is generated, it is validated again against the platform recipe to ensure that it meets the platform's needs and structural requirements before being finalised.

Additionally, all input choreographies passed to this mechanism can be validated by the validator component, in the same way that the choreographies produced by the generator are evaluated.

5.2 Prototype

5.2.1 Information Model Prototype

The Virtual Choreographies (VC) format defines an abstract JSON-based information model for representing entity interactions over time. It is composed of five main sections:

Listing 5.1: Structure of a VC file

```
{
  "@version": "1.0",
  "@context": { ... },
  "@definitions": { ... },
  "@behaviours": { ... },
  "@choreography": { ... }
}
```

5.2.1.1 @context — Semantic Definitions

This section defines the context profile for the choreography. Each declaration maps a local term to a URI, linking the term to a concept in an external vocabulary.

- @types: Declares the actor and object types (e.g., entities like oilspills, persons, abstract concepts).
- @properties: Declares the properties that can appear in entities.
- @verb: Lists the permitted actions in the form of verbs.
- @extensions: Optional extensions used to encode additional data for behaviour statments.

Listing 5.2: Example behaviour

```
"@context": {
    "@types": { "ActorType": "<url>", "LinkedActorType": "<url>", "SceneType": "<
        url>" },
    "@properties": { "propertyA": "<url>", "propertyB": "<url>", "propertyC": "<url
        >" },
    "@verb": { "isAt": "<url>" },
    "@extensions": { "extensionA": "<url>" }
  }
```

Validation rules:

- All keys must have non-empty string values.

- All types, properties, verbs, and extensions used anywhere in the file must be declared here.

5.2.1.2 @definitions — Entity Declarations

Entities are declared here and separated into:

- @actors: Entities capable of performing actions.
- @scenes: Scenes were the actions ocurred.

Each entity contains:

- @type: Refers to a type declared in @context["@types"].
- @const: Static fields that do not change (e.g., name, id, model).
- @state: A mapping of keys to specific combinations of dynamic properties (e.g., geometry or status).

Listing 5.3: Example behaviour

```
{
  "@definitions": {
    "@actors": {
      "actor1": {
        "@type": "ActorType",
        "@const": { "propertyA": "<val>" },
        "@state": { "t0": { "propertyB": "@actor2_t0" } }
      },
      "actor2": {
        "@type": "LinkedActorType",
        "@const": {},
        "@state": { "t0": { "propertyC": "<val>" } }
      }
    },
    "@scenes": {
      "scene": {
        "@type": "SceneType",
        "@const": { "propertyA": "<val>" }
      }
    }
  },
}
```

Validation rules:

- All entities must be assigned a type.

- All used types must be declared in @context["@types"].
- Every entity must define both @const and @state.
- All used properties must be declared in @context["@properties"].
- All dynamic references between entities (e.g., @actor2__t0) must resolve to existing entities and states if the reference entity has dynamic states(mutable entity).
- @const must not contain properties defined in the @state field.
- States must follow a consistent schema.

5.2.1.3 @behaviours — Atomic Interactions

Behaviours define the interactions between entities, following an Actor-Verb-Object statement structure:

Listing 5.4: Example behaviour

```
{
  "1": {
    "@actor": "@entity_id",
    "@object": "@entity_or_scene_id",
    "@verb": "verb_key",
    "@extensions": { ... }
  }
}
```

Validation rules:

- Each behaviour must have a unique key.
- Fields @actor, @object, and @verb are mandatory.
- If @extensions are used, their keys must be declared in @context["@extensions"].

5.2.1.4 @choreography — Temporal Execution Trace

This section defines the ordered sequence of behaviours over time. Each item maps a unique timestamp to a list of behaviour IDs:

Listing 5.5: Example choreography block

```
{
  "0": {
    "timestamp": "<ts>",
```

```
    "behaviours": [ "1", "2" ]
  }
}
```

Validation rules:

- Each item must have a unique timestamp and ID.
- Referenced behaviour IDs must exist in @behaviours.

5.2.1.5 Choreography Recipe

Technologically, the recipe leverages JSON Schema constructs to precisely define the valid ingredients of a choreography. For instance, the anyOf operator is used to declare all possible behaviour patterns or entity states, allowing flexible yet controlled variability in dynamic definitions. Constant fields rely on the const keyword to enforce fixed semantic values (e.g., a given type or property), ensuring strict typing. References between entities are expressed through predictable string patterns validated via pattern constraints. This approach makes the recipe easy for a machine to automatically validate it, acting as a contract for what a choreography instance may contain.

5.2.1.6 Additional Tools

A validator was implemented in Python as a software component to check whether a given file conforms to the validation rules of the information model, assessing if it constitutes a valid choreography.

The profiler, also developed in Python, extracts the recipe of a given choreography in JSON Schema format. It analyses references between entities, the underlying structure of entities and behaviour statements, and extracts the vocabulary associated with the choreography.

5.2.2 Generator Prototype

The generator is implemented in Python as well as the plugin system that accompanies it. They consume templates in JSON the user may configure, declaring the entities, their properties as well as behavior statements, and necessary processing steps including segmentation, behavior evidence identification and data mapping, from the input dataset to the declared choreography ingredients. Alternatively, the system also provides a UI where the user may declare the ingredients and processing steps in a hierarchical tree as shown in figure 5.3. The generator is statically connected to concepts of xAPI and schema.org, as well as custom concepts to be able to structure and semantically describe the choreography.

Through the user interface after configuring the system, the user may export the configuration as a template in JSON or immediately input a dataset to be processed. Alternatively, the system also provides a terminal interface where upon receiving a generation template and an input dataset, the program returns a validated choreography as well as the inferred recipe.

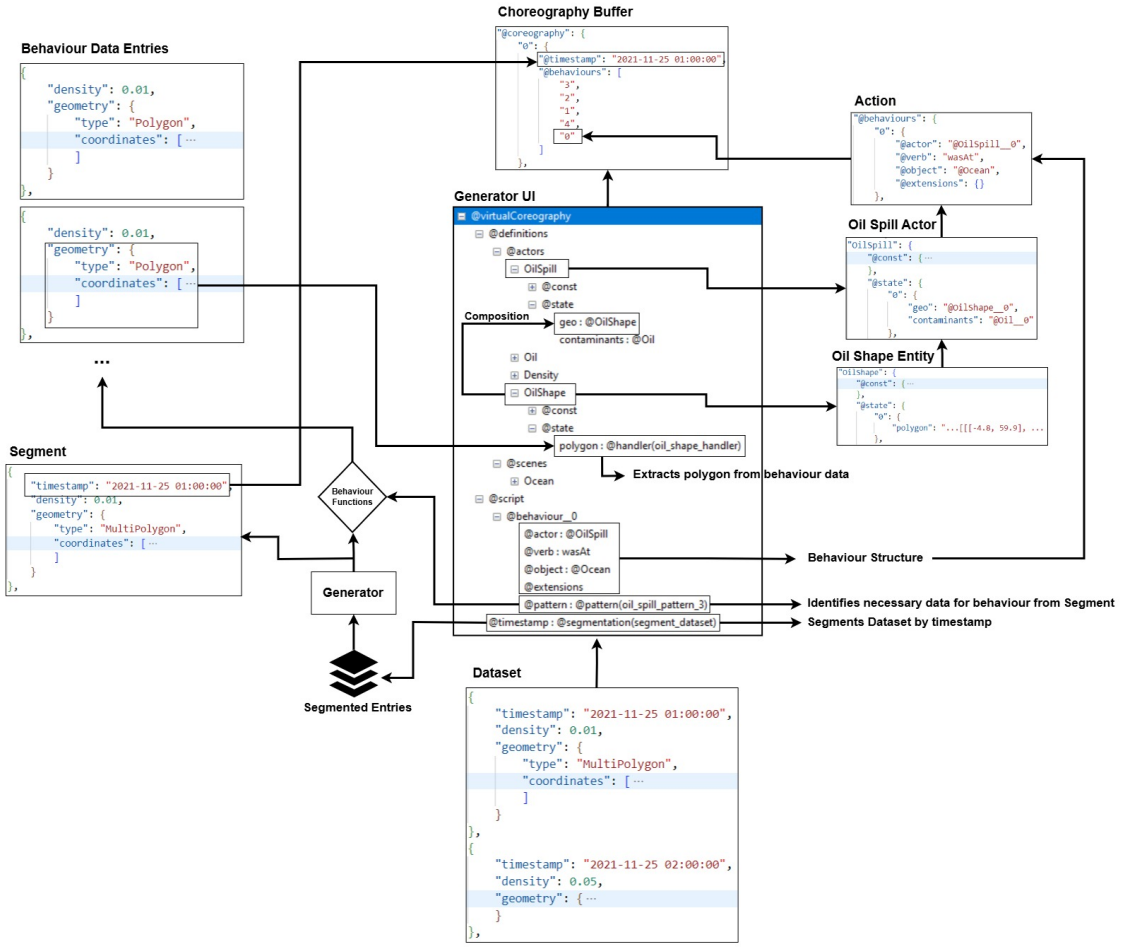


Figure 5.3: Transformer architecture

Figure 5.3 shows how the user can hierarchically describe entities and their relationships. For instance, an oil spill (which is a pollution event) has a property called geo. This geo property is associated with another abstract entity called oil shape, which is a shape type and has a polygon property. The contaminants field references the oil entity.

For this scenario the processing workflow is as follows: In the @timestamp parameter, we select our segmentation function which is applied from the plugin system to segment the dataset. Then each segment is provided one at a time by the generator, where each behavior tries to find evidence of its occurrence using the pattern handler. Finally, the actors and objects are resolved recursively, and data is mapped for each segment by the handler functions as shown in the polygon property. The final choreography is included in the choreography buffer which produces the sequence of events.

5.2.3 Transformer prototype

The transformation system needs to be configured before applying choreography translations, by providing it with three essential artifacts:

- The choreography recipe – which defines the expected structure and constraints of the input choreography.
- The transformation template – which contains the logic for converting a choreography from its generic form into a platform-specific representation.
- The platform recipe – which specifies the requirements the output choreography must satisfy to be considered valid for a specific platform.

Only once all three are loaded does the transformation system become ready to operate. When a choreography is submitted, the system first validates it against the choreography recipe. If it passes, the transformation template is applied. Finally, the resulting output is validated against the platform recipe to ensure compatibility.

The platform recipes are independent of the virtual choreography framework and, for this prototype in particular, are assumed to always be in JSON format due to constraints imposed by the field case. As such, platform recipes are essentially JSON Schemas.

The template engine used is jinja, which enables automatic construction of different view artifacts from predefined templates and enable registration of filters as extensible functionalities. A plugin manager was developed in order to automatically include python functions into the jinja registery.The following plugin extensions were included:

- resolveKey(actor): Splits an actor reference on "__" to identify the base actor name.
- resolve(data, ref_chain): Recursively resolves a string reference that may contain embedded actor states.
- searchActor(data, choreography_id, actor_name): Searches a choreography's behaviours for a specific actor reference (e.g. "@oilSpill").
- getPropertyStates(data, ref): Retrieves all state values for a given property reference such as "@actor.property".
- flag(data, flag, *ignore_keys): Flags the first item in a list and flags subsequent items when differences appear, ignoring specified keys.
- ConvertPlaneToGeodetic(polygon): Transforms geographical points from WGS84 to a projected coordinate system and returns a DataFrame.
- resolvePolygon(polygon, bbox, scale): Applies geodesic conversion, scaling, and translation to a polygon relative to a bounding box.
- parseLocation(location): Splits a coordinate string into floats and returns a (lat, lon) tuple.
- getBbox(polygons): Produces the bounding box from a list of polygons after flattening their coordinates.

5.2.3.1 Platform recipes

One significant problem encountered was that the platforms did not provide any recipes. This makes the field case particularly unique, as virtual choreographies have traditionally been applied to platforms with well-established semantic spaces. Therefore, a recipe was deduced automatically by extracting a JSON Schema from input examples for each system, ensuring that all available features were represented, using the genson Python library. The resulting schemas were then manually adjusted to handle specific cases, such as enforcing that latitude and longitude coordinates contain exactly two values, or validating colour formats through pattern definitions.

5.3 Demonstration

To demonstrate the prototype solution, two scenarios were considered:

- An example with **static oil spills** with 9 different timestamps.
- An example with **dynamic oil spills** with 9 different timestamps where the different oil spill layers are constantly changing their geometry and position.

5.3.1 Choreography and recipe generation

The first step was to extract the choreographies from the datasets. The datasets are restricted by the field case, in the way that they have similar structures, therefore, the same processing configuration was used 5.4.

We can see in the configuration 5.4 that four different actors are declared. The main (parent) physical actor is the OilSpill, which is composed of two properties: a geometry and a contaminant. These properties reference the GeoShape actor and the Oil actor, respectively. The GeoShape actor includes a polygon property, while the Oil actor references a Density actor, which contains a value property.

The defined behaviour for this choreography is the presence of the OilSpill in the ocean scene, defined by the wasAt verb. After defining all the core elements of the choreography, the dataset is segmented into different timestamps. The specific evidence matching the choreography's behaviour is then identified by the behaviour plugin (labelled as "@pattern" in Figure 5.4).

From this evidence segment, the corresponding behaviours are generated. The handler plugins resolve the actors by mapping data from the segment. The density value is assigned to the Density actor, and the geometry is mapped to the GeoShape actor. As a result, each resolved behaviour instance yields an OilSpill actor populated with its physical and geometrical properties. It is important to note that the OilSpill actor is defined by a geometry field associated with one densitiy, and multiple OilSpill instances may coexist spatially according to the choreography recipe produced by this process.

By processing both datasets using this configuration, two different choreographies are obtained. The static choreography reuses the same 5 oispill actors across all 9 timestamps with each

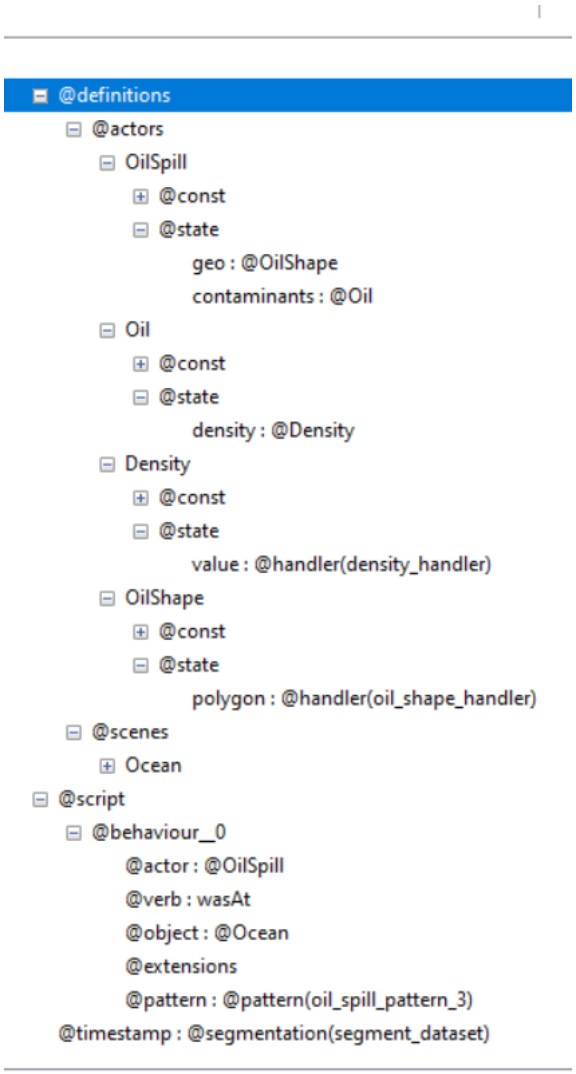


Figure 5.4: Generator Configuration

actor sharing the same geometry, whereas the dynamic choreography generates 36 distinct OilSpill actors, resulting in a non-repeating sequence of events.

Since both choreograhies share the same structure, the inferred recipe produced in both scenarios was equal, therefore we can say both choreograpies share the same recipe.

In order to integrate the choreographies into the different oil spill monitors, different transformation templates were crafted respecting the recipe of the choreograhies, but for the different platform recipes.

5.3.2 Cesium-based platform Integration

The jinja template shown below transforms a choreography into a structured JSON compatible with the Cesium platform. This process can be divided into three conceptual stages:

5.3.2.1 Action Composition

Each action is added to the root array using the following construction:

```
{% for choreography_id, choreography in data["@choreography"].items() %}
{
  "id" : "{{ loop.index0 }}",
  "timestamp": "{{ choreography['@timestamp'] }}",
  "verb": "wasAt",
  "place": "Ocean",
  "actor": [ ... ]
}
```

Here, the id is assigned incrementally, and the timestamp is extracted from each choreography entry. The verb is fixed as wasAt to indicate the actor's presence in the environment. The actor field is dynamically populated in the next stage.

5.3.2.2 Actor Processing

Within each action block, the template iterates over behaviours and extracts actors that match the OilSpill type:

```
{% for behaviour_id in choreography['@behaviours'] %}
{% set actor = data['@behaviours'][behaviour_id|string]['@actor'] %}
{% if actor | vc_resolveKey == "@OilSpill" %}
{
  {% set poly = (vc_resolve(data, actor + ".geo.polygon") | ops_loadJson) %}
  "polygon": {{ poly }},
  "name": "Oil Spill",
  "density": {{ vc_resolve(data, actor + ".contaminants.density.value") + 30 }},
  "color": "#ff690e",
  "type": "oil",
  "url": "",
  "scale": "",
  "altObj": 1
}
{% endif %}
```

This section resolves nested references to polygon coordinates and density values for the OilSpill actor using vc_resolve. The geometry is directly injected into the polygon field, while the density is adjusted to ensure compatibility with platform requirements.

5.3.2.3 Post-Processing

The following metadata guides post-processing:

```
"___metadata___": {
  "post_processors": ["ops_flag this.spill flag timestamp id"],
  "schema_version": "1.0",
  "platform": "cesium"
}
```

The ops_flag post-processor is configured to analyse the spill array and flag entries based on actor changes, excluding timestamp and id from the comparison. This enables detection of evolving states or repeated actors, as required by the platform.

5.3.2.4 Execution and Results

The transformation system is first configured by providing the recipe of the choreographies, the template for the cesium platform and the cesium platform recipe.

Using a command-line interface, both choreographies were provided to the configured transformation system, which produced the following results.

- Both choreographies were successfully transformed into the corresponding platform choreography, without any detected issues;
- When running the static oil spill choreography on the platform, the platform displayed a static oil spill with no movement or behavioural updates, as shown in Figure 5.5;
- When running the dynamic oil spill choreography, the platform displayed an evolving oil spill with different density layers, marking each timestamp as a unique behaviour, as seen in Figure 5.6.

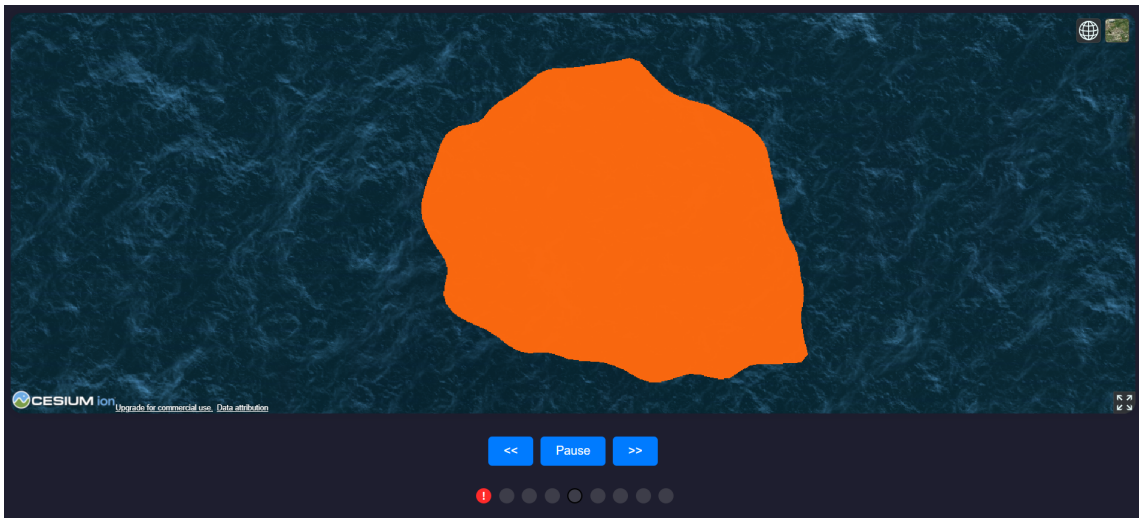


Figure 5.5: Cesium-based Platform(static oil spill)

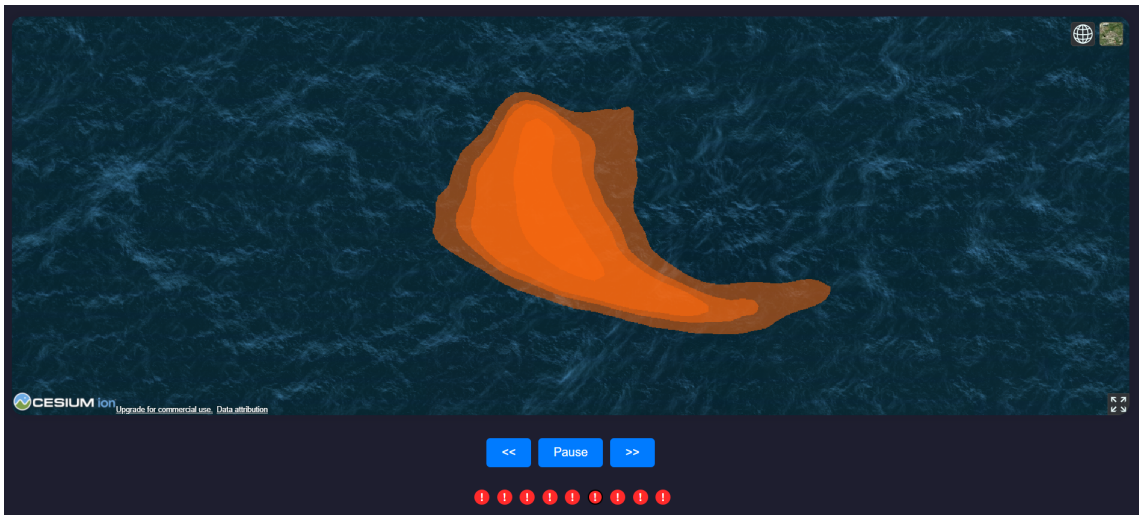


Figure 5.6: Cesium-based Platform(dynamic oil spill)

5.3.3 Unity-based platform Integration

The jinja template shown below transforms a choreography into a structured JSON compatible with the Unity-based platform:

5.3.3.1 Action Composition

Each action is constructed using the following block:

```
{% set bbox = vc_getPropertyStates(data, '@OilShape.polygon') | ops_getBbox %}
{% for choreography_id, choreography in data["@choreography"].items() %}
{
  "id" : "{{ loop.index0 }}",
  "timestamp": "{{ choreography['@timestamp'] }}",
  "verb": "wasAt",
  "color": "#ff690e",
  "texture": "",
  "place": "{{ (bbox[0][1] + bbox[1][1]) / 2 }}, {{ (bbox[0][0] + bbox[1][0]) / 2
      }}",
  ...
}
```

Each action is assigned an incremental *id*, uses the choreography's timestamp, and defines visual parameters such as color and texture. One of the key differences in this platform, compared to the Cesium platform, is that color is applied uniformly to all actors. Additionally, this platform introduces a `place` key, which is crucial for marking the location of events on the minimap.

To obtain this location in longitude and latitude, a bounding box of the scene must be calculated by applying the `ops_getBbox` function to all geometries in the choreography. This bounding box is also essential for each actor, as will be explained in the next section.

5.3.3.2 Actor Processing

For each behaviour in the choreography, if the actor type matches `@OilSpill`, a resolved object is injected using polygon and density values, as shown below:

```
{% set oilSpills = vc_searchActor(data, choreography_id, "@OilSpill") %}
...
{% set actor = data['@behaviours'][behaviour_id|string]['@actor'] %}
{% if actor | vc_resolveKey == "@OilSpill" %}
{
  {% set densities = vc_getPropertyStates(data, '@Density.value') %}
  "polygon": {{ oilSpill_resolvePolygon(vc_resolve(data, actor + ".geo.polygon"),
      bbox, 0.001) }},
  "name": "Oil Spill",
  "density": {{ oilSpill_normDensity(vc_resolve(data, actor + ".contaminants.
      density.value"), densities) }},
  "type": "oil",
  "url": "",
  "scale": "",
  "altObj": 1,
  "position": { "x": 0, "y": 0, "z": 0 }
}
{% endif %}
```

Here, the oil spill's polygon is geospatially transformed relative to the bounding box and a scale using the oilSpill_resolvePolygon plugin. This is a critical step in the transformation process, as the Unity-based platform uses a specific XY axis. The transformation from the original choreography's geographic coordinates (latitude and longitude) must be projected into this XY system and scaled to fit the platform's visual bounds. By using a shared bounding box, all actors can align to a common Cartesian coordinate system, ensuring consistent placement across timestamps. This prevents actors from being repeatedly rendered at the same position in each frame, which would hinder the visualization of the oil spill's progression over time.

Density is also a sensitive parameter in this platform. It must be normalized to a specific interval to be correctly recognized. For this reason, we use the oilSpill_normDensity plugin function to normalize all densities, ensuring visual consistency across the rendered choreography.

5.3.3.3 Post-Processing

At the top of the JSON, metadata defines how to process the output:

```
"___metadata___": {
  "post_processors": ["ops_order_density this.spill", "ops_flag this.spill flag
      timestamp id"],
  "schema_version": "1.0",
  "platform": "unity"
```

```
5 }
```

Two post-processors are applied. First, `ops_order_density` ensures the correct rendering order of density layers by sorting the actors accordingly. Then, `ops_flag` flags actor changes while ignoring timestamp and ID, enabling state transitions to be recognized between steps, similar to how the Cesium platform handles it.

5.3.3.4 Execution and Results

The transformation system is first configured by providing the recipe of the choreographies, the template for the unity platform and the unity platform recipe.

Both choreographies were provided to the configured transformation system, which produced the following results.

- Both choreographies were successfully transformed into the corresponding platform choreography, without any detected issues;
- When running the static oil spill choreography on the platform, the platform displayed a static oil spill with no movement or behavioural updates, as well as marking the correct event coordinates in the minimap, as shown in Figure 5.7;
- When running the dynamic oil spill choreography, the platform displayed an evolving oil spill with different density layers, marking each timestamp as a unique behaviour, as well as marking the correct event coordinates in the minimap, as seen in Figure 5.8.

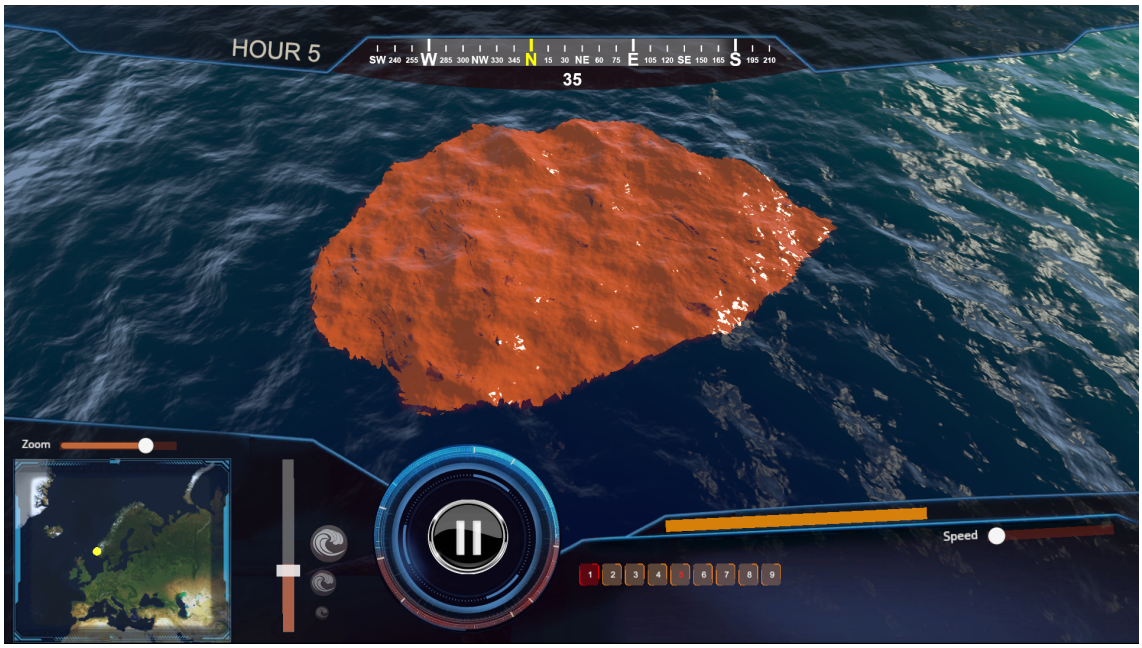


Figure 5.7: Unity-based Platform(static oil spill)



Figure 5.8: Unity-based Platform(dynamic oil spill)

5.4 Evaluation

When analyzing the results of the applied solution, I reached two relevant conclusions.

- First, the choreography effectively represents geometric and physical data related to oil spills by structuring information from the datasets in a behaviour-pattern-oriented format.
- Second, both platforms can reflect, within their respective boundaries, content derived from the same choreography, which in this approach was applied in two distinct scenarios.

These observations validate the proof of concept: virtual choreographies solve the proposed platform integration problem for oil spills. However, the solution is not perfect; several important issues emerged during introspection and discussions with the project team and supervisors.

5.4.1 Something more dynamic

Satisfied with this iteration, I began planning the next approach. My supervisor had a predetermined vision aligned with a graphical representation of choreography construction and transformation. Although very conceptual, he envisioned a system that would allow the intuitive linking of visual elements, enabling choreography authoring and mapping between platform primitives and generic choreographies to configure the transformation process graphically. The idea was promising, and the supervisor's main objective was clear: to make these tools accessible even to non-specialized users.

Building on this vision and my understanding of the latest prototype and the state of the art, I began outlining an approach that aligned with my research problem and addressed the nature

of oil spills. From the start, I identified a disconnection: the proposed authoring vision, though promising, did not match my specific problem. I was not manually creating oil spill events. I could draw polygons or manually design interactions between oil spills and other entities step by step. However, that would be a different problem!

My problem involved processing pre-existing scientific datasets in a batch-processing style, which I was able to offer a solution in the form of a static dataflow. However, this architecture, while simple to execute (segmenting, behaviour identification, property handling), had many limitations. For example, considering the transformation template for the unity platform: a bounding box is directly calculated within the template to represent the geographic zone of the spill event. While this information is particularly relevant for the platform, it is independent of the technology. It is generic information that could be useful in other hypothetical platforms. Couldn't we pass this information to the generic choreography? No, because the static nature of the generator only processes each segment separately, preventing operations that span multiple segments, like bounding box aggregation.

Inspired by the graphical interface vision and the current limitations, the following idea emerged: Instead of a static pipeline, why not a dynamic pipeline configurable through a visual interface? This pipeline would include all the basic primitives of a dataflow architecture but also specialized operators for automatic choreography creation, something novel in this topic. With this solution, users would not need to define each action at the time. They could visually map relationships between entities and actions, crafting a choreography architecture that, given scientific data, would automatically instantiate the elements that make the choreography. Although choreography production would be automatic, the design process would still be manual, as users must create the processing graph in the interface. However, the idea was promising, and it addressed the limitations of the current generator architecture.

5.4.2 Putting it all together

Another significant challenge emerged regarding the mapping between platform concepts and choreography constructs. As identified earlier, neither platform supported a well-defined "recipe". My solution was to extract the technology platform's interface schema. This schema showed all the technological components needed to reproduce a valid input. Could this be the "recipe" the old choreography approaches described [24]? My supervisor asked me this question at the time. I said yes, with the utmost confidence! I knew, however, that a visual interface that directly links concepts would be a great challenge with the platform recipes I had. Choreography recipes could not establish a one-to-one mapping to meet platform requirements, at least not without processing in the middle. Facing this statement, I had to ask myself: Would this data-shaping transformation process, across entities and actions, follow the same dataflow logic as the generator but in reverse logic? Of course it did! Until now, all that processing was buried in the transformation template, a bad practice since those mechanisms were meant to display information, not process it. Another identified architectural problem was that the generator and the transformation engine have independent plugin managers. This means that a component cannot be interchangeably used between

both mechanisms. Plugins could be more easily reused if both the generator and the transformer engine could be coupled within the same architecture. Given these observations, it was clear that both the generation and transformation of choreographies could happen within the same dataflow engine.

5.4.3 Not the right tone

Another debated topic in the recipe context was clarifying terminology. During discussions with supervisors, "recipe" often invoked a cooking analogy. The term proposed by Demetrius et al. [24] describes the semantic space of a given choreography or platform. The term, however, resembled a sequence of events or actions besides a semantic space of "ingredients". A recipe can also be analogous to an algorithm, in this case, a sequence of steps or actions. So, there was confusion between "recipe" and "choreography," as both may be mistakenly assumed to refer to the exact definition. To avoid ambiguity, I used the term "manifest" to refer to the set of characteristics or components that a choreography may include, essentially, its list of ingredients. In contrast, the term "recipe" is redefined to represent the pairing of a choreography with a compatible manifest, forming a complete and valid configuration. Both elements must align structurally and semantically for the recipe to be considered valid.

5.4.4 Don’t reinvent the wheel

From another angle, the project team responded positively to the concept and demonstration, so the solution was successfully integrated into the application package for generating visualizations. However, the project context focused heavily on the interoperability of processes and data via standards, prompting my custom solution to be adapted for community-preferred scientific tools. That shift benefited both the project and my solution. The choreography information model could rely on already implemented tools, avoiding reinventing the wheel and promoting adaptability. The team was already using a tool to validate data in JSON-LD format. SHACL was the name of the technology, and it was explicitly used to validate RDF via shape definitions. The state-of-the-art review of choreography representation technologies revealed RDF as a core theme. This representation technology had already caught my attention due to its triple-based data model, graph-oriented nature, and ability to link vocabularies and represent semantic data. So, in essence, all qualities are directly aligned with various themes and definitions raised in the choreography review. When I realized that, given an RDF, SHACL could serve as a technology analogous to the manifest, it became clear that this pairing was ideal for representing virtual choreographies.

5.5 Communication

The results of this iteration contributed both to the present thesis and to the publication of the article *“A Domain-Agnostic Virtual Choreography Framework for Digital Twins: an Oil Spill Application, IEEE Oceans 2025”*.

5.6 Conclusions

The proof-of-concept confirmed that virtual choreographies can represent oil spill data as behavioural patterns and are interpretable across heterogeneous platforms, validating the integration approach.

5.6.1 Limitations

The static dataflow architecture restricted multi-segment operations and domain adaptability. Generator and transformer engines were decoupled, limiting reuse. Terminological ambiguity in the concept of "recipe" may cause some confusion with the term virtual choreography. Initial implementation lacked alignment with community standards.

5.6.2 Planned Design Iteration

While the problem and objectives remain unchanged, the following design iteration will revise the artefact's architecture and implementation.

Key updates include introducing a dynamic, visual dataflow interface, unifying the generation and transformation engines, and supporting RDF and SHACL to represent manifest and virtual choreography. Terminology will also be refined: a *manifest* will explicitly define the domain structure, while a *recipe* will denote the pairing of a choreography with a compatible manifest.

Chapter 6

Iteration 2 - VC Flow v1

This chapter outlines the second approach, presenting improvements in the design and prototype artefacts. This solution is called the VC Flow, due to its alignment with the data flow architecture, where data is centralised in virtual choreographies and shaped through data flow graphs.

6.1 Design

6.1.1 Canvas

This architecture follows the *dataflow* paradigm, where data processing is modelled as a directed acyclic graph (DAG). Execution is modular and extensible, supported by a *plugin architecture* that allows the introduction of new node types without modifying the system core, as shown in the figure 6.1.

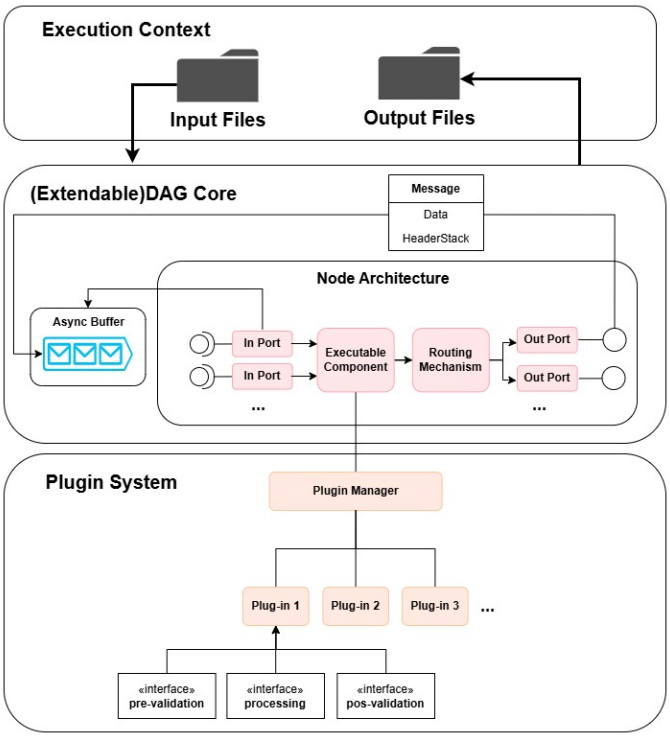


Figure 6.1: DAG architecture

6.1.1.1 Execution Context

The *Execution Context* defines the sources of input data and the destinations for output data. It acts as a bridge between the pipeline and the external environment and is decoupled from the processing logic.

6.1.1.2 Extensible DAG Core

- **Node Architecture**: Each node represents a data processing step. Nodes have input and output ports, allowing them to communicate through connections between their ports. After processing input data, the output can be divided into different items that are routed to specific output ports.
- **AsyncBuffer**: Data received by a node is stored in an asynchronous buffer. This buffer waits until all required inputs are available before processing can proceed.
- **Messages**: All node communication is done through message objects. Each message consists of:
  - **Data**: the resulting payload of a node transmitted to another.
  - **HeaderStack**: a stack of execution headers that track nested or repeated flows.
    - Generator nodes push a new header onto the stack each time they emit data.

    * Coroutine nodes pop headers to ensure synchronization across repeated graph segments.

The central component also has functionalities to verify whether the graph submitted for execution is valid. This includes, for instance, verifying the graph structure to ensure it is acyclical.

6.1.1.3 Node Types

Six distinct types of nodes in the pipeline are allowed:

- **Generators**: emit multiple values, supporting repeated or continuous execution.
- **Coroutines**: consume values from generators and stay synchronized via the `HeaderStack`.
- **Function Nodes**: perform stateless and punctual processing. They execute once per input and may run multiple times if preceded by a generator node.
- **Providers**: Used to import data from the execution context into the graph.
- **Consumers**: Used to export produced data from the graph to the context execution.
- **VCOperators**: implement specific functionality for handling virtual choreographies.

6.1.1.4 Plugin System

The architecture allows the DAG to be extended with new node types through a plugin system. Each plugin defines a new node and can implement up to three interfaces:

- **pre-validation**: executed before the main logic, used to validate input data.
- **processing**: the core logic of the node responsible for processing the data.
- **pos-validation**: executed after processing, used to validate output data before emission.

6.1.1.5 Interface

The canvas alone is not just an execution space. It builds upon the possibility of joining plugin-based nodes in a dataflow architecture to be able to draw the system in a visual interface by performing operations such as dragging and dropping nodes, configuring nodes metadata, and connecting nodes ports as shown in figure 6.2.

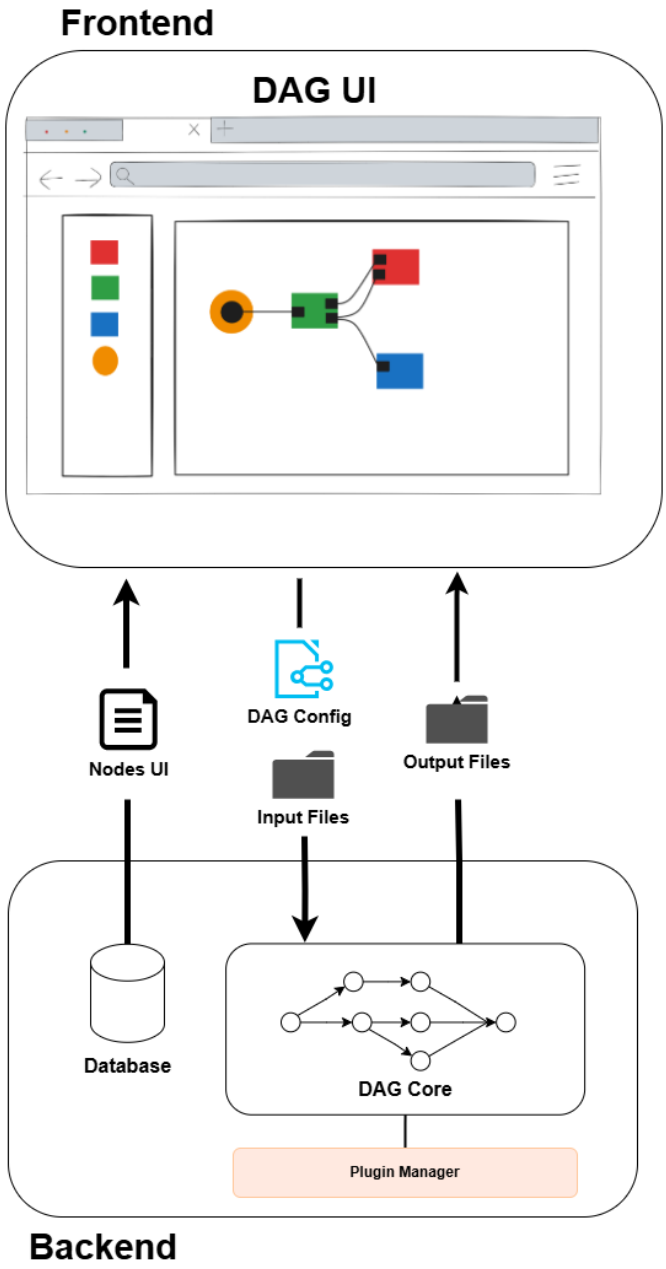


Figure 6.2: DAG building interface(Website mock was built on top of template produced by @dhaval_godwani from excalidraw)

This interface is connected to a database that supplies UI-related information for each node, including the metadata fields required for proper configuration and a unique identifier that links the visual representation of the node to its backend implementation, which is registered via the plugin manager. The frontend is also responsible for parsing the visual model of the pipeline into a DAG configuration compatible with the DAG core.

Additionally, the interface includes a file input mechanism that allows users to upload files into the DAG core's execution context. Once the pipeline is submitted and executed, the resulting

output files are generated within the same context and returned to the frontend, making them available for the user to download.

6.1.2 Transfomer

The transformer retains its execution logic, but instead of relying on template-based files, it now receives a DAG configuration, which is executed by the DAG core. Based on these configurations, it performs the necessary data transformations to the virtual choreography uploaded to the dag core execution context to produce the platform choreography, which is then fetched from the dag execution context output files.

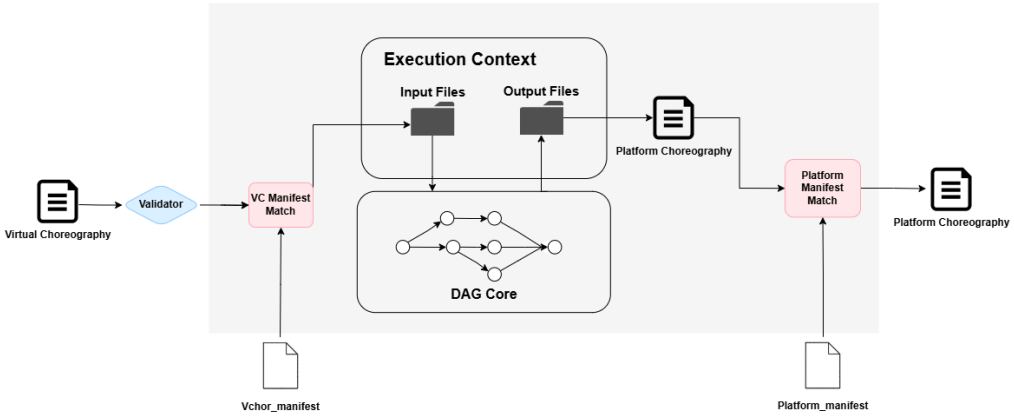


Figure 6.3: Transformer adapted to the DAG core

6.2 Prototype

6.2.1 Information Model Prototype

In the virtual choreography format, each choreography is defined as an RDF (Resource Description Framework) document representing a sequence of actions. It follows an actor-action-object pattern, where entities interact through well-defined actions:

```
@prefix vc: <http://inesctec.org/virtualchoreography#> .
@prefix iliad: <http://inesctec.org/iliad#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://inesctec.org/iliad/Action/abc123> ;
    rdf:type iliad:WasAt ;
    vc:actor <http://inesctec.org/iliad/Entity/def456> ;
    vc:object <http://inesctec.org/iliad/Entity/ghi789> ;
    iliad:timestamp "2023-10-01T12:00:00Z"^^xsd:string .

<http://inesctec.org/iliad/Entity/abc123> ;
```

```
    rdf:type iliad:Polygon ;
    iliad:geometry "[[4.189, 59.813]..."^^xsd:string .

<http://inesctec.org/iliad/Entity/cba321> ;
    rdf:type iliad:Polygon ;
    iliad:geometry "[[4.183, 59.813], [[4.233, 60.113]]"^^xsd:string .

<http://inesctec.org/iliad/Entity/def456> ;
    rdf:type iliad:OilSpill ;
    iliad:polygon <http://inesctec.org/iliad/Entity/abc123> ;
    iliad:density "0.92"^^xsd:decimal .

<http://inesctec.org/iliad/Entity/ghi789>  ;
    rdf:type iliad:Ocean ;
    iliad:bbox <http://inesctec.org/iliad/Entity/cba321> .
```

All choreography instances contain domain-specific entities (e.g., OilSpill, Ocean) that participate in actions (e.g., WasAt), forming a semantically rich representation of interactions in a particular domain.

6.2.1.1 Manifest Definition

Each choreography must be paired with a **manifest** that defines the domain-specific vocabulary and constraints. The manifest:

- Declares which specific entity types exist in the domain (as subclasses of `vc:Entity`)
- Defines which action types are permitted (as subclasses of `vc:Action`)
- Specifies detailed property constraints for each entity and action type

Manifests utilize SHACL (Shapes Constraint Language) to define constraints in:

- **Object Properties**:
  - Type constraints on referenced entities;
  - Cardinality rules;
  - Valid relationship patterns between specific entity types;
- **Literal Properties**:
  - Data type specifications (strings, numbers, dates, etc.);
  - Cardinality rules.

6.2.1.2 Validation Process

The validation process ensures that both the manifest and choreography conform to their respective requirements:

Before a manifest can be used, it must be validated against a `manifest_shape.ttl`, which ensures:

- Each class is properly defined within the `vc:Entity`/`vc:Action` hierarchy;
- Every class is targeted by exactly one shape definition;
- Property definitions have appropriate types and constraints.

Once paired with a valid manifest, the choreography is validated against:

1. **Domain-Specific Rules** from the manifest that enforces the domain constraints.
2. **Core Structural Rules** defined in `vc_shape.ttl`:
   - Every resource must be properly typed as either a `vc:Entity` or `vc:Action`;
   - Each valid choreography must contain at least one instance of `vc:Action`;
   - Every `vc:Action` must define both actor and object properties.

This two-level validation approach ensures that choreographies are both structurally and semantically valid within their specific domains. The validation is conducted using the pyshacl [48] library in python.

6.2.2 Canvas

The canvas is implemented in two parts: the backend, which is responsible for implementing the environment that integrates the DAG core, the plugin manager, the execution context, and all tools required for the validation mechanism; and the frontend, which is responsible for implementing the visuals of the DAG, and supporting user interactions.

6.2.2.1 Technological stack

All node frontend configurations are stored in a**MongoDB** database as JSON documents. Each node contains its visual properties (label, ports, colour), a JSON Schema for generating dynamic metadata forms, and a unique code that links it to backend plugins. This configuration tells the frontend which plugin each node uses defines its metadata structure, and specifies how it appears visually.

The frontend canvas is built with **React-Diagrams**, enabling users to construct dataflows visually. Additional interface components are styled using **MUI** templates. Node metadata is configurable through dynamically rendered forms powered by **react-jsonschema-form**, using the

schema embedded in each node's configuration. The front end communicates with the backend to send pipeline definitions, upload files, and trigger execution. It also transforms the visual diagram into a DAG structure interpretable by the backend.

The backend is implemented in **Python**, exposing RESTful endpoints that handle database operations and client interactions. It receives pipeline configurations, file uploads, and execution commands from the frontend. It is responsible for executing the pipeline exactly as designed in the visual interface, mapping each node code to its corresponding execution logic, and passing the metadata of each node to create/execute the graph.

6.2.2.2 Tool menu and metadata forms

This section introduces the core tools used to build pipelines within the canvas engine. Figure 6.4 highlights its main components. On the left, there's a tool menu containing all available nodes, organised into sections. Nodes can be dragged from these sections and dropped into the canvas, located at the centre. The canvas is where nodes are arranged and connected via their ports.

On the right side, when a node is selected in the canvas, its metadata form appears. This form is generated automatically from the node's user interface JSON configuration and complies with its backend specifications. For example, in the "orderBY" node, the form displays the configuration options relevant to its functionality. In Figure 6.4, the property fields represent the criteria used to order elements in the "groupBy" node.

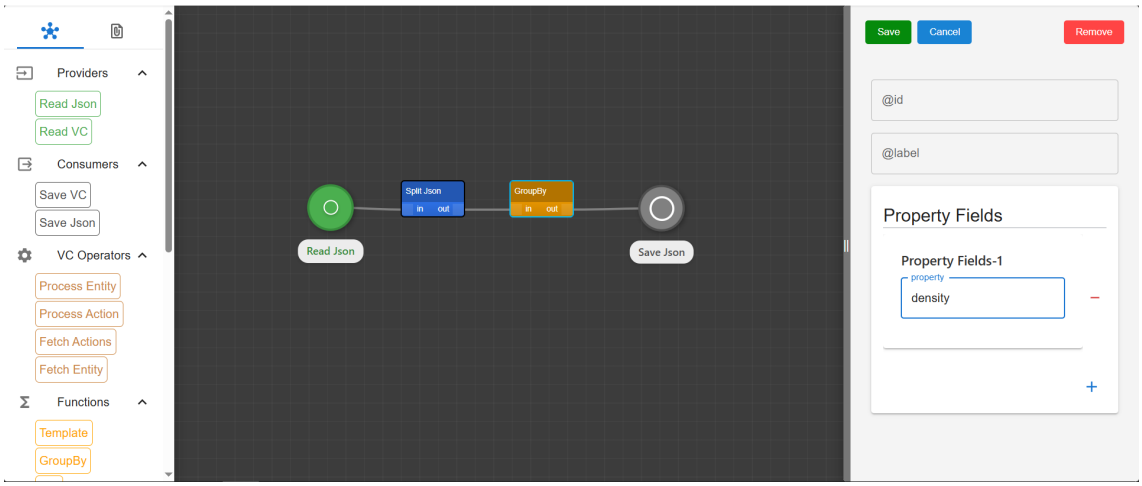


Figure 6.4: Canvas interface

In the menu tool, there is a second slider that allows files to be attached. When the DAG is launched in the backend, these files will serve as the input files for the context execution of the DAG. As shown in Figure 6.5, the metadata for the green provider node automatically displays the list of available files. By selecting a file, a binding is created between the provider and the file that has been input into the system.

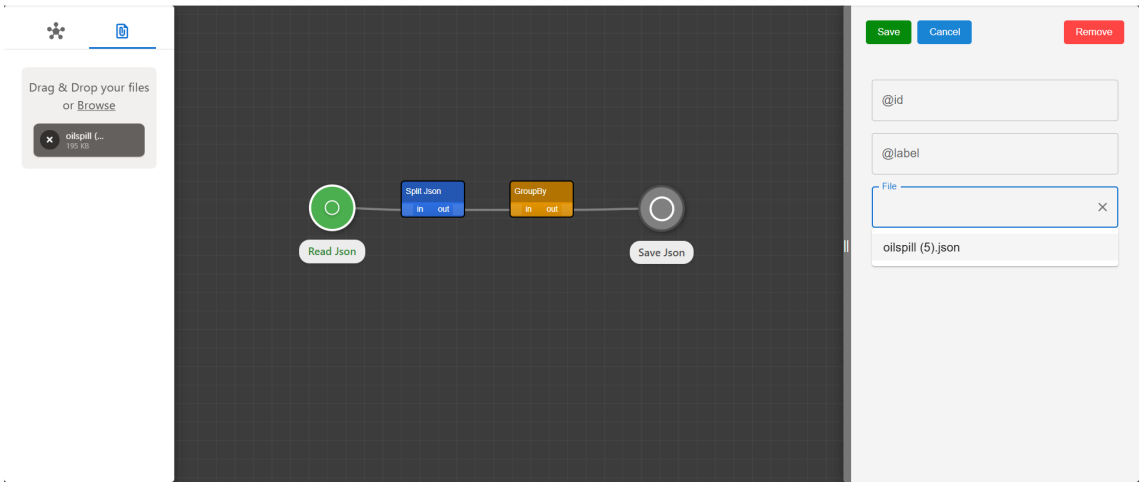


Figure 6.5: File Selector in DAG interface

This functionality is essential for loading choreographies and datasets in order to debug transformation and generation pipelines directly within the interface.

6.2.2.3 Generators and coroutines

Generators and coroutines are essential mechanisms in building the processing directed acyclic graph (DAG).

The generator nodes utilize Python's underlying generator type to provide a fan-out of elements from a given iterator, while the coroutine nodes use Python's underlying coroutine type to await values emitted by the generator until it sends an "EndOfStream" message. These two mechanisms utilize the header stack in the messages they receive to detect if all elements are being received in order. To illustrate this in practice, some scenarios are presented.

In the graph, generator nodes are colored blue, while coroutine nodes are colored red.

In Figure 6.6, we see a simple example where the split node transmits each element of an array one by one. At the coroutine, with each element that arrives, it appends the element to an array. After the split node sends an "EndOfStream" message, the "catch" ends its execution.

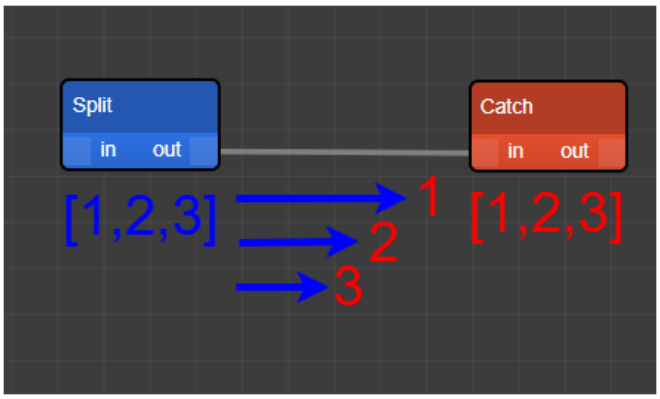


Figure 6.6: Simple generator-coroutine example

Another more useful example is how operations can be incorporated between these two operators. In Figure 6.7, each element from the array transmitted by the "split" node is incremented by one before being sent to the "catch" node. Once the generator sends an "EndOfStream" signal, the "catch" node knows it has received all elements, resulting in an array where each element has been increased by one.

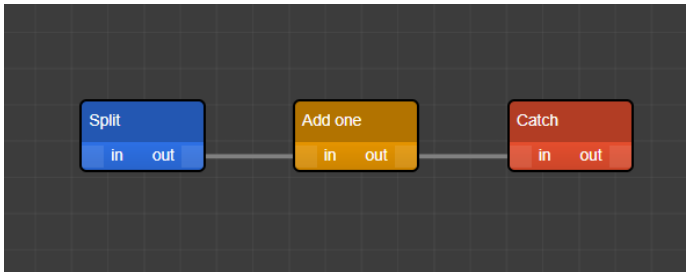


Figure 6.7: Add one to the array operation in DAG

A more complex example is illustrated in Figure 6.8, where nested executions occur. In this scenario, "split 1" receives an array of arrays and sends each array to "split 2" one at a time. "Split 2" then divides the array into individual elements and forwards them one by one to the "Add one" operation. After each element is increased by one, it is caught in "Catch 1." Only when "Catch 1" receives an end-of-stream signal from "Split 2" does it send the resulting array to "MultiArray 5" operation, which multiplies all the elements of the received array by 5. At the same time, "split 2" begins processing the next array received from "split 1," initiating a new inner cycle. This inner cycle continues until all arrays have been sent by "split 1." Once the final array is processed, "Catch 2" concludes with the array of processed arrays.

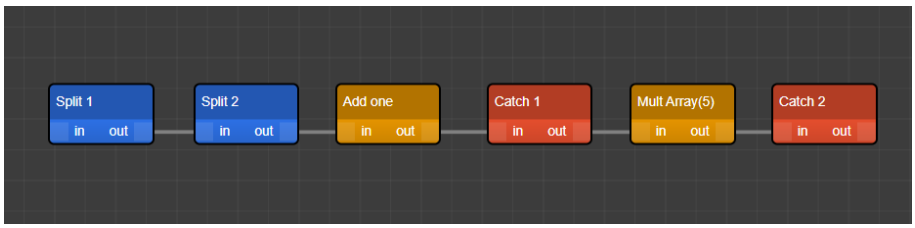


Figure 6.8: Complex operation in DAG

This mechanism will be beneficial for segmenting datasets and iterating actions or polygons.

6.2.2.4 Template Nodes

Template nodes are a special type of function node because they allow for the use of the Jinja template engine to inject data from previous nodes into the template. For instance, in Figure 6.9, there is a hypothetical scenario in which "Node1" outputs its value to the input port of the template node. In the metadata of the template node, we can observe the defined schema, which consists of a dictionary where a key named "value" is injected with the output from "Node1."

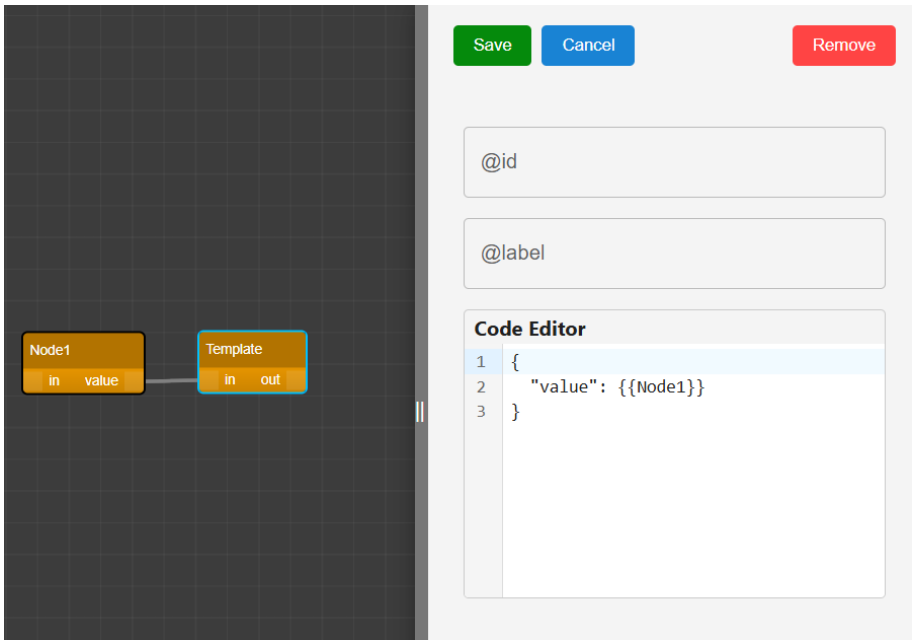


Figure 6.9: Template node in DAG

This mechanism enables the construction of platform elements, which in turn allow for the development of the platform choreography. Additionally, template nodes can interact with other template nodes, meaning that the entire choreography template will not be created in a single node. Instead, it will be orchestrated by multiple nodes working together.

6.2.2.5 Virtual choreographies operators

These operator nodes are the core functionalities that support virtual choreographies; they are meant for different purposes and address different primitives of the virtual choreography.

The first node is known as the "process entity" node. This node is primarily responsible for creating entities within the choreography. The "in" input port connects data to this node. The entity fields define the properties of the entity. Each property must have a name, a type, and a value. The value can either be a constant, which can be directly typed in the node metadata, or it can be derived from the data received at the "in" port by using the format "@map(key)", where "key" corresponds to the value that needs to be mapped from the input data to the property. The "@entityClass" field indicates the class of the entity, such as "OilSpill." The code editor functions as an interface where namespaces and prefixes for the entity class can be added. For example, (iliad:http://inesctec.com/iliad/) will be used to generate a URI (Uniform Resource Identifier) for the entity produced such as (iliad:OilSpill).

The "graph" input port is designed to receive RDF graphs that originate from other "process entity" nodes. When a "process entity" node receives a new graph, it can define a new property in its entity fields, using the format "@ref(entityID)" to establish a nesting relationship between the entity or entities contained in the input graph. In Figure 6.10, the process node for entity B receives a graph containing entity A, which is referenced in the entity fields.

After this process finishes, it outputs the resulting graph from the "graph" output port and forwards the exact same content received at the input port "in" to the output port "out."

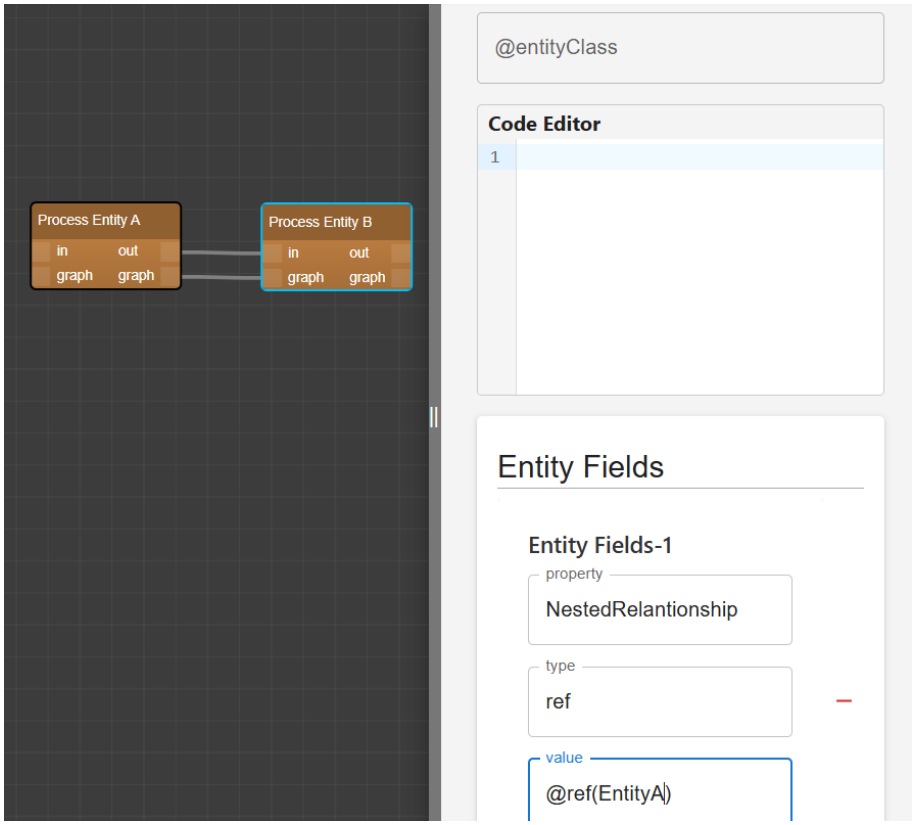


Figure 6.10: Process entity node in DAG

The second node is the "Process Action" node 6.11, which is responsible for creating virtual choreography actions. The "in" input port connects data to this node. The action fields define the extensions of the action. This node employs a similar mechanism for mapping input data to extensions using the format "@map(key)", just like the Process Entity node. Additionally, the Action class specifies the type of action, such as "wasAt," and the code editor serves the same purpose as that in the "Process Entity" node. In order for the action to be valid, it needs to be assigned an actor and an object, so the "process action" node supports directly an input for the actor entity graph and the object entity graph, and produces a graph containing the entities and the configured action.

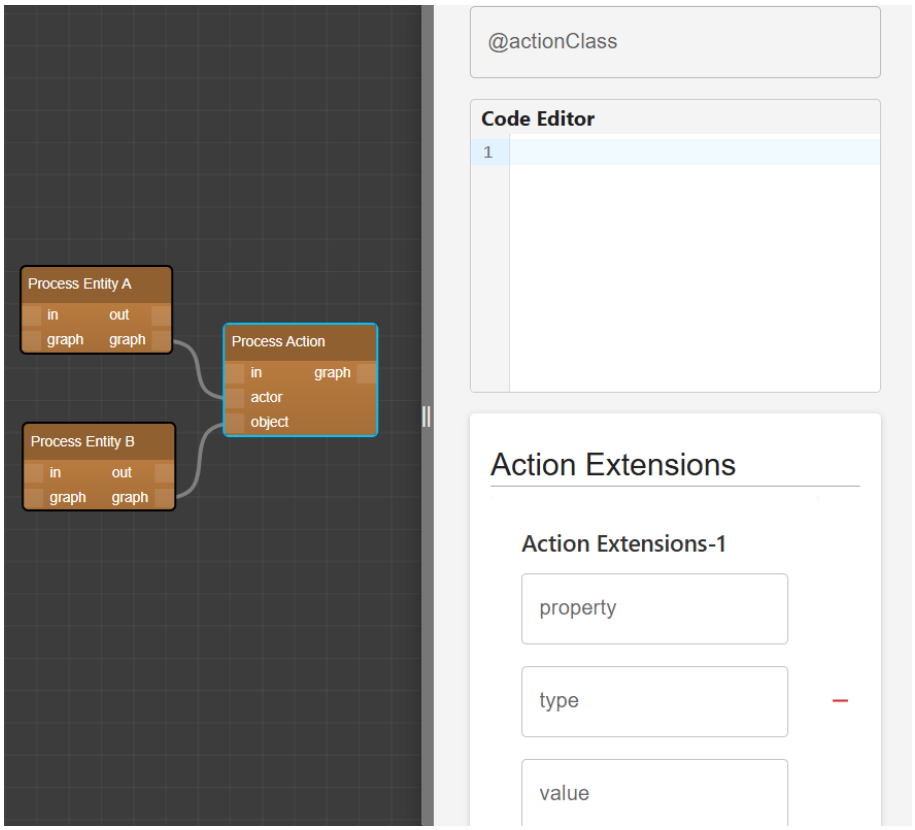


Figure 6.11: Process action node in DAG

The third node is the "Fetch entity" node. This node is meant for the transformation pipeline, in which it interacts directly with the virtual choreography, and given a URI in the "id" port, it retrieves the corresponding entity in the choreography.

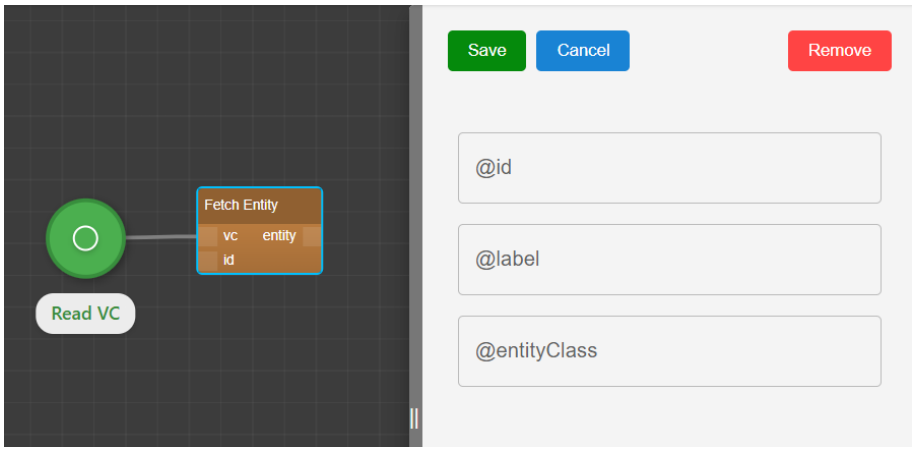


Figure 6.12: Fetch entity node in DAG

Finally, the last vcoperator node is the Fetch Actions node. This node allows a set of actions to be retrieved from a choreography by specifying the actionClass and the corresponding action targets.

Since the choreography is represented as RDF and is based on a triple structure, it's not strictly necessary to specify all action targets. For example, if only the actionClass is provided, all actions corresponding to that class will be retrieved.

Another example, as illustrated in Figure 6.13, is when only the actor class is provided, and the object class is omitted. In that case, only the triples where the actionClass has actor "EntityA" will be retrieved, even if the actions span multiple object classes.

However, specifying both the actor and the object class increases the specificity of the action set returned.

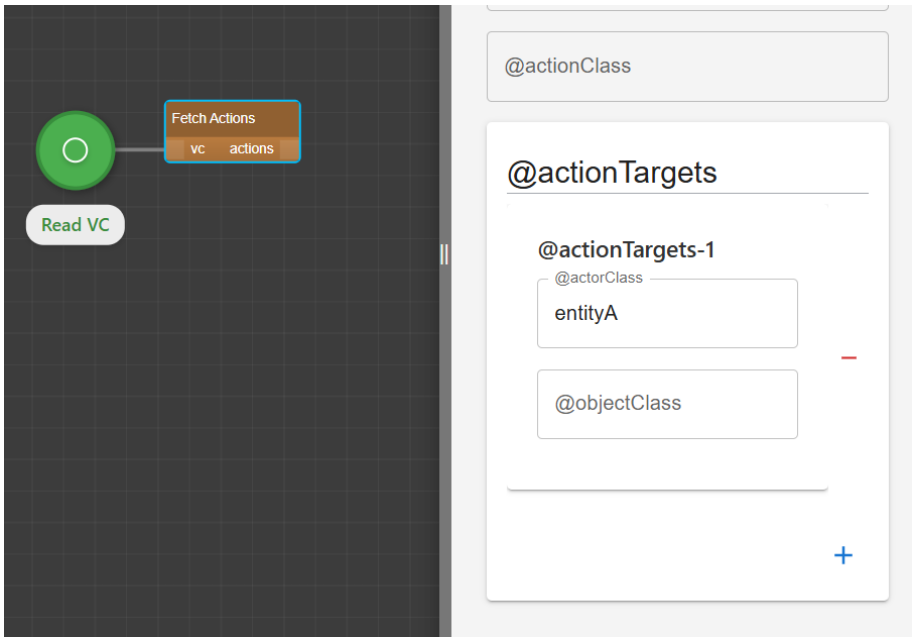


Figure 6.13: Fetch actions node in DAG

6.2.2.6 Additional nodes

Additional nodes and their functionalities are described in a table in the appendix C.

6.3 Demonstration

To demonstrate the prototype solution, the two scenarios from the previous iteration were tested:

6.3.1 Choreography generation

The Figure 6.14 showcases the use of the canvas prototype to generate choreographies from the field case datasets. This pipeline is divided into multiple segments, with each segment responsible for a specific task.

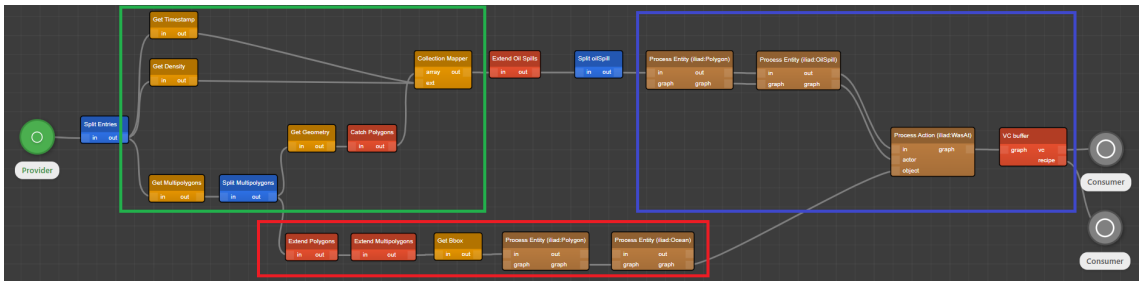


Figure 6.14: Generation pipeline

The green frame is where the pre-processing of the dataset is conducted; here, multipolygons are decomposed into single polygons and are associated with the corresponding density and timestamp from the parent multipolygon.

In the red frame, all resulting polygons are then aggregated to compute the scene's bounding box. A dedicated node generates a single polygon entity from this box, which is subsequently passed to the Ocean entity process. There, the bounding-box polygon is nested within the Ocean entity.

In the blue frame, oil spill entities and their nested polygons are processed. "wasAt" actions are created, using the oil spill as the actor and the Ocean entity, produced in the red frame, as the object. Once all actions have been generated, the choreography buffer assembles the complete virtual choreography and infers the final choreography recipe.

By processing the experimental datasets using this pipeline, two different choreographies are obtained:

- The static choreography, which has 5 different oil spill actors, all sharing the same polygon entity. Only one ocean entity exists and is connected to the only polygon that represents a bounding box. The oil spill entities remain unchanged through time.
- The dynamic choreography, which has 36 different oil spill actors, with each actor having unique density and polygon combinations showing a different behaviour with each timestamp. Only one ocean entity exists and is connected to the only polygon that represents a bounding box.

6.3.2 Cesium-based platform transformation

In Figure 6.15, we can see how the canvas prototype implements the transformation for the cesium platform.

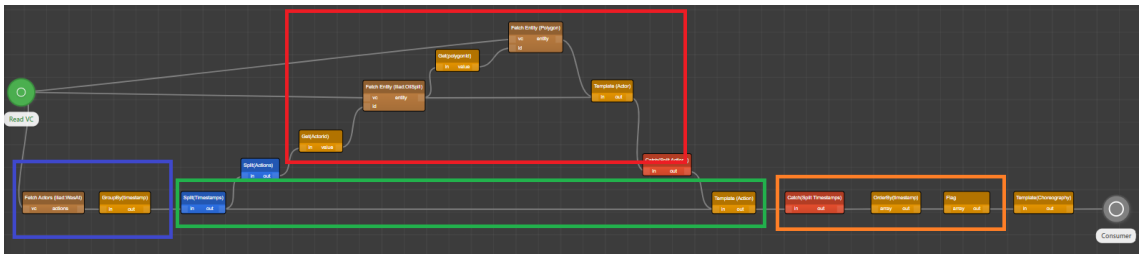


Figure 6.15: Cesium transformation pipeline

The blue frame is responsible for fetching the choreography actions from the choreography provider and grouping them by time.

The ref frame decomposes oil spill actors from each timestamp group to produce the platform actor template that corresponds to the actor in the platform recipe. After processing all the actors from the choreography action, the result is stored in a coroutine.

In the green frame, the coroutine from the red frame feeds the produced platform actors to another template node, which also receives the choreography action timestamp and builds the platform action template in accordance with the platform recipe.

in the orange frame, after all platform actions are processed. They are ordered by timestamp, and each action is flagged if it was different from the previous one.

6.3.3 Unity-based platform transformation

For the unity-based platform, the following transformation pipeline was created:

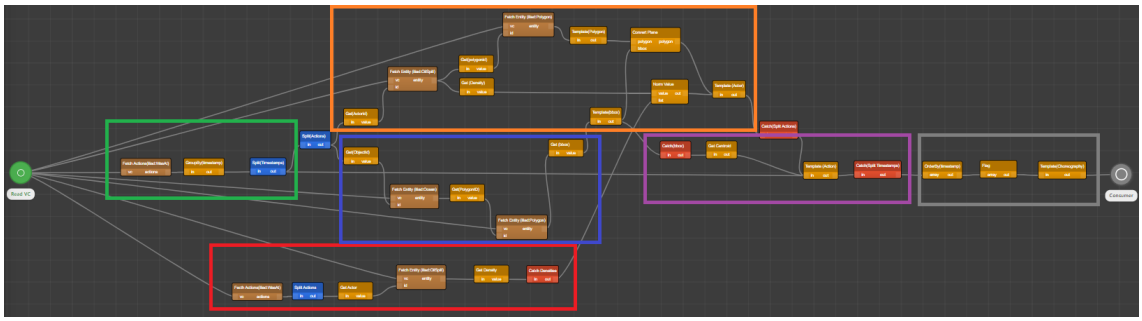


Figure 6.16: Unity transformation pipeline

In the green frame, the choreography actions are fetched from the choreography provider and grouped by time.

In the red frame, the oil spill entities are fetched from the choreography, and all the different densities in the scene are gathered in an array.

In the blue frame, within each choreography action, the ocean object is resolved and the bounding box is retrieved for two purposes. The first is to pass the bounding box into the convert plane node in the orange frame. The second purpose is to pass the bounding box to the purple frame.

In the orange frame, within each choreography action, the oil spill actors are resolved, and their shape is converted from latitude and longitude into the Cartesian XY plane. This conversion also

receives the bounding box from the scene from the blue frame, in order to correctly represent with a universal reference of the scene, the position of the oil spills on the platform. Another operation is the normalisation of the densities, which receives the array of densities from the red frame in order to normalise the density of each choreography oil spill actor, within limits acceptable to the platform. The last operation is to produce each platform actor from the actor template node.

In the purple frame, the bounding box is received from the blue frame, and its centroid is calculated and used to mark the place of the action, in order to correctly display the position of the scene in the minimap of the platform. It is in this frame that the platform actions are generated by consuming the platform actors that were produced in the orange frame, the minimap location in the blue frame and the action timestamp.

Finally, once all the platform actions were processed, the grey frame orders them by timestamp, and each action is flagged if it was different from the previous one.

6.3.4 Transformations

The transformer system is designed to handle a three-phase process:

First, it takes an input choreography along with a corresponding choreography manifest that aligns with the intended transformation. The system checks if both are compatible and confirms a match as shown in Figure 6.17.

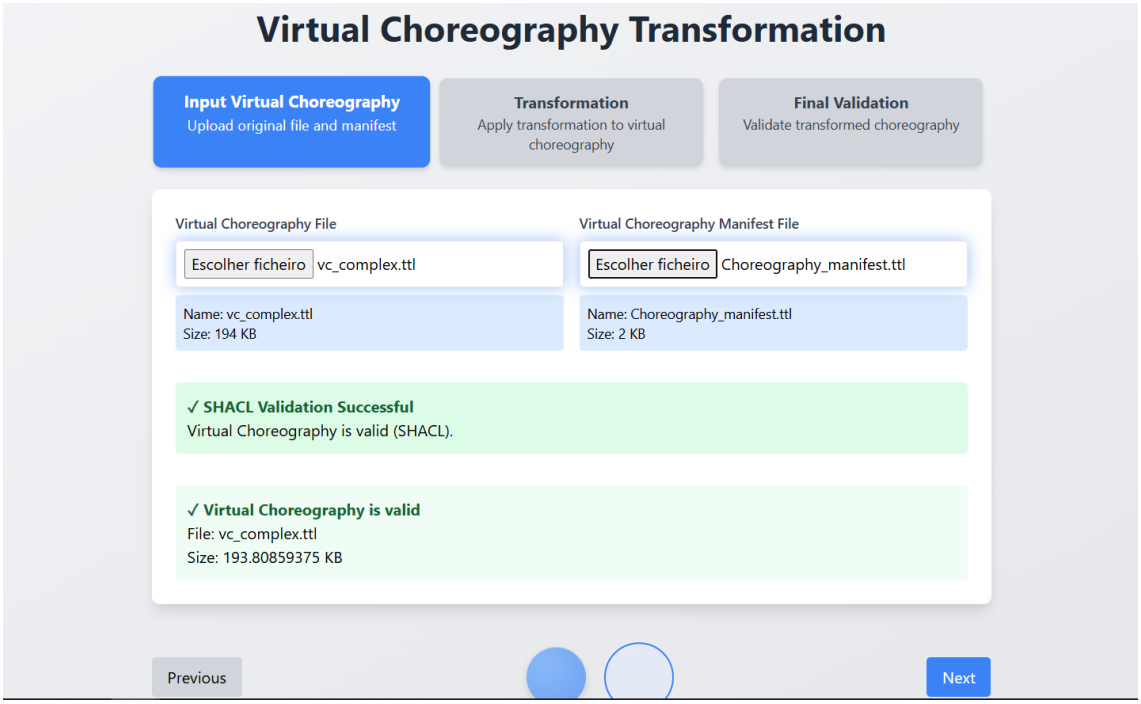


Figure 6.17: First phase of the transformation process

Secondly, the user selects the transformation process by providing the DAG (Directed Acyclic Graph) configuration, which outlines the tasks needed for the transformation 6.18.

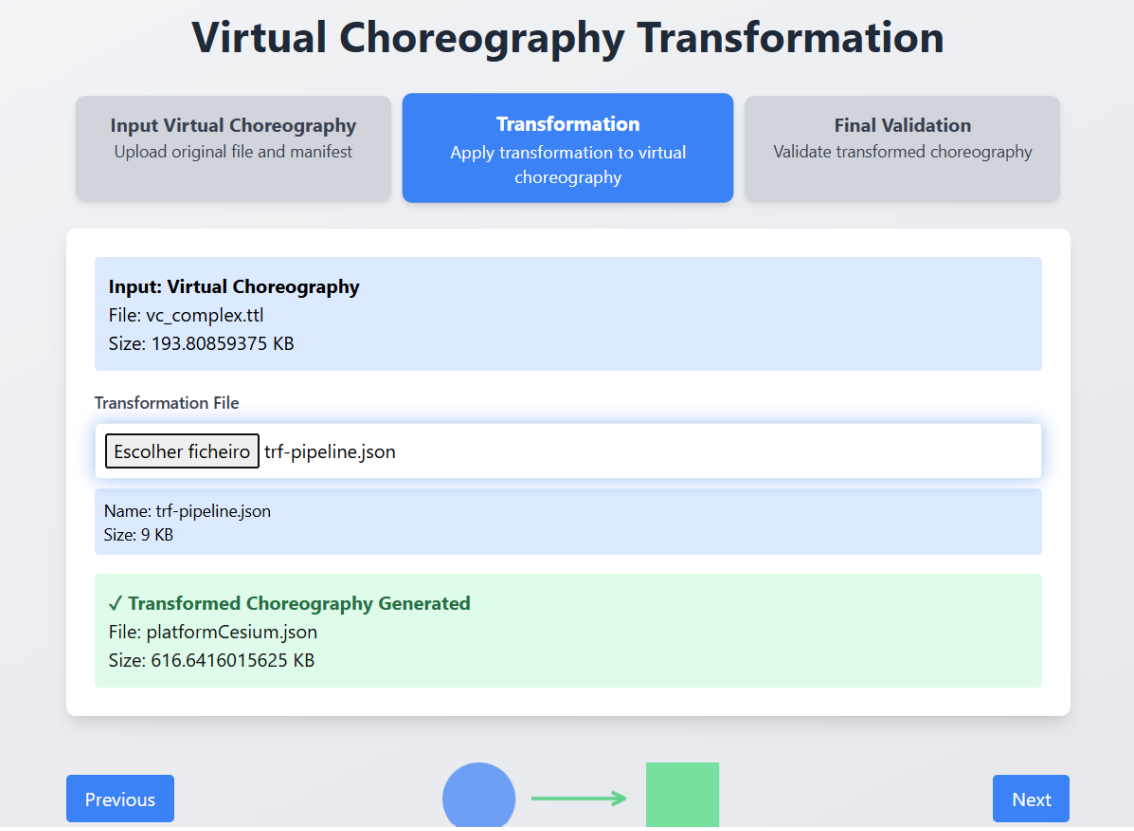


Figure 6.18: Second phase of the transformation process

Finally, if the transformation is successful, the last phase involves validating the platform choreography against a platform recipe. This ensures that the final output adheres to the defined standards and requirements of the platform. Then the user can simply download the platform choreography 6.19.

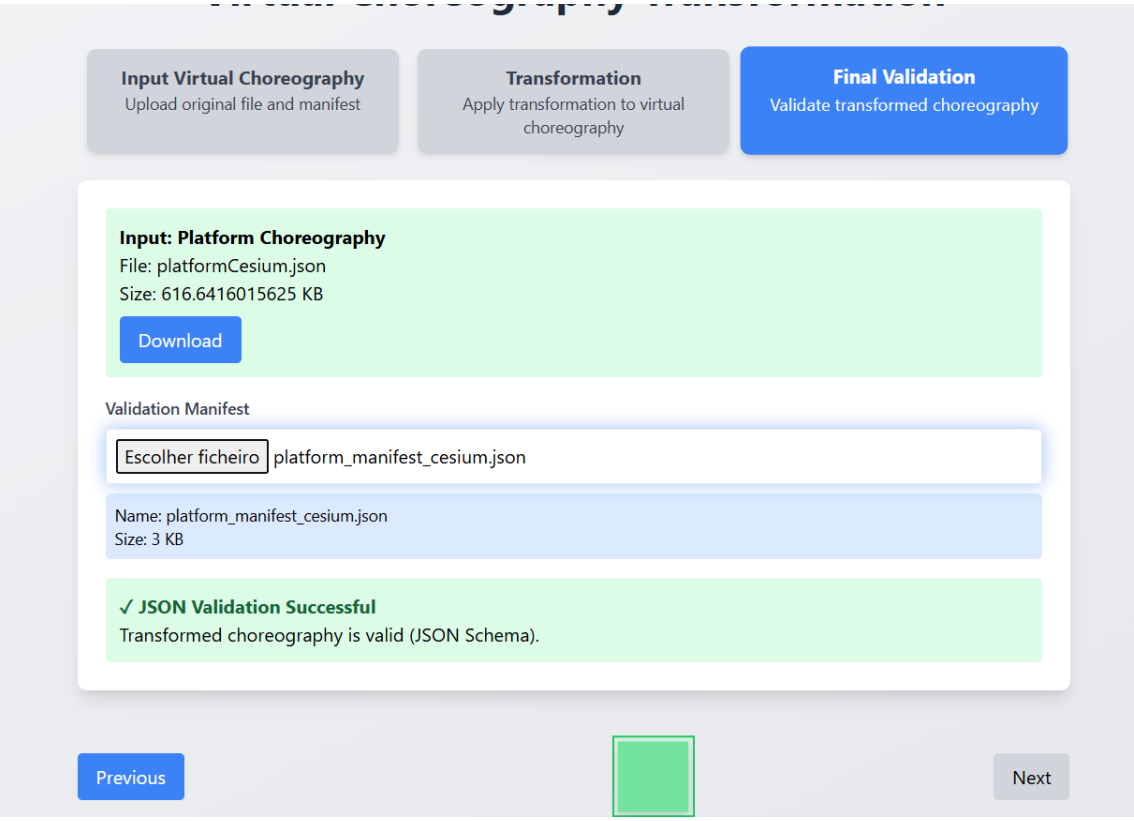


Figure 6.19: Third phase of the transformation process

6.3.5 Platform Results

All the produced platform choreographies were provided in the platforms, to which no errors were detected.

The unity and cesium-based system provided identical outcomes to the previous iteration, both on the simple scenario 5.5 5.7 as well as in the complex scenario 5.6 5.8.

6.4 Evaluation

Just like in the previous iteration:

The generated choreography successfully captures the geometric and physical properties of oil spills by structuring the dataset information around identifiable behaviour patterns.

Both target platforms were able to represent content derived from the same choreography, each within its own constraints, across two distinct oil spill scenarios.

So the concept was once again demonstrated, this time aligned with a more developed dataflow architecture.

6.4.1 Almost perfect

So there I was, standing proud of my advancements. At the weekly meeting, my supervisors looked at the tool, and something was wrong that I was not aware of. The basis of the choreography' manifests was the choreographies! So far, all the choreography manifests have been automatically inferred from the generated choreographies. My engineering nature had betrayed me. While trying to optimise the technology, a theoretical flaw had appeared. If the platform manifest is tailored to the platform, the choreography manifest is tailored to the domain. However, was the choreography manifest not tailored to the domain? No, it was tailored to the information I extracted from the datasets. For the choreography to be truly independent of any technology, its manifest needs to be conceptually generated before creating and transforming choreographies! It was like making the dish without knowing the recipe! The manifest should be used as the basis for these processes, not the other way around. Even to align with the transformation processes, there needs to be a consensus between the generation and the transformation. In a scenario where we have a team developing generation and transformation pipelines, they will do so according to a manifest that is only dependent on the domain.

If there is insufficient data to use the whole manifest, or the platform does not understand the whole domain, this should not limit the manifest. This reflection has even brought about an improvement that could make the system more cohesive. Since the process starts from the manifest, several concepts could improve the tool, such as loading the processing nodes of the choreographies with pre-configured information from a manifest. In this scenario, the pipeline orchestrator would only have to drag the nodes and connect them to other processing units without having to manually configure them, which can lead to a greater risk of errors and thus makes the development process more secure, dependent on a well-defined domain and a semi-automated service or, as addressed in the literature review, semantically guided. Another great advantage of this process is that if the manifest is already used to complete the choreography nodes automatically, then why not annotate those nodes with the ingredients they operate on? This way, the transformation graph would have the semantic annotation of the information it needs from a choreography to perform a transformation.

6.4.2 Where is the play button?

I knew from the beginning that my supervisor was not a big fan of how I was performing the transformations. He wanted the process simplified for whoever was going to use it. Just give a choreography; without knowing what is underneath, you get your platform choreography. Sure, there would need to be an engineer to configure the system, but that would not be the stakeholder's job. Although it showed the concept, the current transformation process was not intuitive enough. It showed how choreographies could be used technically, but not how they could be an advantage for multi-platform environments such as digital twins.

Since manifests can be automatically compared to a choreography, the following solution occurred: given one choreography, I can compare it automatically to a list of manifests. If each of

those manifests is associated with a transformation, I can automatically detect which transformations are possible and consequently which platforms are able to render the choreography. In a user interface, the different possible platforms can be presented to the user, who then can select which visualisation method is desired.

It became clear that there was a need for these processes to become intuitive to whoever uses them, and I finally had a solution for it!

6.5 Communication

The present cycle was communicated through this thesis.

6.6 Conclusions

The choreography-based approach, once again, successfully demonstrated that oil spill data, modelled as virtual choreographies, can be effectively integrated into heterogeneous applications.

6.6.1 Limitations

The system's theoretical foundation was compromised by automatically inferring choreography manifests from generated choreographies, creating unwanted coupling between domain representation and data extraction. The transformation process also lacks an intuitive automated process for non-technical users, requiring manual configuration that hides the end-user advantage of using choreographies.

6.6.2 Next iteration

The objectives, the design and the prototype will change to accommodate a manifest-first architecture where choreography manifests are created independently of data sources or platforms. After manifest comparison, the transformation interface will automatically present compatible platforms, hiding technical complexity from users. The system will incorporate pre-configured processing nodes populated from manifest specifications, reducing configuration errors and creating semantically annotated transformation graphs.

Chapter 7

Iteration 3 - VC Flow v2

This chapter presents the third approach, which involves reformulating the original objectives and introducing enhancements in the design and prototype artifacts. This solution is named VC Flow v2, as it represents an improvement from the previous iteration.

7.1 Reformulated Objectives

The research objectives were reformulated due to a critical theoretical flaw: the manifests were inferred from generated choreographies, inverting the intended dependency. This approach compromised the conceptual integrity of the system by coupling domain representation with data extraction. To address this, the research focused on designing a manifest grounded directly in the oil spill domain and validating its role in representation and integration, within the constraints defined by the field case. Based on this reflection, the research objectives were reformulated accordingly:

- Develop a virtual choreography manifest suitable for the oil spill domain.
- Validate the applicability of the manifest for representing the domain within the constraints defined by the field case.
- Validate the manifest integration with platform-specific implementations within the constraints defined by the field case.

7.2 Design

7.2.1 Oil spill manifest

The manifest captures a simplified but representative view of the oil spill domain, centred on key environmental entities such as oil spills, slick layers, oceans, and their spatial boundaries. The OilSpill concept is modelled as a collection of SlickLayers to reflect its non-uniform structure, where different parts of the slick exhibit varying densities due to environmental conditions [49]. Additional properties include area, perimeter, compaction, dispersion radius, and bearing. These

characteristics further enhance our understanding of the domain while remaining independent of any specific technology or platform. The following excerpt illustrates these core definitions and constraints:

```
iliad:WasAt a rdfs:Class ;
    rdfs:subClassOf vc:Action .
iliad:SlickLayer a rdfs:Class ;
    rdfs:subClassOf vc:Entity .
iliad:OilSpill a rdfs:Class ;
    rdfs:subClassOf vc:Entity .
iliad:Polygon a rdfs:Class ;
    rdfs:subClassOf vc:Entity .
iliad:Ocean a rdfs:Class ;
    rdfs:subClassOf vc:Entity .
...

iliad:SlickLayer
    a sh:NodeShape ;
    sh:targetClass iliad:SlickLayer ;
    sh:property [
        sh:path iliad:density ;
        sh:datatype xsd:decimal ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path iliad:polygon ;
        sh:nodeKind sh:IRI ;
        sh:class iliad:Polygon ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ...

iliad:WasAtShape
    a sh:NodeShape ;
    sh:targetClass iliad:WasAt ;
    sh:property [
        sh:path iliad:timestamp ;
        sh:datatype xsd:string ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] ...

iliad:OilSpillShape
    a sh:NodeShape ;
    sh:targetClass iliad:OilSpill ;
    sh:property [
        sh:path iliad:area ;
```

```
        sh:datatype xsd:decimal ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ]  ...

iliad:OceanShape
    a sh:NodeShape ;
    sh:targetClass iliad:Ocean ;
    sh:property [
        sh:path iliad:bbox ;
        sh:nodeKind sh:IRI ;
        sh:class iliad:Polygon ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] .
```

7.2.2 Canvas enhancements

To implement manifest ingestion, the canvas design enhances existing vcoperator nodes to enable automatic node structuring from manifests and semantic annotation in graphs, as shown in figure 7.1. These enhanced nodes shift the system toward manifest-dependent generation and transformation processes while introducing semi-automatic pipeline-building capabilities.

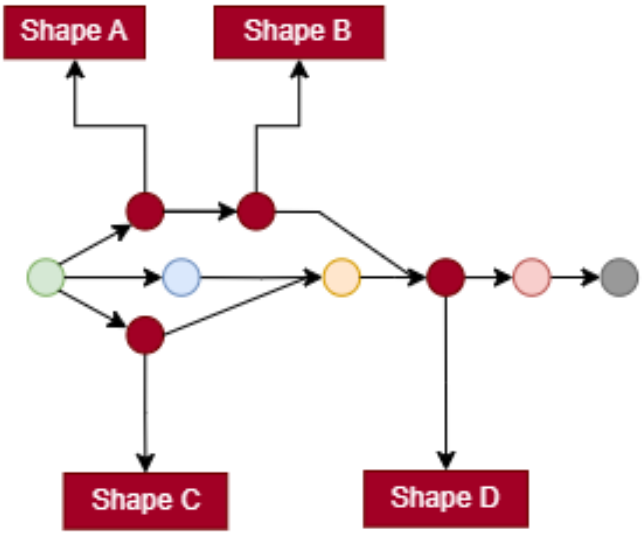


Figure 7.1: Semantically annotated graph

7.2.3 Transformer enhancements

The transformer was enhanced to include not only one reference of transformation, in this case a single manifest, but also a chain of references, each connected to a DAG execution transformation, that is going to apply the necessary operations to obtain the platform choreography as shown in

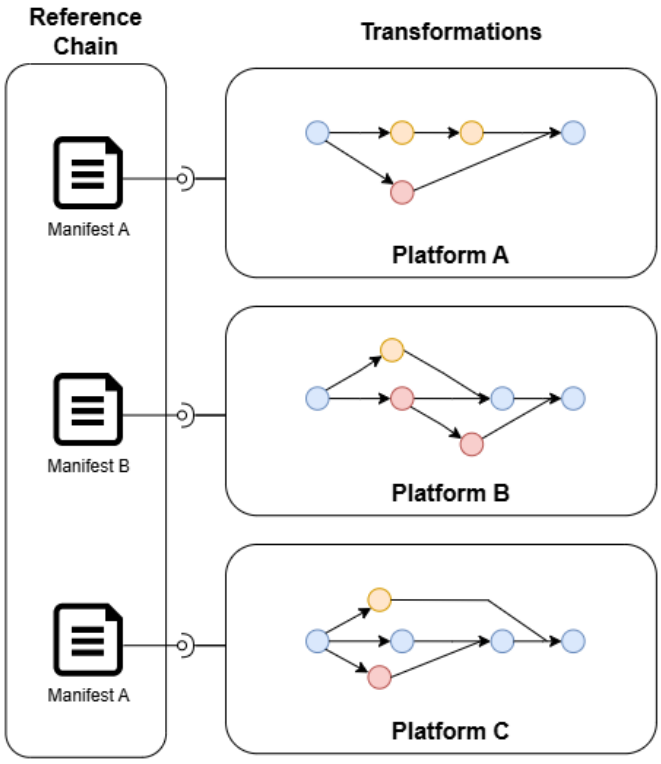


Figure 7.2: Transformer architecture

figure 7.2. Even though it is not included in the figure, platform manifests are still included as a validation mechanism after the pipeline produces a platform choreography, as the pipeline structure is still dependent on the connections that are made from choreography manifest concepts to platform concepts.

The reference chain represents the entire basis for comparison, not just a single manifest. The input virtual choreography is matched against the corresponding manifests and, consequently, the platform transformations. By obtaining a list of possible conversions, it enables a multi-platform transformation orchestrator.

When a manifest is combined with a semantically annotated transformation configuration, it becomes possible to directly determine whether the manifest is compatible with the transformation. This offers an additional validation step, ensuring alignment between the manifest's domain and the domain the transformation is designed to handle. However, this validation only works through direct matching.

7.3 Prototype

7.3.1 Canvas enhancements

To improve the canvas according to the previous iteration evaluation, a new slider was added to the tool menu, where a user can input manifests.

The result is a set of processing functions oriented towards the ingredients declared in each manifest. For entities, the processing entity operator and the fetch entity operator are automatically configured. For actions, the processing action operator and the fetch action operator are also configured. The user can then drag and drop the nodes into the canvas, ready to be directly applied in the graph. Each node is also annotated with the shape configuration in its metadata autonomously, meaning its semantic signature as seen in figure 7.3.

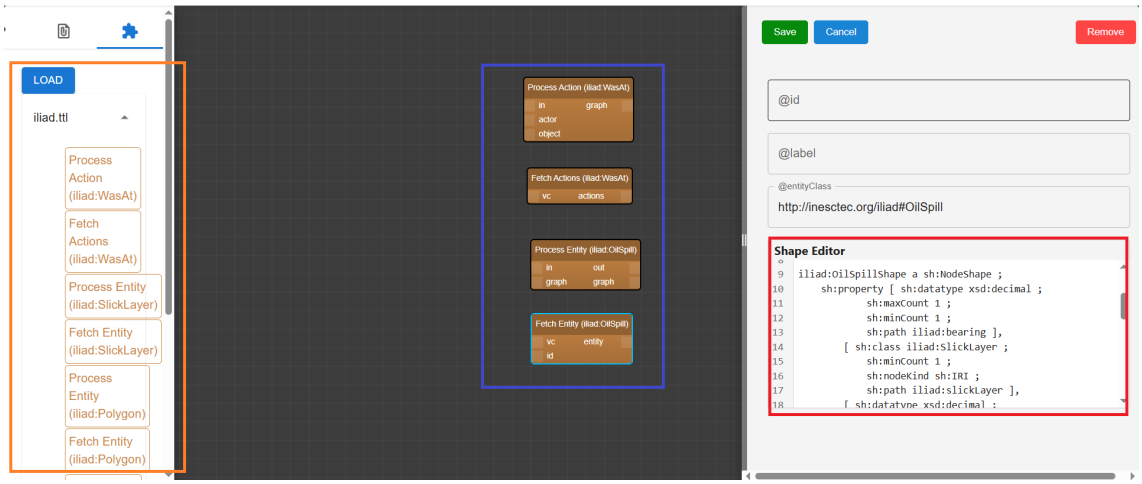


Figure 7.3: Manifest injection

The red frame marks the semantic signature of the processing oil spill node, the orange frame marks the loaded nodes from the manifest in the tool menu, and the blue frame marks canvas nodes that were dragged from the tool menu.

7.3.2 Transformer enhancements

The transformer system was enhanced to orchestrate multiple platform transformations.

The first step in the system is to input the platform manifest along with an identifier of the platform, as shown in Figure 7.4.

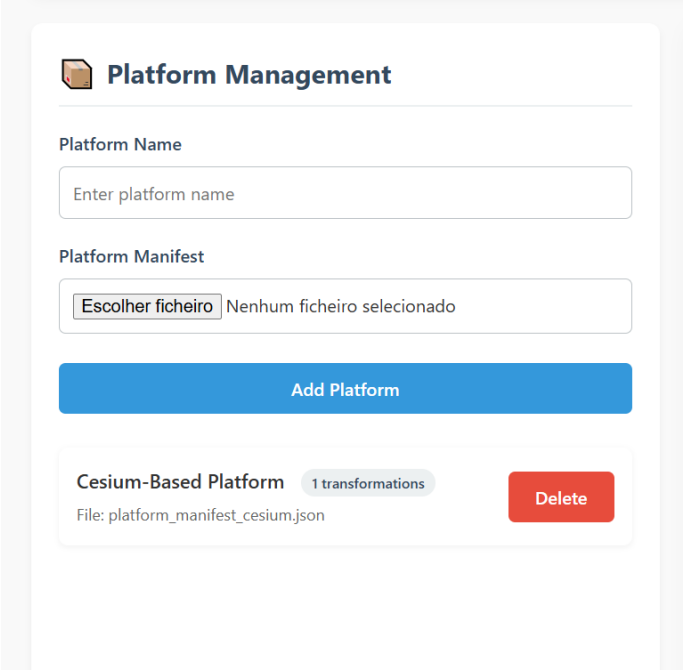


Figure 7.4: Transformer - Platform Configuration

The second step of the system is to configure the transformations. For this, a transformation process in the form of a DAG config must be provided, as well as the corresponding manifest as seen in figure 7.5.

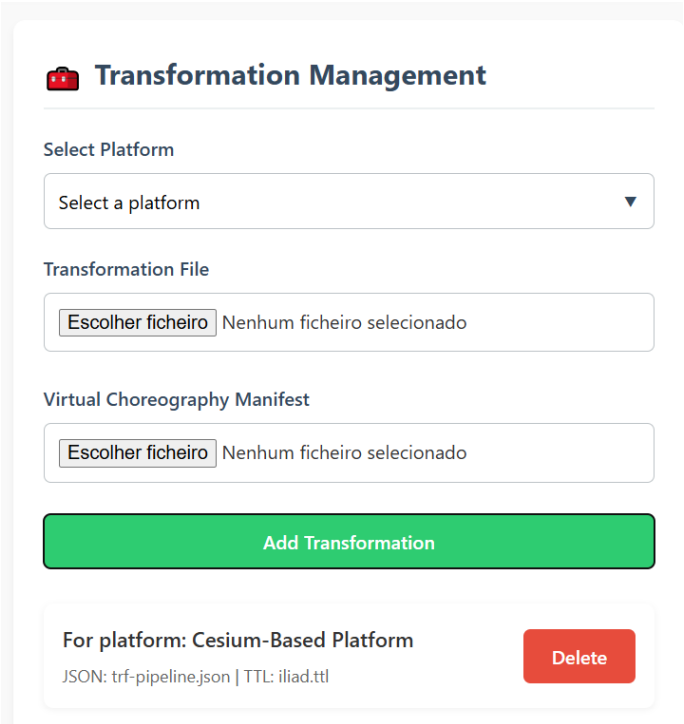


Figure 7.5: Transformer - Transformation Configuration

In this phase, an underlying algorithm on the backend establishes if both artefacts are compatible by extracting the implicit manifest in the graph, extracting the annotated nodes' shape, and establishing a connection with the provided manifest. To showcase how this algorithm reveals compatibility, the following user interface was developed as a way to understand which ingredients are compatible or not. The following example showcases the compatibility between the manifest created during this iteration and the manifest that is implicitly ingrained in the transformation graph of the Cesium transformation in Figure 7.6.

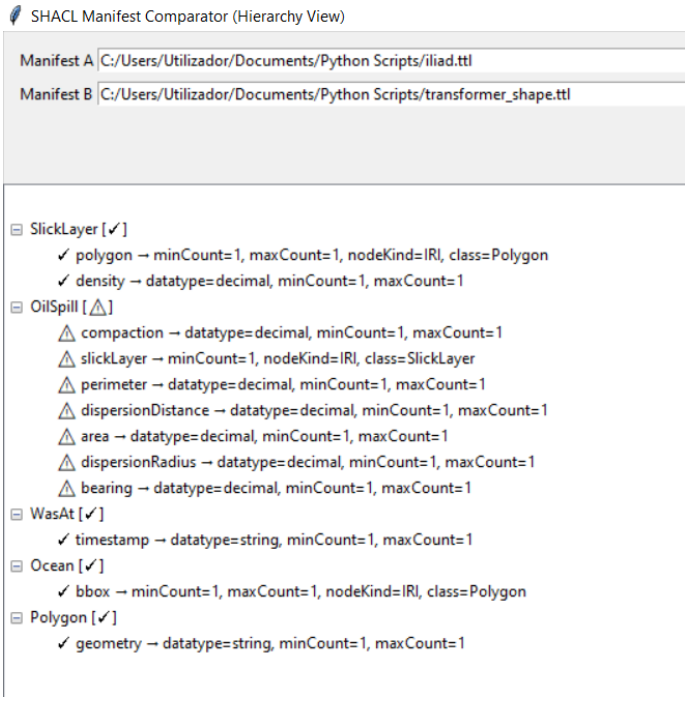


Figure 7.6: Transformer - manifest comparison

What the comparison is showing is that all of the ingredients from the manifest are compatible with the transformation, with the exception of the Oil Spill entity, since the oil spill entity is not used in the cesium transformation. This stems from the fact that the cesium platform only renders the oil spills, as aggregations of density layers with a certain geometry, therefore not having any need for the composition of these layers, as is the case with the oil spill entity. In this case, the excess ingredients are cut out of the manifest. If the manifest does not have the necessary ingredients that the transformation process requires, then the missing ingredients are declared, and the match fails.

After configuring all the manifest transformation pairs, the system is ready to receive virtual choreographies. When the virtual choreography gets inserted into the system, it goes through the reference chain, which is composed of all the manifest declarations in the orchestrator. The matching between a choreography and the manifest is established by asserting two factors:

- the choreography respects the structure of the ingredients it shares with the manifest;
- the choreography supports all the ingredients the manifest requires;

If the virtual choreography has elements out of the scope of the transformation, then these elements are simply ignored, so that the platform is still able to run the choreography.

The final list of available platforms is shown, and the user can select which platform to run the choreography 7.7

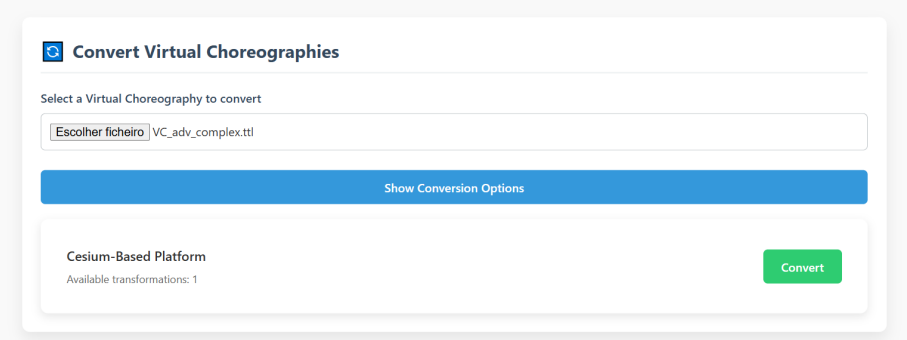


Figure 7.7: Transformer - choreography input

7.4 Demonstration

To demonstrate the prototype solution, the two scenarios from the previous iterations were tested:

7.4.1 Choreography generation

In this iteration, the choreography generation is driven by the manifest, which was built according to the domain, and within the restrictions of the field case.

As such, the vcoperators declared in the generation graph are automatically added from the manifest, and validate the resulting choreography elements they produce in the post_validation interface of the node according to the semantic annotation each node expects. With those elements, the following generation pipeline was created:

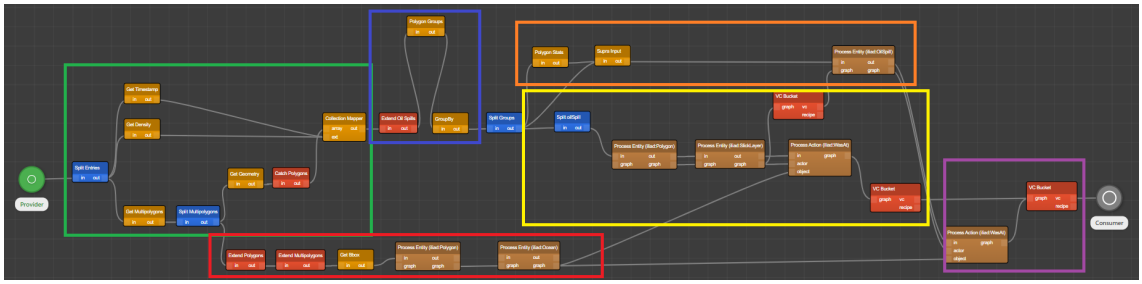


Figure 7.8: Generation Pipeline

Some new challenges occurred; the datasets cannot completely cover all the information from the manifest domain. Instead, they must rely on inferred information from additional processing.

In this pipeline, the nodes in the green frame are responsible for doing pre-processing, in this case, decomposing multi-polygons from the dataset into single polygons, and assigning each polygon with the correct density and timestamp.

The red frame is responsible for grouping all the possible polygons in the dataset and extracting a bounding box that fits all the events from the scene. Then it creates both a polygon entity, with the obtained bounding box and the ocean entity, which directly references the polygon entity, establishing a nesting relationship.

The blue frame is responsible for grouping the polygons that are directly or indirectly connected. For example, if polygon A shares the same topological space with polygon B and polygon B shares the same topological space with polygon C, even if A and C don't topologically superpose, they are still part of the same topological structure.

The yellow frame is responsible for processing each (polygon, density) after the pre-processing and generating the corresponding polygon entity and by nesting those entities in the slick layer entity. This frame is also responsible for creating the actions from the processed slick layer entities and the static ocean generated in the red frame. Finally, this layer is responsible for aggregating all the oil slicks that were grouped in the blue frame and will be nested inside the oil spill entity.

The orange frame receives the grouped polygons from the blue frame and, using the Shapely Python library, extracts statistics such as area and perimeter and redirects that data to the process oil spill node to generate the oil spill entities. This frame also receives the slick layer entities from the yellow frame and nests those entities inside the oil spill entity.

The final purple frame processes the actions between the oil spill entities and the static ocean entity. It stores all virtual choreography elements in the final choreography buffer, and when all generators are empty, redirects the choreography to the consumer.

By processing both datasets using this configuration, two different choreographies are obtained:

- The static choreography has only one oil spill actor, and five different slick layers, all sharing the same geometry and nested inside the single oil spill actor. The oil spill actor also includes the additional data derived from the generator pipeline. Only one ocean entity exists and is connected to the only polygon that represents a bounding box. Both the slick layer entities and the oil spill entity remain unchanged through time.

- the dynamic oil spill has 9 different oil spill actors and with each oil spill representing a unique subset of four different slick layer entities, showing a different behaviour with each action. Only one ocean entity exists and is connected to the only polygon that represents a bounding box.

7.4.2 Cesium-based platform transformation

For this platform transformation, the pipeline is nearly identical to the one built for cesium in the previous iteration, since it shares its ingredients. The only difference from this transformation to

the previous one is the fact that all nodes are semantically annotated, and there is no operation that fechts the oil spill entity form the choreography. Instead, the old oil spill entity fetch process was replaced by the process based on the slick layer entity, in order to respect the domain manifest.

7.4.3 Unity-based platform transformation

This platform transformation reuses the same pipeline as the previous Unity-based transformations, as it relies on the same components and connections. The key differences are: all nodes are now semantically annotated, and the oil spill entity has been replaced by a process centred on the slick layer entity. No use is given to the operator that fetches oil spill entities.

7.4.4 ThreeJS-based platform transformation

This platform transformation presents new challenges and peculiarities in the transformation process, as the platform has specific information related to its implementation, such as points per spill and a uniform density distribution per oil spill.

To map this data, certain unique data flows needed to be established:

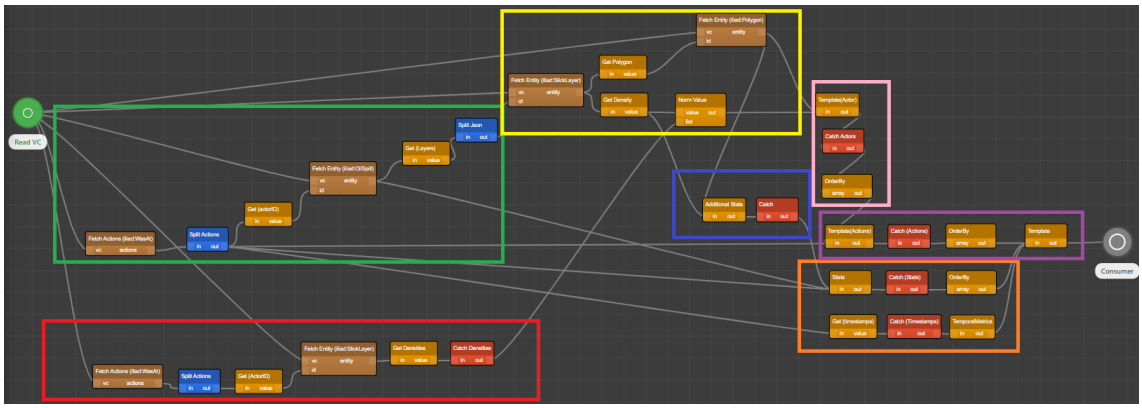


Figure 7.9: ThreeJS transformation Pipeline

The green frame is responsible for fetching the actions from the virtual choreography and resolving the oil spill actor in each action iteration.

The red frame is responsible for gathering all of the slick layer entities and providing an array with all the densities in the scene.

The yellow frame is responsible for resolving the nested entities in the oil spill actor. The slick layer entities are further decomposed into a polygon and a density that is normalised in accordance with the list of densities that are retrieved from the red frame.

The blue frame is responsible for ingesting the slick layer entity attributes and deriving the new platform-dependent information, such as the number of points in an oil Spill, which corresponds to the average number of points in all the slick layer that constitute an oil spill entity.

The pink frame is responsible for implementing template engines that make the direct connection to platform concepts. This frame formats the data from the yellow frame to create the

corresponding platform oil spills. It is also responsible for providing different colours per slick layer and ordering the slick layers based on their density.

The orange frame calculates the temporal metrics and places all of the oil spill statistics in a statistics template compatible with the platform

The purple frame is responsible for creating the platform action concepts based on the previously established actors. It is then ordered by time and is joined with the statistics and the temporal metrics to produce the last platform choreography.

7.4.4.1 Transformations

The transformations are declared in the transformation orchestrator, along with the designated platforms. This involves declaring the platforms along with their technology-dependent manifests, and then each transformation is paired with the virtual choreography manifest, which represents the domain. This setup can be seen in Figure 7.10

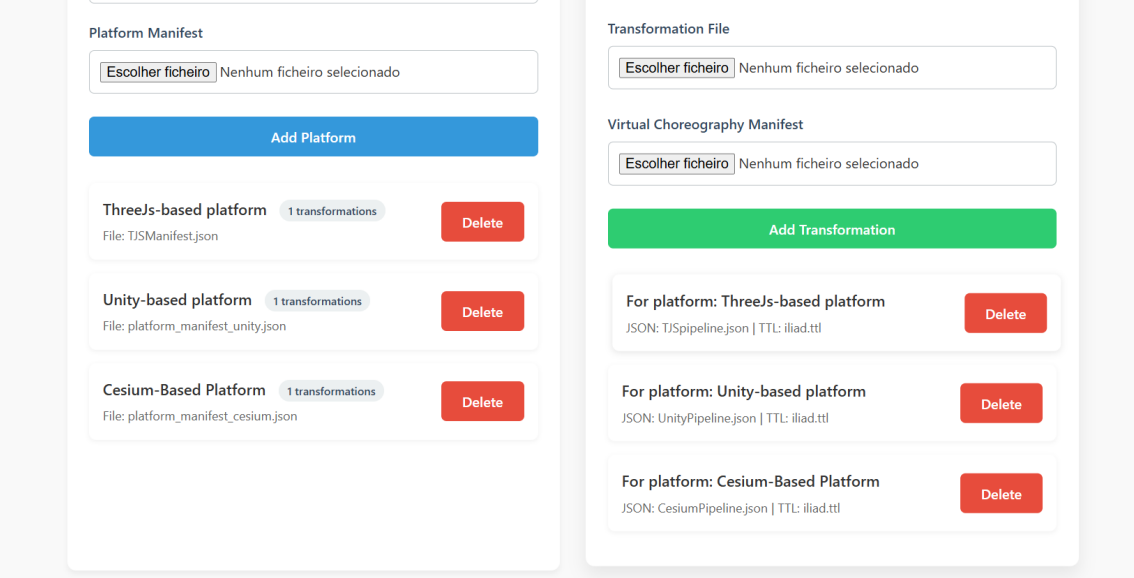


Figure 7.10: Transformation setup

After no errors were detected, both virtual choreographies from each testing scenario were input into the system as the example figure showcases 7.11:

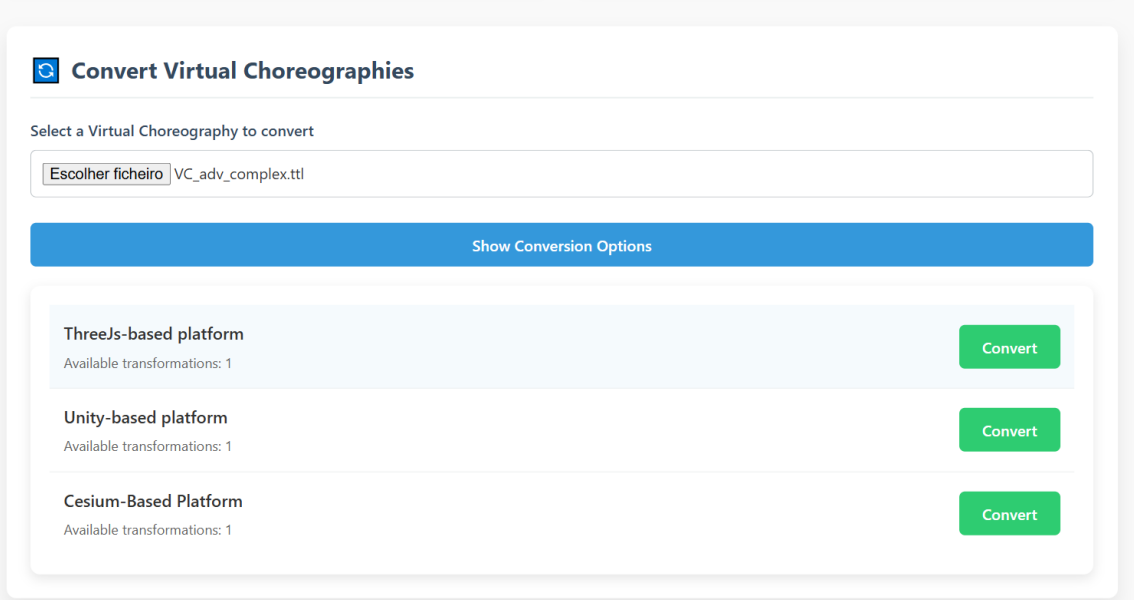


Figure 7.11: Complex Scenario virtual choregraphy input

7.4.4.2 Platform Results

All the platform choreographies were extracted and placed in the corresponding platforms.

The unity and cesium-based system provided identical outcomes to the previous two iterations 5.5 5.6 5.7 5.8.

The threeJS-based platform exhibited the following results.

- Both choreographies were successfully transformed, with only minor issues detected in the statistics field;
- When running the static oil spill choreography on the platform, the platform displayed a static oil spill with no movement or behavioural updates, as shown in Figure 5.5;
- When running the dynamic oil spill choreography, the platform displayed an evolving oil spill with different density layers, marking each timestamp as a unique behaviour, as seen in Figure 7.8.

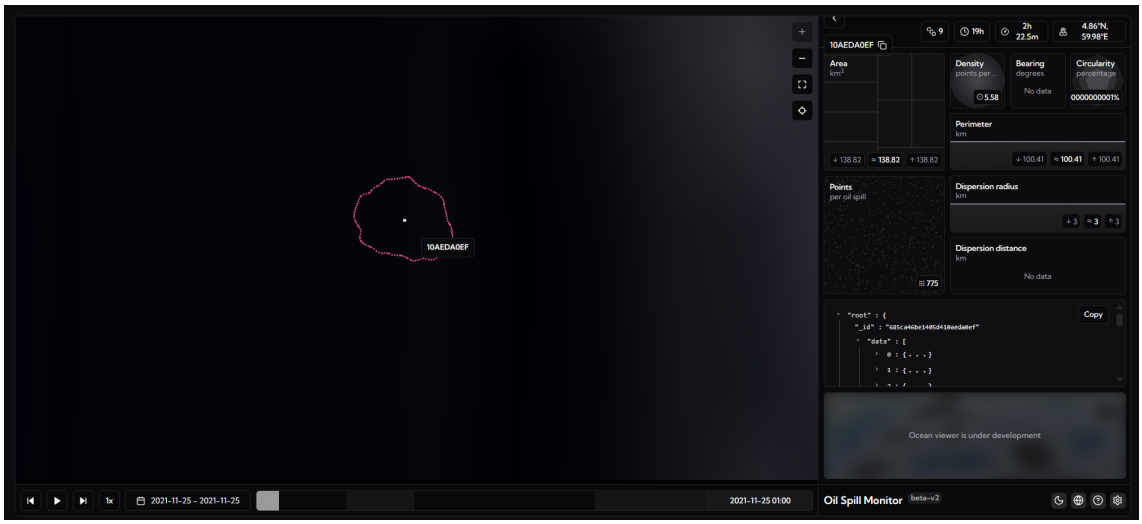


Figure 7.12: ThreeJS-based Platform(static oil spill)

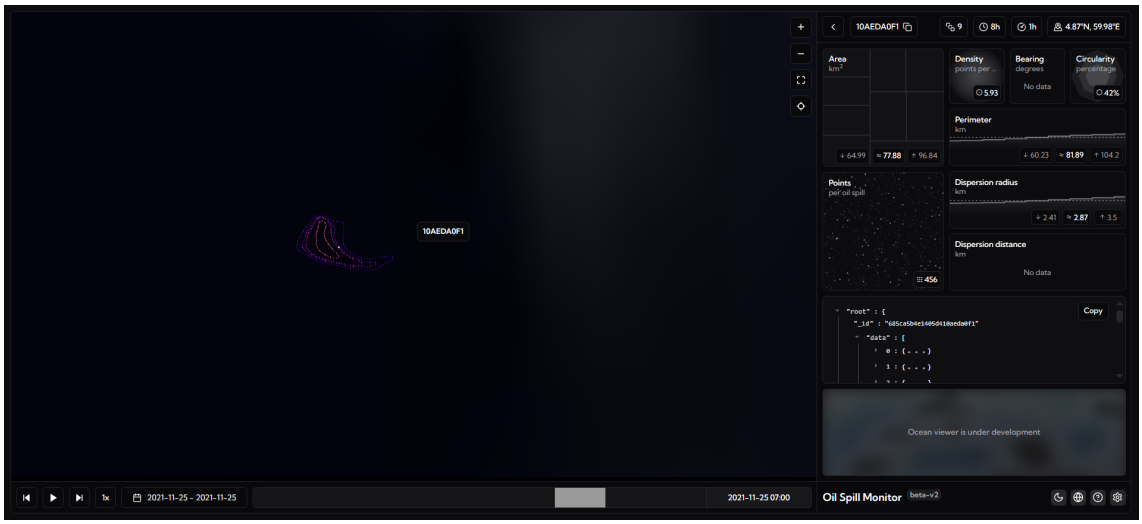


Figure 7.13: ThreeJS-based Platform(dynamic oil spill)

7.5 Evaluation

The choreography manifest outlined for this iteration was successfully validated for the oil spill domain within the restrictions of this field case. However, one technical detail was noted regarding the threeJS-based platform transformation: the statistics data, specifically the dispersion distance and the bearing, revealed no results. Upon careful inspection of the platform, it was found that the dispersion distance was 0 and the bearing was NULL. This indicated that the platform had not correctly evaluated the virtual choreography. After further examination of the platform's results, both values were neither null nor 0, showing that the relationship between the choreography manifest and the platform manifest was correctly aligned. Thus, for all the information that is available about the platform manifest, it is assumed that this result was simply a malfunction of the platform,

through which no control or measurement could be conducted. Therefore, this issue is presented as a limitation of the platform rather than a limitation of the conceptual or choreography manifest.

So the final conclusion of this demonstration is that the choreography manifest concepts were correctly mapped and represented in all three platforms.

7.5.1 It's just polygons

I was looking at the latest results from the unity transformation, and I came across an interesting question. The oil spill in the visualisation had holes in its shape, which wasn't something new that I had noticed; it's a visual feature that is implemented on top of the choreography information and is only dependent on the platform. So even though the underlying oil spill was represented using a single polygon, the platform visually adorned this information by creating these gaps in the oil, in order to get a more realistic result. Okay, that is one way to do it. What if I actually wanted to have oil spills with holes in my choreography? This thought came rumbling down, and with it, so many other questions:

The first question I asked myself was: Why haven't I thought of this earlier? Because all provided oil spill behaviours did not represent this particular scenario, the problem space hadn't demanded this kind of thinking until now.

The second question was: Does my choreography manifest predict this scenario? Unfortunately, it doesn't, since it only considers that the slick of oil is represented by an array of latitude and longitude pairs. This also raised another question about what would be the best way to represent topologies in a platform-independent manner for oil spills. For instance, couldn't an abstract mathematical representation express the topologies of an oil spill? Whatever solution one may opt to take, this limitation in the manifest is not unfixable, as it is only needed to add another ring structure, that is nested inside the polygon. The outermost polygon delineates the oil shape, and the innermost rings delineate the holes inside the slick of oil, similarly to how GeoJSON represents this kind of scenario.

Finally, the third question: Do the platforms even account for this problem? To my surprise, they do not, neither of them does. The concept of the platforms that correspond to an oil spill doesn't provide the capability to place holes inside an oil slick. So, is there some sort of patch solution that could be done? As an engineer, of course, there is. It's not acceptable to assume there is no solution simply because the system wasn't designed to function in a specific way. This can be shown in the Unity platform. If we stack an oil slick on top of another one and give it a density of 0, then it is possible to show holes in the oil spill without directly changing the code 7.14.

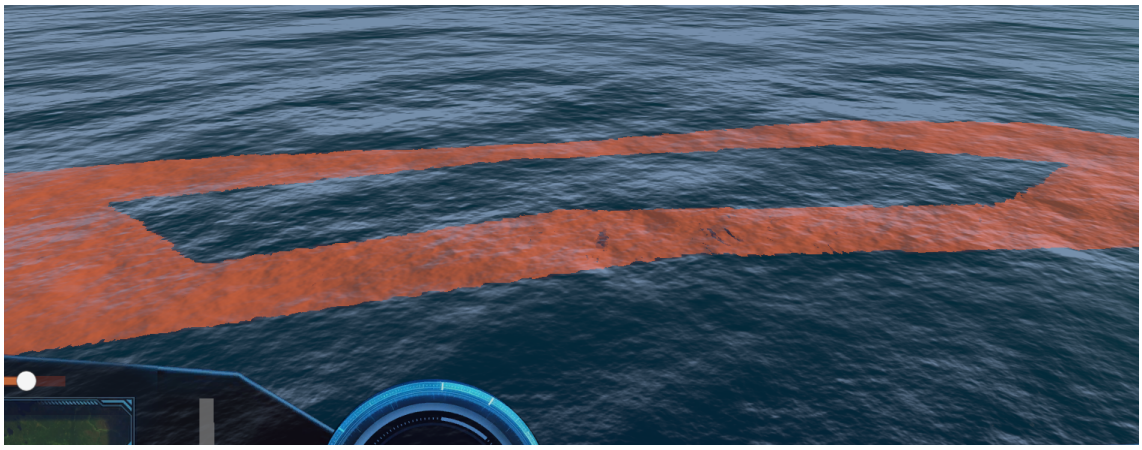


Figure 7.14: Unity patch for oil spills with holes

Another solution that is also possible is to disassemble the oil Slicks with holes into different pieces that are topologically uniform. Each would be a single polygon that could then be placed together as a puzzle in order to build the final oil spill.

In essence, platforms are conceptually modelling objects that are more closely related to polygons than oil spills.

In order to successfully integrate these improvements into the current workflow, three things would need to be done:

- The first is to fix the manifest, as the concept of holes is specific to the domain.
- The second is to change the generation process to reflect the newer changes in the manifest.
- The third would be to apply one of the patch solutions to the transformation pipelines, such as marking holes as polygons with density zero, or by breaking the oil spill into topologically uniform structures, and assembling them in the actors list of each platform.

7.5.2 Platform misconception

One of the early problems outlined in the field case was the lack of a platform manifest. Throughout this thesis, it was considered that the only way to represent this manifest was through a JSON schema that could describe the data interface of a platform. However, as explained in the last section, the concepts of the platform can be too abstract or tied to the technological concepts that make it difficult to compare them to high-level semantic spaces such as the choreographies manifest. Such has been one of the most challenging factors throughout these iterations. In fact, though inspired by Demetrius et al. [24] recipe concept, I couldn't stop considering that maybe I was doing something different. The previous approaches don't apply any form of transformation, just direct alignments between concepts [24, 47]. But for the specific field case presented in this thesis, such a direct approach has not been possible given the structural and technological peculiarities

faced in integrating choreographies into platforms. As I was facing these thoughts, the same keywords came to mind: Technological, data interface, data structure... All these characteristics were not aligned with the very definition of a manifest(previously named recipe):

> The platforms that want to be able to reproduce these choreographies will have to make explicit their semantic space, through a platform recipe. [24]

The platform manifest represents **explicitly** the semantic space on which a platform operates, not something implicit and hidden inside data structures, which are directly used by the code of the platforms. Changing these data structures would eventually mean changing the platform code. We can consider that previous approaches, which have relied on Model-Driven Engineering, have predominantly applied model-to-model transformations, as they use well-defined semantic spaces and ontologies [24, 47] . In fact, the very technological tool I have used up until this point to represent platform choreographies has been templates. I finally understood it: I wasn't performing model-to-model transformations, I was performing model-to-text transformations. Up until this point, I haven't been performing a matching between semantic spaces; I have been staging choreographies by aligning and transforming high-level concepts with low-level technological configurations. So, where was my platform manifest? Instead of having a platform manifest, we can have a transformation manifest, which is the set of semantic annotations that are embedded into the transformation pipelines. Those specifications highlight exactly what concepts every transformation needs. I mean to say transformation, not platform, because multiple transformations can be specified for the same platform. It then depends on how powerful and for what purpose each transformation is. For example, we may have a transformation for the Unity-based platform that doesn't articulate the necessary platform elements to render an oil spill with holes. However, this does not need to be a limitation, as shown in the previous section. Without recoding the application, a transformation could be established to render oil spills with holes. This capability would be directly reflected in the transformation manifest, as it now accepts oil spills with holes at the highest semantic level. This process is essentially the embodiment of low-level technological characteristics into a high-level semantic space 7.15 that is annotated in the nodes of the graph that interface with virtual choreographies.

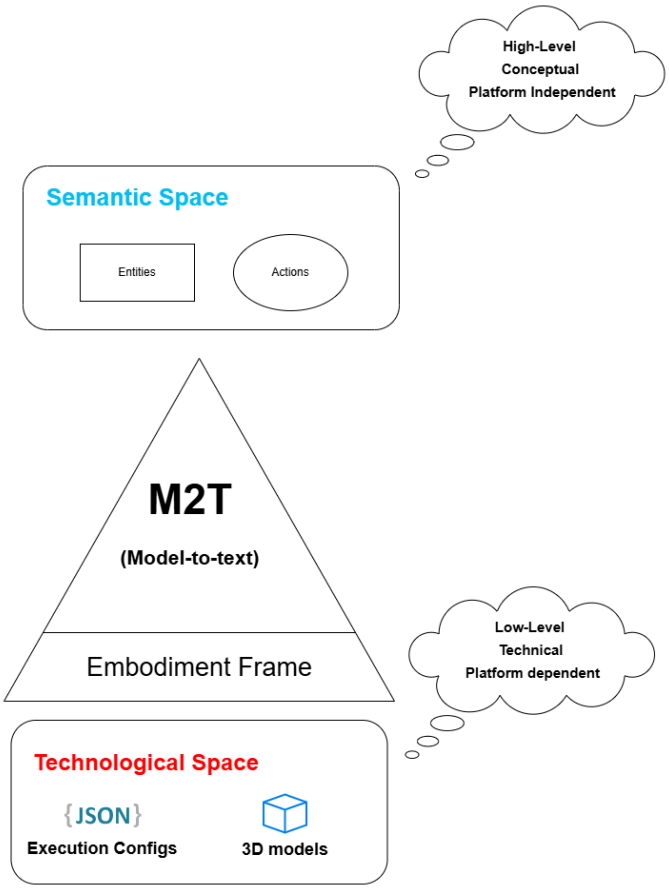


Figure 7.15: Embodiment Frame

While reviewing previous approaches [23, 24, 47] in choreography staging and execution, it is not clear how these processes are conducted. The reason for this is that these approaches primarily focus on the model-to-model transformations and platform choreography validation. No solution in the current state of the art addresses the need to embody low-level technological characteristics into high-level semantics when the platforms don't offer a well-defined semantic space.

In my latest solution, the transformation manifest is associated with the same concepts from the virtual choreography manifest, since I used the same manifest both in generating and transformation. Therefore, the matching mechanism that associates a choreography manifest with a transformation manifest is a direct match of ingredients as shown in Figure 7.16. Which essentially means the proposed manifest for the conversion system is always going to be equivalent to the transformation manifest, since it will either be equivalent to the choreography manifest(e.g threeJS transformation) or a subset(e.g cesium transformation).

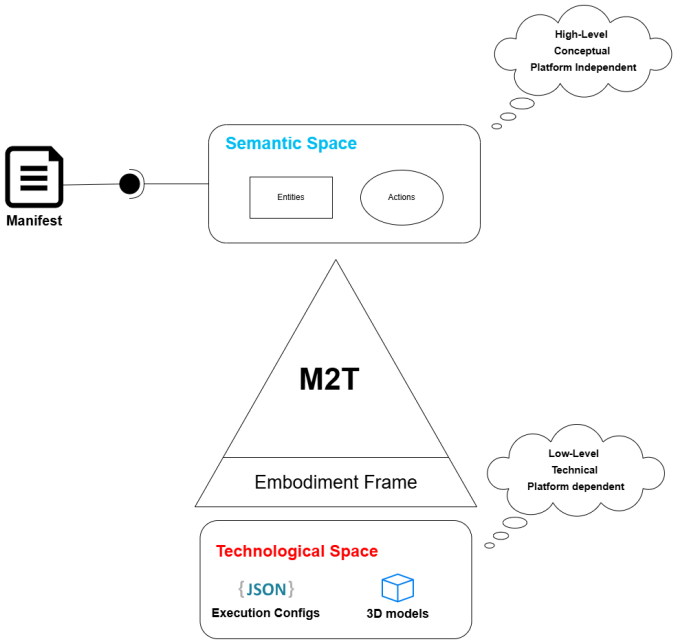


Figure 7.16: Current approach

This isn't exactly a problem, since the embodiment frames can be instantiated with any manifest; however, in many cases, an embodiment frame can be made in accordance with a manifest that is not directly a match with the choreography manifest. In this case, the Demetrius et al solution [24] can enter as a complement for the proposed solution. Since now we have two well-defined semantic spaces, an alignment may be possible 7.17.

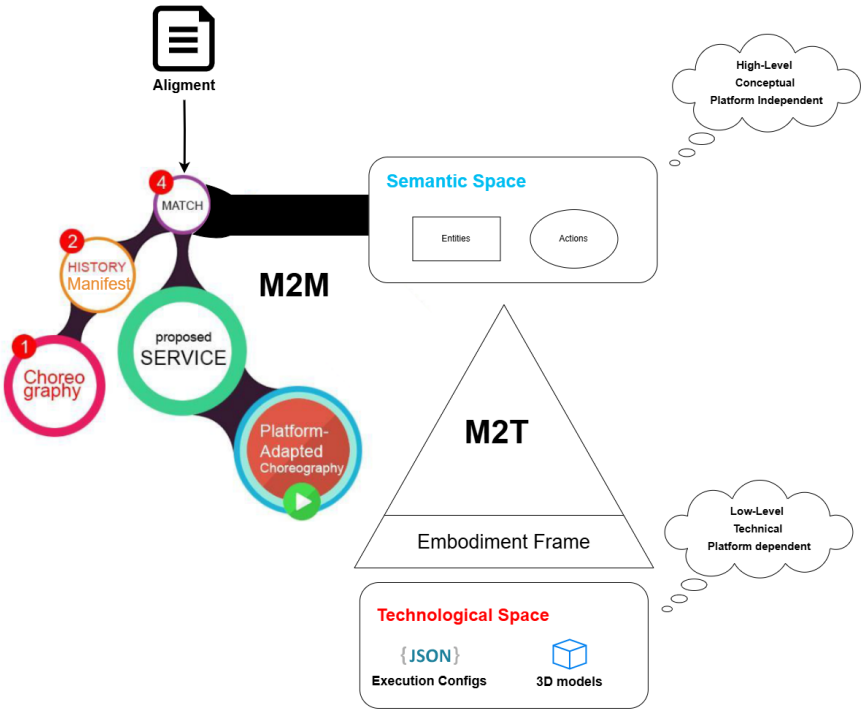


Figure 7.17: Proposed approach

7.5.3 Canvas limitations

One of the key limitations in the current graph-based execution model lies in its inflexibility: every path in the graph is executed, regardless of whether all necessary entities are present in the input choreography. This inflexibility means that even if a choreography lacks certain entities defined in the transformation manifest, the corresponding transformation steps will still be triggered. The result? Errors, since the system tries to process components that simply don't exist. As it stands, choreographies must provide all the ingredients that the graph supports in order to be processed correctly, which, according to the concept, should not be a possibility since a choreography representing a subset of the transformation manifest is still viable. This full-coverage requirement is a major bottleneck for flexibility and reuse. A potential solution is to introduce conditional branching into the graph engine, where certain branches are only activated based on contextual checks, such as the presence or type of an actor. This would enable the system to ignore irrelevant parts of the graph and focus only on what's applicable, allowing the reuse of generic graphs for partial or variant choreographies 7.18. For instance, if a choreography includes only certain semantic actors or actions, only those relevant branches would be executed — the rest would be skipped.

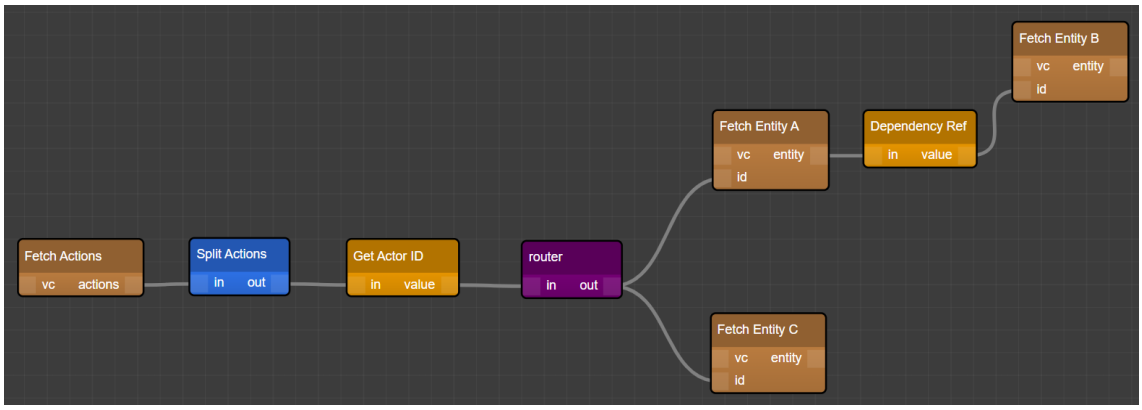


Figure 7.18: Mock router in the canvas

While considering this solution, another issue arises: some entities depend on others due to nested relationships. As illustrated in Figure 7.18, the processing of entity B relies on the processing of entity A. The graph executes the fetch entity operation in a sequence for these different entities to gather their information. Consequently, certain sets of ingredients are interconnected, and if we can verify these sets in the graph, it becomes possible to construct a final list of independent sets. This allows us to identify which sets of ingredients may be executed independently, even if the choreography does not support other sets.

Another improvement to the interface involves managing visual complexity. With many entities — especially nested ones — the canvas can quickly become overcrowded and difficult to navigate. A potential solution is to introduce visual abstraction, where all entity-related nodes are grouped into a single composite component.

7.6 Communication

The present cycle was communicated through this thesis.

7.7 Conclusions

The choreography manifest was successfully validated on all three platforms. Despite encountering technical issues specific to the threeJS platform, the manifest proved to be conceptually sound and functionally consistent. The demonstration confirmed that choreography semantics can be accurately staged and rendered, even across technologically diverse platforms.

The proposed innovation involves rethinking the platform manifest as a transformation manifest. This approach enables alignment through high-level semantics and technological specifications via model-to-text transformations. As a result, this supports richer and more flexible choreography staging than what current solutions can provide.

7.7.1 Limitations

- **Manifest topology limitation**: The current choreography manifest does not support nested ring structures for holes.
- **Graph execution inflexibility**: all branches executed regardless of input choreography coverage.
- **Lack of conditional logic**: The transformation graph cannot selectively activate nodes based on context.

Chapter 8

Conclusions

This chapter outlines the primary conclusions of this dissertation by first answering the research questions, followed by the limitations and future work sections.

8.1 Research Questions

8.1.1 RQ1: How can virtual choreographies be employed in a domain and platform-independent manner to overcome the limitations of current domain-specific approaches?

This work explored how virtual choreographies can be used to model behavioural patterns in a way that is independent of specific domains or platforms. The investigation focused on two core aspects: identifying conceptual similarities with existing modelling paradigms and examining the representational tools best suited to support such abstraction.

8.1.1.1 RQ1.1: What other concepts exist in the literature that share similarities with virtual choreographies?

The literature highlights three conceptually related paradigms: scripts, choreographies, and process models. Scripts provide a narrative-driven perspective, offering semantically enriched representations of behaviour, capturing sequences of actions, participants, and evolving event structures. Choreographies focus on communication patterns among distributed services, modelling interaction through message exchange in decentralised scenarios. Process models, in contrast, prioritise procedural logic, system states, and decision points, supporting executable and machine-readable representations that structure workflows and transitions across multiple entities.

8.1.1.2 RQ1.2: What are the most appropriate specifications and modeling languages for describing behavior patterns in virtual choreographies across different domains?

The virtual choreography state of the art showed a diverse range of modeling approaches suitable for describing behavioural patterns across domains. Process modeling techniques such as

flowcharts and Petri nets are used to express procedural logic and behavioural constraints. Artifact-centric models focus on data lifecycle and workflow structure. Event-based representations use structures like triples and multi-argument models to capture semantic detail. Structural representations rely on graphs, nested structures, and relational models to define relationships and constraints. Standard formats such as XML, RDF, and ontologies enable semantic consistency and interoperability. Behavioural models include state machines, declarative specifications, and procedural definitions to express execution logic. Additionally, specialised representations support discrete and distributed event modeling, action abstraction, and decision structures.

In this work, an RDF-oriented solution was implemented, grounded in minimal primitives: entities and actions. Concepts such as nesting and relational structuring were applied, along with an action-oriented triple representation (subject–action–object) capable of capturing semantic meaning and handling varying conceptual complexity. This structure allows new domains to be instantiated over the same modelling foundation, as demonstrated with the oil spill domain through the use of manifest definitions. While less complex than other domain-specific approaches, it provides a generalisable and semantically expressive framework that goes beyond constrained formats such as xAPI.

8.1.2 RQ2: What are the specific challenges and opportunities presented by the oil spill field case that can inform the broader application of virtual choreographies?

The oil spill domain introduced both conceptual and technical challenges to the application of virtual choreographies. Unlike previous contexts where choreographies revolved around discrete entities, such as agents or characters in virtual environments, the oil spill scenario deals with evolving topologies defined by physical properties like dispersion and density. This demanded a shift in how entities and behaviours are represented, moving from clearly defined actors to dynamic physical phenomena.

8.1.2.1 RQ2.1: How can existing scientific datasets be used to create virtual choreographies?

A dataflow-based architecture was introduced to generate virtual choreographies from scientific datasets. Within this architecture, two types of graphs were explored: a static graph, which enabled simpler logic but with reduced flexibility, and a dynamic graph-building mechanism using directed acyclic structures that allowed the choreography architecture itself to be assembled progressively. In the dynamic DAG, choreography generation begins with the definition of a domain-specific manifest. This manifest acts as a semantic anchor, guiding the construction of the graph and enabling the semi-automatic extraction of virtual choreographies through a semantically driven process.

8.1.2.2 RQ2.2: How can virtual choreographies be effectively integrated into oil spill monitoring platforms?

Integration was achieved through a manifest matching approach. Specifically, the system matches the manifest of a choreography with the manifest of a transformation pipeline. Graphs are semantically annotated, allowing the extraction of transformation manifests directly from their structure. If the matching between the choreography manifest and the transformation manifest is successful, a proposed manifest is generated. This proposed manifest defines the semantic space that choreographies must satisfy in order to be executed within a given transformation. Rather than relying on a single comparison point, this mechanism supports chains of proposed manifests, each linked to one transformation and platform. This enables automatic identification of compatible execution platforms for a given choreography.

This approach contributed both to the oil spill domain by introducing a structured, interoperable way to define and execute domain knowledge and to the field of virtual choreographies itself. The inherent complexity of the platforms and the absence of native platform manifests introduced the need for a new type of transformation layer. This led to the conceptualisation of the embodiment frame, an abstraction mechanism to elevate heterogeneous technical platforms to semantical spaces.

8.2 Limitations

Despite demonstrating the applicability of virtual choreographies in a complex domain, this work has some limitations.

One key limitation lies in the topology of the manifest structure. The current implementation assumes only spills without holes, which prevents the representation of these behaviour patterns.

At the execution level, the graph architecture exhibits limited flexibility. All branches of a transformation graph are executed regardless of whether the input choreography covers the corresponding elements. This unconditional execution prevents the graph from isolating and executing only the relevant parts, meaning that if a single expected element is missing, the entire choreography cannot be accurately represented or executed.

Finally, the validation of the proposed architecture was constrained by limitations imposed by the field case.

8.3 Future Work

Several directions can be pursued to extend and consolidate the results presented in this thesis. A natural next step is the evaluation of both the proposed framework and the underlying information model across new domains beyond oil spill monitoring. This would test the generalisability of the approach and validate its flexibility in capturing different types of choreography semantics.

An additional opportunity lies in developing a choreography "weaver". A visual canvas that allows users to build choreographies step by step by uploading a manifest and manually assembling instances of its components could allow greater flexibility in certain scenarios, rather than creating choreographies automatically by creating a processing pipeline.

Additionally, improving the flexibility of the transformation pipeline remains a key technical goal. The current DAG-based architecture should be extended to support more expressive execution logic, including conditional branching and selective activation of nodes based on context. This would allow the choreography system to accommodate partial scenarios that don't encompass the entire proposed manifest, but are still viable.

Finally, the embodiment frame, introduced in this work as a conceptual layer mediating between semantic definitions and platform-specific implementations, should be validated. Its role and applicability across other domains and transformation scenarios remain open for exploration.

8.4 Final Thoughts

This thesis makes a significant contribution to the oil spill domain by addressing the lack of integration across multiple platforms of the behavioural patterns of oil spills. It introduces a new framework for representing complex environmental behaviours through virtual choreographies. This framework is platform-independent, semantically expressive, and operationally versatile.

As highlighted in the literature review, multiplatform integration is just one of many potential applications for virtual choreographies in this field. The approach presented here represents only one example of a broader capability. Further advancements could be achieved by applying virtual choreographies to other scientific and operational challenges, such as ecological impact modelling and the visualisation of stochastic or multiple future scenarios.

The field case played two roles: it validated the choreography concept by enabling the formal representation and integration of complex, continuous entities such as oil spills, and it simultaneously exposed limitations in current virtual choreography concepts. It highlighted the lack of clarity in how choreographies can be staged, specifically in contexts where platforms offer no clear semantic anchors (e.g., no manifests). This observation led to new conceptual contributions in virtual choreographies within the context of platform integration.

Finally, this work leaves behind a proof of concept and a set of tools, formal constructs, and theoretical advancements drawn from both literature and practical iteration. These assets are open for adaptation by future scientists, engineers, and developers across domains.

This thesis is not the end, but another step for a class of interoperable, declarative, and behaviour-centric modelling systems.

References

[1] Arne J. Berre et al. “Iliad Digital Twins of the Ocean Interoperability Architecture”. In: *IGARSS 2024 - 2024 IEEE International Geoscience and Remote Sensing Symposium*. ISSN: 2153-7003. July 2024, pp. 3568–3571. DOI: 10.1109/IGARSS53475.2024.10641949. URL: https://ieeexplore.ieee.org/document/10641949 (visited on 04/09/2025).

[2] David Burggraf. *OGC KML 2.3*. English. Implementation Standard. Publisher: Open Geospatial Consortium. Aug. 2015. URL: https://docs.ogc.org/is/12-007r2/12-007r2.html (visited on 04/09/2025).

[3] H. Butler et al. *The GeoJSON Format*. Request for Comments RFC 7946. Num Pages: 28. Internet Engineering Task Force, Aug. 2016. DOI: 10.17487/RFC7946. URL: https://datatracker.ietf.org/doc/rfc7946 (visited on 04/09/2025).

[4] Valeria Cardellini et al. “MOSES: A Framework for QoS Driven Runtime Adaptation of Service-Oriented Systems”. In: *IEEE Transactions on Software Engineering* 38.5 (2012), pp. 1138–1159. DOI: 10.1109/TSE.2011.68.

[5] Fernando Cassola et al. “Design and Evaluation of a Choreography-Based Virtual Reality Authoring Tool for Experiential Learning in Industrial Training”. In: *IEEE Transactions on Learning Technologies* 15.5 (2022), pp. 526–539. DOI: 10.1109/TLT.2022.3157065.

[6] Alberto Corbi and Daniel Burgos. “Review of Current Student-Monitoring Techniques used in eLearning-Focused recommender Systems and Learning analytics. The Experience API LIME model Case Study”. In: *International Journal of Artificial Intelligence and Interactive Multimedia* 2 (Sept. 2014), pp. 44–52. DOI: 10.9781/ijimai.2014.276.

[7] Jens Dietrich, John Hosking, and Jonathan Giles. “A Formal Contract Language for Plugin-based Software Engineering”. In: *12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*. July 2007, pp. 175–184. DOI: 10.1109/ICECCS.2007.7. URL: https://ieeexplore.ieee.org/document/4276314 (visited on 06/17/2025).

[8] Inc. Environmental Systems Research Institute. “ESRI Shapefile Technical Description”. en. In: (1997).

[9] Merv Fingas and Carl E. Brown. “A Review of Oil Spill Remote Sensing”. en. In: *Sensors* 18.1 (Jan. 2018). Number: 1 Publisher: Multidisciplinary Digital Publishing Institute, p. 91. ISSN: 1424-8220. DOI: 10.3390/s18010091. URL: https://www.mdpi.com/1424-8220/18/1/91 (visited on 02/20/2025).

[10] Martín Garriga et al. *Web services composition mechanisms: A review*. In: *IETE Technical Review (Institution of Electronics and Telecommunication Engineers, India)* vol. 32 (2015), pp. 376–383. DOI: 10.1080/02564602.2015.1019942.

[11] Roy Grønmo, Michael C. Jaeger, and Hjørdis Hoff. "Transformations Between UML and OWL-S". In: *Model Driven Architecture – Foundations and Applications*. Ed. by Alan Hartman and David Kreische. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 269–283. ISBN: 978-3-540-32093-7.

[12] Yi Han et al. *A survey of script learning*. In: *Frontiers of Information Technology and Electronic Engineering* vol. 22 (2021), pp. 341–373. DOI: 10.1631/FITEE.2000347.

[13] J.S. Hayworth, T.P. Clement, and J.F. Valentine. "Deepwater Horizon oil spill impacts on Alabama beaches". In: *Hydrol. Earth Syst. Sci.* 15.12 (2011), pp. 3639–3649. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84863998205&partnerID=40&md5=6b8b7b2899d560768fccde2b3294bc43.

[14] Alan Hevner et al. "Design Science in Information Systems Research". In: *Management Information Systems Quarterly* 28 (Mar. 2004), pp. 75–.

[15] Alaa Akram Huby, Rafid Sagban, and Raaid Alubady. "Oil Spill Detection based on Machine Learning and Deep Learning: A Review". In: *2022 5th International Conference on Engineering Technology and its Applications (IICETA)*. ISSN: 2831-753X. May 2022, pp. 85–90. DOI: 10.1109/IICETA54559.2022.9888651.

[16] Richard Hull. "Artifact-centric business process models: Brief survey of research results and challenges". In: vol. 5332 LNCS. 2008, pp. 1152–1163. DOI: 10.1007/978-3-540-88873-4_17. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85099426094&doi=10.1007%2f978-3-540-88873-4_17&partnerID=40&md5=770506222ee99936f2afbd5f38920911.

[17] Hamid Jafarzadeh et al. "Oil spill detection from Synthetic Aperture Radar Earth observations: a meta-analysis and comprehensive review". EN. In: *GIScience & Remote Sensing* (Oct. 2021). Publisher: Taylor & Francis. ISSN: 1548-1603. URL: https://www.tandfonline.com/doi/abs/10.1080/15481603.2021.1952542 (visited on 02/20/2025).

[18] M. Janati, M. Kolahdoozan, and H. Imanian. "Artificial Neural Network Modeling for the Management of Oil Slick Transport in the Marine Environments". en. In: *Pollution* 6.2 (Apr. 2020). Publisher: University Of Tehran Press, pp. 399–415. ISSN: 2383-451X. DOI: 10.22059/poll.2020.289549.684. URL: https://jpoll.ut.ac.ir/article_75229.html (visited on 06/15/2025).

[19] Zongchen Jiang et al. "Research on Thermal Infrared Remote Sensing Detection of Oil Spill on Sea Surface". In: *IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium*. 2022, pp. 7970–7973. DOI: 10.1109/IGARSS46834.2022.9883855.

[20] Panagiota Keramea et al. "Modes of Operation and Forcing in Oil Spill Modeling: State-of-Art, Deficiencies and Challenges". en. In: *Journal of Marine Science and Engineering* 11.6 (June 2023). Number: 6 Publisher: Multidisciplinary Digital Publishing Institute, p. 1165. ISSN: 2077-1312. DOI: 10.3390/jmse11061165. URL: https://www.mdpi.com/2077-1312/11/6/1165 (visited on 02/20/2025).

[21] Panagiota Keramea et al. "Oil Spill Modeling: A Critical Review on Current Trends, Perspectives, and Challenges". en. In: *Journal of Marine Science and Engineering* 9.2 (Feb. 2021). Number: 2 Publisher: Multidisciplinary Digital Publishing Institute, p. 181. ISSN: 2077-1312. DOI: 10.3390/jmse9020181. URL: https://www.mdpi.com/2077-1312/9/2/181 (visited on 02/20/2025).

[22] S. Kumaran et al. “Using a model-driven transformational approach and service-oriented architecture for service delivery management”. In: *IBM Systems Journal* 46.3 (2007), pp. 513–529. DOI: 10.1147/sj.463.0513.

[23] Demetrius Lacet, Filipe Penicheiro, and Leonel Morgado. “Magical Board Theatre: interactive stories that can be played on multiple boards -two educational prototypes”. In: *Proceedings videojogos 2020*. 2020, pp. 164–176. ISBN: 978-989-54606-2-5. DOI: 10.13140/RG.2.2.11068.16003.

[24] Demetrius Lacet et al. “Preserving story choreographies across multiple platforms: An approach to platform-independent reuse of characters’ behaviors for games, simulations, animations and interactive videos”. In: *Proceedings of the 9th International Conference on Digital and Interactive Arts*. ARTECH ’19. New York, NY, USA: Association for Computing Machinery, Feb. 2020, pp. 1–4. ISBN: 978-1-4503-7250-3. DOI: 10.1145/3359852.3359916. URL: https://dl.acm.org/doi/10.1145/3359852.3359916 (visited on 06/16/2025).

[25] Danielle Levac, Heather Colquhoun, and Kelly K. O’Brien. “Scoping studies: advancing the methodology”. In: *Implementation Science* 5.1 (Sept. 2010), p. 69. ISSN: 1748-5908. DOI: 10.1186/1748-5908-5-69. URL: https://doi.org/10.1186/1748-5908-5-69 (visited on 11/04/2024).

[26] Chris Little, Jon Blower, and Maik Riechert. *OGC CoverageJSON Community Standard*. 2023. URL: https://docs.ogc.org/cs/21-069r2/21-069r2.html (visited on 04/09/2025).

[27] P Martin et al. “Development of a GIS-based Spill Management Information System”. en. In: *Journal of Hazardous Materials* 112.3 (Aug. 2004), pp. 239–252. ISSN: 03043894. DOI: 10.1016/j.jhazmat.2004.05.014. URL: https://linkinghub.elsevier.com/retrieve/pii/S0304389404002596 (visited on 02/27/2025).

[28] Aitor Mata, Juan M. Corchado, and Dante I. Tapia. “CROS: A Contingency Response multi-agent system for Oil Spills situations”. In: *Applied Soft Computing* 11.3 (Apr. 2011), pp. 3147–3159. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2010.12.017.

[29] Pierpaolo Mincarone et al. “Standardized languages and notations for graphical modelling of patient care processes: a systematic review”. In: *International Journal for Quality in Health Care* 30.3 (Jan. 2018), pp. 169–177. ISSN: 1353-4505. DOI: 10.1093/intqhc/mzx197. URL: https://doi.org/10.1093/intqhc/mzx197.

[30] Torill Moen. “Reflections on the Narrative Research Approach”. EN. In: *International Journal of Qualitative Methods* 5.4 (Dec. 2006). Publisher: SAGE Publications Inc, pp. 56–69. ISSN: 1609-4069. DOI: 10.1177/160940690600500405. URL: https://doi.org/10.1177/160940690600500405 (visited on 06/23/2025).

[31] Saeed Mohammadiun et al. “Intelligent computational techniques in marine oil spill management: A critical review”. In: *Journal of Hazardous Materials* 419 (Oct. 2021), p. 126425. ISSN: 0304-3894. DOI: 10.1016/j.jhazmat.2021.126425. URL: https://www.sciencedirect.com/science/article/pii/S030438942101390X (visited on 02/20/2025).

[32] Leonel Morgado and Dennis Beck. “Unifying Protocols for Conducting Systematic Scoping Reviews with Application to Immersive Learning Research”. In: *2020 6th International Conference of the Immersive Learning Research Network (iLRN)*. 2020, pp. 155–162. DOI: 10.23919/iLRN47897.2020.9155093.

[33] David Murphy et al. “An in-depth survey of the oil spill literature since 1968: Long term trends and changes since Deepwater Horizon”. In: *Marine Pollution Bulletin* 113.1 (2016), pp. 371–379. ISSN: 0025-326X. DOI: https://doi.org/10.1016/j.marpolbul.2016.10.028. URL: https://www.sciencedirect.com/science/article/pii/S0025326X1630844X.

[34] Srini Narayanan and Sheila A. McIlraith. “Simulation, verification and automated composition of web services”. In: *Proceedings of the 11th International Conference on World Wide Web*. WWW ’02. Honolulu, Hawaii, USA: Association for Computing Machinery, 2002, pp. 77–88. ISBN: 1581134495. DOI: 10.1145/511446.511457. URL: https://doi.org/10.1145/511446.511457.

[35] S. Othumpangat and V. Castranova. “Oil Spills”. In: *Encyclopedia of Toxicology (Third Edition)*. Ed. by Philip Wexler. Third Edition. Oxford: Academic Press, 2014, pp. 677–681. ISBN: 978-0-12-386455-0. DOI: https://doi.org/10.1016/B978-0-12-386454-3.00359-6. URL: https://www.sciencedirect.com/science/article/pii/B9780123864543003596.

[36] Gyunam Park, Minsu Cho, and Jiyoon Lee. *Leveraging machine learning for automatic topic discovery and forecasting of process mining research: A literature review*. In: *Expert Systems with Applications* vol. 239 (2024). DOI: 10.1016/j.eswa.2023.122435. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85176244652&doi=10.1016%2fj.eswa.2023.122435&partnerID=40&md5=0dd9b626946bd43b9a16a2afaa01f9ea.

[37] Ken Peffers et al. “A design science research methodology for information systems research”. In: *Journal of Management Information Systems* 24 (Jan. 2007), pp. 45–77.

[38] C. Peltz. “Web services orchestration and choreography”. In: *Computer* 36.10 (2003), pp. 46–52. DOI: 10.1109/MC.2003.1236471.

[39] Maja Pesic, Helen Schonenberg, and Wil M.P. van der Aalst. “DECLARE: Full Support for Loosely-Structured Processes”. In: *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*. 2007, pp. 287–287. DOI: 10.1109/EDOC.2007.14.

[40] Micah D. J. Peters et al. “Guidance for conducting systematic scoping reviews”. In: *JBI Evidence Implementation* 13.3 (Sept. 2015), p. 141. ISSN: 2691-3321. DOI: 10.1097/XEB.0000000000000050. URL: https://journals.lww.com/ijebh/fulltext/2015/09000/guidance_for_conducting_systematic_scoping_reviews.5.aspx.

[41] R. Rew and G. Davis. “NetCDF: an interface for scientific data access”. In: *IEEE Computer Graphics and Applications* 10.4 (July 1990), pp. 76–82. ISSN: 1558-1756. DOI: 10.1109/38.56302. URL: https://ieeexplore.ieee.org/document/56302 (visited on 04/09/2025).

[42] N. Ritter and M. Ruth. “The GeoTiff data interchange standard for raster geographic images”. In: *International Journal of Remote Sensing* 18.7 (May 1997). Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/014311697218340, pp. 1637–1647. ISSN: 0143-1161. DOI: 10.1080/014311697218340. URL: https://doi.org/10.1080/014311697218340 (visited on 04/09/2025).

[43] Alberto Rodrigues da Silva. "Model-driven engineering: A survey supported by the unified conceptual model". In: *Computer Languages, Systems & Structures* 43 (Oct. 2015), pp. 139–155. ISSN: 1477-8424. DOI: 10.1016/j.cl.2015.06.001. URL: https://www.sciencedirect.com/science/article/pii/S1477842415000408 (visited on 06/16/2025).

[44] Richard M. Rosenfeld, Richard N. Shiffman, and Peter Robertson. "Clinical Practice Guideline Development Manual, Third Edition". In: *Otolaryngology–Head and Neck Surgery* 148.S1 (2013), S1–S55. DOI: https://doi.org/10.1177/0194599812467004.

[45] Simen Ellingsen Rustad et al. "Exposing Military Sensor Data using SpatioTemporal Asset Catalog (STAC)". In: *2023 International Conference on Military Communications and Information Systems (ICMCIS)*. May 2023, pp. 1–7. DOI: 10.1109/ICMCIS59922.2023.10253573. URL: https://ieeexplore.ieee.org/document/10253573 (visited on 04/09/2025).

[46] Rami Al-Ruzouq et al. "Sensors, Features, and Machine Learning for Oil Spill Detection and Monitoring: A Review". In: *Remote Sensing* 12.20 (2020). ISSN: 2072-4292. DOI: 10.3390/rs12203338. URL: https://www.mdpi.com/2072-4292/12/20/3338.

[47] Emanuel Silva, Nuno Silva, and Leonel Morgado. "Model-Driven Generation of Multi-user and Multi-domain Choreographies for Staging in Multiple Virtual World Platforms". In: *Model and Data Engineering*. Ed. by Yamine Ait Ameur, Ladjel Bellatreche, and George A. Papadopoulos. Cham: Springer International Publishing, 2014, pp. 77–91. ISBN: 978-3-319-11587-0. DOI: 10.1007/978-3-319-11587-0_9.

[48] Ashley Sommer and Nicholas Car. *pySHACL*. Version v0.26.0. Apr. 2024. DOI: 10.5281/zenodo.10958008. URL: https://doi.org/10.5281/zenodo.10958008.

[49] Malcolm L. Spaulding. "State of the art review and future directions in oil spill modeling". In: *Marine Pollution Bulletin* 115.1 (Feb. 2017), pp. 7–19. ISSN: 0025-326X. DOI: 10.1016/j.marpolbul.2017.01.001. URL: https://www.sciencedirect.com/science/article/pii/S0025326X17300012 (visited on 02/20/2025).

[50] M. M. Mahbubul Syeed et al. "Pluggable Systems as Architectural Pattern: An Ecosystemability Perspective". In: *Proceedings of the 2015 European Conference on Software Architecture Workshops*. ECSAW '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 1–6. ISBN: 978-1-4503-3393-1. DOI: 10.1145/2797433.2797477. URL: https://dl.acm.org/doi/10.1145/2797433.2797477 (visited on 06/16/2025).

[51] Eugene Syriani, Lechanceux Luhunu, and Houari Sahraoui. "Systematic mapping study of template-based code generation". In: *Computer Languages, Systems & Structures* 52 (June 2018), pp. 43–62. ISSN: 1477-8424. DOI: 10.1016/j.cl.2017.11.003. URL: https://www.sciencedirect.com/science/article/pii/S1477842417301239 (visited on 06/16/2025).

[52] John M. Teal and Robert W. Howarth. "Oil spill studies: A review of ecological effects". In: *Environmental Management* 8.1 (1984), pp. 27–43. DOI: 10.1007/BF01867871.

[53] Mojtaba Vaismoradi et al. "Theme development in qualitative content analysis and thematic analysis". In: *Journal of Nursing Education and Practice* 6 (Jan. 2016), pp. 100–110. DOI: 10.5430/jnep.v6n5p100.

[54] Rodrigo N. Vasconcelos et al. “Deep Learning-Based Approaches for Oil Spill Detection: A Bibliometric Review of Research Trends and Challenges”. In: *Journal of Marine Science and Engineering* 11.7 (2023). ISSN: 2077-1312. DOI: 10.3390/jmse11071406. URL: https://www.mdpi.com/2077-1312/11/7/1406.

[55] Daxa Vekariya et al. “A Survey on Oil Spill Detection using SAR images in Machine Learning”. In: *2024 Parul International Conference on Engineering and Technology (PICET)*. May 2024, pp. 1–7. DOI: 10.1109/PICET60765.2024.10716148.

[56] Dongwen Wang et al. “Representation primitives, process models and patient data in computer-interpretable clinical practice guidelines:: A literature review of guideline representation models”. In: *International Journal of Medical Informatics* 68.1 (2002), pp. 59–70. ISSN: 1386-5056. DOI: https://doi.org/10.1016/S1386-5056(02)00065-5.

[57] Andreas Weiß et al. “Model-as-You-Go for Choreographies: Rewinding and Repeating Scientific Choreographies”. In: *IEEE Transactions on Services Computing* 13.5 (2020), pp. 901–914. DOI: 10.1109/TSC.2017.2732988.

[58] Jeremy Wilkinson et al. “Oil spill response capabilities and technologies for ice-covered Arctic marine waters: A review of recent developments and established practices”. en. In: *Ambio* 46.3 (Dec. 2017), pp. 423–441. ISSN: 1654-7209. DOI: 10.1007/s13280-017-0958-y. URL: https://doi.org/10.1007/s13280-017-0958-y (visited on 02/20/2025).

[59] Yi-Jie Yang, Suman Singha, and Ron Goldman. “A near real-time automated oil spill detection and early warning system using Sentinel-1 SAR imagery for the Southeastern Mediterranean Sea”. In: *International Journal of Remote Sensing* 45.6 (Mar. 2024), pp. 1997–2027. ISSN: 0143-1161. DOI: 10.1080/01431161.2024.2321468. URL: https://doi.org/10.1080/01431161.2024.2321468 (visited on 02/28/2025).

[60] Tuğrul Yazar. “Design of Dataflow”. en. In: *Nexus Network Journal* 17.1 (Apr. 2015), pp. 311–325. ISSN: 1522-4600. DOI: 10.1007/s00004-014-0222-8. URL: https://doi.org/10.1007/s00004-014-0222-8 (visited on 06/16/2025).

[61] Piotr Zaborowski et al. “The Role of Standards in The Environmental Digital Twins Architectures”. In: *IGARSS 2024 - 2024 IEEE International Geoscience and Remote Sensing Symposium*. ISSN: 2153-7003. July 2024, pp. 271–273. DOI: 10.1109/IGARSS53475.2024.10640598. URL: https://ieeexplore.ieee.org/document/10640598 (visited on 04/09/2025).

[62] Bin Zhang et al. “Construction and Application of Event Logic Graph: A Survey”. en. In: *Database Systems for Advanced Applications. DASFAA 2022 International Workshops*. Ed. by Uday Kiran Rage, Vikram Goyal, and P. Krishna Reddy. Cham: Springer International Publishing, 2022, pp. 160–174. ISBN: 978-3-031-11217-1. DOI: 10.1007/978-3-031-11217-1_12.

[63] Lin Zhao, Timothy Nedwed, and Douglas Mitchell. “Review of the Science behind Oil Spill Fate Models: Are Updates Needed?” In: *International Oil Spill Conference Proceedings* 2021.1 (Dec. 2021), p. 687874. ISSN: 2169-3366. DOI: 10.7901/2169-3358-2021.1.687874. URL: https://doi.org/10.7901/2169-3358-2021.1.687874 (visited on 02/20/2025).

Appendix A

Code analysis for virtual choreographies literature review

Article	Code
[12]	Script
[12]	Semantic Frame
[12]	Event Segment
[16]	Workflow
[36]	Process Model
[36]	Workflow
[36]	Behavior Patterns
[62]	Event Logic Graph
[10]	Coreography
[10]	Orchestration
[10]	Behavioural Model
[10]	Workflow
[56]	Guideline Representation Models(GRM)
[29]	Process Models

Table A.1: Codes for Research Question 1

Article	Code
[12]	A script is a structured representation capturing sequences of typical real-life events.
[12]	Scripts model the connections between events and their participants within a scenario.
[12]	Scripts are organized sequences of events arranged in a temporal order.
[12]	A script is a schema or frame that focuses on typical events within a scenario.
[12]	Scripts can be viewed as multiple semantic frames that represent event sequences.
[12]	A semantic frame represents an abstract description of an activity, including its roles, at the lexical level.
[12]	An event segment is a portion of a script containing multiple events within a single event chain.
[12]	A script is a type of schema that outlines activities as ordered event sequences.
[12]	A Script can capture multiple narrative sequence of events.
[16]	Workflow in electronic business processes, focusing on artifact-centric approaches.
[36]	Process models can be described as modeled behaviours.
[36]	Workflows can be described as process models.
[62]	An Event Logic Graph is a knowledge base in the form of a directed graph representing the laws and patterns of evolution between events.
[10]	Orchestration is an executable business process that interacts with services at the message level.
[10]	Choreography tracks message sequences among multiple parties collaboratively.
[10]	Workflow defines information flow and integrates atomic services into composite services.
[56]	A GRM models guidelines and it's representative primitives in a machine readable format including actions, decisions, entity and execution states.
[56]	A GRM facilitate standard aproaches by providing generic knowledge that can be locally adapted.
[56]	A GRM defines a process model by combining different represention primitives.
[29]	Process models describe the logic and sequences of activities in processes.
[29]	Process models involve a network of heterogeneous agents with adaptive capabilities, influenced by self-organized behaviors.
[29]	Process models describe complex settings and interactions.
[38]	Orchestration is a long-lived, transactional, multi-step process model, including business logic and task execution order.

Table A.2: Codes for Research Question 2

Article	Code
[16]	Declarative artifact-centric workflows lack visualization and manipulation techniques.
[16]	Artifact schemas and service attributes drive the automated generation of sequence web pages.
[16]	Dashboards deliver real-time feedback to process users.
[36]	Some Techniques improve process model visualization through graph layout algorithms.
[36]	Some Techniques support social network visualization using diagrams like Chord diagrams.
[62]	A fact graph correlates event causes and consequences, forming event chains for user presentation.
[10]	Monitoring to evaluate composite services' performance and reliability.
[10]	Simulation to evaluate composite services' performance and reliability.
[29]	Process models can be visualized in the graphical languages used to define them.
[22]	Automatic generation of user interface through model transformation based on a hidden Markov model of user interaction
[34]	KarmaSim can be used to simulate and visualize web service composition behavior.
[44]	Guidelines can be graphically displayed as decision logic or sequence of activities

Table A.3: Codes for Research Question 3

Article	Code
[12]	Event components can be extracted using NLP tools.
[16]	Procedural artifact-centric workflows are implemented efficiently.
[16]	Declarative workflows pose challenges for efficient implementation.
[16]	Declarative workflows might use interpretive methods, to identify eligible services for invocation.
[16]	Workflow schemas can be automatically synthesized from high-level goals using artifact schemas, pre/post conditions, and regressions.
[16]	Artifacts are discovered by subject matter experts.
[16]	Stakeholders define the macro-level lifecycle of artifacts based on business-relevant stages from creation to archiving.
[16]	Process mining can be expanded for artifact discovery.
[36]	Trace variant analysis analyse unique activity sequences across different process instances from event logs.
[36]	Process discovery techniques include region-based methods.
[36]	Process discovery techniques include genetic algorithm-based methods.
[36]	Process Discovery aism to extract process models from event logs

[62]	Event logic graph construction involves extracting events, identifying relationships, determining causality direction, and combining similar events to form the final graph.
[62]	Event extraction methods include pattern matching.
[62]	Event extraction methods include machine learning, and deep learning approaches.
[62]	Pattern matching-based event extraction uses predefined language patterns, automated pattern acquisition, and active learning for pattern enhancement.
[62]	Machine learning extraction based on patterns of event features.
[62]	Neural networks are applied to event extraction, using trigger word and event type identification an.
[62]	Event relationship extraction includes feature learning-based methods using traditional classifiers and deep learning-based methods like LSTM and CNN
[10]	Manual composition involves user-driven editing of workflow scripts for execution engines.
[10]	SoaML offers a UML profile for designing and specifying service structures.
[10]	REST necessitates manual inspection of service catalogs for resource discovery/composition
[10]	Service composition often demands programming to develop orchestration models.
[10]	Planning can be exploited for automatic Composition
[10]	Semiautomated composition offers semantic guidance but requires user-driven service selection and linking.
[10]	Automated composition autonomously builds service composition from semantic-enriched descriptions.
[10]	Process algebra/calculus offers a mathematical foundation for service composition systems
[10]	Linear Integer Programming can be used to create a optimal execution plan for compositive services
[56]	A GRM is a reflect of specific patient status that is recorded and constraintly modified automatically or manually.
[29]	Graphical languages can be used to create process models.
[4]	Moses can be used as an orchestration model that drives the adaptation of a compositive service to meet certain demands
[11]	Model Driven Architecture can be used to develop compositions of web services.

Table A.4: Codes for Research Question 4

Article	**Code**
[12]	Event components include the verb predicate and related noun phrases.

[12]	An event description is the lexical realization of a script event, composed of verb predicates and noun phrases.
[12]	Verb predicates define the actions in an event and noun phrases identify the entities.
[12]	Event representations can be discrete.
[12]	Event representations can be distributed.
[12]	Events can be described by the structure predicate(subject, object)
[12]	Sequence of <verb, dependency> pairs connected to a protagonist.
[12]	In Protagonist Representation, the verb is the events action and the dependency describes the relationship to the protagonist.
[12]	Events can be represented using a relational triple structure <Arg1: subject, Relation: phrase, Arg2: object>.
[12]	Events can be represented as relational atoms in the form of v(es, eo, ep), where v is the verb, and es, eo, and ep represent the subject, object, and prepositional relations.
[12]	Multi-argument representations create a unified sequence for an entire document.
[12]	Multi-argument representations can incorporate multiple entities to provide detailed event meanings.
[12]	The multi-argument representation can be extended with a preposition.
[12]	Fine-grained represention combines elements from multi-argument and protagonist representation.
[12]	Two aditional fine-grained properties capture sentence-level sentiment and animacy.
[12]	Additional elements like negation labels, particles, and clausal complements can be incorporated into predicates
[16]	Artifact -centric workflow modeling is a data-centric approach to structure workflows.
[16]	Artifact-centrict workflow modeling focuses on augmented records(business artifacts).
[16]	Business artifacts represent key objects in workflows, their lifecycles, and interactions with tasks/services.
[16]	The artifact-centric Workflow model can be structured in business artifacts, lifecycles, services and associations.
[16]	Changes on business artifacts can be constrained using flowcharts.
[16]	Changes on business artifacts can be constrained using finite state machines.
[16]	Changes on business artifacts can be constrained using petri nets.
[16]	Changes on business artifacts can be constrained using declarative rules.
[16]	Each business artifact has a unique identity and is tracked.
[16]	Business artifacts contain attributes and data that are initialized and updated.
[16]	The runtime state of a business process is defined by the snapshot of all artifacts at a given point in execution.

[16]	Artifact data models include nested name-value pairs.
[16]	Artifact data models include relational schemas.
[16]	Artifact data models include Entity-Relationship models.
[16]	Artifact data models can be based on XML.
[16]	Artifact data models can be based on RDF.
[16]	Artifact data models can be based on OWL.
[16]	The macro-level lifecycle of artifacts is modeled using finite state machines, where states represent distinct lifecycle stages.
[16]	Hierarchical state machines help represent specialized variations of generic business process schemas.
[16]	Artifact lifecycles can be represented using high-level state machines, hierarchical state machines, or state charts.
[16]	Services can be specified with semantic web service standards like OWL-S.
[16]	Services can be specified through flowcharts.
[16]	Procedural models can express the application steps of combined processes.
[16]	Declarative models express condition action rules of service invocation and state transitions.
[16]	XML framework can be used to model data combined with processes.
[16]	The Vortex framework provides a declarative framework for data-centric workflows
[36]	Process models can be formalized as BPMN.
[36]	Process models can be formalized as petri nets.
[36]	Process models can be formalized as UML Diagrams.
[62]	Event logic graph nodes represent events, with edges defining relationships such as succession, causality, condition, and hierarchy.
[62]	An Event Logic Graph is a knowledge base in the form of a directed graph.
[62]	Events as structured as triples (P: action, S: subject, O: object).
[10]	Service composition standarts include BPEL4WS and BPML.
[10]	Service composition standarts include QoS-focused models such as Meteor-S and QoSOnt
[10]	WSCI defines choreography protocols for external message exchanges in web services.
[10]	OWL-S represent the semantic basis of web services.
[10]	WSMO(web service modeling ontology) represent the semantic basis of web services.
[10]	Finite State Machines provide an abstract behavioral model.
[10]	Service composition approaches may use UML.
[10]	Service composition may use specific ad-hoc languages
[10]	Adapters transform messages through data mapping based on predefined rules.

[10]	Service business protocol defines ordering constraints for message exchange sequences.
[10]	Re-binding involves modifying composition in response to context changes, QoS issues, or component failures.
[56]	Patient States are an abstract GRM primitive that describe the clinical status of a patient.
[56]	Execution states describe the current state of an executable task in the GRM system.
[56]	Actions are a GRM primitive used to represent a task in a guideline application.
[56]	Actions can be encoded in recursive tasks.
[56]	Actions can be instantaneous or continuous, with continuous actions taking time to complete.
[56]	Interventions are actions with direct impact on patients.
[56]	Data collections are actions of indirect impact that retrieve information from patients.
[56]	Wait actions are passive actions that change patients.
[56]	Decision is a GRM primitive used to represent decision making based on predefined criteria.
[56]	Models can represent decisions based on logical criteria or decision analysis.
[56]	Models can represent automatic or manual assisted decisions.
[56]	Scheduling constraints represent the temporal order of primitive executions.
[56]	Scheduling constraints can be defined as sequential, where primitives are executed one by one.
[56]	Scheduling constraints can be defined as concurrent, where primitives are executed in parallel.
[56]	Scheduling constraints can be represented using flowcharts.
[56]	Continuation criteria in scheduling constraints can be specified with logical expressions.
[56]	Scheduling constraints can be defined as a partial order.
[56]	Scheduling constraints have two demensions, ordering constraints and continuation condition.
[56]	Scheduling constraints can be defined as alternatives, whose primitives execution depends on specific criteria.
[56]	Scheduling constraints can be defined as loops, with repetitive execution of primitives
[56]	Scheduling constraints can be modeled using Petri Nets.
[56]	Scheduling requirements can be modeled using constraint satisfaction graphs.
[56]	Scheduling constraints can be modeled using state transition diagrams
[56]	Nesting of guidelines allow multiple abstract levels and granularities of a guideline representation.
[56]	Standard patient data follow a standard encoding and a standard patient model.

[56]	Controled terminologies can be used as standard data encoding
[56]	Ontologies can be used for standard patient modeling
[56]	A mapping between the implementation system and a reference information model can be made.
[29]	Business process diagram is a flowchart based technique used to represent process models.
[29]	Business process execution languages can represent actions of process models with web services.
[29]	UML is a standard language for graphical modeling with a capacity to represent different abstract views.
[29]	BPMN is a standard language for graphical modeling with a capacity to represent different abstract views.
[29]	Activity diagram is an enhanced for of flowcharts used to represent workflows.
[29]	State machine diagrams represent the changes of states of an object in a process.
[29]	Timing diagrams represent conditions changes of lifelines
[29]	Sequence diagrams represent interactions between entities with a sequential map of messages over time
[39]	DECLARE provides a declarative language based on temporal logic for process model development

Table A.5: Codes for Research Question 5

Appendix B

Theme analysis for virtual choreographies literature review

Article	Code
[12]	Script
[12]	Semantic Frame
[12]	Event Segment
[16]	Workflow
[36]	Process Model
[36]	Workflow
[36]	Behavior Patterns
[62]	Event Logic Graph
[10]	Coreography
[10]	Orchestration
[10]	Behavioural Model
[10]	Workflow
[56]	Guideline Representation Models(GRM)
[29]	Process Models

Table B.1: Themes for Research Question 1

Theme	**Definition**
Script	This concept of script is similar to our concept of virtual choreography, in that it captures and defines behavioral patterns by modeling the laws and patterns of event evolution, participants, actions and multiple narrative sequences of events within a scenario, while providing semantically enriched representations of activities.
Choreography	This concept of Choreography is similar to our concept of virtual choreography, in that it captures a behavioral pattern by capturing message sequences in a multi-service scenario.
Process Model	This concept of Process Model is similar to our concept of virtual choreography, in that it models behavioral patterns, sequences of activities, and process logic in complex and dynamic scenarios, while also being able to interact with services, progress in a flow across multiple entities, and represent actions, decisions, entities, and execution states in a machine-readable format.

Table B.2: Themes for Research Question 2

Theme	**Definition**
Graph	A Virtual Choreography can be visualized as a graph presenting a process over time.
Web Page Sequence	A Virtual Choreography can be visualized as a sequence of web pages generated from a pattern definition.
Dashboards	A Virtual Choreography can be visualized as Dashboards that deliver real-time feedback about a process.
Graphical Simulation	A Virtual Choreography can be visualized in a graphical simulation environment that depicts the behavior of a system.
Chord diagram	A Virtual Choreography can be visualized as Chord diagrams that analyze the relationships and interactions between entities.
Automatic User Interface	A Virtual Choreography can be presented through user interfaces that are automatically generated based on user interactions.
Sequence of Activites	A Virtual Choreography can be graphically displayed as a sequence of activities.
Graphical Languages	A Virtual Choreography can be presented through the graphical languages that are used to describe them.
Temporal diagrams	A Virtual Choreography can be presented using temporal diagrams as a solution to show condition changes or represent actions between entities over time.

Table B.3: Themes for Research Question 3

Theme	**Definition**
Service Adaptation	A Choreography can be updated by modifying composition upon context change or errors, or to adjust to certain needs.
Automatic Extraction/Definition	A Choreography can be Automatically extracted or defined from semantic-enriched descriptions, regressions, high-level abstract goals/conditions, and Planning.
Semi-Automated Extraction/Definition	A Choreography can be extracted or defined by the user, with the aid of semantic guidance.
Manual Extraction/Definition	Choreography extraction or definition is Human based(visual or textual).
Pattern Matching	Choreographies can be extracted by identifying patterns that follow strict rules.
NLP	Choreographies can be extracted from text by identifying patterns using NLP tools.
Machine Learning Extraction	Choreographies are extracted by identifying Patterns using Machine Learning, including specific Deep Learning techniques.
Region-Based discovery	Choreographies can be discovered using region-based methods
Genetic algorithm discovery	Choreographies can be discovery using Genetic algorithm-based methods
Trace Variant Analysis	Choreographies can extracted using trace variant analysis, a method used for activity sequence extraction.
Process Algebra/Calculus	Choreographies can be defined using a mathematical foundation.
Linear Integer Programming	Choreographies can be defined with Linear Integer Programming for optimal behavior patterns.
Model Driven Architecure	Choreographies can be created using Model Driven Architecture.
Process Mining	Behaviour patterns can be identified using process mining.

Table B.4: Codes for Research Question 4

Theme	**Definition**
Graphical Modeling Languages	In this theme, standard high-level graphical languages can be used to express process models at different abstraction views.

Artifact-Centric Worflow Modeling	This representation provides a Data-Centric approach to structure workflows, based on lifecycles, service, associations and business artifacts, with the last element being an identifiable, trackable, and dynamic model of states and interactions.
Event representations	This general theme includes representation techniques used in the protagonist, multi-argument or fine-grained representations, and the triple structure representation.
Rule-Based Data Mapping	The technology is used to transform message representation using a Rule-Based format converter.
Graph	In this theme, Graphs can model behavioral constraints, Knowledge bases and a network of events and their relantioships.
Nesting	This representation is a Data Structure that allows direct nesting, multiple levels of abstraction and granularities.
Relational model	This representation follows relational Database logic.
XML	In this theme, XML can be used a file format define artifacts
RDF	In this theme, RDF can be used as a data model to define artifacts.
Ontologies	This theme describes ontologies as a tool to model entities with strict vocabulary and formal semantic definitions.
State Machines	This theme provides a representation for abstract behavioral models and constraints.
Flowcharts	This theme offers a procedural representation of workflows.
Petri Nets	In this theme, Petri nets are used to represent behavioral constraints and formalize process models.
Declarative specification	Technological specification based on high-level condition-action rules for service invocation and state transitions.
Discrete Representations	In this theme, events are represented as an unbreakable atomic unit
Distributed Representations	In this theme, events are represented as dense vector of real values.
Constraint Representation	This definition captures the essence of defining and managing interaction constraints, focusing on the ordering of actions/decisions, both sequential and parallel, execution sequence management and constraint-driven interaction logic
ad-hoc languages	This theme involves the use of specific languages for composition specification.
Procedural specification	Procedural specification establish a series of steps for combined processes.

Runtime State	This theme defines the runtime state as a Snapshot of a process at a given instant.
WSCI	This tool is used to define choreography protocols for web services.
Action representation	This definition encapsulates multiple abstraction levels of actions, state modification capabilities and diverse interaction mechanisms, including passive actions, data collections and direct actions.
Decision representation	This theme describes decision representation to model decision-making based on predefined criteria, using logical or analytical methods for automatic or assisted choices.

Table B.5: Themes for Research Question 5

Appendix C

Canvas plugin-nodes table

Name	Type	Description
Split Json	Generator	Generates individual dictionary outputs (one by one) from an input JSON list.
Fetch Actions	Function	Extracts “Action” instances from a Virtual choreography (RDF) graph, matching given SHACL-like criteria.
Read Json	Provider	Reads and parses a JSON file from execution context.
Polygon Groups	Function	Groups polygons based on their spatial intersection, assigning a unique group ID to each connected set.
Tag Rings	Function	Identifies each polygon as an outer polygon or hole, depending on containment.
Save VC	Consumer	Serializes and saves an RDF graph (Virtual Choreography dataset) to an output file in Turtle or JSON-LD format in the execution context.
Norm Value	Function	Normalizes a numeric value (e.g., using logarithmic scaling) within a specified range based on a reference list.
GroupBy	Function	Groups a list of dictionaries by specified properties, returning an array of groups.
Collection Mapper	Function	Extends each item in a list with additional properties (an “extensions” dictionary).
TemporalMetrics	Function	Calculates duration and temporal frequency from a list of timestamps.
OrderBy	Function	Sorts an array of dictionaries by one or more specified fields.
Flag	Function	Iterates over a list of dictionaries, setting a boolean “flag” whenever certain fields change.

Get	Function	Extracts the value of a specified property from an input dictionary.
Get Centroid	Function	Takes one or more polygons (GeoJSON) and returns the centroid as a GeoJSON point.
Save Json	Consumer	Saves a received dictionary or list to a JSON file in the output folder of the execution context.
Fetch Entity	Function	Retrieves all predicate–object pairs (triples) for a specified entity from an RDF graph.
Process Action	Function	Creates an Action node in an RDF (Virtual Choreography) graph, combining "actor" and "object" references with additional data.
Template	Function	Applies Jinja2 template logic to generate a JSON-like output from input data.
Catch	Coroutine	Accumulates either a list of items or appended values, depending on configuration.
VC Buffer	Coroutine	Merges Virtual Choreography (RDF) graph inputs while optionally removing provenance or structural information.
Polygon Stats	Function	Computes statistics for merged polygons, such as area, perimeter, and dispersion.
Process Entity	Function	Creates or extends an Entity in a Virtual Choreography (RDF) graph, applying shape validation and mapping input fields to RDF properties.
Convert Plane	Function	Validates a polygon within a bounding box, then transforms and scales coordinates based on metadata.
Read VC	Provider	Loads a Turtle-based Virtual Choreography (RDF) file from execution context into an internal RDF graph.

Table C.1: Canvas plug-ins