VNiVERSiDAD D SALAMANCA

DOCTORAL THESIS

Time-Constrained Ontology Evolution for Predictive Maintenance

Author:

Alda Canito

Supervisors:

Prof. Dr. Goreti Marreiros

Prof. Dr. Juan M. Corchado

Doctor Degree in Informatics Engineering

DEPARTAMENTO DE INFORMÁTICA Y AUTOMÁTICA

LÍNEA DE INVESTIGACIÓN: SISTEMAS INTELIGENTES

FACULTAD DE CIENCIAS

2025

Assinado por: **Maria Goreti Carvalho Marreiros**
Data: 2025.07.16 10:00:08 +0100

Firmado por CORCHADO RODRIGUEZ JUAN MANUEL - ***7831** el día 16/07/2025 con un certificado emitido por AC FNMT Usuarios

Assinado por: **Alda Renata Fangueiro Canito**
Data: 2025.07.16 11:13:59+01'00'

This work has received Portuguese National Funds through Portuguese Foundation for Science and Technology under the project UIDP/00760/2020 and the Ph.D. scholarship with reference SFRH/BD/147386/2019.

This document is the result of my very long and very draining academic pursuits and if I were to make an exhaustive list of people to thank for pushing me in the right direction, I would probably need to write a couple more hundred pages. I would probably have to go all the way to my high-school philosophy teacher in high-school, for telling me what a premise is.

In the spirit of keeping this section short, I want to begin by thanking and blaming Professor Nuno Silva for telling me what an ontology is. This is all your fault. Our infrequent chats are behind some of the main topics explored in this thesis, illuminating potential ideas worth exploring.

I also want to thank my colleague Gabriel, who was my partner in crime during Professor Nuno's classes and with whom I crafted my first ontologies. The fact that we, somehow, still get along 10-ish years later never ceases to amaze me. That we are still talking about ontologies is even more amazing.

I want to thank my colleague Marta, whom in her insistent pestering made me a much better researcher (and person) and without whose confident support this thesis certainly would not even exist. She is, perhaps unwillingly, an infinite source of inspiration. She dragged my rotting corpse back into academia and systematically and consistently attacks all my ideas and convictions, and I cannot thank her enough for it.

I also want to thank the mOeX team at INRIA for welcoming me into their group to the point of making me believe in academia again. Their warm support made my experience in Grenoble unforgettable and fruitful. I especially want to thank professor Jérôme David, whose insights made me realise the potential of my own work and who guided my focus long after I had left mOeX. Without him, I am sure, this thesis would not have come to fruition the way it has and would certainly be a lot less interesting. My words cannot express how profoundly indebted I am to his continuous support.

I want to thank Professor Goreti for the opportunity that even allowed me to pursue this PhD, and for always holding the door open for me.

Finally, I must thank my family for their warm incentives and for indulging me in my many, many silly interests (the PhD included). I will especially remember my two grampas, who had the audacity to pass away before I could finish writing this. I wish we could have had a couple more glasses of wine together.

Acknowledgments

This work has been supported by different projects, namely:

- By the European Union under the Next Generation EU, through a grant of the Portuguese Republic's Recovery and Resilience Plan Partnership Agreement under the projects PRODUTECH R3 and DigiMaTRIA
- The EUREKA – ITEA3 Project PIANISM (Itea-17008) and its Portuguese counterpart, PIANISM (ANI|P2020 40125)
- The European Union's Horizon 2020 research and innovation program, under grant agreement No 832969 – SATIE.

Furthermore, this work has also received Portuguese National Funds through Portuguese Foundation for Science and Technology in the Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development (project UIDP/00760/2020) and the Ph.D. scholarship with reference SFRH/BD/147386/2019.

Finally, this work benefited from the informed and pertinent insights by the mOeX team during my research stay in INRIA Montbonnot, Grenoble, and the efforts made by Wikidata's Project Pokémon[1].

[1] https://www.wikidata.org/wiki/Wikidata:WikiProject_Pokémon

Abstract

With the introduction of the Internet of Things, maintenance practices have been moving from reactive to proactive and predictive approaches. The identification of faults often relies on the analysis of real-time data provided by streams and unstructured sources. Ontologies have been applied to the maintenance field, adding a semantic layer to the data that facilitates interoperability and semantic data mining processes. In such a time-sensitive domain, it is important that ontologies go beyond static representations of the domain and allow not only for the incorporation of time related knowledge, but must also be able to adapt to new knowledge and evolve. Evolving an ontology involves re-learning, re-enriching and re-validating knowledge in the face of changes to the domain, and techniques applied for them can be adapted to ontology evolution. This thesis aims to contribute to these fields by using streams of ontology individuals as the trigger for ontology evolution processes – facing challenges tied to the incomplete and transient nature of these data. As such, this thesis introduces an architecture for time-constrained ontology evolution called *TICO*, or Time Constrained instance-guided Ontology evolution. New versions of ontology classes and properties are reified through a 4D-Fluents approach, thus allowing reasoning over old data and accessing older conceptualizations of the domain. For the identification of property axioms, the possibilistic approach to axiom scoring was adapted to a scenario in which it is not always possible to query all individuals at once. Results show the effectiveness of the approach in accepting/rejecting axioms for the ontology's properties. To identify patterns in data that could trigger the creation of new classes and enrich existing ones, a Formal Concept Analysis-based approach is employed. Using two different concept lattices that are updated with each individual, it is possible to identify a set of axioms to add to the ontology and uncover implicit relationships between old and new classes.

Keywords: Axiom Scoring, Class Learning Problem, Ontology Evolution, Ontology Learning, Predictive Maintenance, Time-Sensitive Data.

Resumen

Con la introducción del IoT, las prácticas de mantenimiento han ido pasando de orientaciones reactivas a proactivas y predictivas. La identificación de fallas a menudo se basa en el análisis de datos en tiempo real proporcionados por flujos y fuentes no estructuradas. Las ontologías se han aplicado al campo del mantenimiento, añadiendo una capa semántica a los datos que facilita la interoperabilidad y los procesos de minería semántica de datos. En un ámbito tan sensible al tiempo, es importante que las ontologías ultrapasen las representaciones estáticas del dominio y permitan no sólo incorporar conocimientos relacionados con el tiempo, sino que también deben ser capaces de adaptarse y evolucionar. Evolucionar una ontología implica reaprender, re-enriquecer y re-validar el conocimiento y las técnicas aplicadas para ellas pueden adaptarse a la evolución de ontologías. Esta tesis pretende contribuir a estos campos utilizando flujos de individuos RDF como desencadenante de procesos de evolución de ontologías, enfrentándose a retos ligados a la naturaleza incompleta y transitoria de estos datos. Como tal, esta tesis introduce una arquitectura para la evolución de ontologías limitada en el tiempo llamada *TICO* (Time Constrained instance-guided Ontology evolution). Las nuevas versiones de las clases y propiedades de la ontología se reifican mediante 4D-Fluents, lo que permite razonar sobre datos antiguos y acceder a conceptualizaciones anteriores del dominio. Para la identificación de axiomas de propiedades, se adaptó el enfoque posibilista de cualificación de axiomas a un escenario en el que no siempre es posible obtener la descripción completa del conjunto de datos. Los resultados muestran la eficacia de la solución en aceptar/rechazar axiomas para las propiedades de la ontología. Para identificar patrones en los datos, crear nuevas clases y enriquecer las existentes, se emplea un enfoque basado en el Análisis Conceptual Formal. Utilizando dos redes de conceptos diferentes, es posible identificar un conjunto de axiomas para añadir a la ontología y descubrir relaciones implícitas entre clases antiguas y nuevas.

Palabras Clave: Aprendizaje de Clases, Aprendizaje de Ontologías, Cualificación de Axiomas, Datos sensibles al tiempo, Evolución de Ontologías, Mantenimiento Predictivo.

Contents

List of Figures

List of Tables

Acronyms

(n)UNA	(not) Unique Name Assumption
ARI	Acceptance/Rejection Index
AS	Asymmetry
AVF	AllValuesFrom
CbM	Condition-based Maintenance
CDM-core	CREMA Data Model Ontology
Ce	Class Expression
CeH	Class Expression Hypothesis
CeL	Class Expression Learning
CM	Corrective Maintenance
CMT	Microsoft Conference Ontology
CQ	Confirmation Query
CSV	Comma-Separated Values
DL	Description Logics
ERP	Enterprise Resource Planning
exaC	Exact Cardinality
F	Functionality
FCA	Formal Concept Analysis
FN	False Negative
fod	Full Object Description
FP	False Positive
Gen	Generation
HS	HasSelf
HV	HasValue
IF	Inverse Functionality
IoT	Internet of Things
IR	Irreflexiveness
IRI	Internationalized Resource Identifier
J2OIM	JSON to Ontology Instance Mapper
JSON	JavaScript Object Notation
maxC	Maximum Cardinality

minC	Minimum Cardinality
NQ	Negation Query
OntoDM	Ontology for Data Mining
OWA	Open World Assumption
OWL	Web Ontology Language
PdM	Predictive Maintenance
PIANISM	Predictive and Prescriptive Automation in Smart Manufacturing
PM	Preventive Maintenance
pod	Partial Object Description
R	Reflexiveness
RDF	Resource Description Framework
S	Symmetry
SPARQL	SPARQL Query Language, initially Simple Protocol And RDF Query Language
SSN	Semantic Sensor Network Ontology
SVF	SomeValuesFrom
T	Transitivity
TICO	Time-Constrained Ontology Evolution
TN	True Negative
TP	True Positive
URI	Uniform Resource Identifier
UUID	Universally Unique Identifier
XML	Extensible Markup Language
XSD	XML Schema Definition
$\boldsymbol{\Pi}$	Possibility
$\boldsymbol{N}$	Necessity
$\boldsymbol{i}$	Ontology individual
$\boldsymbol{P}$	Ontology property
$\boldsymbol{Ax}$	Ontology axiom
$\boldsymbol{C}$	Ontology Class
$\boldsymbol{t}$	timeframe
$\boldsymbol{S}$	RDF stream

CHAPTER 1

Introduction

1. Introduction

The application of Internet of Things (IoT) technologies in industry facilitated the adoption of predictive and proactive approaches to equipment maintenance in place of mainly reactive or scheduled interventions [1,2]. The goal of predictive maintenance (PdM) is to intervene in equipment before faults effectively take place, which often requires the continuous monitorization of equipment condition and identification of anomaly and/or future faulty states [1]. For this to happen, it is important that data regarding equipment condition is obtained and processed in useful time, as well as be properly understood by all intervening parties in all parts of the process – from its acquisition to its processing, analysis and delivery to the end users. Here, ontologies have found applications in facilitating the interoperability between systems and information integration by adding a semantic layer to the data that can be used to trigger reasoning and inference processes. Some architectures propose using ontologies to describe data captured through sensors [3,4], processes [2,5] or for the delivery of results [6]. By providing a formal description of the application domains, reasoning with ontologies has found applications in providing diagnostics, recognition of qualitative fault states and distinguishing between different types of failure [3,4,7,8]. Furthermore, by specifying the set of constraints the data must adhere to, they facilitate the discovery of inconsistencies both in data and on the insights obtained by other processes, allowing for the materialization of implicit knowledge that can be combined with other mechanisms for predictive maintenance purposes and to enable semantic data mining processes.

1.1.1 The Time-dimension in Predictive Maintenance

Predictive maintenance activities require time-sensitive and time-aware processes. A lot of information arrives in real-time and through streams, which may then be analysed through machine learning and data mining algorithms and bound to change its meaning over time – in a form of concept drift –, as the characteristics that define normal or abnormal behaviour slowly transform. Information provided in real time through streams – which is often the case with real-time sensor readings, but also, for example, with financial data – whenever constituting a timeseries, carry an implicit or explicit time dimension [9,10] and is often used by machine learning and data mining algorithms for predictive purposes [11]. The time dimension of this data needs to be addressed for its

semantization to be used to its full potential. Similarly, any outcomes of the algorithms and models applied will also have a temporal validity associated, and the insights they provide may vary over time – meaning that any newly acquired knowledge that could be incorporated in the ontology should have a temporal dimension as well. While there is a considerable amount of work regarding the applications of ontology to the predictive maintenance field, they often fail to understand and model these requirements. Most of the available ontologies focus on the description of a static domain and do not make much use of temporal representation or temporal constraints. For ontologies to change over time and adapt to the dynamic domains they aim to describe, they must be able to evolve in some way.

1.1.2 Ontology Evolution and Ontology Learning

Ontology evolution is described in [12] as the process through which an ontology adapts to the changes in a domain in a timely fashion, while consistently propagating the changes to other artifacts that depend on it. This process often relies on the identification of which changes occurred in the domain that made the ontology outdated, the materialization of those changes into specific evolutionary actions and the execution of those, followed by consistency checks to ensure that the new ontology version is logically sound. Predictive maintenance solutions often must model the gradual change in equipment behaviour as its components wear down with use. As such, there is a variation in what are the characteristics of normal/abnormal behaviour over time, and, similarly, of what constitutes a failure. New sensors can be introduced – and new features may result of analytical processes made by data experts. Furthermore, existing classes and properties may be used differently over time, according to the necessities of the users managing the data. Ontology evolution processes, therefore, can be employed to identify and model these changes, ensuring that the domain description is up-to-date and remains accurate. Many existing approaches to ontology evolution, however, fail to provide formalisms for temporal abstract notions [13].

Ontology evolution – especially when performed automatically or semi-automatically – cannot be detached from other subfields of ontology studies in computer science, such as ontology learning, enrichment, and validation [14,15]. Many steps and techniques are transversal to these fields; consider, for example, how evolutionary processes occur: new knowledge needs to be learned so it can

be formalized into the ontology, and data can be used to further enrich the evolving schema into a more expressive and precise result. Ontology evolution is, in a way, the process of re-learning, re-enriching and re-validating an ontology in the face of changes to the domain, particularly when these are triggered by the data itself. This means the ontology is not learned once, but that learning and evolving the ontology are inextricably linked, and the data used to trigger those processes is both limited and transient. However, ontology learning and evolution solutions tend to focus on the identification and materialization of changes in concepts and roles and the hierarchies between them, and the same attention has not been given to the axiomization of said structures [16], particularly when dealing with the inherent incompleteness that comes with data obtained via streams.

1.1.3 Research Questions

Considering that ontology evolution is a fundamental but understudied part of predictive maintenance solutions whenever they make use of semantic technologies, and that ontology evolution is intrinsically connected to ontology evolution, the working hypothesis being considered in this work is the following:

> *It is possible to establish an Ontology Evolution/Learning methodology to generate expressive, formal ontologies from semi-structured sources, encompassing both user and automatic methods of classification and assessment of possible evolution points.*

Considering this hypothesis, the following research questions will be addressed during the remainder of this document:

- **RQ1:** To which extent is it possible to identify changes in classes and properties used by individuals provided via stream and reflect them in an ontology that represents those individuals during a specific period?
- **RQ2:** What degree of expressivity can be achieved in this fashion, considering that it is not possible to analyse all individuals from a stream at the same time? More specifically:
  - **RQ2.1:** How can we identify changes in property axioms?
  - **RQ2.2:** How can we identify new class expressions?
  - **RQ2.3:** Is it possible to identify the need to have super/subclass relationships that were not in the ontology?

1.2 Main Contributions and Thesis Outline

The doctoral work has resulted in the development of a time-sensitive tool for ontology evolution from streams of individuals named *TICO* (TIme Constrained instance-guided Ontology evolution). This tool receives as input an ontology and a stream of individuals (described by the supplied ontology) and analyses periods of time to derive evolutionary actions that will extend the original ontology to more closely describing the data from that period. This includes adding new classes that describe groups of similar individuals, enriching existing classes, modifying property axioms, and identifying the need for subsumption relationships between classes.

This document is comprised of ***six*** main chapters. This first one, ***Introduction***, contextualizes the problems addressed throughout my work and provides the motivation for the following research.

Chapter 2, ***Fundamentals***, provides an overview of the basic concepts upon which the remainder of this thesis relies. It reviews the notions of ontology and ontology evolution in computer science and the use of Semantic Web Technologies for the description of knowledge and knowledge domains.

Chapter 3, ***Time-Constrained Ontology Evolution***, introduces the similarly-named *TICO*, a tool that allows for real-time acquisition of evolutionary changes that can be applied to produce new versions of an ontology to describe changes in data from streams. It explores its main functionalities and internal architecture. *TICO* is expanded and improved in the subsequent chapters, which focus on the solutions to specific problems related to ontology learning/evolution.

Chapter 4, ***Property Axiom Learning***, presents a set of SPARQL queries that can be used over a sliding window of individuals obtained via streams in order to identify patterns in data that correspond to known property axioms. Furthermore, it discusses the applicability of the possibilistic approach in the streaming scenario, and employs the Acceptance/Rejection Index to assess which of those property axioms are sufficiently supported by the data to be included in the ontology.

Chapter 5, ***Identifying New Classes and Enriching Class Definitions*** establishes mechanisms that allow for the elicitation of hypotheses about which class restrictions could be added to the definition of existing classes. These hypotheses, should they be confirmed by the individuals from the stream, are used to feed two Formal Concept Lattices. These are then used to derive

implications from which new axioms can be extracted – including subsumption relationships, domains and ranges of properties and expanding existing classes by adding new restrictions to their definition.

Finally, Chapter 6, ***Conclusions***, showcases the main outcomes and findings of the doctoral work, along with a discussion of the solution's flaws, potential improvements and future work.

1.3 Origin of Materials

The results of this work have been published in 5 scientific papers, either in conference proceedings or in international journals. Parts of these papers have been adapted into this thesis and are the primary source for some chapters, as explained next.

Parts of the Introduction are based on a systematic review that served as motivation for much of the work:

- Alda Canito, Juan Corchado, and Goreti Marreiros. "A systematic review on time-constrained ontology evolution in predictive maintenance." *Artificial Intelligence Review* 55.4 (2022): 3183-3211.

Chapter 3 is based on two papers, the second of which is an extension of the first:

- Alda Canito, Armando Nobre, José Neves, Juan Corchado and Goreti Marreiros. "Applying Time-Constraints Using Ontologies to Sensor Data for Predictive Maintenance." *World Conference on Information Systems and Technologies*. Cham: Springer International Publishing, 2022.
- Alda Canito, Armando Nobre, José Neves, Juan Corchado and Goreti Marreiros. "Using sensor data to detect time-constraints in ontology evolution." *Integrated Computer-Aided Engineering* 30.2 (2023): 169-184.

Chapter 4 is based on:

- Alda Canito, Jérôme David, Juan Corchado and Goreti Marreiros. "Applying possibilistic axiom scoring to instance-guided ontology evolution in RDF streams." (submitted).

Annex 1 includes a portion of the following paper, which describes the reasoning behind the ontology used in Chapter 2:

- Alda Canito, Juan Corchado, and Goreti Marreiros. "Bridging the gap between domain ontologies for predictive maintenance with machine learning." *World Conference on Information Systems and Technologies*. Cham: Springer International Publishing, 2021.

CHAPTER 2

Fundamentals

2. Fundamentals

This chapter contextualizes the thesis by reviewing the main definitions necessary to understand the remainder of the work. It includes a brief description of Predictive Maintenance and where it stands against other maintenance approaches; a quick overview of ontologies as artifacts to describe domain knowledge and some Semantic Web technologies associated with them.

Predictive Maintenance

Maintenance activities are an important and fundamental task in any manufacturing process. Different maintenance strategies can be applied, depending on the type of equipment and the industry considered, but they are mainly aggregated under the following categories [17]:

- ***Corrective Maintenance*** (CM): the more traditionally applied strategy, in which equipment runs until it breaks; maintenance is executed in response to equipment failure.
- ***Preventive Maintenance*** (PM): the equipment is inspected for signs of failure and wear. The inspection often occurs periodically, meaning replacements may be made regardless of the actual remaining useful life (RUL) of the equipment.
- ***Condition-based Maintenance*** (CbM): with access to sensors, which are often provided by the machines themselves, it is possible to monitor for physical signs of wear and apply maintenance to the equipment as they suggest.
- ***Predictive Maintenance*** (PdM): expands on CbM by using the data from sensors and other sources – such as Enterprise Resource Planning (ERP) software, which can provide context regarding which tasks are being performed by an equipment at a given time – with data mining and machine learning algorithms to effectively predict when failures will occur, making it possible to pre-emptively schedule the replacement of parts and avoid failure altogether.

- ***Prescriptive Maintenance***: builds upon PdM by not only predicting failures but also providing means to estimate and increase RUL and make suggestions on how to use the equipment more efficiently.

Ontology

In computer science, an ontology is an artifact that semantically describes a specific domain. A common definition in the literature is provided in [18]:

> *"An ontology is an explicit specification of a conceptualization."*

The terms involved in the definition are further explained in [19] as:

- ***Conceptualization*** is a rational and abstract model of a certain domain. It includes the definition of concepts, properties and relationships between them.
- ***Specification*** is the meaningful, detailed, accurate, solid and consistent description of the domain.
- ***Explicit*** regards the representation of this knowledge in a way that intelligent agents may understand and reason upon.

Two more terms are introduced in [20], namely "formal" and "shared". The definition is thus improved with:

- ***Formal***, meaning that both humans and machines must be able to read, understand and process the ontology in a similar manner, i.e., should be able to reach the same conclusions from the data.
- ***Shared*** refers to the fact that the ontology, as the description of a domain, should be accepted in consensus by a group and not just by one individual.

Regardless of the formalism used to describe an ontology, it will feature the following entities:

- ***Classes***, also known as Concepts or Types, represent categories or groups of objects in the domain. Classes can be defined *extensionally* (by specifying which elements belong to it) or *intensionally* (by specifying the attributes an object must have to belong to the class).
- ***Properties***, also known as Roles, represent relationships between the objects. Together with classes, they are the vocabulary used to describe a domain. They may specify a relationship between two objects – an `ObjectProperty` – or between an object and a particular literal attribute

(e.g. a number) – i.e., a *DatatypeProperty* (often simplified as *DataProperty*).

- ***Axioms***, rules which impose restrictions on how the vocabulary can be used. By providing rich semantics, they also allow for reasoning mechanisms to infer implicit knowledge from the descriptions. The number and types of axioms allowed and how they are explored directly correlates to the expressivity of the ontology.
- ***Individuals***, or Instances, are specific instantiations of a class. Individuals may not necessarily be part of an ontology, but the ontology is a means through which the classification of individuals can be made.

The set of all entities that comprise an ontology constitutes the ontology's *signature*.

Ontology Evolution

Ontology evolution is described in [12] as the process through which an ontology adapts to the changes in a domain in a timely fashion, while consistently propagating the changes to other artifacts that depend on it. This process often relies on the identification of which changes occurred in the domain that made the ontology outdated, the materialization of those changes into specific evolutionary actions and the execution of those – followed by consistency checks to ensure that the new ontology version is logically sound. Two main approaches to the identification of change can be found in literature, namely:

- Comparing the concepts and roles in the ontology with those in external corpora – establishing if new ones are necessary, if existing ones need change, and which ones have become obsolete (as is the case in [21]).
- Comparing different versions of the same ontology and identifying existing modifications in its structure (e.g. [22–24]).

In [23], the authors describe two main ways changes can be applied to an ontology:

- Naïve approaches, in which changes are materialized into a new ontology version and all copies are fully stored (e.g. [25]).
- Change-based approaches, in which a reference version is maintained, and individual changes are formalized into computable actions – i.e. ***Evolutionary Actions*** – that can be applied to the reference version in order to produce the modified ontology version. For this purpose, it is

necessary to properly describe each evolutionary action in some formal fashion, and ontologies may be used for this end (e.g. [24,25]).

Semantic Web Technologies: RDF, RDFs, OWL and SPARQL

There are different languages through which an ontology may be expressed, each with their specific limitations and expressivity. The Semantic Web [26] is an extension to the World Wide Web that aims to make the internet machine-readable by enabling the addition of semantics to the data. To achieve this end, the World Wide Web Consortium has forwarded a set of standards for metadata representation[2], including the Resource Description Framework (RDF), RDF Schema (RDFs), the Web Ontology Language (OWL) and the SPARQL Query Language (SPARQL).

RDF and ***RDFs*** are the triple data model <subject, predicate, object>[3] to represent the facts – or resources – of a domain. RDFs extends RDF by adding some basic constructors, including defining classes of objects, subsumption relationships and the Ranges and Domains of properties. Entities are described by an ***Internationalized Resource Identifier*** (IRI), such as https://www.wikidata.org/wiki/Q130859144. For ease of reading, IRIs can be represented in the form of *namespace:Entity*. Should the namespace *wd* refer to the prefix https://www.wikidata.org/wiki/, the IRI above could be abbreviated into *wd:Q130859144*. When it comes to RDF triples, the following are possible:

- Subjects: can be either an IRI, representing a domain entity, or a blank node – i.e., an anonymous resource that is used, among other things, for reification purposes and to describe multi-component structures.
- Predicates: the IRI of a property denoting the relationship.
- Objects: may be another IRI (i.e. the Subject of another triple), or a Literal, which represents a primitive type constant (e.g. a specific number). Literals are always used as objects.

An Individual is the group of RDF facts with the same subject.

SPARQL [27] is a declarative query language designed to query data following the RDF standard. SPARQL queries search for triple patterns in a triple store; to do so, the queries can combine conjunctions, disjunctions, optional

[2] Web Standards | W3C - https://www.w3.org/standards/
[3] Often abbreviated as <s,p,o>.

patterns and variable binding. For example, consider the SPARQL query shown in Code Snippet 1, which is designed to obtain four variables (but only distinct combinations of them, avoiding repetition), one of which is part of an optional triple pattern. The query declares the triple patterns that must be matched for each variable, and returns only the results that match the filters.

Code Snippet 1 – SPARQL Query to obtain all Pokémon from Wikidata (and their Johto Pokédex numbers, if they have one)

```
SELECT DISTINCT ?pokemon ?pokemonLabel ?pokedexNumber ?johtodexNumber
WHERE
   {
      ?pokemon wdt:P31/wdt:P279* wd:Q3966183.
      ?pokemon rdfs:label ?pokemonLabel.
      ?pokemon p:P1685 ?statement.
      ?statement ps:P1685 ?pokedexNumber; pq:P972 wd:Q20005020.

      OPTIONAL
      {
          ?pokemon p:P1685 ?johtoStmt.
          ?johtoStmt ps:P1685 ?johtodexNumber .
          ?johtoStmt pq:P972 wd:Q11310550.
      }

      FILTER(LANG(?pokemonLabel) = "en").
      FILTER(STRSTARTS(?pokemonLabel, "M")).

} ORDER BY (?pokedexNumber)
```

OWL [28] builds upon RDFs by providing more expressive semantics, following Description Logics (DL) knowledge representation formalisms. It introduces:

- Property Axioms: Functionality, Inverse Functionality, Transitivity, Symmetry, Asymmetry, Reflexivity and Irreflexivity. These specify how a property can be used and what can be inferred from its use.
- Class Expressions (Ce), as the way to define a class. The signature of a Ce is the set of elements that it employs. A Ce may be as simple as the name of the class, or be composites that use operators such as Intersection, Union and Negation to establish new classes in relation to existing ones. Furthermore, it also allows to define a class by the properties its elements must have and under which circumstances, namely:
  - Universal Quantification: when **all** uses of a property must have the specified class as range.

- Existential Quantification: when **at least one** use of a property has the specified class as range.
- Individual Value: the specified property must be used to relate all individuals of a class with a specific named individual.
- Self-Restriction: all individuals of the class must relate to themselves through the specified property.
- Cardinality and Qualified Cardinality Constraints: the number of times a specific property may be used by an individual, and the number of times a property may be used by an individual to relate to a specific range class, respectively.

There are different 'dialects' or Species of OWL, which are defined by their expressivity, such as OWL-Lite, OWL-DL, OWL-Full and OWL2 – the latter of which is the one employed in this work. The dialect chosen for a specific application scenario is often dependent on its requirements. The more expressive the dialect, the more verifications need to be made by reasoners to guarantee its consistency; and therefore, the choice of dialect has a significant impact on computational requirements. Some dialects, like OWL-Full, do not guarantee decidability, meaning a reasoner may not be able to reach all possible conclusions and classifications for the ontology. ***Reasoners*** allow users to take advantage of the semantics of an ontology, guaranteeing its internal consistency and classifying individuals according to their properties and classes – a process which can be used to infer implicit knowledge from data.

Open World Assumption and (not) Unique Name Assumption

In DL, the lack of knowledge about a fact does not necessarily mean that the fact is false – there is the assumption that the facts representing a domain are fundamentally incomplete, i.e., an ***Open World Assumption*** (OWA). To prove that something is false, therefore, there must be a fact explicitly establishing so. Inversely, most database systems operate under a Closed World Assumption (CWA), in which what is not known is automatically considered false. This assumption is often necessary for knowledge mining and induction processes.

Another assumption in DL is that of the Not Unique Name Assumption (nUNA): even if two individuals have different unique identifiers, it does not necessarily mean they are disjunct; in fact, it is possible for a reasoner to conclude that two individuals refer to the same entity, i.e., may be shown to be equivalent.

Opposite of it is the ***Unique Name Assumption*** (UNA), which is not enforced in OWL.

CHAPTER 3

Time-Constrained Ontology Evolution

3. Time-Constrained Ontology Evolution

This chapter presents an architecture for time-constrained ontology evolution comprised of two tools: the *J2OIM* (JSON to Ontology Instance Mapper), which transforms JavaScript Object Notation (JSON) objects into RDF individuals to populate an ontology, and *TICO* (Time Constrained instance-guided Ontology evolution), which analyses streams of RDF individuals and attempts to identify potential changes to their definitions to trigger evolutionary processes. These tools help compensate for identified gaps in literature in instance mapping and modular versioning. The case-study for these tools involves a PdM scenario in which near real-time data sensor, enriched by contextual data, is continuously transformed into ontology individuals to trigger ontology evolution mechanisms. Results show it is possible to use the instance mapping mechanisms in an incremental fashion while assuring no duplicates are generated and the aggregation of similar information from distinct data points into intervals. Furthermore, they show how the ontology evolution processes effectively detect variations in ontology individuals, generating and updating existing concepts and roles.

While ontologies can be used to establish and enforce logical relationships between data points, which can then be used by a reasoner to uncover implicit knowledge, many data sources, however, often provide information in an unstructured or semi-structured fashion – using formats such as JavaScript Object Notation (JSON), Comma-Separated Values (CSV) or even plain text, which do not carry explicit semantics. While tools to automatically populate ontologies from said sources exist [29,30], they only allow for one mapping to take place at a time, which leaves the task of combining the results to the end user and adds extra steps to the ontology population process. This makes the tools harder to use in complex scenarios where automation is necessary, which is key when dealing with real-time data. As part of the solutions presented in this chapter, the proposed architecture employs a tool that transforms JSON data into ontology individuals – which not only allows for incremental population of repositories, but multiple mappers to be used simultaneously. Furthermore, this tool takes into consideration the time-sensitive nature of the data, using known reification patterns for the representation of time and making sure no duplicate

or redundant time information is generated. It also aggregates multiple identical readings during a period into the same ontology individual associated with only one interval.

Data scientists often run several experiments while exploring datasets, applying different machine learning and data mining algorithms so they can understand and utilize the data. This includes finding the most expressive set of features to use for the models, which can involve engineering additional features [31]. The nature of the experiments depends on the goals of the data scientist and the nature of the tasks at hand [31,32], and therefore it is not reasonable to expect any ontology to cover all possibilities pre-emptively. To ensure the ontology can describe the new features as they are needed, it must be updated over time, preferably in an automatic way. While ontology evolution [12] is a field focused on identifying and materializing changes in domain, existing approaches often fail to account for the time-sensitive nature of said changes, and approach evolution as going from one version of the entire ontology to the next [16]. This means that if a version of a given concept is necessary to describe a particular part of the data, it cannot be simultaneously used with concepts present in a different version of the same ontology (e.g. when answering a query). Additionally, identifying the version of the ontology that best answers a specific query is often left out of the question, and usually only the last version is kept in use.

To address this gap in existing versioning tools, this work proposes giving each ontology concept its own independent version history that can be accessed at any time, allowing a reasoner to decide which version of the concepts must be combined depending on the desired outcome. The framework for ontology evolution hereby proposed shall analyse streams of data containing individuals described through means of an ontology – an initial version of an ontology, so to speak – which, upon analysis, may reveal different applications of concepts and properties that require updating the ontology.

To answer **RQ1:** *To which extent is it possible to identify changes in classes and properties used by individuals provided via stream and reflect them in an ontology that represents those individuals during a specific period?*, this chapter will focus on two main tasks:

- Identifying existing sources that provide timeseries data from semi-structured sources and represent that information through means of ontologies that fully describe the temporal factors implied. To achieve

this, the data must be continuously and incrementally transformed from JSON into ontology individuals to populate the ontology.

- Use the now semantically-described ontology individuals to guide an ontology evolution process that reflects the changes in domain introduced by experiences with feature engineering and variations in data sources over time.

3.1 Background

The architecture presented next has been developed in the scope of the Predictive and Prescriptive Automation in Smart Manufacturing (PIANiSM) project [33], which aimed to facilitate the implementation of prescriptive and predictive maintenance approaches in the plastic extrusion industry. Achieving the goals of the PIANiSM project requires the incorporation of different fields of knowledge, technologies and data sources, whose domains range from industrial maintenance strategies and industrial Internet of Things (IoT) to data science. In this scenario, real-time data is acquired from sensors installed in industrial equipment [34], serialized in JSON, subsequently processed according to temporal representation and constraints defined by the ontology, and finally used to generate predictions that can inform maintenance activities. In this later step, machine learning and data mining algorithms have been used to process and analyse temporally represented data and identify irregular or atypical patterns [35], which can be symptoms of abnormal component behaviours that may lead to potential failures [36].

For a better grasp of the proposed architecture, it is important to understand the choices involved in their development. As such, the following sections will provide some context on the four main tasks that the architecture aims to accomplish, namely:

1. The ontologies chosen to describe the predictive maintenance domain.
2. The temporal representation pattern applied.
3. Assessing whether existing tools for mapping JSON entities to ontological ones are sufficient for this use-case.
4. A brief outlook on the processes of ontology evolution that allow for the description of domain changes over time and to establish if existing tools can be applied in this scenario.

3.1.1 Ontologies for Predictive Maintenance

The distinct data sources utilized in this work are characterized by individual ontologies, representing time, sensors and manufacturing equipment, among others. Some of these domains were described by pre-existing ontologies, which were imported into a new ontology that describes the greater project scope and whose details can be found in Annex 1 and further detail in [35].

Transforming temporal data from multiple and diversified sources into semantically structured data, despite the existence of a significant number of tools, is nonetheless a complex task, and no industry standards have been defined. Furthermore, when it comes to its semantic representation, while time can be represented through several popular ontologies, such as [37], and through different reification approaches [38,39], when it comes to representing the temporal aspects of data provided in real-time via streams there is still much work to be done [40].

3.1.2 Time-Representation

The change in classes and properties over time is an important factor in the representation of temporal entities in ontologies. A property may have different values at different time points, which often requires some level of reification. Additionally, properties must maintain the original semantics when converted to temporal relations. CHRONOS [39] is a Protégé[4] plug-in that facilitates the transformation of static entities into dynamic temporal entities and follows a W3C recommendation for time representation. While this tool easily adds a temporal dimension to data, it can currently only be used with specific, older versions of Protégé – which is not the ideal scenario for transformations that should occur in an automatic process, and not manually generated in an ontology development environment. As such, the temporal transformation processes applied to static data originating from plastic extrusion processes represented in this work had to be re-implemented, but were modelled corresponding to the approach explored in CHRONOS and its subsequent reasoning tool [41]. To do so, the 4D-Fluents pattern is used to represent the persistence of objects through time [38] by reifying the temporal part of an object. The changes in objects' qualities over time are suitable for describing the time-sensitive data acquired from sensors in this scenario. This is represented in Figure 1, which exemplifies

[4] https://protege.stanford.edu/

the conversion of a static relation of entities *Machine* and *Manufacturing Operation* into the 4D-Fluents pattern: by creating a new individual of class *Event* related to a new *Time Interval*, with the *Start Instant* and *End Instant* during which the *Event* occurred.

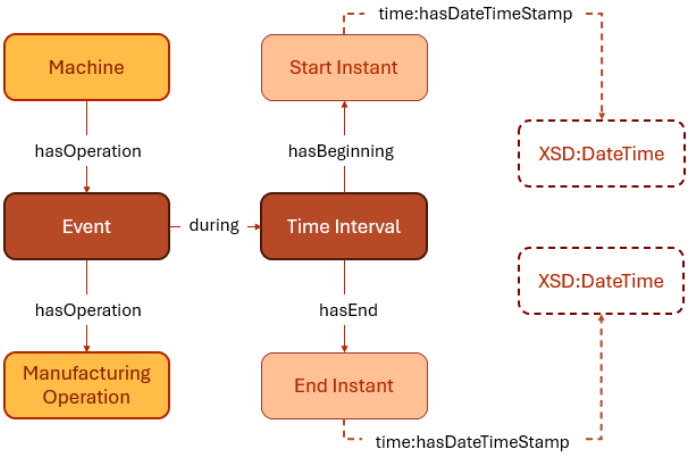

Figure 1 – Temporal object property between entities[5]

Event represents the temporal part of *Machine*, connecting *Machine* and *Machine Operation* during a specific *Time Interval*. As such, the same *Machine* may be related to a set of *Events* that take place at different times. Following the representation pattern of CHRONOS, the same property, *hasOperation*, is used both to connect *Machine* to *Event* and *Event* to *Machine Operation*. *Event* takes place during a specific *TimeInterval*, with starting and ending *Instants* (using the Object Properties *hasBeginning* and *hasEnd*), which have their own individual time stamps, represented through the Datatype Property *hasDateTimeStamp* and connecting them to the XSD:DateTime value.

3.1.3 Mapping JSON files to ontological entities

A semantic representation of data is proposed in [29], applying a set of transformation rules to extract an OWL Ontology originating from JSON documents. This approach only has the capability of transforming a single JSON document or file, not considering joining or establishing relationships between data stored in several files into a single ontology. Furthermore, the automatic transformation rules are based on the JSON's structure, the key labels, and the

[5] For the remainder of this section and for the sake of simplicity, please consider the full lines to represent OWL Classes and Object properties, and the dashed ones to represent Datatype properties and Literal values.

nesting level of the document. This approach does not contemplate configurations or templates and does not require human intervention in the transformation process: the structure of the produced ontology is always the one provided by the input JSON file. In [30], the semantic transformation of data also starts from a single JSON source. Here, the goal is to transform JSON documents into a predefined ontology structure in terms of OWL classes, object and data properties, without the need to perform any changes to the original JSON Schema. To achieve this, an intermediary JSON Schema serves as template to map the original JSON properties to existing OWL classes, object and data properties. This template therefore allows for the automation of the necessary mapping process to perform the semantic transformations. However, much like in [29], the JSON Schema mapping only works with a single JSON document and does not allow for the interrelation of several JSON document sources into the same ontology, making it harder to incrementally add data to it over time in an automatic fashion.

3.1.4 Representing Ontology Evolution

In [42], the Open Provenance Model is applied to show the changes between two versions of a given ontology, allowing to identify how, when and why a specific evolutionary action was taken. In [43], this idea is applied to stream data, under the assumption that all axioms may be eventually replaced with new ones as time progresses. Similarly, [44] introduces a queryable a framework that identifies when changes occurred.

It should be noted, however, that the approaches mentioned here consider the ontology a monolithic structure: each step of the evolution gives rise to a new version of the ontology, and it is not possible to combine definitions from different versions when executing a query. The architecture described next aims to fill this gap in the literature by allowing each entity to have its own version history and all of them accessible at once – leaving the task of choosing which is the correct option to the reasoner.

3.2 Architecture

Detecting changes in data is in the core of many predictive processes, as they can be a symptom of component wear and potential future failure states. Not only can these processes generate new insights and add new knowledge to the

known domain, but they can also demand the execution of data transformation processes that may differ from the initial conceptualization of the domain.

The proposed architecture is comprised of a set of tools and ontologies, which will obtain data directly from sensors, translate it from JSON to ontology entities and then use those as a potential trigger for ontology evolution. This flow is depicted in Figure 2.

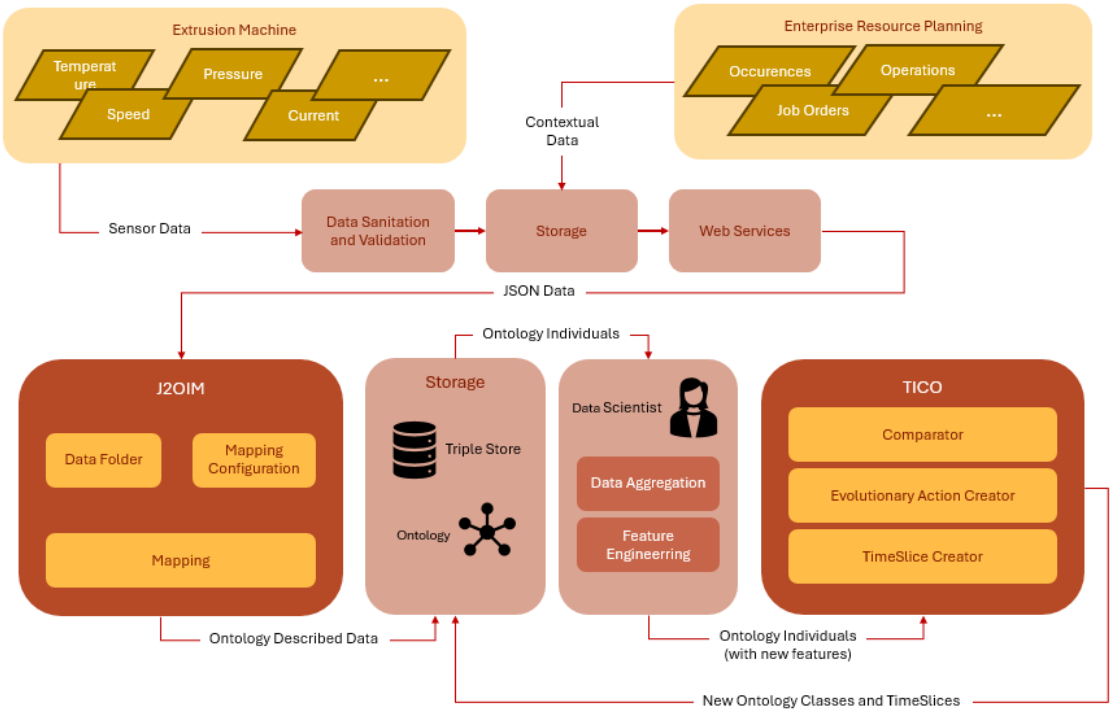

Figure 2 – Data flow, from the individual sensor and software sources to the triple store, including feature engineering processes and identification of new ontological entities

The scenario considers two main sources of data: three plastic extruders, providing information regarding each extrusion head via several sensors, and the ERP software, which adds contextual data to this information – regarding equipment definitions, manufacturing orders and occurrences, among others. This data is stored in a relational database and made available through means of a webservice, in JSON format. The webservice also allows to customize some queries, such as allowing to query specific time intervals and which parameters can be queried for each machine (as they have different sensor types). The webservice contains a total of 20 endpoints, 5 of which pertain to information provided by the ERP, with the remainder corresponding to different sensor data from two different extrusion machines.

To make use of the ontological definitions provided by the ontologies employed, the data acquired by the sensors must be semanticized through some

process. This is where *J2OIM* (JSON to Ontology Instance Mapper) comes into play, by establishing a set of mappings between JSON structures and ontology entities. One of the main objectives of its implementation was the representation of time-sensitive data related to extruder machines supplied by sensors in temporal N-ary relations whenever applicable. This transformation should also be able to happen in near real-time, maintaining the time-sensitive aspect of the original stream – and thus facilitating the ontology individuals themselves via an RDF stream $\mathcal{S}$ as well.

Definition 1 (RDF Stream)

An RDF stream is a (possibly infinite) sequence of triples <Subject, Predicate, Object>, in which: Subject is an IRI or blank node; Predicate is an IRI, and Object is an IRI, blank node or literal.

The individuals provided by $\mathcal{S}$ will then be the target of subsequent processes and experiences. Since different experiments may require a number of different features to be engineered, and it is important to keep track of how and when those were created. However, since said features are highly dependent on the context they were created for – e.g. the experience they are part of – it is unreasonable to expect the original ontology design to reflect all possibilities. The data analyst may introduce and run new processes that change the data on the stream, while not necessarily reflecting those changes in the ontology. The analysis of the individuals of the stream may, nonetheless, highlight the need to automatically update the ontology to reflect any changes. Because $\mathcal{S}$ is unbounded – with unknown start and ending points and potentially infinite –, it is necessary, at some point, to assume sufficient analysis has been done and decisions that may result in ontology evolution can be made. With this in mind, the analysis of the individuals delivered by $\mathcal{S}$ is executed during a particular period of time (or timeframe) t, at the end of which an evaluation of the results is made to generate evolutionary actions. t represents a period in the stream that can be determined by the timestamps on the individuals themselves, a specific number of individuals or an actual time interval.

Definition 2 (Timeframe)

An RDF stream can be divided into subsequences (t_1, t_2, …, t_n, …) of triples called timeframes. Data about a particular individual i (i.e., triples for which i is the Subject) are bound to a single timeframe, and arrive sequentially.

The *TICO* tool will then analyse the individuals from $\mathcal{S}$ and assess in which ways they are different from the definitions of the concepts they belong to, and create new classes in the ontology that reflect those changes as presented in Figure 3. Each individual is only analysed once per timeframe, and no records about the individuals or the statistics they informed are maintained between timeframes. As such, should the same individual feature in more than one timeframe, they are considered distinct.

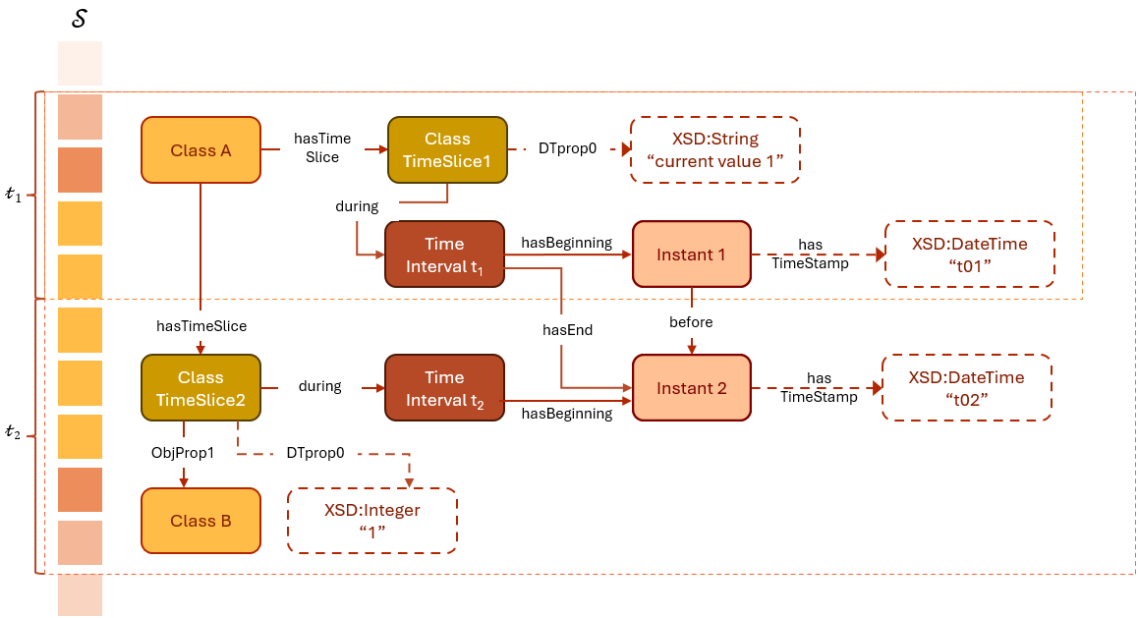

Figure 3 – Representing changes in the definition of a class (Class A) between continuous timeframes

The new concepts will be reified through different classes, each of them representing a definition of the concept for a specific interval, allowing for an ontologist to potentially make further changes to the ontology that use the new features more easily, and have those changes be used only in the adequate period. Because the ontology changes are described as new time slices of existing concepts with clearly defined beginnings and endings – and no overlaps –, there is only the need to **add** axioms to the ontology. Removing a property would basically entail adding an ending time to that specific property's time slice – meaning changes can simply be added to the triple store as well.

3.2.1 J2OIM

J2OIM is this thesis proposal of an architecture that supports the transformation of the JSON data acquired from the different sources into instances represented through ontologies. In this context, the ontology described in [35] is used for the representation of Machine and ERP data for predictive maintenance – in particular, that of plastic extrusion machines –, with time-sensitive data representation following a 4D-fluents pattern [38]. It is worth noting this does not correspond to the total data produced by the machines or available through the ERP, as some of it was kept confidential. These two data sources were considered for their relevance in PdM processes – real-time data obtained from the machine's sensors may show variations that are compatible with equipment wear and indicative of potential failure, and the data from the ERP can further contextualize it. For example, the machine's noise and vibration patterns when executing one type of operation with a specific material may be different from those obtained under different circumstances. By using the ERP information about manufacturing orders and materials, the reason for the variation in the patterns can be established and they will not be considered a symptom of potential failure.

Table 1 discriminates the types of data and total records found in the sample.

Table 1 – Total records of sample data

DATA ORIGIN	FILE TYPE	RECORDS
ENTERPRISE RESOURCE PLANNING DATA	Machines	3
	Sensor Types	15
	Occurrences	17
	Manufacturing Operations	9110
	Manufacturing Orders	3.157
SENSOR DATA	Sensor readings	1.001.965
TOTAL		**1.014.267**

In total, the webservices were queried and the results converted into files containing 5 types of process data and representing data from 15 different sensors. These must be aggregated, related, and transformed into a concise semantic model. The data samples used in this work consisted of 3 machines, with 24 variables each, describing the machine, sensors, occurrences (such as maintenance operations), and manufacturing operations. The two types of data

are structured differently. Records provided by sensors always take the form of a tuple (s, ts, rd), in which s is one of the known sensors, ts is the timestamp in which the record was captured and rd is the value registered. On the other hand, records obtained from the contextual sources have a more complex representation [35] and establish relationships with sensor data.

Overall, the obtained data is not normalized and has no inter-file relation, rendering its interpretation difficult. To cope with this problem, a JSON mapping file was developed to map each property of each file to a triple subject-predicate-object. Once a property is mapped, it can be transformed into an individual with object properties and data properties, which will allow the insertion into an ontology file and a triple store.

In J2OIM, mappings are executed by a function $map(OT, f, c, p) \rightarrow OI$ such that:

1. OT is the target ontology;
2. f is the source dataset, in JSON;
3. c is one of $\{c_1, \dots, c_n\} \in C$ where C is the set of concepts defined in OT;
4. p is a list of mappings between the JSON keys and the properties, assuming the form $(prop_{JSON}, r_{OT})$ in which:
   i. $prop_{JSON}$ is a JSON key in f and
   ii. r_{OT} is one of $\{r_1, \dots, r_n\} \in R$ where R is the set of properties defined in OT.
5. Finally, OI is the resulting set of ontology individuals described by OT.

J2OIM has three main modules that perform the transformation processes, namely: (1) the Main module, which orchestrates the transformation process, (2) the Data Loader module, which loads the data according to the defined Model and (3) the Data Writer module, which persists the data as defined in the mapping configuration.

Following Figure 4, the Main module orchestrates the process. First, a controller is launched to execute the loading process of the configuration and data files based on the defined models. Afterwards, the loaded data is forwarded via a Batch Loader to an Observable interface.

The Data Loader fulfils the responsibilities regarding import and error handling. These are based on different configuration files, specifying how the importing process will take place, where are the source and mapping files and how to condensate and aggregate data. It guarantees that all machine data has been properly read, processed and sorted.

In the Data Writer module, the Ontology and Triple Store Writers are observing the Batch Loader, and for each record passed to the Observable interface, the writers are triggered and persist the data as defined in the mapping configuration file.

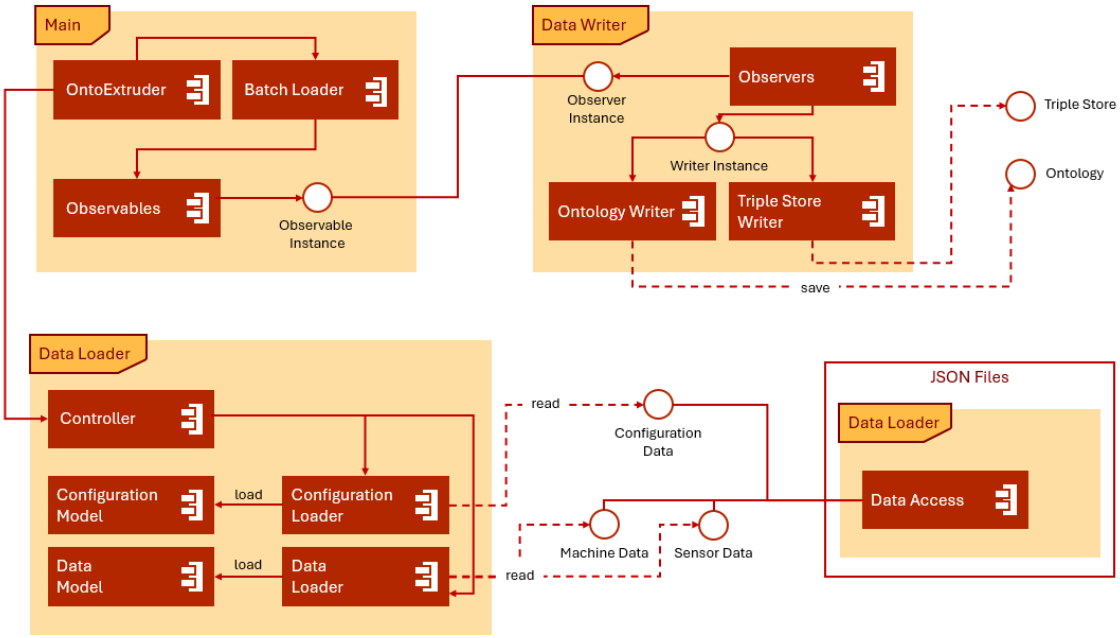

Figure 4 – Component Diagram of J2OIM's architecture

While data can be consumed in near real-time or in batches – meaning it is possible for data to arrive late – *J2OIM* only attempts to correlate individuals within the same timeframe (which could constitute a batch). However, the Universal Unique Identifiers (UUID) of individuals and *Intervals* are generated through means of a function that guarantees the same UUID is generated for any *Intervals* with the same starting and ending dates. Still, in the unlikely scenario that late arriving data concerning a new *Interval* has the same beginning and ending dates as an existing one, *J2OIM* will replace the previously stored value – i.e., the last arriving version prevails. A similar UUID generation process guarantees that any two identical individuals of a particular class will have the same UUID, ensuring no repeated values are stored – only the latest-arriving version of the data.

Finally, it is also important to note that while *TICO* makes changes to the ontology, *J2OIM* maps JSON instances always to the first known version of the ontology, as the mapping configurations are not automatically updated by *TICO* and that is not the focus of this work. As such, it is not possible to update the format of new instances to match the changes in the ontology; meaning that the

version of the concepts that should be used to classify the instances must be inferred through means of a reasoner.

3.2.2 TICO

Equipment behaviour is bound to change over time, and thus the definitions of what is considered normal and abnormal regarding it evolve as well. Furthermore, as new analyses are performed over data, new features may need to be computed, either by extending or combining existing ones. Because of this concept drift, the history of previous states cannot be queried using the latest version of the ontology; at any given historical moment, it is important to assess the version of the ontology that must be considered, by applying only the changes that are relevant for the considered interval. This also means that many of the existing concepts of the ontology should also include a time dimension. While, as previously mentioned, reification is a common approach in a scenario such as this, the particular reification approach depends mainly on whether the quality assignments, which are only valid for a given time interval, can be overlapped with other quality assignments, and if such an overlap is always complete (with the same starting and ending timepoints) or partial. *TICO* is a novel framework that aims to incorporate and represent these changes in knowledge through means of ontology evolution practices.

TICO is guided by incoming ontology individuals that arrive through an RDF stream. By analysing the frequency of roles and concepts used to describe those individuals and a set of user-defined thresholds, *TICO* determines if and when there has been sufficient variation during a time period to warrant modifying the ontology, and executes a set of evolutionary actions to bring those changes about. Because these individuals have been previously reified into 4D-Fluents by *J2OIM*, it is easy to query their time dimension, which can then be used to establish the *Instant* when a pattern arose and when it stopped being significant (if this information is not available, real time can be used). *TICO* reifies this through 4D-Fluents as well, by creating a *TimeSlice* of the concept that is added to the ontology which has a starting and, potentially, an ending date, as illustrated in Figure 5.

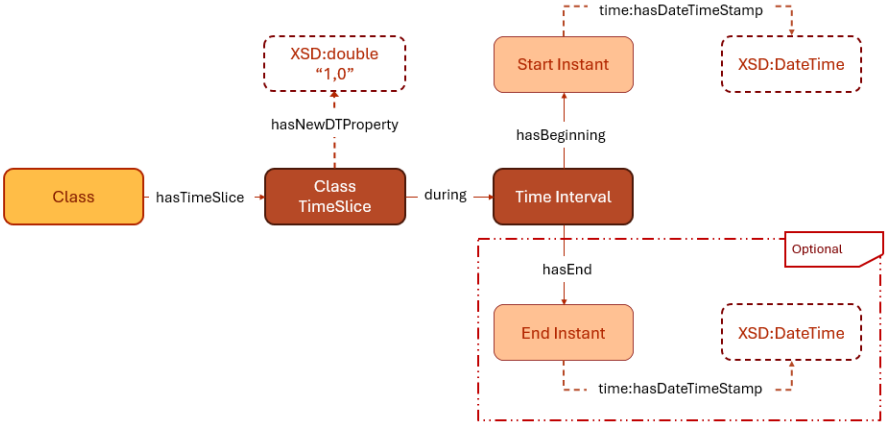

Figure 5 – Example of a TimeSlice using the 4D-Fluents reification approach

As pictured in Figure 6, *TICO* has three main modules it uses to generate and execute evolutionary actions, namely:

1. the Comparator, which reads the inputs, orchestrates the data flow and produces the new version of the ontology at the end of the timeframe,
2. the Metrics module, which reads user configurations and stores metrics relative to ontology structures during the period of analysis and
3. Evolutionary Actions, which creates said actions and generates *TimeSlices* for concepts.

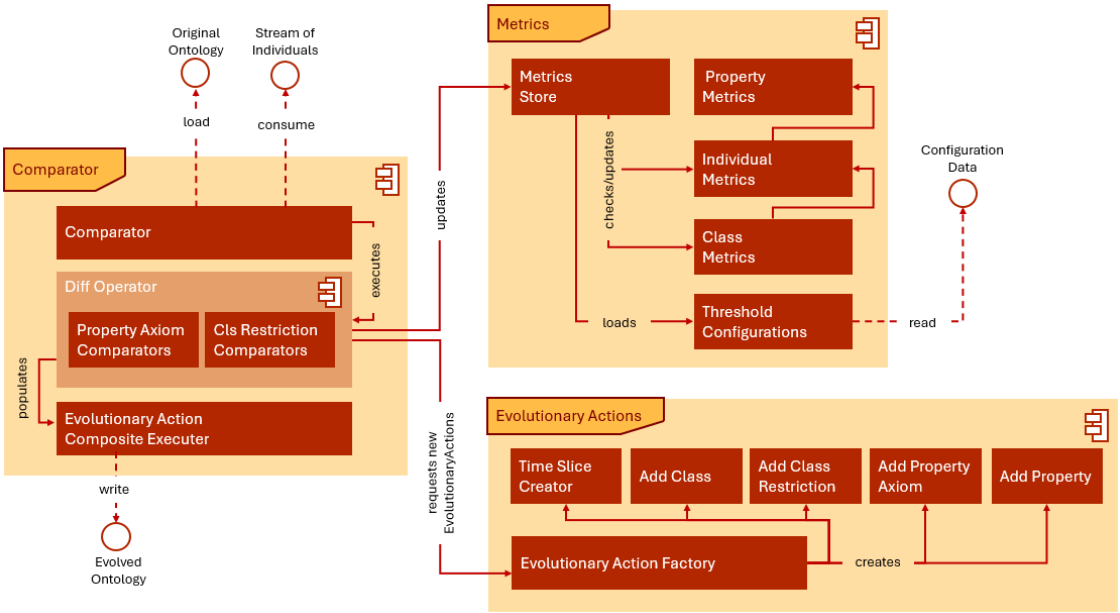

Figure 6 – Component Diagram of TICO's architecture

Here, the Comparator module kickstarts the process by having the similarly-named component read both a seed ontology – the Original Ontology – and a stream of individuals that would prompt modifications – the Ontology Individuals. The comparison process flow that follows is illustrated in Figure 7.

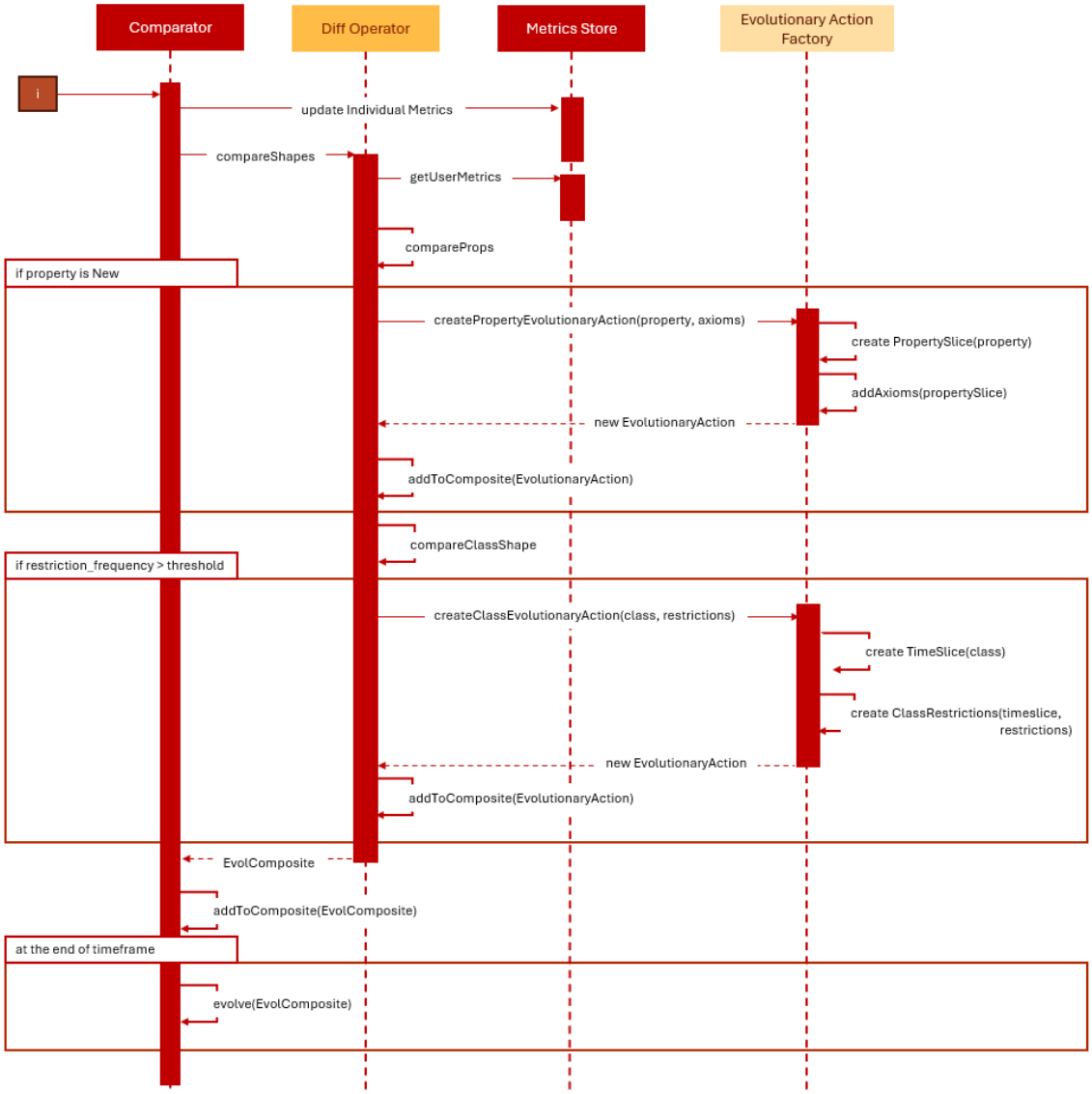

Figure 7 - Overview of the Comparison Process

The Comparator class triggers the Diff Operator component, which is responsible for analysing whether the original version of an individual's describing concept matches with the individual's definition and properties. To do so, it triggers an additional set of comparators, which will assess, among others, the differences in restrictions, classes and roles employed. The metrics and configurations employed to assess these differences are provided by the components of the Metrics module, which reads configuration files and calculates and stores different metrics across sets of individuals. Here, a user-defined threshold is used to determine whether enough individuals (either by percentage or in absolute numbers) with the same structure have been identified, and if their structure is different from the one in the ontology. The result of the application of the Diff Operators will be a set of one or more Evolutionary Actions, which are created by an Evolutionary Action Factory and stored through

means of a Composite. These actions include, for the time being, the addition of properties and classes, along with *TimeSlices* of new or existing concepts and restrictions – but not the identification of potential property axioms, nor relationships between classes. Finally, when the Composite executes its set of Evolutionary Actions, the resulting modified ontology is stored in an additional model, described in Figure 6 as Evolved Ontology Model.

At this point, if a new *TimeSlice* $TS(cls)$ of a given class cls is created, the following metrics are analysed for the individuals of cls:

- For each property P used by the individuals of cls, the maximum and minimum number of times it was used per individual. At the end of the timeframe, these are used to establish Cardinality-based class expressions that are added $TS(cls)$.
- For each property P, the number of times P was used by individuals of cls for each specific range class. These are used to derive $AllValuesFrom$ and $SomeValuesFrom$ class expressions to add to $TS(cls)$ – by assessing whether only one class was used as the range of P. If this is the case, an $AllValuesFrom$ class expression is added, otherwise $SomeValuesFrom$.

The *TimeSlices* created by *TICO* do not overlap: when a new *TimeSlice* is discovered, the previous one is updated with an ending date corresponding to the start date of the new one. This ensures that an individual cannot be classified as belonging to two different *TimeSlices*, preventing potential inconsistencies, as their definitions will be different. Because the individuals arrive sequentially, as provided by the webservice and *J2OIM*, there is no expectation that incoming data will be producing *TimeSlices* that would overlap previously existing ones, although such possibility will be considered in the future to account for potentially late-arriving data.

3.3 Experiments and Results

The experimentation process described next can be separated into three main parts:

1. ***Ontology instantiation***, in which the JSON instances are converted to ontology individuals by *J2OIM*
2. ***Feature engineering***, in which new features are created by data scientists and incorporated into the dataset and, finally,
3. ***Ontology evolution***, in which all individuals, including the ones created in the second step, are analysed to fuel ontology evolution processes and generate either new concepts or new versions of existing concepts by *TICO*.

3.3.1 Ontology Instantiation

In this phase, the webservices are queried for the data collected from sensors and management software and transformed from JSON into ontology individuals.

All the data had to be tagged with an identifier or reference to establish relationships via object or datatype properties. For the present implementation, each data record read from the source JSON files was tagged with a UUID generated in runtime during the transformation phase. To provide readable information regarding each individual or subject created in the ontology and triple store, each subject reference is composed of the *subjectUri* defined in the mapping configuration, appended with a UUID based on its data contents. This ensures that any subject, even if inserted or referenced multiple times, maintains the same reference.

Not all entities identified in the data pertain to time-sensitive data; some entities have non-temporal relations with others. The identified temporal relations associated with the extruder machine regard manufacturing operations and manufacturing occurrences, and the temporal relations associated with a sensor (with all the sensed output values that occurred over time).

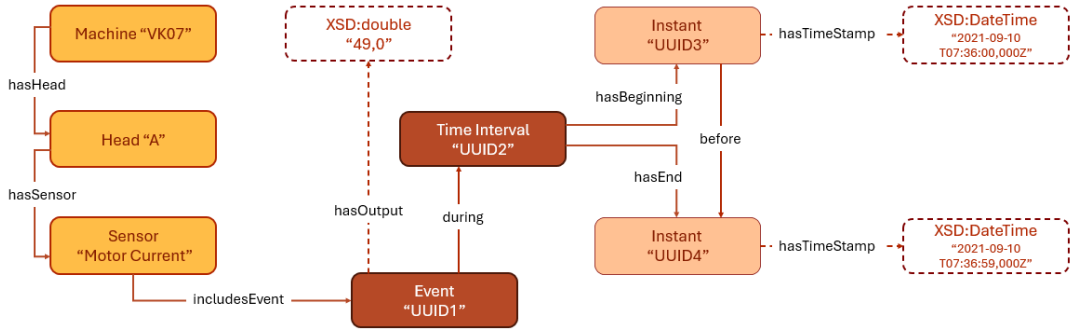

Figure 8 – Events occurring in sensors

Figure 8 shows the different ontology individuals created when a JSON sensor reading is processed by *J2OIM*. The machine is now related to the sensors present in the head, and sensor data is related to a precise time interval, with a beginning and ending *Instant*, during which the reading provided by the *hasOutput* datatype property is valid. Figure 9, on the other hand, shows how the several *Events* observed on a particular *Machine* are now all interrelated and temporally represented. In this case, a *ManufacturingOperation* (executing a specific *ManufacturingOrder)* and an *Occurrence* are associated with the *TimeIntervals* in which they occur. Additionally, it should be noted that the individual *Instants* and *Events* are ordered – i.e., they use the role *before/after* to ensure the direction of time between them.

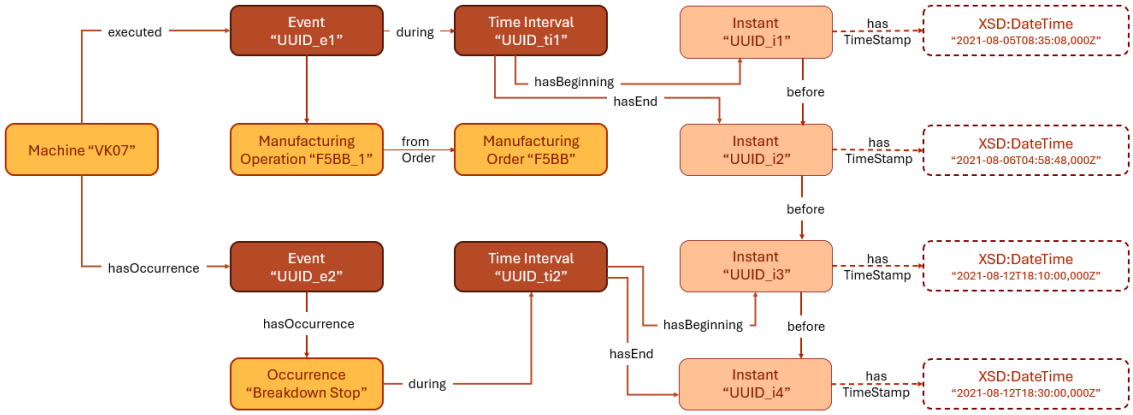

Figure 9 – Events occurring in machines

Table 2 presents an overview of all the individuals (or subjects), in the form of triples, differentiating those involved in non-temporal and temporal relations present in the plastic extrusion use-case. Here, the initial non-temporal relation *Sensor-hasOutput-Value* is now represented as a temporal relation *Sensor-includesEvent-Event, Event-hasOutput-Value, Event-during-TimeInterval*.

Table 2 – Relations between subjects

SUBJECT	PREDICATE	OBJECT	RELATION TYPE
Machine	has head	Head	non-temporal
Head	has sensor	Sensor	non-temporal
Sensor	includes event	Event	temporal
Manufacturing Operation	from order	Manufacturing Order	non-temporal
Machine	includes event	Event	temporal
Event	has output	Value	temporal
	executed	Manufacturing Operation	temporal
	occurrence	Manufacturing Occurrence	temporal
	during	Time Interval	temporal
Time Interval	has beginning	Time Instant	temporal
	has end	Time Instant	temporal
Time Instant	before/after	Time Instant	non-temporal

Once inserted in the ontology, extrusion head sensors instances have temporal relations to represent the output values occurring over time. In Code Snippet 2, the individual $j.4$: $erpSensor\_d7d0646d$ represents an extrusion head sensor that has two temporal events (*extEvent*): representing the readings obtained by the sensor "Motor Current" of the head "D" in extruder machine "VK07" at two different, sequential intervals. The *TimeInterval* is comprehended between the *Instants* represented by the timestamps. The resulting instances (data properties, in the case of sensor data) obtained from the data transformation are listed in Table 3. These show that non-temporal data from the ERP resulted in the same number of instances as the initial data, as expected, while sensor readings resulted in about the half of the initial values – from 1.001.965 shown in Table 1 to 91.055 seen in Table 3 – due to the aggregation of repeating values over the same time interval and by averaging the values over each minute[6]. On the other hand, because of the reification approach employed, 252.945 instances were created to describe *Intervals* and *Instants*.

[6] Aggregation here is done as part of data pre-processing and uses user-defined configurations to establish the periodicity.

Table 3 – Total resulting instances

DATA ORIGIN	INSTANCES	RECORDS
ENTERPRISE RESOURCE PLANNING DATA	Machines	3
	Sensors in extrusion heads	121
	Occurrences	17
	Manufacturing Operations	9.110
	Manufacturing Orders	3.157
SENSOR DATA	Sensor readings (data properties)	91.055
TEMPORAL DATA FROM ERP AND SENSORS	Events	78.322
	Intervals and Time Stamps	252.945
FEATURE ENGINEERING DATA		12.885
TOTAL		**447.615**

Code Snippet 2 – Individual of class Extrusion head Sensor (MotorCurrent) and sample output Events

```
j.4:erpSensor_d7d0646d
   rdf:type                      j.2:Sensor ;
   rdfs:label                    "erpSensor_VK07_D_MotorCurrent" ;
   sensors4ExtruOnt:sensorName "MotorCurrent" ;
   j.5:includesEvent             "erpEvent_1953c8e2" ;
   j.5:includesEvent             "erpEvent_2310f1b2" ;
   CM_ontology:hasSensorID       "d7d0646d" .

j.4:erpEvent_1953c8e2
   rdf:type                      j.0:extEvent ;
   rdfs:label                    "erpEvent_1953c8e2" ;
   j.3:after                     j.4:erpTimeInstant_0f739594 ;
   j.3:before                    j.4:erpTimeInstant_c0135577 ;
   j.3:intervalDuring            j.4:erpTimeInterval_4afbee81 ;
   j.5:hasOutput                 "44.807500000000005" .

j.4:erpEvent_2310f1b2
   rdf:type                      j.0:extEvent ;
   rdfs:label                    "erpEvent_2310f1b2" ;
   j.3:after                     j.4:erpTimeInstant_c0135577 ;
   j.3:before                    j.4:erpTimeInstant_a912cc14 ;
   j.3:intervalDuring            j.4:erpTimeInterval_2dab33d1 ;
   j.5:hasOutput                 "49.000000000000015" .

j.4:erpTimeInterval_4afbee81
   rdf:type                      j.3:DateTimeInterval ;
   rdfs:label                    "erpTimeInterval_4afbee81" ;
   j.3:hasBeginning              j.4:erpTimeInstant_0f739594 ;
   j.3:hasEnd                    j.4:erpTimeInstant_c0135577 .

j.4:erpTimeInterval_2dab33d1
   rdf:type                      j.3:DateTimeInterval ;
   rdfs:label                    "erpTimeInterval_2dab33d1" ;
   j.3:hasBeginning              j.4:erpTimeInstant_c0135577 ;
   j.3:hasEnd                    j.4:erpTimeInstant_a912cc14 .
```

3.3.2 Feature Engineering

The instances generated by this process can then be used for different data science experiments, as illustrated in Figure 2**.** In this case-study, sensor data is aggregated by the minute and new features are generated following the experiments described in [45]. In that work, two new features were devised: the running average (with sample size of 5) of the Motor Current feature, and Feed Rate, which is given by combining values of Throughput and Speed. For the purposes of this experiment, consider Motor Current's running average as an extension of the existing Motor Current generated *Events*, and Feed Rate as a new type of *Event* – a Virtual Event – that correlates the values of two *Events* generated by two existing *Sensors* at a given time. Figure 11 and Figure 10 highlight the generated data – new classes and relationships represented in red – and how it relates to the existing structures.

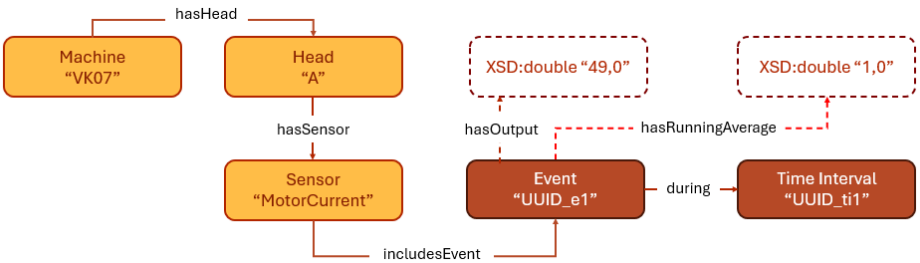

Figure 10 – Extending the Motor Current sensor's Events

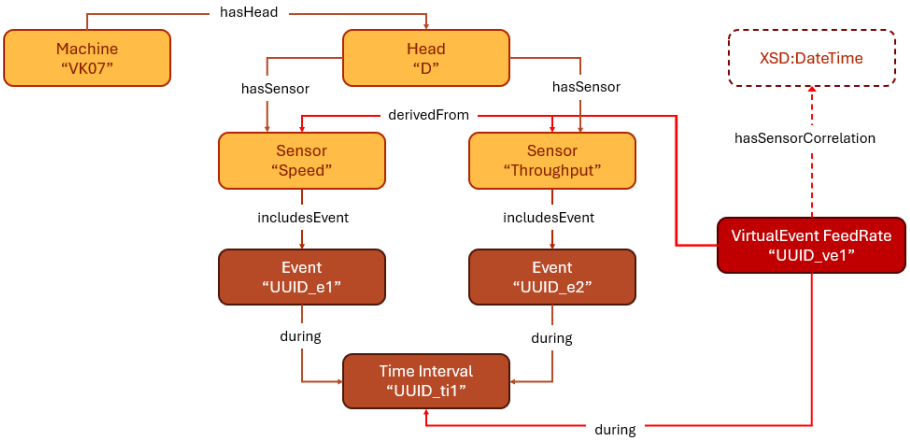

Figure 11 – Using "Speed" and "Throughput" Events to create a VirtualEvent "Feed Rate"

Since these two new features were created on demand, they are not part of the initial ontology used to describe the data and cannot be used in potential subsequent reasoning processes. These new instances, with their new features, are then fed to *TICO*, which tries to identify both the new ontological structures and changes to existing ones, and update the ontology accordingly in an automatic fashion. Furthermore, these experiments mean that, potentially, there

may be different versions of the ontology at hand: the original one, described in [35] and representing the instances as they are output by *J2OIM*; a second version which includes a new type of *Event* – the *VirtualEvent* and the *derivedFrom* role; and a third version that also describes the *hasRunningAverage* role and how it relates to Motor Current generated Events.

3.3.3 Ontology Evolution

In order to simplify the comprehension of the ontology evolution process undertaken by *TICO* and the following examples, a seed ontology was generated, which contains only the minimal definitions used to describe the initial sensor data dataset and that may, therefore, be affected by evolutionary processes, as illustrated in Figure 12.

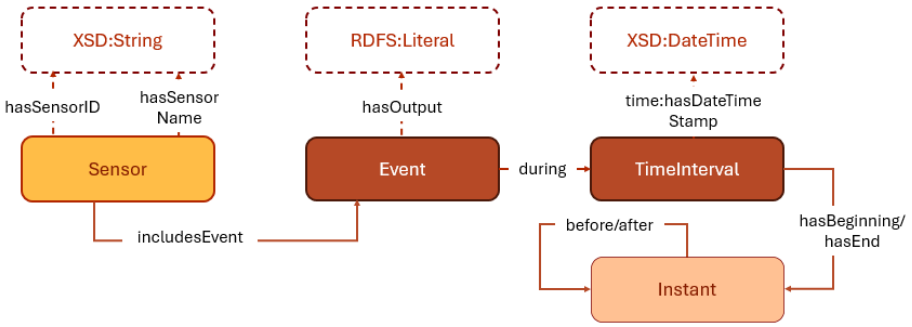

Figure 12 – The "seed" ontology

The seed ontology is a subset of the ontology described in [35], including only four concepts (the *Sensor, Event, TimeInterval* and *Instant* classes) and ten roles, namely: six object properties (*before, after, during, hasBeginning, hasEnd* and *includesEvent*) and four datatype properties (*hasTimeStamp, hasOutput, hasSensorName* and *hasSensorID*). It was created for this experiment with the purpose of keeping the number of concepts low, as to allows us to keep a better track of the evolutionary processes and their results.

The alterations to the data fed to *TICO* are distributed according to three different experiment periods, or three timeframes, each comprised of a sample of 2188 *Events*. The first round was used in a scenario in which the Feed Rate feature was required; the second one includes *Motor Current*'s running average; and the third round uses both new features simultaneously.

The timeframes were supplied sequentially to *TICO*, which allowed it to generate reports regarding the evolutionary actions created and executed following each timeframe. Table 4, Table 5 and Table 6 summarize the reports for each round, detailing the evolutionary actions that were effectively triggered. As

shown in Table 4, a total of 19 Evolutionary Actions were triggered when comparing the seed ontology to the structures present in the first set of instances.

Table 4 – Evolutionary Actions triggered by the first set of instances (timeframe 1)

EVOLUTIONARY ACTION	VALUES
Composite 1: 2 Evolutionary Actions	
Add ObjectProperty	hasSensorCorrelation
Add ObjectProperty	derivedFrom
Composite 2: 6 Evolutionary Actions	
Add TimeSlice	TS__Event__1
Add Cardinality Restriction	hasSensorCorrelation exactly 1 Thing
Add AllValuesFrom Restriction	hasSensorCorrelation only Sensor
Add SomeValuesFrom Restriction	TS__Event__1 isTimeSliceOf Event
Add HasValue Restriction	after "2021-09-09T12:15:00.000Z"
Add EquivalentClass	Event AND (after "2021-09-09T12:15:00.000Z")
Composite 3: 7 Evolutionary Actions	
Add Class	VirtualEvent
Add Cardinality Restriction	derivedFrom exactly 2 Thing
Add SomeValuesFrom Restriction	derivedFrom some Event
Add TimeSlice	TS_VirtualEvent_1
Add Cardinality Restriction	derivedFrom exactly 2 Thing
Add SomeValuesFrom Restriction	derivedFrom some Event
Add SomeValuesFrom Restriction	TS_VirtualEvent_1 isTimeSliceOf VirtualEvent
Composite 4: 4 Evolutionary Actions	
Add TimeSlice	TS__Sensor__1
Add SomeValuesFrom Restriction	TS__Sensor__1 isTimeSliceOf Sensor
Add SomeValuesFrom Restriction	includesEvents some Sensor
Add Cardinality	hasSensorID exactly 1 rdfs:Literal
TOTAL EVOLUTIONARY ACTIONS	**19**

Two object properties were created to accommodate for the roles present in the *VirtualEvent* classes and its relationship with existing *Events*, and three *TimeSlices* were created in total. The first one pertains to the *VirtualEvent* class, which is effectively new and must be added to the ontology, along with its first

TimeSlice. The necessary conditions for this class are assumed to be the roles that appear in all individuals of the class analysed in this period.

Because *derivedFrom* appears always twice per individual and its range is consistently an individual of class *Event*, two restrictions are created (Cardinality and $SomeValuesFrom$ restrictions). While there are no changes between the original version of *VirtualEvent* just discovered and its first *TimeSlice*, they are effectively copies of each other – the only difference being that the first *TimeSlice* has a $HasValue$ restriction on its list of necessary and sufficient conditions. Note that *VirtualEvent* could potentially be considered a subclass of *Event*, but *TICO* does not have evidence to support such relationship.

A *TimeSlice* for the *Event* class is also created, to accommodate for the appearance of the *hasSensorCorrelation* role. As before, since it consistently appears in relation to the *Sensor* class, the $AllValuesFrom$ restriction was created with that range. However, while *TICO* managed to detect that this role only appears once per *Event*, it failed to specify the range of the Cardinality restriction it created. The same could have been achieved by making the *hasSensorCorrelation* role functional. While it may seem that *TICO* has opted for the least elegant option, it is important to note that it is not known whether the property may be used for a different cardinality restriction later, which could potentially conflict with a functional definition.

As for the existence of the remaining *TimeSlice* – of the *Sensor* class – while Figure 12 may suggest that no changes to this class should have occurred, it is important to note that neither of the roles represented in the figure were part of the class's definition. As such, and because they were always identified in conjunction to *Sensor*, a *TimeSlice* adding them to the necessary conditions of said class has been created. It is also interesting to note here that, since both *Sensor* and *VirtualEvent* are not temporally reified the same way *Event* is, the framework could not identify a starting point for the generated *TimeSlices* that it could add to the classes' necessary and sufficient conditions.

The second timeframe, summarized in Table 5**,** resulted in a much smaller number of Evolutionary Actions, with a total of seven. *TICO* correctly identifies *hasRunningAverage* as a Datatype property, but fails to infer the details of its type, going for the broader type rdfs:Literal. Because in this period the individuals of *Event* all display the new role, a new *TimeSlice* of the class is created with it in its list of necessary conditions. Similar to the previous round, the new *TimeSlice* is equivalent to the original class, as long as it takes place after the *Instant* that marks

the moment when the framework initially identified the pattern. The major change is the modification of the *TimeSlice* created in the previous timeframe, which is modified to have both a beginning and ending date by being equivalent to the intersection of the *after Instant* previously defined and the new, *before Instant* – the same that marks the beginning of the new *TimeSlice*.

Finally, since no new individuals of *VirtualSensor* have been found, no modifications to its *TimeSlices* have been produced, and the same happens for the *Sensor* class, since all its individuals maintain the same structure.

Table 5 – Evolutionary Actions triggered by the second set of instances (timeframe 2)

EVOLUTIONARY ACTION	VALUES
Composite 1: 1 Evolutionary Action	
Add DatatypeProperty	hasRunningAverage
Composite 2: 5 Evolutionary Actions	
Add TimeSlice	TS__Event__2
Add Cardinality Restriction	hasRunningAverage exactly 1 rdfs:Literal
Add SomeValuesFrom Restriction	TS__Event__2 isTimeSliceOf Event
Add HasValue Restriction	after "2021-09-09T03:01:00.000Z"
Add EquivalentClass	Event AND (after "2021-09-10T03:01:00.000Z")
Composite 4: 1 Evolutionary Action	
Change EquivalentClass (of TS_Event_1)	Event AND (after "2021-09-09T12:15:00.000Z") AND (before "2021-09-10T03:01:00.000Z")
TOTAL EVOLUTIONARY ACTIONS	**7**

Finally, Table 6 describes the Evolutionary Actions triggered by the last set of individuals, which include both new features. The most desired, and potentially more efficient scenario would be editing the existing definition of the first *TimeSlice* of *Event* to ensure it covers the two different intervals – this is, as the definition of *Event* is effectively the same now as it was when the first *TimeSlice* was created. However, *TICO* is unable to identify this and generates a third *TimeSlice* for *Event*, whose definition is similar to the first one but with a different starting date instead. It also edits the second *TimeSlice* to have a similar ending date, which may not be ideal, as it assumes there is no overlap between the two *TimeSlices* and no two different definitions of *Event* should exist simultaneously.

Table 6 – Evolutionary Actions triggered by the third set of instances (timeframe 3)

EVOLUTIONARY ACTION	VALUES
Composite 1: 6 Evolutionary Actions	
Add TimeSlice	TS__Event__3
Add SomeValuesFrom Restriction	hasSensorCorrelation exactly 1 Thing
Add AllValuesFrom Restriction	hasSensorCorrelation only Sensor
Add SomeValuesFrom Restriction	TS__Event__3 isTimeSliceOf Event
Add HasValue Restriction	after "2021-09-10T08:14:59.000Z"
Add EquivalentClass	Event AND (after "2021-09-10T08:14:59.000Z")
Composite 2: 1 Evolutionary Action	
Change EquivalentClass (of TS_Event_2)	Event AND (after "2021-09-09T12:15:00.000Z") AND (before "2021-09-10T08:14:59.000Z")
TOTAL EVOLUTIONARY ACTIONS	**7**

At the end of the last timeframe, the ontology has three *TimeSlices* that represent conceptualizations of the *Event* concept at different times: the *Event* concept was first modified from its original definition to include *hasSensorCorrelation* as part of its necessary and sufficient conditions during the first period, then having a new definition that sees different conditions (with the *runningAverage* role), and finally returning to a state in which *hasSensorCorrelation* was implicated again. The *VirtualEvent* concept is also discovered and associated with two *derivedFrom* roles, with no known ending date.

3.4 Discussion

J2OIM's goal is the creation of instances required for temporal representation, events, and intervals. About a million new instances were created to temporally represent sensor readings, occurrences, and manufacturing operations; which is likely to impact future reasoning processes that may need to be applied, and future tests will be able to determine the full impact of the chosen approaches. On the other hand, it is important to note that *J2OIM* ensures *Instants* are never duplicated, and can aggregate continuous and identical sensor readings into the same interval.

TICO makes use of these temporal representations to correctly identify new classes and properties in incoming instances and analyse their structures to define potential necessary conditions for classes during a specific timeframe. As such, whenever experiments are made that modify the data – either by extending existing features or including new ones, which is often a requirement when analysing and experimenting for PdM – the structure of the ontology is modified to accommodate them. This allows a user not only to visualize the modifications but to use a reasoner to automatically classify instances according to the concept versions that were valid at the time of the experiment. *TICO* achieves this by supplying a number of Evolutionary Actions with specific triggers, which range from the creation of classes to properties, time slices and restrictions.

While *TICO* is thus far able to infer Evolutionary Actions by comparing the ontology individuals to known descriptions of concepts, it does show some limitations that should be addressed in future work. First of all, as is evident from the results of timeframe 3, the framework is unable to identify that two versions of the same class may occur simultaneously. This is a direct result of including roles employed by individuals of a class into its necessary conditions – which may not always be a good solution, and may even be overly restrictive. Efforts should also be made to ensure that no excess of *TimeSlices* are created, especially if existing *TimeSlices* can be edited to include more than one interval, as the same set of restrictions may occur more than once. The reification approaches applied in this work already amplify the number of individuals and concepts, and this should not be exaggerated, both for computational and human readability reasons.

The interesting case of the *VirtualEvent* concept, which is not on itself reified to have starting and ending dates but to link to an *Event* that does also shows a

potential improvement point for *TICO*, in which could analyse not just the direct descendants of a concept for an *Instant* to use as pivot, but also check the concepts the initial one relates to, and decide which *Instant* to use based on those.

When it comes to deciding which class expressions should be added to a *TimeSlice*, more attention needs to be given by the possibility that not all of them are equally relevant, and some constructors (e.g. $AllValuesFrom$) could benefit from an analysis that considers each individual in the timeframe a confirmation or a counterexample in order to decide if the constructor should be added to the class definition.

3.5 Conclusions

J2OIM allows for the parameterizable transformation of JSON instances into different ontological instances, which can be either temporally dynamic or static. This chapter follows how this tool is used to transform the data acquired through different webservice endpoints and how the generated ontology individuals are used to populate ontologies and triple-store repositories. While the reification approach chosen for temporal representation generates additional ontology individuals that can affect the efficiency of reasoning processes, the aggregation of similar captures into intervals and the removal of duplicate entries such as instants with the same timestamp mitigates said effects.

Regarding the research question that motivated this chapter, **RQ1:** *To which extent is it possible to identify changes in classes and properties used by individuals provided via stream and reflect them in an ontology that represents those individuals during a specific period?*, *TICO*, as an ontology evolution tool, uses ontology individuals and analyses their features – namely their restrictions and properties – to establish whether the original definitions in the ontology need to be updated. To represent the changes in concepts over time, it creates new concepts when needed and *TimeSlices* with clearly defined beginning and ending instants for each different definition. By using ordered 4D-Fluents for the description of the time information associated with the ontology individuals, reasoners can use the *TimeSlices* to guarantee that the correct version of each concept is used to classify individuals.

All changes considered thus far have been addictive only – a new version of a concept is added to the ontology as enough changes are detected to trigger it – meaning there is no need to make subtractions to the ontology: it is possibly to

merely declare the concept has reached the end of the period in which it is valid. However, future work will involve a deeper and more systematic approach to restriction creation and the identification of potential inconsistencies that may arise from that process and, as such, subtraction methods may become a necessary feature of *TICO* for inconsistency resolution.

In this initial form, *TICO* merely performs a counting of how many property uses or property-value pairs are employed by individuals of a class to determine which class expressions should be added to new *TimeSlices* of that class. This is done exclusively to assess the proper creation of *TimeSlices* and their ability to represent the changes in class expressions. To properly address **RQ2.2:** *How can we identify new class expressions?*, subsequent chapters will further explore how to identify class expressions in a more nuanced way – for instance, by taking in consideration both positive and negative evidence for each class expression.

The topic of automatic identification of subclasses (and by extension, superclasses) – which *TICO's* current form is unable to assess – is yet another interesting case that must be studied with future experiments and improvements to the tool. Additionally, for the purposes of the work described here, not much attention was given to how to provide a time dimension to properties. Therefore, the future expansion of the tool will also include more work on the identification of domains and ranges of properties, and the axioms related to them (e.g. symmetry, transitivity, etc.).

As such, the two following chapters will focus on expanding *TICO* to:

- Include the identification of potential property attributes,
- Identify potential domain and range of properties,
- Identify the need for new classes that are not named in the data, but could describe groups of individuals, and
- Establish subsumption relationships between classes.

While some restriction types are created with this version of *TICO*, the following chapters will detail the particulars of how property axioms can be identified in stream data so that they can be added to the ontology, and how to use formal concept analysis to derive implications that can be used to extend and modify existing classes.

CHAPTER 4

Property Axiom Learning

4. Property Axiom Learning

In this chapter, *TICO* is extended to identify changes in the axioms associated with the properties of an ontology. It achieves this by adapting the possibilistic approach to axiom scoring to the context of RDF data streams for ontology evolution, a scenario which forcefully deals with incomplete and time-dependent data. Possibilistic axiom scoring is used in two distinct scenarios: (1) with previously known property axioms, allowing for the exploration of the effectiveness of the approach in a scenario in which no incorrect data was present; and (2) in an evolving knowledge scenario, in which neither the properties nor the property axioms were known and the dataset was obtained from publicly available sources, possibly simultaneously incomplete and with errors. Results show the effectiveness of the approach in accepting/rejecting axioms for the ontology's properties. The different approaches to possibility and necessity proposed in literature are recontextualized in terms of their bias towards selective confirmations or counterexamples – showing that some axioms benefit from a more lenient approach, while others present a lower risk of introducing inconsistencies by having harsher acceptance conditions.

This chapter focuses on how *TICO* was expanded from its original design to identify ontology property constructors. This is done by analysing extensional evidence for and against each of the ontology roles' constructors and ascertain if the data shows enough support for their inclusion in newer versions of the ontology. The work presented in this chapter aims to assess the degree to which it is possible to identify changes in OWL property axioms through the analysis of incomplete, unbounded and changing RDF data provided by streams, and to establish which metrics and assumptions/constraints are more suitable for this task. To do so, axiom testing analysis from both statistical and possibilistic perspectives will be executed. Additionally, when applying the solution to an ontology evolution scenario, this work aims to assess if and how the knowledge already present in the ontology should affect the decision to include/exclude suggested axioms.

This chapter aims to answer **RQ2.1:** *How can we identify changes in property axioms?*, by focusing on two main topics:

1. Adaptation of the possibilistic approach to axiom testing as described in the works of Tettamanzi et al [46] to the context of streams of RDF individuals/instances, followed by an extensive, in-depth analysis of the robustness of the proposed metrics. This includes the analysis of the effects of the number and variety of the individuals that can be analysed simultaneously when searching for potential axioms in RDF data streams.
2. Combining Acceptance/Rejection Index with other metrics in order to accommodate for the potential existence of errors in data: the percentage of selective confirmations w.r.t the support and an evolving form of ARI which is informed by previous versions of the ontology. For this purpose, the approaches will be tested against an ontology generated from publicly available data, for which no *a priori* information about property characteristics is known.

4.1 Background

Changes to domain knowledge must first be identified and quantified, so that they can be materialized into executable actions that will modify an ontology into a new version of itself – i.e., evolutionary actions [42,47]. To identify if there is a need for a change at the ontology level – if and which evolutionary actions should be considered – it is often necessary to analyse data that is external to the ontology [21,48]. Examples of this include identifying changes in the way end users apply the concepts in the ontology to the data [42], evaluating corpora regarding the domain to extract new concepts and compare them to existing ones [21,48,49] (frequently through the application of Natural Language Processing algorithms) and the analysis of datasets of structured data [42,47] (such as RDF datasets). It is possible to conceive ontology evolution as ontology learning with extra steps: new concepts and roles must be derived from existing data, but they also must be compared and made consistent with previously established ones, or otherwise update their definitions.

The identification and execution of evolutionary actions when they concern the addition of classes and properties – and the hierarchies between them – are relatively well documented in literature [16,42,50]. The identification of the axioms that could enrich them (e.g. class expression restrictions and property axioms), however, is not as popular – potentially stemming from the fact that many ontologies are often used as taxonomies and lack axiomatic expressivity [50]. This axiomatic complexity, however, is particularly relevant for lower-level ontologies, which need to describe the intricacies of their domains with varying degrees of detail and can be used to determine the consistency of data they describe, and to derive implicit information using reasoners [51]. For that, it is necessary not only to identify changes in concepts and roles and their relative novelty, but also to enrich them with axioms: ontology evolution is the more useful the more it applies the precepts of ontology learning. Furthermore, [16,50] detail the relative popularity of different types of evolutionary actions as they are described in literature, noting that while identifying new properties and property hierarchies is often considered, identifying and changing their property axioms in particular is still a relatively understudied field.

4.1.1 Ontology Learning

Ontology learning can be defined as the processes and techniques applied to design ontologies either automatically or semi-automatically [15]. According to [52], said techniques can be classified into two main categories: (1) linguistic-based approaches and (2) machine learning-based approaches, which can be further divided into statistics or logic-based.

Linguistic-based approaches focus on the analysis of large corpora of text to identify potential concepts and the relationships between them [52–54]. To do so, they seek for patterns and syntactic information in the text – making them particularly language-dependent. Machine learning-based approaches, on the other hand, can use different types of input data for their training: both structured and unstructured. Statistics-based approaches are usually applied to identify the co-occurrence of terms, association rules and hierarchies [50,52]. Logic-based approaches, on which the work described in this chapter is grounded, use logical inference or inductive logic programming to derive rules from positive and negative examples found in structured datasets. This approach is particularly suited for learning rules and formalizing axioms. On the Ontology Learning Layer Cake [55], which is used to describe the layers of the learning

process and, by extension, the possible tasks it encompasses, rule and axiom extraction is depicted as the final sub-task in the process and the least explored in literature [52].

4.1.2 Axiom Testing

An axiom is a statement that expresses what is true in the domain described through the ontology. The number and type of axioms directly affect the expressivity of the ontology by adding more information to the description of classes, properties, and assertions, among others, and the relationships between them. For example, an ontology that can specify if a certain property is a mandatory element of a class, or how many times that role can be applied to an individual, is more expressive (and potentially more complete) than one in which those assertions are not established [15].

Axiom testing is the process of evaluating the credibility of a given hypothesis concerning the relationships in a domain – the property axiom – by assessing whether the individuals of said domain (e.g. facts of an RDF dataset) confirm or deny a hypothesis [46] – i.e., whether they are confirmations or counterexamples of the axiom. The selective confirmation [56] principle can be put into effect as well: a fact selectively confirms a hypothesis when not only it favours that hypothesis but also fails to confirm its negation. Not all facts are equally relevant for axiom testing, and, as such, the number of examples needs to be considered not in the context of all analysed instances, but only of those that do entail the relationships under scrutiny [46].

Property characteristics, from the perspective of axiom suggestion for ontology enrichment purposes, have been described in [57]. This work describes enrichment methods that have been implemented as part of the DL-Learner framework [58]. These methods use different queries to look specifically for axiom support in a triple store, considering both the count of examples and the average of the 95% confidence interval when suggesting axioms to the user. However, while the approach tackles the discovery and materialization of ontology axioms, including property axioms, it depends on (large) knowledge bases containing a complete collection of samples that can be analysed as a whole – and by analysing only these samples to generate possibilities it is, in a way, working under a closed world assumption. Similarly, [59] describes how to

identify and materialize changes in property axioms but requires large and contained datasets to do so, falling under the same assumption.

4.1.3 The Possibilistic approach to Axiom Scoring

The possibilistic approach has been applied in the works of Tettamanzi et al. [46,60–63] for axiom testing against RDF facts in ontology learning and validation/evaluation scenarios. As the name indicates, the possibilistic approach deals with the degree of possibility of an event, such as an axiom – which falls in a range between impossible and possible [0,1]. Possible, unlike probable (from probability distributions), does not mean that the axiom must be true: only that it is compatible with the known state of the world. While probability theory is suited for the representation of random and observed phenomena, the possibilistic theory better reflects how to deal with incomplete knowledge. In [63] (and, by extension, [46]), the authors detail how using such an approach is more suitable for candidate axiom scoring than statistical/frequentist approaches, claiming that the nature of the inductions necessary for ontology learning and evolution makes it such that statistical analysis ends up with largely arbitrary and unobjective results. Through the analysis of the results obtained in section 4.3, the same conclusion was reached, and the possibilistic approach will be used to complement the statistical analysis. Being an unsupervised approach, the possibilistic approach is particularly suited for application to RDF streams and can accommodate for the incomplete nature of the data they provide. However, to the best of the author's knowledge, the possibilistic approach has only been applied for ontology validation, and its applicability to identify new axioms from data analysis is either understudied or non-existent.

4.2 Learning and Evolving Property Axioms

The application scenario of the *TICO* framework involves the existence of several streams of sensor data arriving in near real-time which are subject to transformations by data scientists, introducing new and unpredicted changes that are not in the original ontology. As such, not only new properties may be introduced, but the way they are used may change – e.g., a property that was previously considered functional may now be used in a way that is incompatible with that axiom. The following sections will describe the ways in which *TICO*

was adapted and extended to identify changes in property axioms within a timeframe, and the structures involved in that process.

Consider, for example, a stream $\mathcal{S}$ of individuals describing familial relationships between people as seen in Figure 13. The individuals are separated into three consequent groups, which correspond to different periods of analysis, at the end of which the ontology that represents these individuals may be updated. When analysing the first group, one can conclude, for example, that the properties *hasMother* and *hasFather* should de functional, and the property *hasBrother* is both symmetrical and transitive. However, upon analysing the second group of individuals, this assessment must be revised: the property *hasBrother* is still symmetrical, but no longer transitive, i.e., the brother of my brother is not necessarily my brother as well.

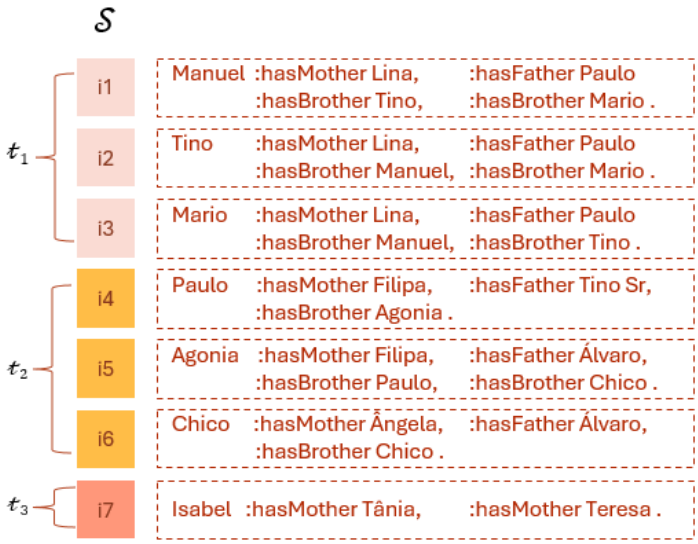

Figure 13 – Familial relationships between individuals in a stream

Now consider that, as society grows more open to different familial structures, individuals in the stream may now in fact have two legal parents of the same gender (represented by the third group in Figure 13); in order to accurately reflect reality, the application of the property *hasMother* must now accommodate for this necessity, and this change in how the property is used means its functionality is obsolete and should be reconsidered: the ontology should evolve to accommodate for the change in how the property is effectively used.

This chapter focuses on the changes introduced to *TICO* that go beyond the detection of new properties and into recognizing the property axioms that should be added with them. Unlike other logic-based approaches, *TICO* does so by

analysing positive and negative evidence regarding a set of known and predetermined rules – the property axioms – to ascertain if they should be added to the ontology. To better understand the property axiom testing process on which this chapter focuses, consider Figure 14 and the descriptions that follow.

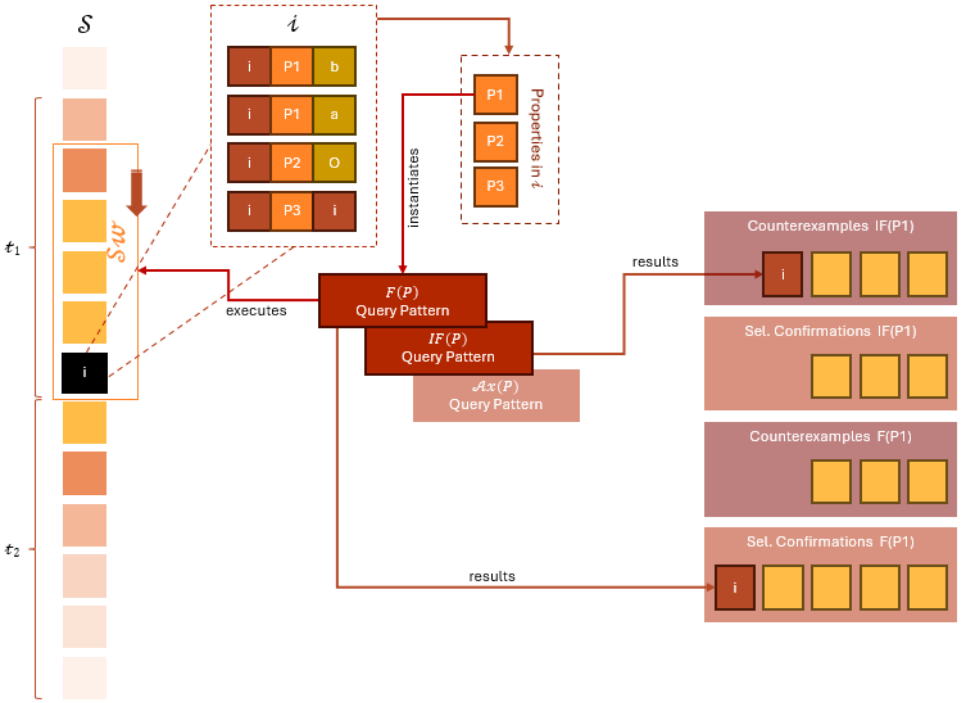

Figure 14 – Data structures concerning the property axiom testing process

The analysis is triggered by the arrival of a new individual on the stream. The visualization provided by Figure 14 shows:

- The timeframe t, the period in which the stream S is analysed and from which to draw conclusions.
- Sw, the sliding window upon which queries are executed.
- The named individual i, which is composed by a set of RDF triples with the same Subject.
- The use of i and its properties to instantiate query patterns for specific $Ax(P)$.
- The query results being used to classify i as selective confirmation or counterexample for each $Ax(P)$.

These concepts will be further described in the remainder of this section and the definitions presented next.

Sliding Window

With the exception of Irreflexivity, all property axioms pertain to the relation between one individual and individuals other than itself: it is therefore almost always necessary to have access to more than one individual to test the potential property axioms. Since individuals arriving on a stream are otherwise lost after arrival and keeping a permanent copy of each is not sustainable nor desirable, they are stored in a temporary structure upon which the analysis is executed – in a sliding window $\mathcal{Sw}$ of fixed length. In a first-in-first-out fashion, every time $\mathcal{S}$ delivers a new individual, it is added to $\mathcal{Sw}$, until its maximum size is reached – i.e., the size of the sliding window is measured by the total number of individuals it can store, regardless of how many triples compose them. Afterwards, for each new arrival, the oldest individual is removed from $\mathcal{Sw}$ and forgotten.

Property Constructor Axiom

Property axioms $\mathcal{Ax}$ – or *characteristics* [57] – can be used to describe how the property must be employed. There are seven property axioms in OWL, namely: Functionality, Inverse Functionality, Transitivity, Reflexivity, Irreflexivity, Symmetry and Asymmetry. The definition of Reflexivity, however, is too strong to evaluate properly considering the constraints of this solution – requiring the analysis of all individuals at the Class level, which is not compatible with the property-oriented approach – and it is not in the scope of this thesis. Focusing exclusively on the application of property axioms in Object Properties, $\mathcal{Ax}$ denotes one of the possible OWL property axioms among:

FunctionalObjectProperty (F), InverseFunctionalObjectProperty (IF),
TransitiveObjectProperty (T), SymmetricObjectProperty (S),
AsymmetricObjectProperty (AS), IrreflexiveObjectProperty (IR)

and therefore:

$$\mathcal{Ax} \in \{ F, IF, T, IR, S, AS \}$$

All property axioms for a given property P can be expressed as first order logic implications in the form:

$$\mathcal{Ax}(P): \forall i, \forall x_1, \dots, \forall x_n$$
$$B(P, i, x_1, \dots, x_n) \rightarrow H(P, i, x_1, \dots, x_n)$$

where i is an individual in the domain of a property P, and $x_1, \dots, x_n$ are other individuals that may be required to evaluate the relationship between the body B and the head H. For example, the functionality of P could be written as such an implication, in which the B shows the simultaneous application of the same property more than once for the same individual – $B(P, i, x_1, x_2) : P(i, x_1) \wedge P(i, x_2)$ – and the H is the resulting implication that the ranges of said property must therefore be the same – $H(P, i, x_1, x_2) : x_1 = x_2$.

Confirmation, Selective Confirmation and Counterexample

In testing a property axiom hypothesis $\mathcal{A}x(P)$, we assume the original ontology is consistent and does not neither entail $\mathcal{A}x(P)$ nor its negation. Contrary to Tettamanzi's approach [46], substitutions occur not at the triple level, but at the individual level: a named individual i counts as either one confirmation or one counterexample, regardless of how many triples compose it.

For a named individual a_0 and a property P, substitutions $i/a_0, x_1/a_1, \dots, x_n/a_n$ for $B(P, i, x_1, \dots, x_n)$ and $H(P, i, x_1, \dots, x_n)$ may be found, such that:

- If there's at least one substitution for B and the negation of H under the UNA: the named individual a_0 is a **counterexample**;
- There is a substitution for both B and H, and no substitution for the negation of H: a_0 **selectively confirms** the hypothesis;
- There is a substitution for B, but no substitution for H nor for $\neg H$: a_0 does not contain enough information to selectively confirm nor deny the hypothesis, and therefore is a **weak counterexample**;
- If there are no substitutions for B, a_0 is a confirmation, but not a selective one, and thus not considered and it is ignored.

This distinction between confirmation and selective confirmation follows from the selective confirmation principle [56]: a ***selective confirmation*** is a confirmation in which i is in the domain of P, i.e., selective confirmations are a subset of all possible confirmations. Considering both UNA and the fact that only individuals in the domain of the property are counted, functionality, inverse functionality, irreflexivity and asymmetry cannot generate weak counterexamples – the imposed constraints guarantee that an individual is always classified as either a selective confirmation or counterexample – there are always substitutions for either H or $\neg H$. The unknown data brought upon by the constraints of stream and the sliding window does not affect the classification of

individuals, as they are assumed to arrive complete – furthermore, counterexamples rely on more individuals being accessible in $\mathcal{Sw}$, meaning there is no uncertainty regarding their correct classification.

On the other hand, the openness of the world and unknown facts about individuals in the range of the properties greatly affect the classification process for the transitivity and symmetry axioms – consider, for instance, that the individuals in the range of the properties may not be present in the sliding window when the query is executed, and therefore it is not possible to assess if that individual is also in the domain of the property (necessary for both axioms). Selective confirmations are indisputable, as their mere existence implies that the data was, in fact, complete. Counterexamples, on the other hand, imply proving a negative, making them particularly harder to identify under the constraints and assumptions of the methodology employed. As such, any substitutions that do not selectively confirm these axioms are treated as *weak* counterexamples.

Constructor Query Pattern

To classify i w.r.t. $\mathcal{Ax}(P)$ as a confirmation or counterexample, it is first necessary to ascertain if P is used by i. If that is the case, i is then used to instantiate a specific constructor query pattern – a SPARQL query which enforces the pattern associated with $\mathcal{Ax}(P)$. This query will enforce the substitutions mentioned previously, meaning reasoners are not employed.

Code snippet 3 shows one possible query (for symmetry), in which `iURI` and `pURI` are known and correspond, respectively, to i and P. Each i is used to instantiate the query patterns only once, although it may influence the results of queries coming afterwards (as Object). When the query is executed over $\mathcal{Sw}$, any results it obtains classify i as a selective confirmation of $\mathcal{Ax}(P)$ (i.e. both the triple and its mirror are present). In this case, as it is symmetry, if the query does not obtain at least one result, then i is considered a *weak* counterexample of $\mathcal{Ax}(P)$.

Code snippet 3 – Query representing Symmetry

```
SELECT * WHERE {
{
  SELECT (COUNT(?obj) as ?B) WHERE {  <iURI> <pURI> ?obj . }
}
{
  SELECT (COUNT(?obj) as ?H) WHERE
    { <iURI> <pURI> ?obj .  ?obj <pURI> <iURI> . }
}
FILTER (?B = ?H) }
```

4.2.1 Property Axiom Analysis

The analysis of real-time data can unravel patterns regarding a property's usage that may hint at its formal definition. Each property axiom, per virtue of its definition, corresponds to a different pattern in the data (cf. the end of this section). Consider the generic task *computeAxiom*, which aims to gather statistics about property axioms and is summarized in Algorithm 1. E^+ and E^- reflect, respectively, how many individuals are categorized as selective confirmations and counterexamples of $\mathcal{A}x(P)$ in $\mathcal{S}$ during t. Support (E^0) is provided by the sum of $E^+$and E^- and reflects the total number of individuals in the domain of P.

Algorithm 1 – computeAxiom

```
Input: 𝒮, 𝒮𝓌, 𝒜x, P
Output: E+, E−
for each i in 𝒮 do
  if 𝒮𝓌 is full then
   dequeue 𝒮𝓌
  queue i to 𝒮𝓌;

  dP ← get distinct properties used by i;

  if P in dP then
    𝒬𝒜x ← get query pattern for 𝒜x;
    instantiate 𝒬𝒜x using i and P;

    if 𝒬𝒜x has results in 𝒮𝓌 then
      increase E+;
    else
      increase E−;
```

Table 7 – Example individuals in a $\mathcal{S}w$ and respective properties

ID	HASNAME	HAS EVOLUTION GROUP	HASTYPE	HATCHES
i_1	Onix	"Onix & Steelix Line"	Rock, Ground	Mineral EG
i_2	Ditto		Normal	Ditto EG
i_3	Caterpie	"Caterpie Line"	Bug	Bug EG
i_4	Metapod	"Caterpie Line"	Bug	Bug EG
i_5	Tropius		Flying, Grass	Grass EG, Mineral EG
i_6	Bug EG			Caterpie, Metapod
i_7	Grass EG			Tropius, Maractus
i_8	Maractus		Grass	Grass EG

Consider a $\mathcal{Sw}$ with a size of 8, whose current individuals are described in Table 7. Individuals are complete but may contain references to individuals that are not in the $\mathcal{Sw}$ at the time $computeAxiom$ (Algorithm 1) is executed.

Regarding the potential Inverse Functionality of the *hasEvolutionGroup* property, i.e. $IF(hasEvolutionGroup)$, the individuals are classified as follows:

- Samples: i_1, i_3 and i_4, as they use the property being analysed. $E^0 = 3$.
- Selective confirmations: i_1, as the domain of the property is only used once per value. $E^+ = 1$.
- Strong counterexamples: i_3 and i_4, as they have the same property value for different individuals. $E^- = 2$.

Individuals i_2, i_5, i_6, i_7 and i_8 do not use the property and are not used in the analysis of $IF(hasEvolutionGroup)$. They may be used in the analysis of axioms for different properties.

Consider the potential symmetry of the $hatches$ property. For $S(hatches)$:

- Samples: i_1, i_2, i_3, i_4, i_5, i_6, i_7 and i_8. $E^0 = 8$.
- Selective confirmations: i_7, i_8, as both $hatches(i_7, i_8)$ and $hatches(i_8, i_7)$ are present. $E^+ = 2$.
- Weak counterexamples: i_1, i_2, i_3, i_4, i_5 and i_6 as they use the property in relation to other individuals, but whether those individuals also use it in relation to them is not made explicit in the data. $E^- = 6$.

The implications – i.e., the pattern to look for in the data – formulas, definitions of selective confirmations and counterexamples and the assumptions considered for the identification of each property axiom are described next.

Functionality

Under Functionality, P can have one and only one value per i. P may be employed more than once by i, provided the ranges are the same individual.

- **Formula**: $\forall\, i, y_1, y_2 : P(i, y_1) \wedge P(i, y_2) \rightarrow y_1 = y_2$
- **Selective Confirmation** ($\boldsymbol{E^+}$): i for which P has only one value.
- **Counterexample** ($\boldsymbol{E^-}$): i for which P is used more than once and the ranges of P are distinct. Counterexamples of $F(P)$ are **strong** counterexamples.
- **Assumptions**: Partial closure of the world through UNA allows for the existence of strong counterexamples.

Inverse Functionality

Under Inverse Functionality, P cannot have the same entity on its range for more than one i (e.g. a unique ID cannot be shared).

- **Formula**: $\forall\, i_1, i_2, y: P(i_1, y) \wedge P(i_2, y) \rightarrow\ i_1 = i_2$
- **Selective Confirmation** ($\boldsymbol{E^+}$): i for which the value of y has not been the range of P in any previously analysed individual in $\mathcal{Sw}$.
- **Counterexample** ($\boldsymbol{E^-}$): i for which the value of y was found in the range of P of other previously seen individuals. Counterexamples of $IF(P)$ are **strong** counterexamples.
- **Assumptions**: Partial closure of the world through UNA allows for the existence of strong counterexamples.

Transitivity

With Transitivity, if P holds between any two individuals in a sequence, it holds between all individuals of that sequence.

- **Formula**: $\forall\ i, y, \mathrm{z} : P(i, y) \wedge\ P(y, z) \rightarrow P(i, z)$
- **Selective Confirmation** ($\boldsymbol{E^+}$): i for which P has propagated between any set of three individuals i, y, z, for which a $P(i, z)$ exists.
- **Counterexample** ($\boldsymbol{E^-}$): i for which P does not propagate between any set of three individuals i, y, z. Non-confirmations of $T(P)$ are considered **weak** counterexamples, as a strong counterexample would have to prove $\neg P(i, z)$.
- **Assumptions**: CWA allows for the elevation of non-confirmations to weak counterexamples.

Irreflexivity

Irreflexivity is the characteristic of a property that cannot relate i with itself: the domain and range of the P must not be the same individual.

- **Formula**: $\forall\ i : \neg P(i, i)$ or $\forall\ i_1, i_2 : P(i_1, i_2) \rightarrow\ i_1 \neq\ i_2$
- **Selective Confirmation** ($\boldsymbol{E^+}$): i in which P is used to relate with individuals other than itself.
- **Counterexample** ($\boldsymbol{E^-}$): i in which P is used to relate with itself. Counterexamples of $IR(P)$ are strong counterexamples.
- **Assumptions**: Partial closure of the world through UNA allows for the existence of strong counterexamples.

Symmetry

A property is Symmetric when it is its own inverse: meaning that if $P(a, b)$, then $P(b, a)$ must also be true.

- **Formula**: $\forall\ i, y : P(i, y) \rightarrow P(y, i)$
- **Selective Confirmation** ($\boldsymbol{E^+}$): i in which the range of P relates back to the individual through P.
- **Counterexample** ($\boldsymbol{E^-}$): i for which the range of P does not relate back to the individual through P. Non-confirmations of $T(P)$ are considered **weak** counterexamples, as a strong counterexample would have to prove $\neg P(y, \mathrm{i})$.
- **Assumptions**: CWA allows for the elevation of non-confirmations to weak counterexamples.

Asymmetry

A property is Asymmetric when it cannot be its own inverse: if $P(a, b)$, then $P(b, a)$ cannot hold true.

- **Formula**: $\forall\ i, y : P(i, y) \rightarrow \neg P(y, i)$
- **Selective Confirmation** ($\boldsymbol{E^+}$): i in which the range of P does not relate back to the individual through P.
- **Counterexample** ($\boldsymbol{E^-}$): i for which the range of P relates back to the individual through P. Counterexamples of $AS(P)$ are **strong** counterexamples.
- **Assumptions**: Partial closure of the world through UNA allows for the existence of strong counterexamples.

4.2.2 The Possibilistic Approach to Axiom Scoring

The work described in [46] also introduces the concept of an acceptance/rejection index (ARI) and its application to axiom scoring: positive values suggesting acceptance, negatives suggesting rejection and values close to zero indicating ignorance. For any $Ax(P)$, it is possible to calculate:

- the necessity N: the degree to which the axiom is corroborated by the data while not being contradicted by it; and
- the possibility Π: the degree to which it is not contradicted by the data.

ARI can thus be calculated using [63]:

$$ARI(Ax(P)) = N(Ax(P)) + \Pi(Ax(P)) - 1$$

N and Π are calculated differently depending on the approach to the data being used. The original reasoning behind them can be found in [63]. For the purposes of this thesis, the computation of N and Π depends on the relative weight given to selective confirmations and counterexamples.

If counterexamples are strong, a single counterexample is sufficient to quash N, regardless of how many selective confirmations are found. Ergo, counterexamples have more weight in the decision-making process than selective confirmations – when *strong* counterexamples are available, the strong form of N and Π is applied, as follows [63]:

$$N\big(Ax(P)\big) = \begin{cases} 1, if\ \mathrm{E}^{-}_{\mathrm{Ax(P)}} = 0 \\ 0, if\ E^{-}_{Ax(P)} > 0 \end{cases}$$

and

$$\Pi(Ax(P)) = 1 - \sqrt{1 - \left(\frac{\mathrm{E}^{+}_{\mathrm{Ax(P)}}}{\mathrm{E}^{0}_{\mathrm{Ax(P)}}}\right)^2}$$

in which $E^{0}_{Ax(P)}$ is the total number of individuals in the domain of P, corresponding to the sum of $E^{+}_{Ax(P)}$ – the number of selective confirmations, – and $E^{-}_{Ax(P)}$ – the number of counterexamples.

In the cases in which counterexamples are not sufficient to exclude a hypothesis – being *weak* counterexamples – the possibility of an axiom being true is directly related to the existence of selective confirmations. N and Π are calculated according to [63]:

$$N(Ax(P)) = \sqrt{1 - \left(\frac{\mathrm{E}^{-}_{Ax(P)}}{\mathrm{E}^{0}_{\mathrm{Ax(P)}}}\right)^2}$$

and

$$\Pi(Ax(P)) = \begin{cases} 0, if\ \mathrm{E}^{+}_{\mathrm{Ax(P)}} = 0 \\ 1, if\ E^{+}_{Ax(P)} > 0 \end{cases}$$

This second set of formulas ensures the decision is influenced more by the selective confirmations – with Π being at its highest whenever selective confirmations are present. N cannot be set to 0, as counterexamples are *weak* and

the assumption that they are, in fact, actual counterexamples cannot be made under OWA. For the sake of simplicity, the first set of formulas are considered the stronger forms and the second the weaker forms, after the strength of their respective counterexamples.

Table 8 summarizes which forms of N and Π (ARI form for brevity) to use per each property axiom. These reflect not only the assumptions mentioned previously, but also the constraints of the approach itself (under UNA, weak confirmations become strong for functionality and inverse functionality, for example).

Table 8 – ARI form to apply for each property axiom

PROPERTY AXIOM	ARI FORM
Functionality	Strong
Inverse Functionality	Strong
Transitivity	Weak
Symmetry	Weak
Asymmetry	Strong
Irreflexivity	Strong

Returning to the example in Table 7, not all individuals in the stream are equally available to be analysed – e.g. there is mention of an individual *"Caterpie Line"* that is not among the individuals in $\mathcal{Sw}$. E.g., in the case of the *hatches* property, one can see that it relates several individuals in a symmetrical fashion: i_8 relates to i_7 using this property and vice-versa. These would, indeed, count as selective confirmations of $S(hatches)$ (and for $T(hatches)$). However, all individuals in the sliding window use the property, which results in the data in Table 9.

Table 9 – Axiom scoring for the *hatches* property using the strong forms of N and Π

$Ax(hatches)$	E^+	E^-	N	Π	ARI
F	6	2	0	0,34	-0,66
IF	5	3	0	0,22	-0,78
AS	3	5	0	0,07	-0,93
IR	8	0	1	1	1

By not having information about individuals such as *Mineral EG* and *Ditto EG*, it is not possible to have all the data needed for the remaining individuals to count as selective confirmations. In the cases of T and S, incomplete data may lead to false negatives, and selective confirmations are much harder to find as they require more individuals to be on the same sliding window. When using the weaker forms to compute ARI for T and S, the results reflect the potential for those axioms to be present (see Table 10).

Table 10 – Axiom scoring for the *hatches* property using the weak forms of N and Π

$Ax(hatches)$	E^+	E^-	N	Π	ARI
T	5	3	0,93	1	0,93
S	2	6	0,48	1	0,66

While ARI can be updated with each new individual on the stream, it only influences the ontology evolution process at the end of t. Algorithm 2 shows how the decision to include or exclude a property axiom from an ontology can be made, considering the existence of different thresholds for the weak and strong forms of ARI (wf_t and sf_t, respectively).

Algorithm 2 – ARI Decision and evolving ontology

```
Algorithm is executed at the end of t.
Input: ontology, set of <Ax(P),E+,E->, sf_t, wf_t
Ouput: ontology (evolved)
for each Ax(P) do
  if Ax one of {T, S} then
     compute N(Ax(P)) using weak form
     compute Π(Ax(P)) using weak form
     t ← wf_t
  else
     compute N(Ax(P)) using strong form
     compute Π(Ax(P)) using strong form
     t ← sf_t

  ARI(Ax(P)) ← N(Ax(P)) + Π(Ax(P)) − 1

  if ARI(Ax(P))  >= t then
     add Ax(P) to ontology
  else
     remove Ax(P) from ontology
```

4.2.3 Applying ARI to evolving ontologies

Considering the ontology evolution application scenario, in which a potential new version (or at least for some of its axioms) is created at the end of each t, any conclusions reached over the application of the queries must be made in the context of available knowledge – i.e., previous known versions of the axioms in the ontology. As such, we use a weighted average between the newly calculated ARI and the previous state of the axiom, which will be referred to as Evolving ARI (ARI_e).

$$ARI_e\big(Ax(P)\big) = \begin{cases} ARI\big(Ax(P)\big), & t = 0 \\ d\big(Ax(P)\big)_{t-1} * w_p + ARI\big(Ax(P)\big) * (1 - w_p), & t \geq 1 \end{cases}$$

in which:

1. ARI is the Acceptance/Rejection Index calculated during t;
2. $d\big(Ax(P)\big)_{t-1}$ is one of {0,1}, depending on whether the axiom in question was (or was not) missing from the previous known version of the ontology. 0 indicates the axiom was rejected and 1 indicates it was accepted (thus $accept = 1$ and $reject = 0$ in the equations below);
3. w_p is the relative weight of previous knowledge [0,1]. Lower weights should allow for more versatility in the evolutionary process, favouring new axioms, while higher ones value a more conservative approach.

A w_p of 0 would imply that previous knowledge does not affect the current decision-making, but a w_p of 1 would equally mean that no changes to the axioms in the ontology would ever be possible, regardless of how much evidence for it is found.

Both ARI and ARI_e are on the [-1,1] interval, with positive results suggesting acceptance of the axiom and the inverse for negative results. When it comes to decision-making, however, this may not be sufficient (maybe not all positive results are strong enough to force asserting an axiom, and it may be interesting to allow for errors in the data to exist).

For strong-form using properties, the decision d to accept or reject an axiom is informed by:

$$d\big(Ax(P)\big) = \begin{cases} accept, & E^+_{Ax(P)} > cf_t \\ accept, & E^+_{Ax(P)} \leq cf_t \ and \ ARI > sf_t \\ reject, & otherwise \end{cases}$$

$E^{+}_{Ax(P)}$ is the percentage of selective confirmations (w.r.t. the Support), cf_t the minimum percentage of selective confirmations required, and sf_t is the threshold to be applied for the strong form of ARI computation.

For weak-form using properties (i.e., T and S) the decision is informed exclusively by ARI and a minimum threshold wj_t. Since the existence of counterexamples does not have the same weight as for the strong form, the reasoning to include the percentage of support in the decision-making process does not apply. As such, axiom acceptance is informed by:

$$d\big(Ax(P)\big) = \begin{cases} accept, & ARI > wf_t \\ reject, & otherwise \end{cases}$$

As F and T axioms are incompatible, and F is computed on stricter terms (using the stronger-form), F is considered to take precedence over T in case of both being flagged for inclusion. As such, whenever the algorithm concludes that the same property could be both F and T, it will only classify it as F. Similarly, an axiom cannot be simultaneously T, S and IR. In these cases, precedence is given to T and S.

The following sections will illustrate the application the possibilistic approach to axiom scoring in two different scenarios:

1. ***Experiment I***: in this experiment, ARI will be compared to traditional information retrieval metrics to ascertain its applicability to axiom scoring in an RDF stream scenario. For this purpose, existing ontologies for which the property axioms are known will be used. This experiment will also assess the effects of different sliding window sizes in the proper classification of individuals as selective confirmations or counterexamples.
2. ***Experiment II***: experiments will be conducted using an ontology automatically generated from publicly available datasets to establish the applicability of the solution in an ontology evolution/learning scenario. Information regarding property axioms is not known *a priori*, and the suitability of the proposed axioms will be analysed by employing a reasoner and verifying if inconsistencies are introduced. This experiment will show the suitability of the solution in cases in which the data is both potentially incomplete and with errors. Furthermore, the individuals analysed will be segregated into different, pre-established timeframes, allowing for the application of Evolving ARI between them.

4.3 Experiment I: Effects of Sliding Window size and suitability of ARI for axiom scoring

To establish the strength of the devised solution, experiments were first conducted against a dataset in which certain property constructors were previously known. With the ontologies known and the datasets curated, there is no expectation of counterexamples to be found for the assertions present in the ontology.

For a better understanding and discussion of the results, they have been separated according to the following questions:

1. How many individuals should be included in the sliding window in order to effectively learn property axioms and how does the number of samples (i.e., individuals that use the property under scrutiny) in the stream influence this number? Furthermore, the following subquestion will be investigated:
    i. How does ARI compare with precision, recall and f-measure? How does it compare to the application of selective axiom confirmations exclusively?
2. How well do the weak forms of N and Π compensate for the difficulty in identifying selective confirmations under OWA?

As the data stream used in Chapter 3 was obtained from a real-life scenario and not sufficiently complex in terms of employing property axioms – or having them change over time – the experiments described below were carried out using the set of ontologies in Table 11, which were selected for their property axioms and population sizes.

Table 11 – Properties and their respective axioms by class

	TYPE	PROPERTY	*Ax*
CMT [64]	Paper (4301 individuals)	hasAuthor	F
		hasDecision	F
		readByMetaReviewer	F
		rejectedBy	F
	Person (4255 individuals)	addedBy	F
		addProgram	IF
		CommitteeMember	
		assignedByReviewer	F
		assignExternalReviewer	IF
		rejectPaper	IF
		writePaper	IF
		writeReview	IF
WINE [65]	Region (36 individuals)	adjacentRegion	S
	Region (as range)	locatedIn	T
PLANT [66,67]	Thing (82 individuals)	part_of/has_part	T

Each populated ontology was split into samples of 3000 random individuals, which are then sequentially subjected to the analysing queries. To simulate an RDF stream scenario, the individuals of the ontology are queued (and dequeued when necessary) into the sliding window one at a time. Queries are executed over the sliding window, guaranteeing they never have access to all existing individuals in the ontology simultaneously and therefore operate under the original restrictions of *TICO*. For performance reasons, only the properties that displayed the characteristics mentioned above were analysed: as it is impossible for a property to employ all property axioms simultaneously, they can serve as counterexamples for those they do not display.

For the purposes of this work, a modified version of the queries described in [57] are used as confirmation queries, in which the differences account for the streaming nature of the use case and the granularity of the search (here at the individual level and not at the triple level). Two different types of queries can be used for each property axiom: the confirmation queries (CQ), which ascertain if

a property axiom could be present, and the negation queries (NQ), which ascertain the opposite. Each class of query (CQ or NQ) provides its own confusion matrix, and the meaning of their solutions varies depending on whether the true class corresponds to the existence or non-existence of the axiom. This reasoning is illustrated in Table 12.

Table 12 – Confusion Matrix and respective query results

		HAS SOLUTION?	TARGET	
			AXIOM	¬AXIOM
APPLIED QUERY	CQ	Yes	True Positive (TP)	False Positive (FP)
		No	False Negative (FN)	True Negative (TN)
	NQ	Yes	False Positive (FP)	True Positive (TP)
		No	True Negative (TN)	False Negative (FN)

The confirmation query looks for positive cases of functionality: i.e., if a given individual is compatible with the axiom, either by having only one use of the property or the object being a duplicate. An individual that can be selected with this query is a selective confirmation of the functionality axiom. If the property is indeed functional – i.e., if the true class in Table 12 is AXIOM – then this result is a true positive. On the other hand, if the axiom was present but the query does not yield any results, it is a false negative. If the true class is ¬AXIOM – i.e., it is known that the functionality axiom is not present – and the query returns a result, it is a false positive. Following the same reasoning, an empty set here is the correct result for the individual and therefore a true negative.

The negation query, on the other hand, looks for explicit counterexamples: for an individual to result in a non-empty set when queried, it must definitively have at least two uses of the property, and their objects be distinct. If the true class in Table 12 is AXIOM and the negation query returns a non-empty set of solutions, it must forcefully be a false positive – there should not have been any solutions for the query, as functionality is present. If it returns an empty set, it is a correct identification and a true negative. In the same vein, the true class is ¬AXIOM – and the query has results, it is doing so correctly and, therefore, a true positive. If it returns none – claiming the axiom is present when it should not be – it accounts for a false negative.

Code snippet 4 and Code snippet 5 show one possible difference between confirmation and negation queries, using the functionality axiom as an example. Each query is executed for each individual being tested and generates a result set with a size of either 0 or 1 query solutions.

Code snippet 4 – Confirmation Query for Functionality

```
SELECT ?o1 WHERE
{
          <iURI> <pURI> ?o1.
   FILTER NOT EXISTS
          {
               <iURI> <pURI> ?o2. FILTER ( ?o1 = ?o2 )
          }
}
```

Code snippet 5 – Negation Query for Functionality

```
SELECT ?o1 WHERE
{
          <iURI> <pURI> ?o1. <iURI> <pURI> ?o2.
          FILTER ( ?o1 != ?o2 )
}
```

Traditional information retrieval statistics, e.g. precision, recall and f-measure are computed as follows:

$$precision = \frac{TP}{TP + FN} \quad recall = \frac{TP}{TP + FP}$$

$$f - measure = \frac{2 * precision * recall}{precision + recall}$$

To ascertain how "quickly" and effectively the queries can identify the possibility/probability of an axiom being present for a given individual, three different sliding windows sizes are analysed: 10, 50 and 100 individuals (corresponding, roughly, to 0.2%, 2.3% and 4.7% of all individuals in the sample, respectively). Theoretically, the more individuals in the sliding window that can be used to answer a query, the more precise the classification of each individual as a selective confirmation or counterexample should be. However, for performance reasons, it is interesting to see if reliable conclusions can be made from as little data as possible.

While all properties used by each class were studied, for the sake of brevity, we will focus on presenting the results for a property from a class with a high variance in the usage of its properties – i.e., not all individuals of the same type will use the same properties – and those for a class with lower variance, in which all individuals always use the same properties.

4.3.1 Effects of window size and relevance of individuals in axiom support

This section analyses the effects of both the window size and the influence of support in the sample. As previously explained, for an individual to constitute support, it must be the domain of the property being investigated. However, even if a stream is comprised only by individuals of the same type, there is no guarantee that all of them will employ the same properties. For the following experiments, two classes from the CMT ontology were selected – *Paper* and *Person* – as the first always employs all its properties and the second has more variety to it, with some properties being used in a very low fraction of its individuals. To ascertain the influence of the window size (w_s) of the $\mathcal{Sw}$, three different values (of 10, 50 and 100 individuals) are applied to each of the classes.

Considering that the dataset is clean, and no counterexamples can be found, the results in favour of an axiom are fairly evident regardless of the window size applied (perfect precision combined with necessity, possibility, and ARI of 1). However, it is important to investigate the more interesting cases in which counterexamples are indeed possible, by analysing the evolution of the metrics in the cases where an axiom is known not to be present.

The values presented in Table 13 are applied on the following experiments, assessing how the relevance of each individual for support affects the evolution of the metrics and the potential results.

Table 13 – Number of individuals on the stream to analyse, window sizes, and percentage of support in each sample for each of the studied properties

VARIABLE	VALUE
INDIVIDUALS ANALYSED	2150
w_s (SIZE OF $\mathcal{Sw}$)	10 / 50 / 100
PROPERTY	% of Support
READBYMETAREVIEWER	100%
REJECTEDBY	≈50%
ADDEDBY	≈25%

The *readByMetaReviewer* property does not feature any of the explored property axioms, but it is present in all individuals of the *Paper* class. The study of the evidence for and against the IF axiom for this property shows how the

proper identification of each individual as a selective confirmation or a counterexample is affected by the changes in window size – that as more individuals are analysed, it becomes apparent that the same property is used by more individuals with the same value – and the number of counterexamples starts to increase.

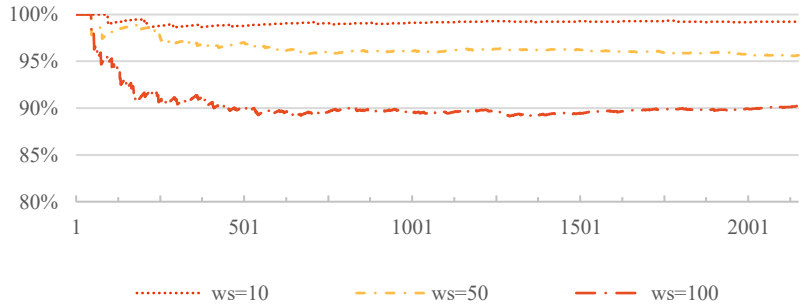

Figure 15 – Evolution of the selective confirmations of IF for the *readByMetaReviewer* property with w_s of 10, 50 and 100

Most of the changes occur when analysing the first 60 individuals out of the sample, as shown in Figure 15, with most individuals (around 99%) being incorrectly categorized as selective confirmations due to lack of information to the contrary. However, by increasing w_s to 50, the categorization improves significantly: the number of selective confirmations lowers to around 96%, and to 90% when w_s is increased to 100. With recall being 100% regardless of the window size chosen, Figure 16 compares the evolution of precision, showing similar improvements although with a high propensity for misclassification (stabilizing around 10%).

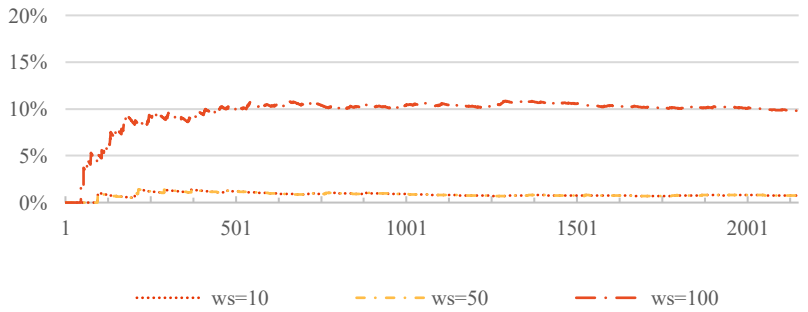

Figure 16 – Evolution of the precision of IF for the readByMetaReviewer property with w_s of 10, 50 and 100

ARI is always steadily negative, with the improvements in the classification of individuals changing how negative it skews, improving from -0.36 to -0.71, as shown in Figure 17 (featuring only the first 60 individuals, after which the metrics

stabilize). The metrics evolve differently when not all individuals in the stream are considered support.

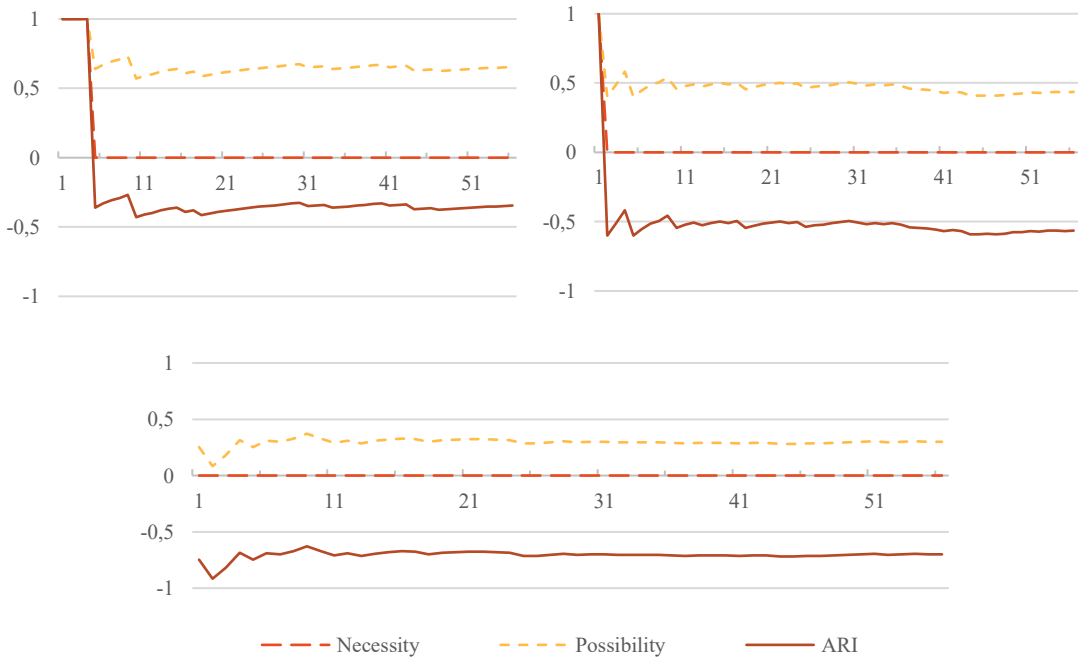

Figure 17 – Evolution of N, Π and ARI of IF for the readByMetaReviewer property with w_s of 10 (first graph), 50 (second graph) and 100 (third graph)

Consider the results in Figure 18, which were obtained when searching for inverse functionality of the *rejectedBy* property, which is used by 1279 individuals of the *Paper* class (≈50%).

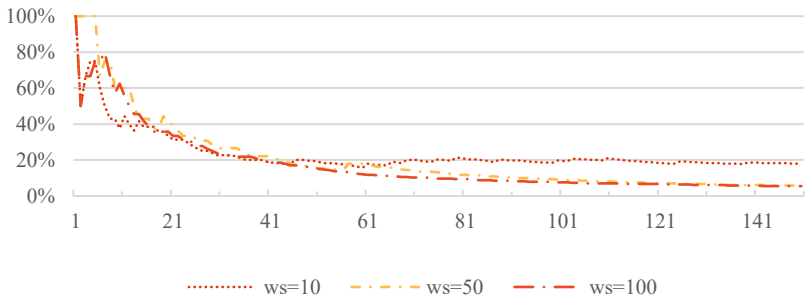

Figure 18 – Evolution of selective confirmations of IF for the rejectedBy property with w_s of 10, 50 and 100

Selective confirmations for the IF axiom start high (at 100%), as a single individual with a single use of the property cannot be flagged as a counterexample. After analysing 60 individuals, counterexamples account for more than 70% of the samples; 80% after 120, and their number finally stabilizes

around 85% as the analysis progresses. With a sliding window of 50, there should be more available evidence for each query to ascertain if the axiom is present. This seems indeed to be the case: while the 70% threshold obtained with a w_s of 10 is equally obtained after analysing around 60 individuals, with a w_s of 50 the 70% threshold is reached after analysing only 16 samples. The percentage of counterexamples also stabilizes at a higher point (at around 94%, after circa 75 individuals). Increasing w_s to 100 shows very minor increases in both speed and accuracy. The number of counterexamples on the sample reaches the 70% threshold earlier than with w_s of 50 – after 12 individuals – and stabilizing at a negligibly slightly higher point – at 97%.

Precision, recall and f-measure show similar evolution patterns as those of support, as seen in the first graph of Figure 19. All metrics start low, with a quick but unsteady increase until around 50 individuals have been analysed, and equally stabilizing as the analysis progresses.

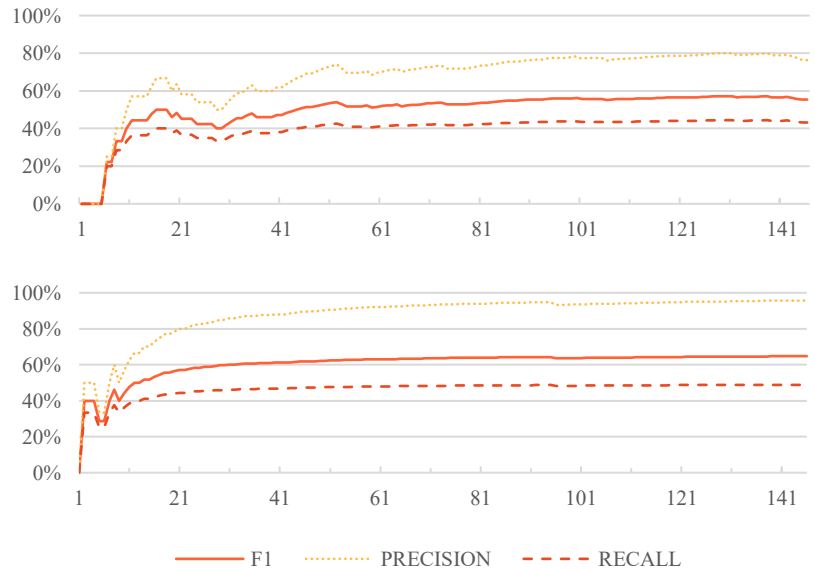

Figure 19 – Evolution of precision, recall and f-measure of IF for the *rejectedBy* property with w_s of 10 (first graph) and 50 (second graph)

Precision peaks at around 85%, as expected from the results presented in Figure 18. Recall stabilizes around 45% and f-measure at 60%. With a similar evolution to that of the support for the same window size, as illustrated in the second graph of Figure 19, similar levels of precision, recall and f-measure are obtained but with their values stabilizing higher and later (f-measure reaching 60% after circa 40 samples and 65% after 100). This effectively shows that by having a bigger sliding window – i.e., by having more individuals available for each query to search in – it is possible to track more nuances in the data and better

classify each individual as a selective confirmation or counterexample. In further increasing w_s to 100, no significant improvements are made. Precision, recall and f-measure stabilize faster, but improvements account for little more than 1%.

If one were to trust these metrics by themselves, it could be concluded that the approach can identify if the property is not IF with a certainty of 65%. Here, necessity, possibility and ARI can provide a faster and more complete idea of the classification that must be done when giving the proper weight to the counterexamples. The following figures illustrate the evolution of ARI using the same three window sizes as before.

Figure 20 shows necessity starting at 1, as only one individual has been analysed, and it is not a counterexample. However, as the second individual provides a counterexample, it immediately drops and stays at 0. Possibility, as determined by the number of selective confirmations, suffers a gradual loss as less and less selective confirmations are found in the data, and stabilizes very close to 0 (but with a positive value, as individuals that do not explicitly contradict the axiom can still be found) after around 50 individuals.

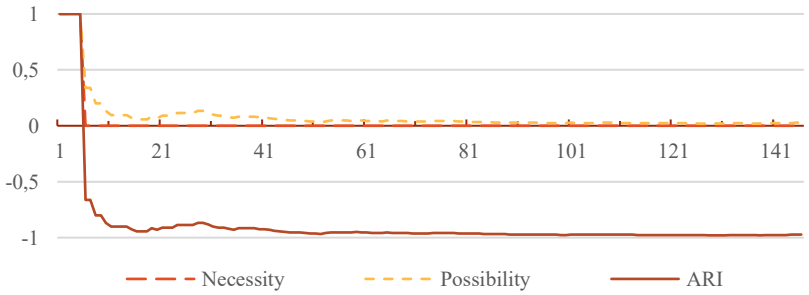

Figure 20 – Evolution of N, Π and ARI of IF for the *rejectedBy* property with w_s of 10

ARI, as a function of the other two metrics, shows a similar evolution from 50 individuals onwards, stabilizing very close to -1, which strongly advocates for axiom rejection, as seen in Figure 21.

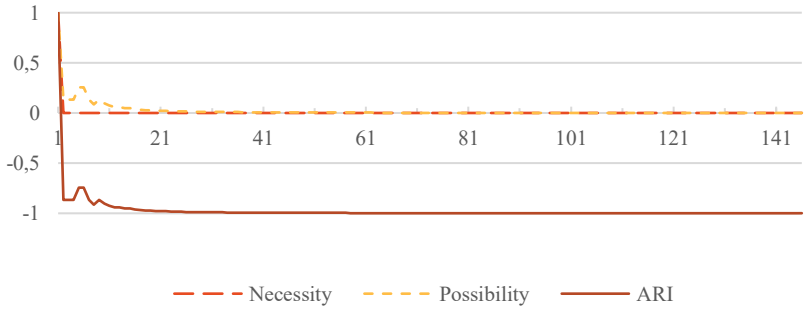

Figure 21 – Evolution of N, Π and ARI of IF for the *rejectedBy* property with w_s of 50

There are no discernible changes in the evolution of necessity, possibility and ARI when the window size is increased, as the number of selective confirmations and counterexamples is not as relevant for these metrics as the mere presence of counterexamples is. The metrics support the reasoning that a bigger window provides a better classification of each sample as either a selective confirmation or a counterexample, but only to a certain extent.

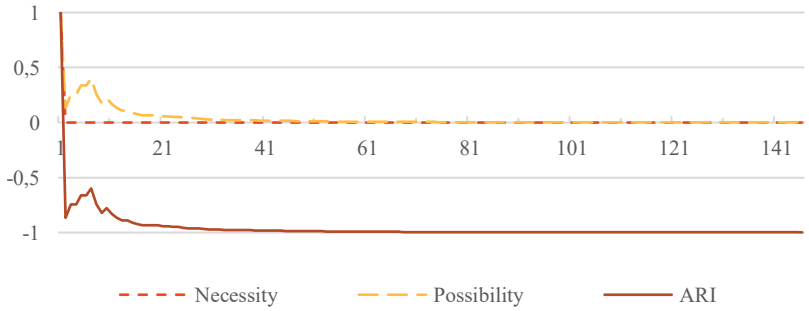

Figure 22 – Evolution of N, Π and ARI of IF for the *rejectedBy* property with w_s of 100

Similar to support and information retrieval metrics, Figure 22 displays how ARI evolves slightly faster but is not significantly improved. Unlike the *Paper* class previously explored, *Person* has even more variance in the application of its properties – e.g. the property *addedBy* is used by 734 of the 3000 individuals studied (≈25%), while *rejectPaper* is used only by 4 of them (≈0.1%) – although it is relevant to note that *rejectPaper* is the inverse property of *rejectedBy*, which is more widely used. While this change in frequency affects the evolution of the metrics being studied, the results follow the same trends as before: by allowing the queries to access more individuals, metrics are improved, but only until a certain point. The speed at which they improve, however, is indeed affected by the variety in the data, as shown in Figure 23.

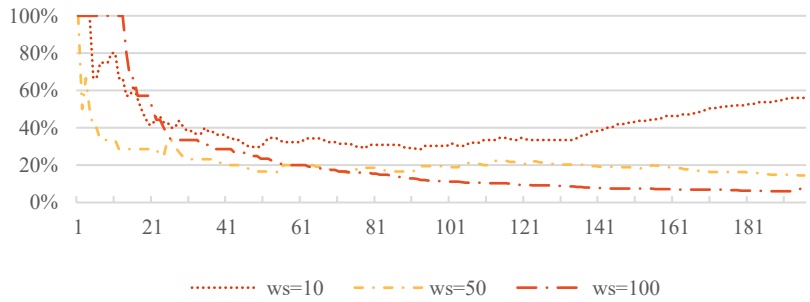

Figure 23 – Evolution of confirmations of IF for the property addedBy with w_s of 10, 50 and 100

The results show that if potential variations in use of the properties can be expected, increasing w_s allows for better classification as selective confirmation or counterexample. The same conclusion can be drawn from the analysis of the differences in precision, recall and f-measure for the three values of w_s, shown in Figure 24.

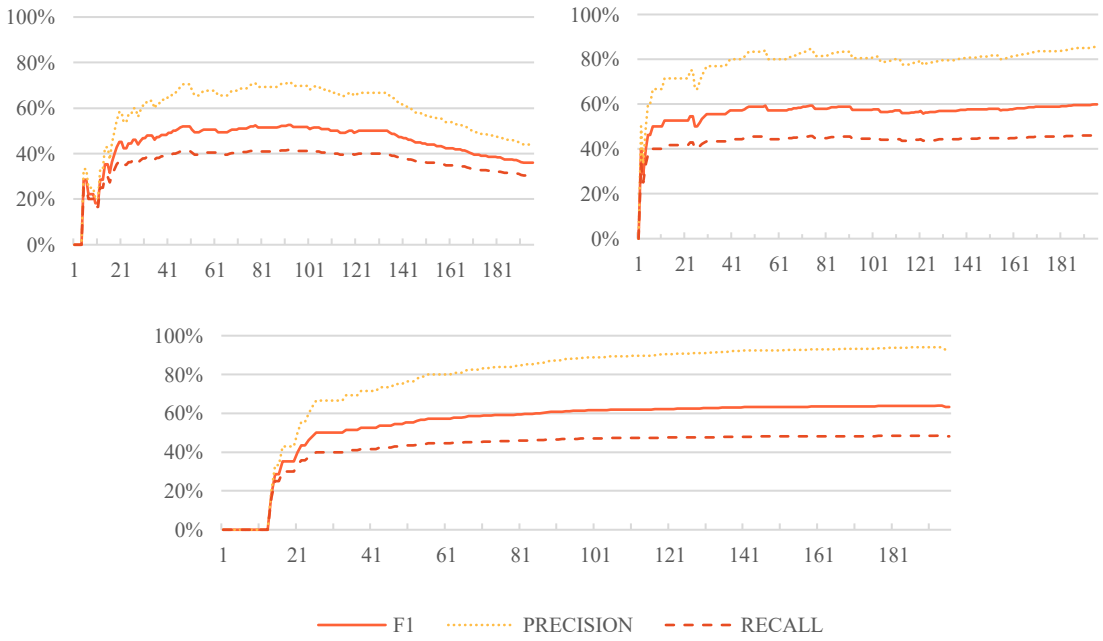

Figure 24 – Evolution of precision, recall and f-measure of IF for the property addedBy with w_s of 10 (first graph), 50 (second graph), and 100 (third graph)

Furthermore, it is important to note that for the smaller values of w_s, the evolution of the metrics is not monotonic (see the first graph of Figure 24). This suggests a lower w_s can be detrimental when parsing a large number of individuals.

Finally, the changes in the evolution of necessity, possibility and ARI follow the information retrieval ones, by showing how a small window size for a type with high property variety takes longer to stabilize, as seen in Figure 25.

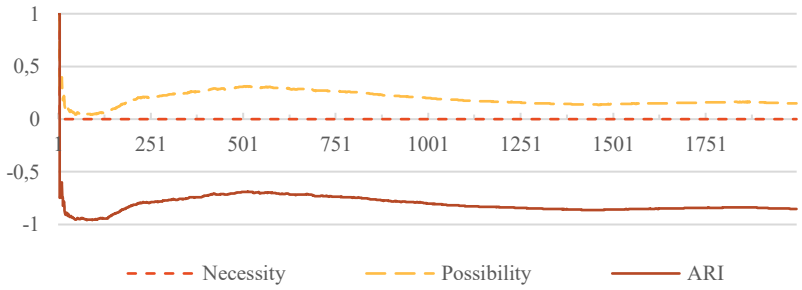

Figure 25 – Evolution of N, Π and ARI of IF for the property *addedBy* with w_s of 10

However, while there are significant improvements when w_s is increased from 10 to 50, the same cannot be said from increasing it from 50 to 100 (much like in the studies for the *Paper* class), as seen in Figure 26.

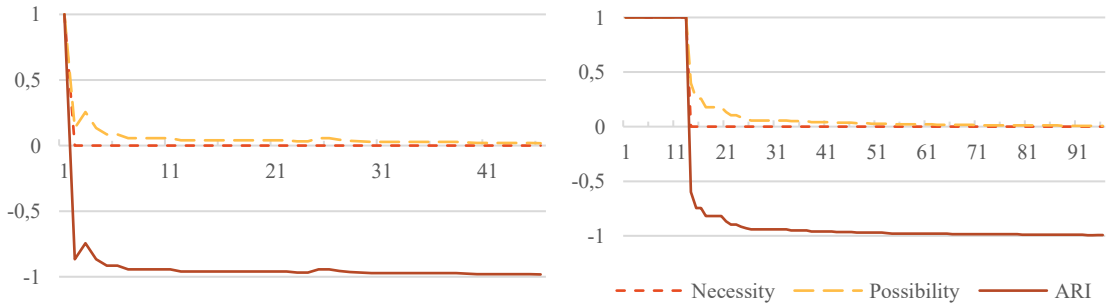

Figure 26 – Evolution of N, Π and ARI of IF for the property *addedBy* with w_s of 50 (first graph) and 100 (second graph)

In conclusion, in the cases where there is a very limited number of counterexamples, ARI may never reach its lower threshold; but remains reliably negative, even when f-measure is at its lowest – showing that the possibilistic approach is indeed robust and applicable when scoring axioms in streams and with limited data, and more so than a strictly probability-based one. It nonetheless benefits from increased number of counterexamples, implying that a w_s of 50 may provide sufficiently reliable results while also accounting for the negative effects of variations in property use.

Depending on the completion of the data, the incompleteness of the ontology, or simply because no such cases have ever been documented – the same sample can easily selectively confirm multiple property axioms. Consider, for example, that falsifying both F and IF axioms rely on the existence of more than one sample, or at least that the one sample uses the same property more than once.

This is an unreasonable expectation to have about the data, and any decision to include the axioms needs to consider the fact that absence of evidence is not evidence of absence.

Since a bigger window size allows for better categorization of samples, it is possible to see the effect in evolution of IF's ARI for the functional property *hasAuthor*. Consider the difference between the graphs in Figure 27.

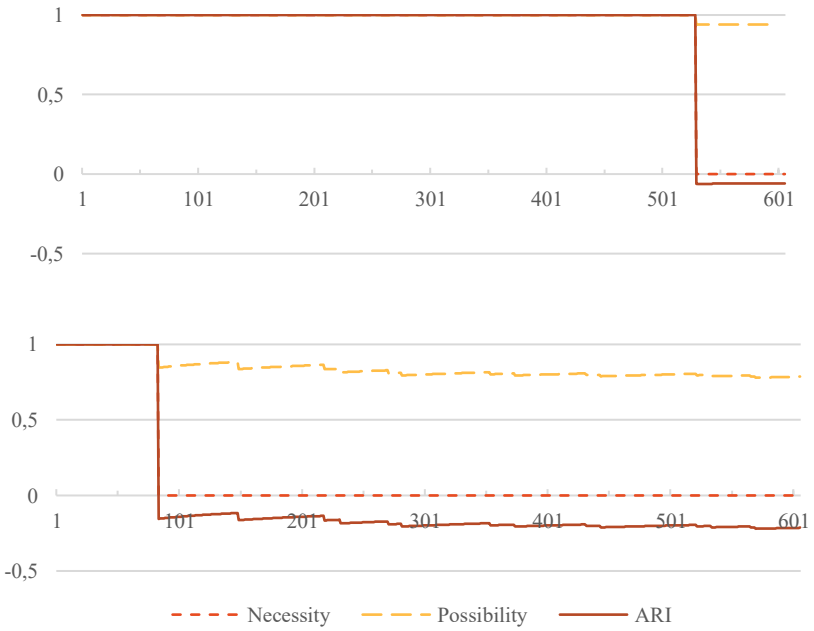

Figure 27 – Evolution of N, Π and ARI of IF for *hasAuthor* with w_s of 10 (first graph) and 50 (second graph)

Not only is the number of counterexamples very low, but they also take some time to occur in the stream. This means that while there are counterexamples, since there are so many sequential selective confirmations – and they continue even after a few counterexamples are encountered – N may drop to 0, and Π remains high to accommodate for potential errors in data. Therefore, while ARI skews negative, it remains relatively close to 0 (full uncertainty). However, it is once again interesting to note the benefits of the bigger window in the correct categorization of samples, as it allows for the negative ARI to be reached sooner (and more negative) as the window size increases.

Again, it is important to reiterate that this is a fortunate case: while the counterexamples may have taken longer to arrive and be few, they were still identifiable. When such is not the case, it has to be considered that just because a property is only used once per sample, it does not necessarily mean that it must

certainly be *both* functional and inverse functional (although it is possible to be both at once) – even if their Π, N and ARI are always at 1.

4.3.2 Using the Weak Forms of Possibility and Necessity

Transitivity was studied using the Wine and Plant ontologies, with similar results. Since the Wine ontology had more available individuals with transitive properties (a total of 82), the following results reflect those exclusively.

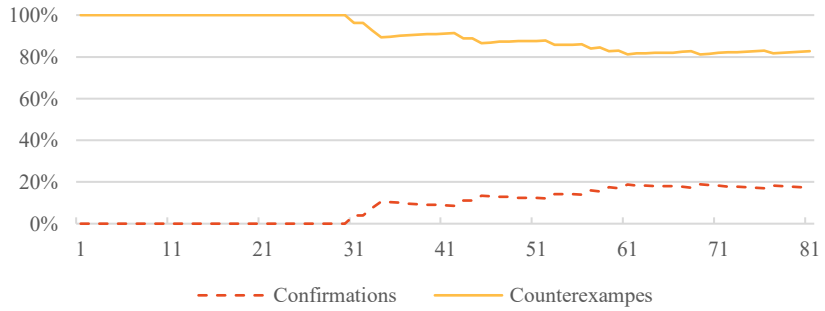

Figure 28 – Evolution of confirmations and counterexamples of T for the property *locatedIn* with w_s of 50

Figure 28 shows how the lack of explicit confirmations affects the support for an axiom. It was necessary to analyse at least 30 individuals until a confirmation could be found, with the number increasing steadily until it reaches around 15%. Information retrieval metrics, as seen in Figure 29, also illustrate the clear lack of selective confirmations, which informed the decision to apply the weak forms of Π and N.

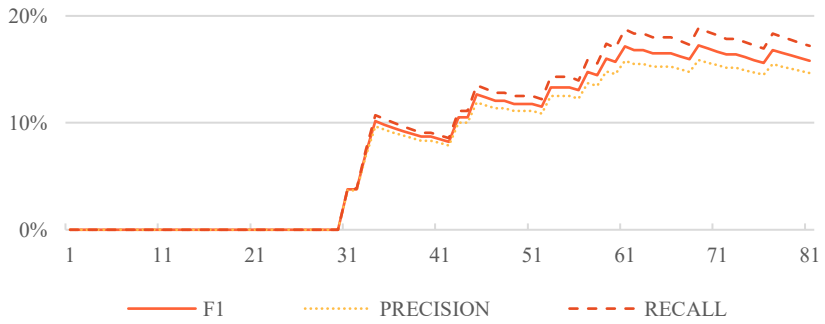

Figure 29 – Evolution of information retrieval metrics of T for the property *locatedIn* with w_s of 50

By giving more weight to the very hard to find selective confirmations than to the very easily incorrectly identified counterexamples, Figure 30 shows it is possible to obtain a positive, albeit conservative, ARI.

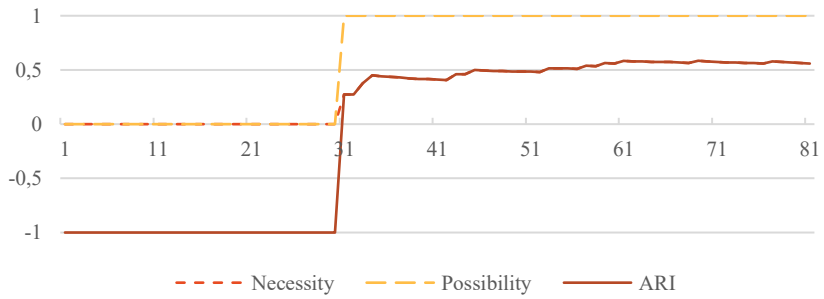

Figure 30 – Evolution of the weak forms of Π, N and ARI of T for the property *locatedIn* with w_s of 50

Should the strong form of possibility and necessity been applied instead, ARI would tend towards extremely negative (circa -1), even if the possibility was seen increasing (very) slowly.

4.3.3 Discussion

Of the three options in window size studied – of 10, 50 and 100 individuals – the results show that there is a significant improvement between going from 10 to 50, but hardly any from 50 to 100. We find 50 individuals to be a good middle ground; while the results can be improved by employing bigger sliding windows, they also validate our assumption that it is possible to efficiently learn property axioms from a relatively small number of samples.

The observation of the previous results show that the computation of ARI contains more information than just the percentage of selective confirmations and could potentially be used in its place. Additionally, ARI shows better performance than traditional information retrieval metrics, achieving sharper results and considerably earlier. However, since the existence of a single counterexample immediately skews ARI towards negative, $E^{+}_{Ax(P)}$ cannot be altogether excluded, especially considering that potential errors in data could be classified as counterexamples. Furthermore, the results suggest that some axioms benefit from the application of a higher threshold for ARI than others.

For axioms that are relatively easy to falsify, but not so easy to prove, like F and IF, a higher threshold for ARI should allow for some errors in the data while still strongly advocating for the inclusion of said axioms. On a window size of 50 individuals, we argue that ARI should be of 1 for an axiom to be considered for inclusion, but have a proportion of $E^{+}_{Ax(P)}$ of at least 95%. Alternatively, for axioms for which there may be an overabundance of negatively-identified

counterexamples and therefore make use of the weak forms of N and Π, a lower threshold for ARI should be considered: allowing an axiom to be considered for inclusion even though there is relatively little evidence for it. For these, a threshold for ARI of around 0.5 is proposed.

There is a second part to the discussion of the existence of counterexamples: axioms that are the opposite of one another. While a property can be both functional and inverse functional at the same time, it cannot be both symmetrical and asymmetrical. Furthermore, while a selective confirmation of S is a counterexample of AS, a selective confirmation of either does not necessarily mean that the property axiom must to be present; any decision-making processes need to take this in consideration. This is especially evident in the case of property *hasDecision* of the CMT ontology, which is functional, but shows N, Π and ARI of 1 for F, IF, AS and IR.

4.4 Experiment II: Accommodating for errors in data and the effects of previous knowledge in axiom-inclusion decisions

The following experiment pertains to show the applicability of the solution for the purposes of ontology enrichment/evolution. This is done by analysing a stream of individuals in which not only new properties are added over time, but the known properties are also applied in different ways to describe the data. As such, a suitable dataset needs to support the following: (1) have several properties that connect the individuals in diverse ways, such that all different property axioms being studied could be tentatively discovered, (2) the application of said properties should change over time, as the domain naturally evolves, (3) the property axioms are not previously known.

The goal of this experiment is to answer the following questions:

- When combining ARI and Support, are any inconsistencies introduced? To which extent? Are the inconsistencies the result of incorrect data or marginal, but valid uses of the properties?
- If previous knowledge is considered in the decision to include/exclude a property axiom, does it prevent or increase the number of inconsistencies introduced?

The ontologies in use pertain to the Pokémon domain as it is described in Wikidata [68]. Pokémon is a series of games about cataloguing fictional wild creatures of the same name. Over the years, several games have been released, and each generation of games – a total of nine, at the time of writing, – introduces (and refines) mechanics, regions and creatures. Much of this information is publicly available on Wikidata. New subproperties of those present in Wikidata were created (but not characterized) that pertain to specific relationships in the Pokémon domain – Wikidata's properties, by design, are high level and lack the nuance necessary to describe specific domains beyond very simple connections between its individuals (e.g. part of, instance of).

The experiments detailed below show how axiom scoring can be used to determine which axioms could be associated with each property, and how their usage changes with each new generation of games. Table 14 shows the values

and thresholds employed, and the application of both ARI and the percentage of selective confirmations (cf_t) for axiom scoring are described below. Each generation will be analysed as a different version of the ontology, provided by its own timeframe, and introductions and changes to its properties are documented. As previously established, an acceptation threshold of 0.5 will be used for both strong and weak forms of ARI (sf_t and wf_t, respectively) and a window size (w_s) of 50.

Table 14 – Values and thresholds employed in the experiments

VARIABLE	DESCRIPTION	VALUE
cf_t	Percentage of selective confirmations w.r.t. support	95%
sf_t	Threshold for the Strong form of ARI	0.5
wf_t	Threshold for the Weak form of ARI	0.5
w_s	Sliding window size	50

4.4.1 ARI and Support

4.4.1.1 Generation I

Generation I (Gen I) features games with relatively simple mechanics. It introduces one region, one Pokédex (the Pokémon encyclopaedia) and 151 Pokémon. Using the data available on Wikidata, enough information was extracted to ascertain the properties described below. Table 15 shows the names, percentage of selective confirmations, proposed property axioms and ARIs for each.

bordersWith, a property that establishes a connection between two locations (e.g. cities or roads) that share a border, seems a target candidate for S, but the results do not support it (selective confirmations amounted to 1%). Interestingly, the results show there is sufficient positive evidence for T, and adding this axiom to the ontology does not make it inconsistent – although including it would allow for entailing incorrect conclusions about the data. Counterexamples were found for AS and IR. *locatedIn* is another property related to the geography of the region. However, since only one region has been introduced as of Gen I, it is classified as F, with no contradictions on the data.

Table 15 – Gen I properties

PROPERTY NAME	AXIOM	%CF	ARI
bordersWith	T	51%	0.87
hasEvolutionGroup	AS	100%	1
	F	100%	1
hasMoveType	F	100%	1
	AS	100%	1
	IR	100%	1
hasPart	IF	100%	1
	AS	100%	1
hasPokedexEntry	F	100%	1
	IF	100%	1
	AS	100%	1
	IR	100%	1
introducedIn	F	100%	1
	AS	100%	1
	IR	100%	1
locatedIn	F	100%	1
	AS	100%	1
	IR	100%	1
partOf	F	100%	1
	AS	100%	1
	IR	100%	1
presentIn	F	100%	1
	AS	100%	1
	IR	100%	1
hasValue	F	100%	1
hasHeight	F	99%	-0.11
hasWeight	F	96%	-0.28
hasFacet	F	100%	1
hasName	F	100%	1
hasPokedexNumber	F	100%	1
hasColor	F	99%	-0.16

With each Pokémon belonging to a single evolution group (described by the *hasEvolutionGroup* property) but each group having more than one creature, the expected axiom for this property would be F, which the results support.

Of the datatype properties, three of them were classified as F even though counterexamples were found – since the percentage of selective confirmations was considered sufficient, and the deviations should pertain to possible errors in the data. In this case, we can verify if this was the cause, by using the reasoners provided by Protégé (in this case, HermiT[7]) and adding said axioms to the ontology and analysing the explanations provided in case inconsistencies are found.

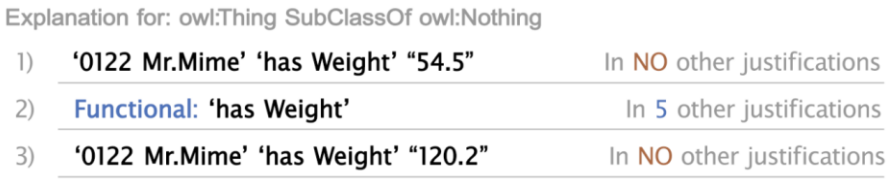

Figure 31 – Reasoner's explanation for inconsistency for property *hasWeight*

Figure 31 shows the inconsistency explanations for the *hasWeight* property, in which it is possible to see there are 6 individuals with more than one entry, and the duplicates' values seem to correspond to the same value under different representation systems (metric vs imperial), which suggest that using the same property to represent both may not be adequate.

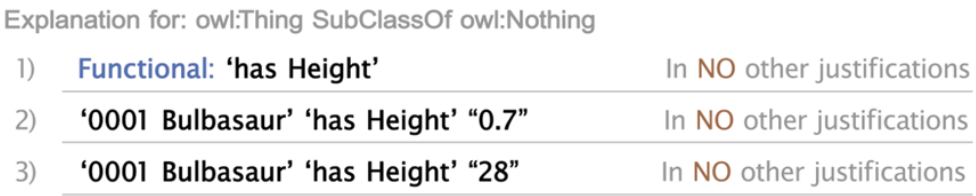

Figure 32 – Reasoner's explanation for inconsistency in property *hasHeight*

Figure 32 shows how there is one single individual that contradicts the functionality of the *hasHeight* property, with the same apparent justification as the *hasWeight*.

Figure 33 – Reasoner's explanation for inconsistency in property *hasColor*

[7] http://www.hermit-reasoner.com/

Figure 33, on the other hand, shows there are 2 individuals with more than one colour, both belonging to the same evolutionary group. This may be an oversight in the data acquisition from Wikidata, which often mixes the information of all generations.

4.4.1.2 Generation II

Generation II (Gen II) improves on Gen I by introducing a new region (adjacent to the first one), while still allowing the player to visit the one introduced in Gen I. The Pokédex is expanded to accommodate for the new region: the player now effectively can access not one, but two Pokédexes, one at the national level (with all creatures) and a regional one (with the creatures inhabiting the new region exclusively, which may or may not be new). Therefore, the same creature may now be associated with more than one Pokédex entry, but each entry will be associated to its own numbering system (and potentially, description). Information about the creature's shape is also added, and the number of Pokémon grows from 151 to 251. Additionally, some new game mechanics are included: creatures can now have one of three genders (female, male, and unknown), and can reproduce within a given group (not necessarily only with members of the same species).

Information about the properties present in Gen II is displayed in Table 16. For the sake of brevity, only changes in axioms are shown. If an axiom is removed, it is preceded by a – symbol, and by a + otherwise. *hasPokedexEntry*, which relates a Pokémon to its corresponding Pokédex information, loses the F axiom – as the new region introduced a new Pokédex, and therefore a Pokémon may have more than one entry. However, it retains the IF axiom, as each entry relates to a single creature. *presentIn*, a property which describes in which generation a given entity or mechanic is featured, can now point to more than one option and, therefore, is no longer F. Of the new properties, *alternateDexEntry* connects any two entries in different Pokédexes that describe the same creature and therefore should be classified at least as S. The results show not only this happens, but it also does not contradict the T, F and IF property axioms.

With the introduction of genders, each species displays one of several possible gender ratios (via the *hasGenderRatio* property), and as such has been classified as F. As no evidence was found against it, it is also classified as AS and IR. *hatches*, which relates a creature with its reproductive group, and each reproductive

group with the Pokémon in it, has sufficient ARI to be classified as T, but not S. Finally, the only datatype property added in Gen II, *hasShape*, does not meet the criteria for any of the property axioms.

Table 16 – Gen II properties

PROPERTY	AXIOM	%CF	ARI
bordersWith	+T	65%	0.94
hasEvolutionGroup	+F	100%	1
hasMoveType	+F	100%	1
hasPokedexEntry	–F	0%	-1
presentIn	–F	37%	-0.93
alternateDexEntry	+F	50%	0.86
	+IF	100%	1
	+T	50%	0.87
	+S	50%	0.87
hasGenderRatio	+F	100%	1
	+AS	100%	1
	+IR	100%	1
hatches	+T	25%	0.66
hasShape	N/A	N/A	N/A

When adding the proposed axioms to the object properties in the ontology of Gen II, no inconsistencies are generated. However, the same cannot be said for the datatype properties.

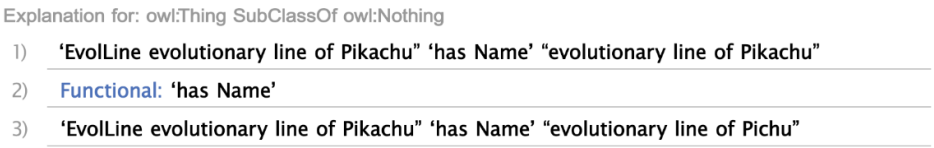

Figure 34 – Inconsistencies with the use of the *hasName* property

According to Figure 34, there are some inconsistencies regarding the use of the *hasName* property, namely when describing the evolutionary groups. Because new Pokémon were introduced in between generations and added to existing evolutionary groups, some of their names to have undergone changes that have not been corrected. As seen in Figure 35, in addition and similar to the inconsistencies present in Gen I, there are a few (six) entities with two or more uses of the property *hasColor*.

Figure 35 – Inconsistencies with the use of the *hasColor* property

4.4.1.3 Generation III

Generation III (Gen III) introduces two more regions, and no longer features those of Gens I and II. It also introduces two new mechanics: abilities and contests. Each Pokémon can now have one or two of the 77 possible abilities – some of which are considered "signature abilities", as they are only shown for specific Pokémon or specific evolutionary groups. Furthermore, 135 new Pokémon are added (to a total of 386). The changes and additions to the properties and their axioms are shown in Table 17.

Table 17 – Gen III properties

PROPERTY	AXIOM	%CF	ARI
hasMoveType	–F	0%	-1
alternateDexEntry	–F	65%	-0.76
	–IF	53%	-0.9
	–T	35%	0.76
hasSignatureAbility	+F	100%	1
	+IF	100%	1
	+AS	100%	1
	+IR	100%	1
isSignatureAbilityOf	+F	100%	1
	+IF	100%	1
	+AS	100%	1
	+IR	100%	1

With the introduction of contests, moves now have no longer only a type in battle, but a type that shows in contest (the two are not related). As such, the

property *hasMoveType* is no longer compatible with functionality. Once again, with the introduction of a new region and a new Pokédex, the same Pokémon can have multiple entries – and although these are still alternatives to each other, and therefore symmetrical, there is now more than one possible alternative and the property can no longer be classified as either F or IF.

Finally, *hatches* maintains its T. Evidence against IR was below the minimum threshold (selective confirmations amounting to 95% of the data) – but as it was already classified as such, the axiom is not added. Following the definition of a signature ability discussed above, the property is properly classified as F (each Pokémon/evolutionary group has only one signature ability) and IF (each signature ability is used by only one Pokémon/evolutionary group). Since no counterexamples for IR and AS are found, these axioms are also added to the property. As the percentage of counterexamples found for F in *hasWeight* has once again lowered below the minimum threshold, the property axiom is reinstated.

As with Gen II, when the axioms discovered are added to object properties in the ontology for Gen III, they do not produce inconsistencies. However, with support for *hasColor* at 98% and *hasHeight* as 99%, counterexamples are rare but produce some inconsistencies.

4.4.1.4 Generations IV and V

Generation IV (Gen IV) introduces once again a new region and, with it, comes a new catalogue and new Pokémon (totalling 493). There is also the introduction of 47 new abilities and 113 new moves. Several evolutionary groups from Gen I were expanded with new elements, and moves are now classified not only according to their previously known contest and type categories, but with an additional damage category that is fully independent from its type. It is also in this generation that the games make use of alternate forms of the same Pokémon (including, but not limited to, differences by gender). Generation V (Gen V) raises the total number of Pokémon to 649, and once again introduces a new region while limiting access to the previous ones, but does not introduce any new properties.

Changes and additions to the properties of the ontology for Gen IV are displayed in Table 18:

Table 18 – Gen IV properties

PROPERTY	AXIOM	%CF	ARI
alternateDexEntry	+T	47%	0.85
hasAlternateForm	+T	62%	0.92
	+S	27%	0.69

The inclusion of these axioms in both Gen IV and Gen V ontologies does not cause inconsistencies beyond those already discovered in previous generations.

4.4.1.5 Generation VI

Generation VI (Gen VI) introduces 72 new Pokémon (to a total of 721), 58 new moves and 24 new abilities (to a total of 617 and 188, respectively). It also adds a new battle mechanic, the Mega Evolution, which is a type of alternative form that can be (temporarily) triggered in battle. Like previous generations, Gen VI introduces a new region and a new Pokédex, while also revisiting the region first introduced in Gen III. Table 19 shows the changes in the property axioms in Gen IV:

Table 19 – Gen VI properties

PROPERTY	AXIOM	%CF	ARI
hasMegaEvolution	+F	98%	-0.21
	+IF	100%	1
	+AS	100%	1
	+IR	100%	1
uses	+F	96%	-0.23
	+IF	100%	1
	+AS	100%	1
	+IR	100%	1

hasMegaEvolution, a relationship between a Pokémon and its Mega Evolution alternate form, is both F and IF (theoretically meaning that a Pokémon can only have one mega evolution and that evolution belongs to only one Pokémon). The mechanic is triggered by the usage (with the *uses* property) of different battle items, and each of them causes a specific evolution to occur.

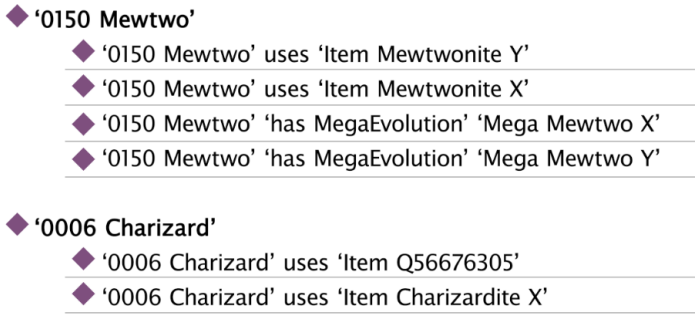

Figure 36 – Rare, but valid individuals which do not support F for the *hasMegaEvolution* and *uses* properties

When the axioms are added to the Gen VI ontology, they do not produce inconsistencies – contrary to what would be expected when the percentage of confirmations is below 100%. However, upon closer analysis, it is possible to see that there are indeed few, but valid, cases in which a Pokémon can have more than one mega evolution, caused by the application of more than one item. In the dataset for Gen VI, there are two such occurrences, shown in Figure 36. The reasoner fails to flag these as inconsistencies unless the individuals are explicitly stated as being different (which would always be the case under the UNA).

4.4.1.6 Generation VII

Generation VII (Gen VII) sees the introduction of regional forms, as the same creature adapts to different habitats: effectively another specific type of alternate form. It also increases the number of Pokémon to 802 and introduces a new region and its respective Pokédex. It revisits the region introduced in Gen I, adding new alternate forms to some previously known Pokémon.

Table 20 – Gen VII properties

PROPERTY	AXIOM	%CF	ARI
hasRegionalForm	+F	97%	-0.24
	+IF	98%	-0.19
hasSignatureAbility	–F	94%	-0.33
isSignatureAbilityOf	–IF	95%	-0.32
uses	–IF	71%	-0.7

With the introduction of new abilities, Gen VII alters how signature abilities work, and a few Pokémon can now have more than one signature ability – as such, *hasSignatureAbility* can no longer be F, and *isSignatureAbilityOf* can no longer be IF. Thankfully, in this case, the number of counterexamples is above

the threshold and the axioms are correctly removed. If the axioms are added to the ontology, by UNA they generate some inconsistencies. Figure 37 shows one such case, in which a valid selective confirmation of a creature has more than one known regional variant.

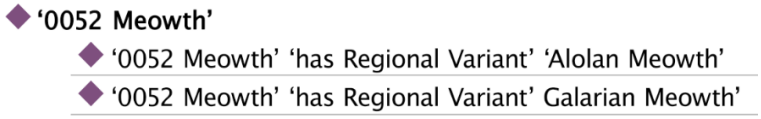

Figure 37 – Rare, but valid individuals which do not support the F and IF for the *hasRegionalForm* property

4.4.1.7 Generations VIII and IX

Generation VIII (Gen VIII) introduces two new regions, each with its individual Pokédex. The national one, which catalogues all creatures, now goes up to 890, with 19 new regional forms. The mega evolution mechanic is removed from the games. Generation IX (Gen IX) introduces another region and its respective catalogue, and 103 new creatures (raising the total to 1008). While it introduces the concept of convergent evolution – creatures that fill the same ecological niches also sharing physical similarities – information about this was not present in Wikidata at this time. Finally, this generation introduces a few more regional forms and a new type of alternative form that is not well described in Wikidata. Because of alterations on how signature abilities are assigned to evolutionary groups between games, Table 21 shows the changes in the associated properties.

Table 21 – Gen VIII and Gen IX properties

PROPERTY	AXIOM	%CF	ARI
hasSignatureAbility	+F	96%	-0.28
isSignatureAbilityOf	+IF	96%	-0.27

Once again, this is a case in which the inconsistencies refer to valid uses of the property, meaning the application of the axioms is in the wrong, as shown in Figure 38.

Figure 38 – Evolutionary line with more than one signature ability, a counterexample to the F of *hasSignatureAbility*

4.4.2 Evolving ARI

Experiments show that allowing for some leeway in terms of inconsistency can result in the discovery of errors in the data, such as duplicates, or potential mistakes in modelling that do not account for the possible alternatives. It also shows that not all inconsistencies are caused by said incorrections, and some valid, but outlier information can get flagged as inconsistent.

Evolving ARI depends not on a minimum threshold on the percentage of selective confirmations, cf_t, but on the evidence found in the current timeframe and the conclusions obtained on the previous one (which may be a state of total ignorance, if the property was not present).

The following experiments are divided in two different sections, namely:

1. Applicability and robustness of ARI for axiom scoring. Considering the dataset was obtained from Wikidata, which is often incomplete and sometimes offers incorrect or even contradictory information, we consider the percentage of selective confirmations does not need to equal 100% for the axiom to be accepted. Furthermore, in this case, the analysis is done from a point of complete ignorance, in which no previous versions of any axiom are considered. This should allow for a more informed decision about which thresholds to consider for axiom inclusion in the following experiments.
2. Analysis of ARI_e over several different timeframes, in which previous versions of the ontology affect the scoring of the new axioms. Using the previously defined thresholds for axiom inclusion and window size, it is possible to measure the effect of more conservative vs progressive approaches to knowledge evolution.

First, we must consider how conservative the approach will be by defining the relative weight of previous knowledge, w_p. Then, we must establish a threshold for inclusion or rejection of the hypothesis considering the computed

ARI_e (sf_t or wf_t, depending on which form was applied). These two values cannot be completely independent of one another, since we are considering only scalar values for the previous state and it is possible for new data to directly contradict said state. Should the threshold for inclusion be too high, changing the axiom between timeframes could become impossible. For such changes to be a possibility, sf_t , wf_t and w_p should be inversely proportional. Since ARI cannot go over 1, for ARI_e to allow for evolution, sj_t should be below or equal to 0.5. Using ARI_e also allows for some data in a timeframe to contradict the axiom that is asserted in that period.

Table 22 shows the thresholds employed for the following experiments – maintaining the acceptance thresholds for the strong and weak forms of ARI (sf_t and wf_t, respectively) and a window size (w_s) of 50 –, in which different weights of previous knowledge for ARI_e (w_p) will be compared.

Table 22 – Values and thresholds employed in the experiments

VARIABLE	VALUE
sf_t, wf_t	0.5
w_p	0.8 / 0.7 / 0.4
w_s	50

ARI_e depends on the results obtained in previous iterations, so its application can be measured from Gen II onwards. When previous decisions (valued 0 if not present, and 1 for present) about the property axioms are not known, an ARI_e of 0 is assumed, and the decision for each previous non-existent axiom is scored at 0.5, both to reflect a state of ignorance. The rest of this section presents and discusses the cases in which the decision supported by ARI_e is different than when using the combination of cf_t and ARI.

First of all, it is interesting to see the effects of the different weights affect the evolution of ARI_e. Figure 39 shows how a more conservative approach tends to favour whichever initial decisions were taken, while lower values require less evidence for a decision to be revoked. In this case, the conservative approach would be wrong, as classifying *hasAlternateDexEntry* as F would generate a large number of inconsistencies that would only grow with each generation.

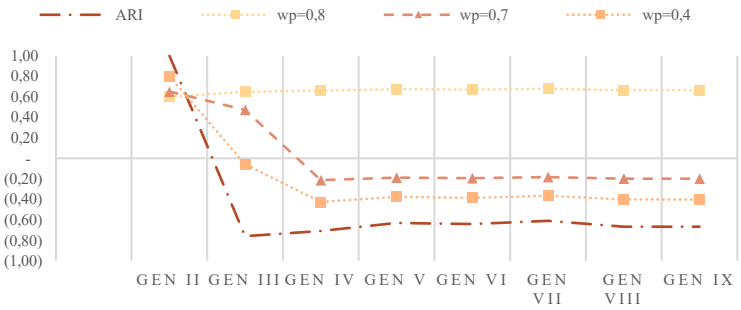

Figure 39 – Evolution of ARI_e for *hasAlternateDexEntry*'s F with different relative weights

Starting with the datatype properties, which were the ones creating the most inconsistencies since early generations, we can see that without taking cf_t in consideration, the duplicates are taken in consideration and the decision, across all generations, is not to include the F axiom. Figure 40 shows the results for the *hasColor* property, although they are similar for the others previously mentioned (*hasWeight* and *hasHeight*). This effectively means that F can no longer be employed to identify errors in data that resulted in the inconsistencies described above.

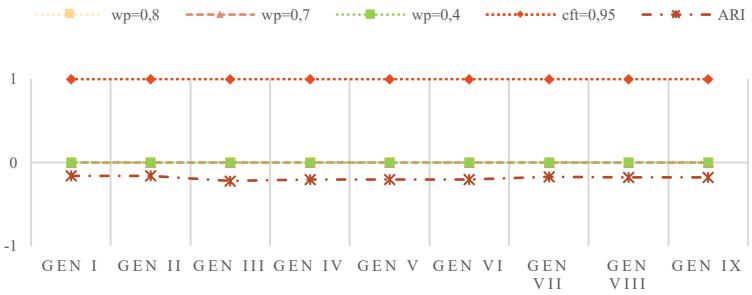

Figure 40 – Comparison of the decisions made using ARI_e and ARI+cf_t for F in *hasColor*

Differences in the decisions reached using ARI, ARI+cf_t and ARI_e are shown by the *hatches* property, as seen in Figure 41 for T.

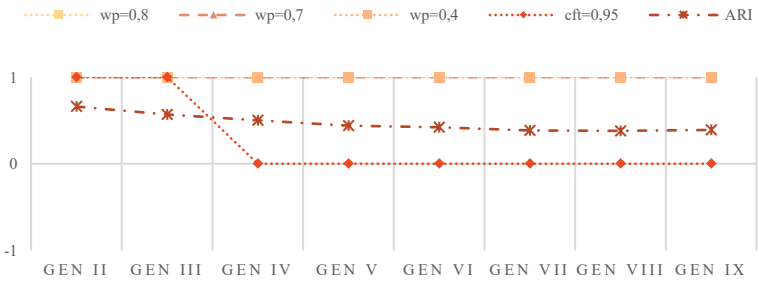

Figure 41 – Decisions regarding the T of the *hatches* property

With the more conservative approach to change provided by ARI_e, the property is equally classified as T for the first two generations, but manages to

maintain said status afterwards, despite the gradual decrease in ARI. In this case, adding the axioms to any of the versions of the ontology does not generate any inconsistencies.

In some generations, *isSignatureAbilityOf* and *hasSignatureAbility* were shown to have valid counterexamples that were not considered because of the chosen value for cf_t. Figure 42 shows the effect of the different w_p in the decision-making process. The results show that employing lower w_p results consistently in a Functional *hasSignatureAbility*, which is known to produce some inconsistencies from Gen VIII onwards. The opposite happens with heavier w_p: by making it harder to change opinion, even though ARI_e gets slightly higher – and in some cases, even positive – values than ARI, the evidence is not sufficient for a change. Figure 43 shows similar trends for IF of *isSignatureAbility* property.

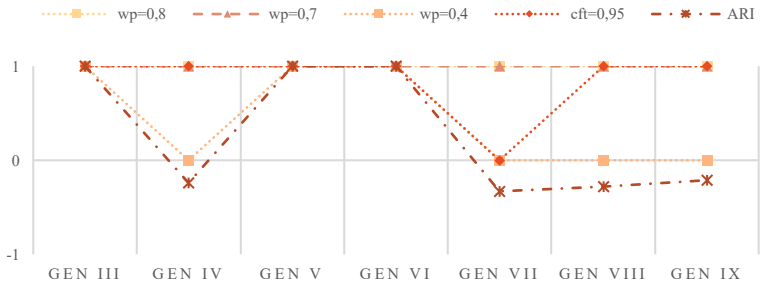

Figure 42 – Comparison of the decisions to include/exclude the F axiom for the *hasSignatureAbility* property

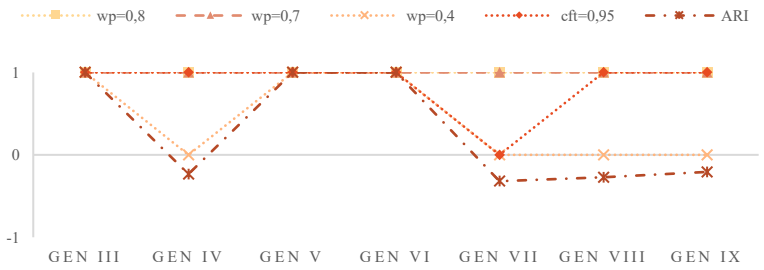

Figure 43 – Comparison of the decisions to include/exclude the IF axiom for the *isSignatureAbilityOf* property

hasRegionalForm, which was considered F, IF and S, is now instead classified as T and S for all generations in which it is featured. This happens because ARI_e cannot sustain F, as seen in Figure 44.

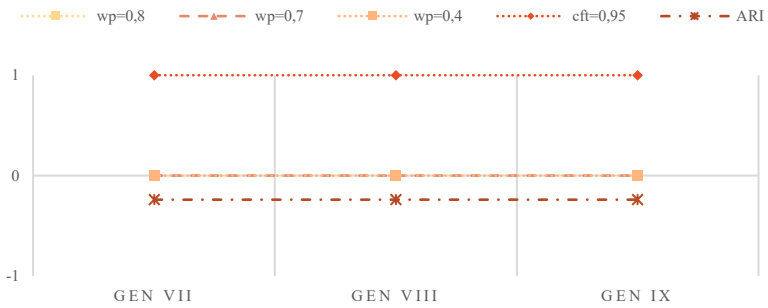

Figure 44 – Comparison of the decisions to include/exclude the F axiom for the *hasRegionalForm* property

Since F is no longer associated with the property, it can assume the T axiom, for which it did have sufficient ARI and ARI_e, as Figure 45 shows.

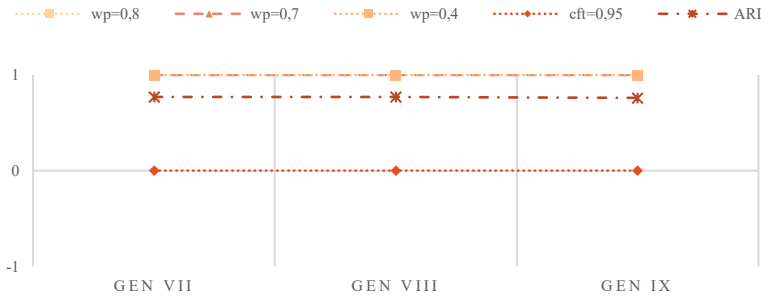

Figure 45 – Comparison of the decisions to include/exclude the T axiom for the *hasRegionalForm* property

The inclusion of the T and S axioms to the ontologies of Gens VIII and IX does not produce inconsistencies.

4.4.3 Discussion

Using ARI by itself does not allow for inconsistencies to arise, which in turn means that in spite of its utility for axiom scoring, it does not allow for the very likely possibility of errors and inconsistencies in the data – which is especially relevant when developing an ontology from publicly-available information from sources such as Wikidata. By combining ARI and cf_t, we allow for some level of counterexamples to be possible while still accepting an axiom hypothesis. In these cases, only the analysis of the data can show if the decision is correct or incorrect – for example, by investigating any generated inconsistencies to ascertain if they effectively correspond to errors, mistakes, or valid information. Nonetheless, we find the approach helpful for deciding on the inclusion of property axioms on evolving ontologies.

The results show that is still possible to counter some errors in data when using ARI_e, but not as effectively as the combination of ARI and cf_t. This is because, by favouring previous knowledge, ARI_e may have difficulty changing its opinion even in the face of overwhelming evidence. On some occasions, this may be beneficial: the experiments show that, when applied for T and S – the two property axioms that use the weak-form of ARI because of their inherent missing information – their inclusion is favoured earlier, and is harder to dismiss, with no inconsistencies resulting from it. However, for the properties using the strong form, the effect is diametrically opposite: axioms are maintained, even though they may introduce an increasing number of inconsistencies over time. Finding a balance between these two outcomes may be possible by tinkering with different values for sf_t and wf_t.

4.5 Conclusions

This chapter presented an adaptation of the possibilistic approach to axiom scoring to the context of RDF data streams for ontology evolution, in order to address the research question **RQ2.1:** *How can we identify changes in property axioms?*

The different approaches to possibility and necessity proposed in literature were recontextualized in terms of their bias towards selective confirmations or counterexamples, and the assumptions regarding the openness of the world under which they operate. Some property axioms, namely transitivity and symmetry, benefit from a more lenient approach, relying more on selective confirmations than on counterexamples – while the others benefit from stricter acceptance conditions to prevent the proliferation of inconsistencies.

To test the applicability of the solution, it was applied in two distinct scenarios: (1) a first one, in which the property axioms were previously known, and which allowed for the exploration of the effectiveness of the approach for their discovery in a scenario where no incorrect data was present; and (2) a second one, in which neither the properties nor their axioms were known, and the dataset was obtained from publicly available sources, possibly both incomplete and with errors.

Regarding Experiment I, results show that possibilistic approach is well suited to suggest potential axioms for ontology properties in an instance-guided

ontology evolution scenario, achieving conclusions about inclusion/exclusion of axioms from a relatively small sample of individuals (roughly 2,3%), and substantially faster than using traditional information retrieval metrics – making it particularly suitable for quickly learning new axioms from streams of RDF data. This, however, is not achieved without some caveats: the approach is not sufficiently robust to cases in which there are inconsistences or errors in data, with the strong approach being particularly unable to recover from counterexamples that may reflect said errors. No approach can effectively deal with all potential negative side effects of dealing with an open world and with incomplete knowledge; if ARI is used independently of the support of the selective confirmations, it cannot account for missing or incorrect information.

Additionally, this experiment shows that the size and variety of the individuals in the dataset affect the speed and accuracy of the identification of axioms, suggesting that some finetuning of parameters may be necessary to achieve the best results depending on the application scenario.

Experiment II aimed to overcome the issue with ARI identified in Experiment I in two ways: (1) by combining it with a minimum percentage of selective confirmations and (2) by attributing weight to previous knowledge between timeframes. Axioms that are not 100% supported by ARI may therefore be accepted: this allows for the identification of potential errors in data but may also erroneously suggest discarding valid information. Acknowledging the information provided by previous versions of the ontology when deciding for or against an axiom is helpful in identifying and maintaining property axioms for which positive evidence is inherently harder to find – however, it also allows for the continued integration of incorrect axioms. Here, the combination of the support of selective confirmations and ARI seems to achieve the better results when it comes to introducing the least amount of inconsistencies over time; the downside being that it may erroneously consider valid, but marginal uses of properties as being irrelevant, and therefore the results still need to be analysed on a case-by-case basis. Ultimately, the experimental results show that ARI can be made more resistant to errors in data, benefiting from being combined with other metrics – but more work is needed in this regard to further explore which metrics – and in which way – can be combined in a more systematic fashion that is less dependent of empiric analysis of specific use-cases.

CHAPTER 5

Identifying and Enriching Class Definitions

5. Identifying New Classes and Enriching Class Definitions

In this chapter, we extend *TICO* to identify patterns in property usage that may suggest expanding the definition of existing classes to include more restrictions, or to derive new subclasses of them. This process is done in three main steps: (1) the elicitation of Class Restrictions Hypothesis by individuals provided via RDF streams, (2) using those individuals to confirm or deny previously raised Class Restriction Hypothesis, and (3) building Formal Concept Analysis to build Concept Lattices using the confirmed Class Restrictions Hypothesis to determine which must be added to the ontology. The lattices are built using a modified version of the AddIntent algorithm that considers the logical entailment of the intents. The effectiveness of the solution is explored using a dataset obtained from publicly available sources.

In this chapter, *TICO* is further extended to employ Formal Concept Analysis (FCA) mechanisms to derive potential new classes and to extend existing classes in order to describe the individuals in a stream during a period of time. To do so, individuals are analysed from both intensional and extensional standpoints – and each of their properties will be used to raise, confirm and deny hypotheses about how the individual's current class definition may be extended. This chapter thus aims to ascertain to which degree FCA is suitable to identify the need to add and modify class expressions in an OWL ontology by analysing individual data from an RDF stream, which is both fundamentally incomplete and unbounded. Furthermore, it also aims to identify if checking for the entailment of intents when building the context lattice affects the implications that can be derived from it. The solution is applied to a scenario involving data obtained from public sources, described through means of an ontology that does not include complex class expressions.

In order to answer the research questions **RQ2.2:** *How can we identify new class restrictions?* and **RQ2.3:** *Is it possible to identify the need to have super/subclass*

relationships that were not in the ontology?, this chapter focuses on the two following topics:

- Identifying the different types of class expressions and the patterns in the data they correspond to, in order to properly allow each individual to elicit new class expression hypotheses and confirm/deny previously raised ones. Because different class expressions can be used for different purposes – enrichment of existing classes versus addition of new classes – different methods will be used to evaluate the validity and relevance of the class expression hypothesis and also how they are confirmed or denied by individuals from the stream.
- Combining two different FCA lattices, whose implications will be used for the different purposes: the first lattice provides new classes for the individuals in the stream, which can be reclassified using a reasoner, and the second one, enriching both the new and the existing classes at the end of the timeframe. The AddIntent algorithm for building lattices is adapted to use an inclusion operator based on logical entailment (as provided by a reasoner). This approach is tested against an ontology for which no complex class expressions or subsumption relationships between classes are known *a priori*.

5.1 Background

5.1.1 Formal Concept Analysis

A domain can be described by its objects and their attributes. Formal Concept Analysis can be used to derive a concept hierarchy between objects sharing sets of attributes. For each formal concept (A, B), the set of objects sharing the attributes (A) is called the ***extent***, and the set of attributes shared by the objects (B) is called the ***intent***. A formal context is a triple $\mathcal{K} = (G, M, I)$, in which G is the set of objects, M is the set of attributes and $I \subseteq G \times M$ is the binary relationship denoting the incidence of the attribute in relation to the object, i.e., if an object has an attribute [69]. Furthermore, for the subset of objects $A \subseteq G$ and the subset of attributes $B \subseteq M$, two derivation operators are defined:

- $A' = \{ m \in M \mid (g, m) \in I \ \forall g \in A \}$, the set of all attributes shared by all objects in A.

- $B' = \{ g \in G \mid (g, m) \in I \ \forall m \in B \}$, the set of all objects that share all attributes in B.

The application of which, sequentially, constitutes two closure operators, namely:

- The extent closure: $A \mapsto A'' = (A')'$ for $A \subseteq G$ and
- The intent closure: $B \mapsto B'' = (B')'$ for $B \subseteq M$.

And thus $A \subseteq G$, $B \subseteq M$ imply that $A' = B$ and $B' = A$, i.e., every object in A has every attribute in B and every object not in A is missing one or more of the attributes in B. Similarly, an attribute is not in B if any of the objects in A is missing that attribute.

According to [70], in DL-based ontology engineering, ontologies are often designed following a top-down hierarchical approach to the description of a domain: starting with abstract classes and properties, which are refined and combined into more specific and more specialized ones. This approach is considered *intensional*, as it deals with definitions of domain objects. On the other hand, FCA works from a bottom-up perspective [71]. First, a binary table of the existing objects and attributes of the domain is drawn – the context –, and groups of similar objects can then be aggregated into potential classes. Since this approach is grounded on finding sets of objects sharing the same set of attributes, i.e., the extents, it can be described as *extensional*.

Table 23 depicts a binary table of the incidence relation between objects and attributes of the Pokémon domain – in this case, the classes and properties. The ♥ marker depicts that the object has the attribute, i.e., that the class includes the property in its definition.

Table 23 – Context depicting the incidence relation between classes and properties

		INTENT						
EXTENT		used for	uses	has name	found in	learns move	has type	in TM
	Pokémon			♥	♥	♥	♥	
	MegaEvolution		♥	♥	♥	♥	♥	
	Move			♥	♥		♥	♥
	Item	♥		♥	♥			

Relational Concept Analysis (RCA) builds upon FCA by having the binary tables not only include relationships between objects and individuals, but also between the individuals themselves – effectively reifying the relationships between individuals [71,72]. According to [73], in a formal context, the incidence of a relationship states that it definitively exists, and lack of incidence meaning the relationships does not exist. In DL, due to OWA, however, that is not necessarily the case: just because the relationship is not present, it does not mean that it cannot exist, unless there is some assertion of that impossibility. Nonetheless, FCA finds many useful applications in ontology engineering, enrichment and completion [74–76], helping ontologists discover implications between properties and identifying the need for new classes, class and property hierarchies, and how to best describe groups of individuals according to their shared attributes [76,77].

In trying to prevent Open World Assumption problems with FCA, [73] introduces the concepts of Partial Object Description (*pod*) and Full Object Description (*fod*). A *pod* is a tuple (A, S) that describes an object in terms of the set of attributes it satisfies (A) and the set of attributes it definitively does not satisfy (S) – i.e., A is always completely disjoint of S. The *fod* corresponds to the union A and S. For both cases, there is the assumption that the set of attributes can be known *a priori*. *Pods* and *fods* can be used to refute implications between attributes. The goal of the approach is to materialize all implications between attributes that cannot be refuted; after this process and approval of the implications by the knowledge engineer, the knowledge base is considered complete.

5.1.2 The Class Learning Problem

The Class Learning Problem, or Class Expression Learning (CeL), is defined in [78] as finding the class expression for a target class that is most suited to describe the set of individuals assigned to it. [79] and [73] add to this definition, stating that the class expression must also not classify the individuals that do not belong to the target class. In CeL, refinement operators [80] and reasoners are often used, alongside with heuristics, to determine which class expressions are more useful to describe the individuals of the domain [81,82]. Shorter class expressions are more desirable than more complex ones as a matter of human readability: longer class expressions are harder to decipher and their relevance

and validity more difficult to measure [78]. Additionally, they may prove harder for a reasoner to ascertain if an individual should be classified with their class or not, leading to longer computational periods.

In [78], the authors present CELOE, the Class Expression Learning for Ontology Engineering tool, which is part of the DL-Learner framework [58] and is considered a search-based approach to CeL. Proposed class expressions are tested against the set of individuals for inconsistencies before they are considered acceptable, and the search space for new class expressions is equally dependent on the number of individuals supporting it – the less individuals effectively described by the class expression, the less likely it is to be proposed to the ontologist. Like other FCA-based approaches, as stated in [73], CELOE is fundamentally reliant on an CWA to derive conclusions, while also assuming the set of individuals is both complete and unchanging. ROCES [79], the Robust Class Expression Synthesis, is another such tool, which improves upon search-based approaches like CELEO by using neural networks that can learn from arbitrary numbers of positive examples. However, while synthesis-based approaches are adequate for scenarios in which large datasets are available, search-based approaches are more effective at learning from limited numbers of positive examples [79].

5.1.3 Concept Lattices

All contexts have a corresponding concept lattice. A complete ***lattice*** is a partially ordered set in which every subset of the lattice has both a supremum (or join) and an infimum (or meet) [69]. It is a diagram graph that explicitly orders concepts from the most general (⊤) to the most specific (⊥). While in literature the elements of the lattice are often called concepts, to avoid confusion with the concepts from the ontology – as they are not necessarily the same – and since the lattice is a graph, for the remainder of the document the lattice's concepts will be called ***nodes***.

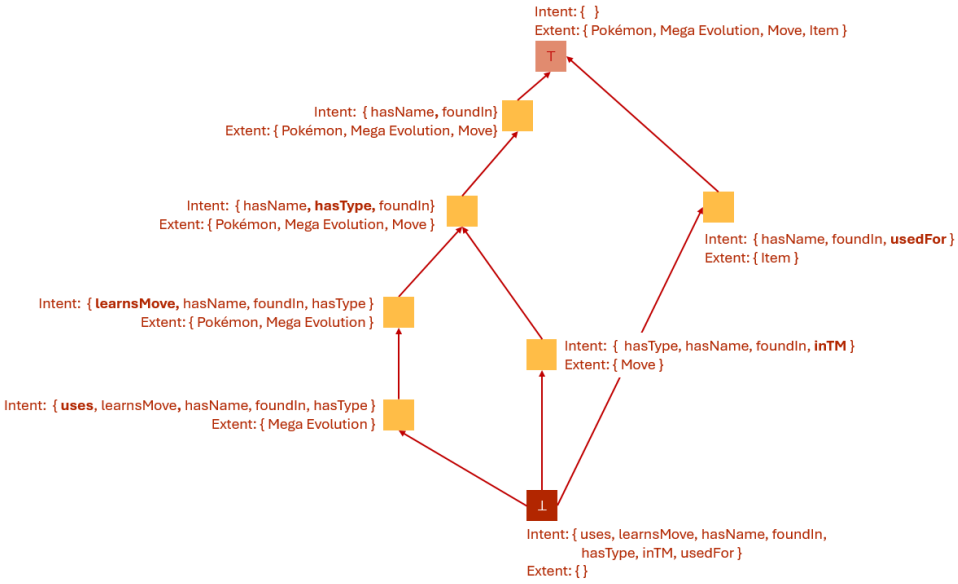

Figure 46 – Lattice corresponding to the context in Table 23

Figure 46 depicts the lattice for the context in Table 23, showing how each node in the lattice "extends" the ones above it with additional attributes. We could interpret, in this case, the extents as representing classes and the intents their properties: and therefore, the relationship between the nodes in the lattice as a subsumption – i.e., that *MegaEvolution* is a subclass of *Pokémon*, and the two of them are subclasses of a potential new class [83]. It also suggests introducing two new concepts corresponding to groups of shared attributes that are not currently used to describe any classes. Alternatively, it is possible to not only use classes and properties to build a context, but RDF individuals as well [83]. In this case, the individuals are the extents and the properties the intents – and the class of the individual is considered another of the individuals attributes. There are several algorithms for building lattices from a formal context, the performance of some has been studied in [84]. Many of the performance issues noted tie themselves to the number of attributes, which directly affect the number of nodes in the lattice, making it harder to navigate when searching in which one to include the objects.

5.1.4 AddIntent

AddIntent [85] is a lattice construction algorithm that is particularly suited for application with streams of objects – in this case, individuals –, as it constructs the lattice iteratively and recursively. Using this algorithm, a lattice can be

transformed into a new one as new objects from the context are evaluated: the algorithm identifies the correct node in which to insert the next object, or otherwise generates a new node for it. For this, it considers four types of nodes: *modified* nodes, *generator* nodes, *new* nodes and *old* nodes[8]:

- A node is considered ***new*** if its intent is not equal to the intent of any existing node in the lattice.
- A node is considered ***modified*** if its extent has been expanded through the introduction of a new object.
- For each new node, there is at least one ***generator*** node, i.e., the node whose intents and extents it expands upon. A new node may have several generators, the most general of which is considered the canonical generator.

The goal of AddIntent is therefore twofold: to identify which nodes to modify with the new object, or identify the canonical generator in case a new node needs to be created – i.e., the most specific generalization that describes the object. Since the nodes are ordered by generality and both $\top$ and $\bot$ are always known, the algorithm can recursively transverse the lattice in a bottom-up fashion to check of the parent nodes of $\bot$ to ascertain if they need to be modified or until it declares a node the generator before introducing a new one.

5.2 Class Expression Learning

This chapter focuses validating general concept inclusion (GCI) for the classes of an evolving ontology. A class C can be described through a set of class expressions $\{Ce_1, \dots, Ce_n\} \in \mathcal{R}$, where these can express necessary conditions $C_1 \sqsubseteq \{Ce_1, \dots, Ce_n\}$, or necessary and sufficient conditions $C_2 \equiv \{Ce_1, \dots, Ce_n\}$ – i.e., the set of class expressions Ce of C can be used to evaluate if any individual meets the necessary or necessary and sufficient conditions to belong to C.

The analysis of the class expressions is comprised of two main steps:

1. The elicitation and evaluation of class expression hypothesis (CeH)
2. The application of FCA to combinations of confirmed Ce in order to establish potential new classes and implications suggested by the data.

[8] In the literature: modified concepts, generator concepts, new concepts and old concepts.

The remainder of this section will present a set of definitions pertaining to the structures and processes involved in this analysis. Figure 47 provides a visualization of the structures involved. These structures include:

- t, the period in which the stream S is analysed and from which to draw conclusions. All the processes described in this section occur within a single t, at the end of which Evolutionary Actions are created.
- The set of facts associated with the current individual i.
- The use of i and its properties to raise new class expression hypotheses $CeH(C)$ regarding each class C of i.
- The classification of i as confirmation or counterexample for each previously raised $CeH(C)$.

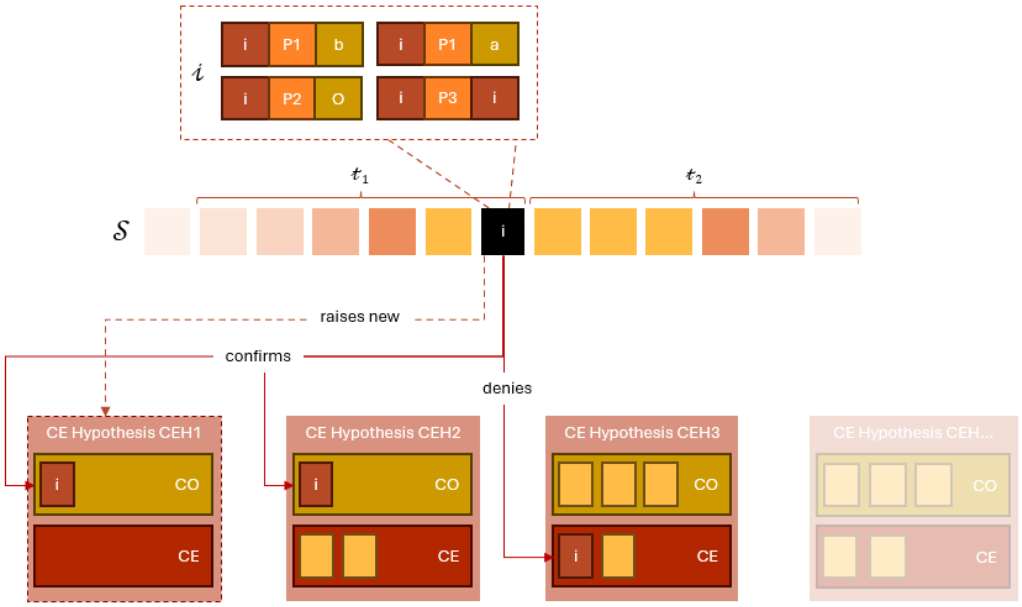

Figure 47 – Data structures concerning the *CeH* elicitation and testing process

CeH are automatically generated through the analysis of the individuals on the stream. Each individual fuels two processes: it raises potential new class expression hypotheses and confirms/denies existing ones.

OWL allows for different **types** of class expressions and class expression axioms, namely:

SomeValuesFrom (SVF)
AllValuesFrom (AVF)
HasValue (HV)
HasSelf (HS)
MaximumCardinality ($maxC$)
MinimumCardinality ($minC$)
ExactCardinality ($exaC$)
Class (C)

which can be combined into more complex class expressions by using the following operators: IntersectionOf (⊓), UnionOf (⊔), ComplementOf (¬) and OneOf (∈). Each of these can be either applied to Object Properties or Datatype Properties. Class expression axioms include the $SubClassOf$, $EquivalentClass$, $DisjointClass$ and $DisjointUnion$. The $Class$ axiom merely states that the class has a specific name and does not express any specific characteristics the individuals that belong to it must have – other than being explicitly stated as belonging to that class. To make the subsequent processes easier to understand, a distinction can be drawn between simple class expressions – composed only by one atomic class expression axiom – and complex class expressions – which result from combining the class expression axioms through one or more of the operators described above.

For the purposes of this work, only the class expressions which are associated with the object properties used by an individual are analysed and then combined through IntersectionOf.

As such, Ce denotes one of the possible OWL class expressions among:

$$Ce \in \{ ObjectSomeValuesFrom, ObjectAllValuesFrom, ObjectHasValue, ObjectHasSelf, ObjectMinCardinality, ObjectMaxCardinality, ObjectExactCardinality \}$$

or, in a more summarized way:

$$Ce \in \{ SVF, AVF, HV, HS, minC, maxC, exaC \}$$

For each i and its set of C, one can elicit a list of Ce that could potentially describe i by analysing its properties. Each Ce can be interpreted as a rule that may or may not be followed by i. A short description of those rules and the patterns in the data associated with each rule can be found at the end of this section. While the different class expressions can be described through means of first order logic implications, these can be separated into three main groups:

- Quantification-Based Class Expressions (SVF and AVF)
- Cardinality-based Class Expressions ($minC$, $maxC$ and $exaC$)
- Value-based Class Expressions (HS and HV).

Assuming each i can raise all the hypotheses in the $Ce(C)$ set, only a subset of them will effectively be confirmed by the individual. Consider two individuals i_1 and i_2 of classes C_1 and C_2 respectively. Should the triple $<i_1, p_1, i_2>$ be the only

property usage present in i_1, the following class expression hypotheses could be raised for C_1:

- $C_1 \sqsubseteq exaC(p_1, 1)$: as the property p_1 is used exactly one (1) time by i_1;
- $C_1 \sqsubseteq SVF(p_1, C_2)$: as p_1 has a range of class C_2 at least once;
- $C_1 \sqsubseteq AVF(p_1, C_2)$: as p_1 always has class C_2 as range, and
- $C_1 \sqsubseteq HV(p_1, i_2)$: as p_1 always has the individual i_2 as range.

Each i raises the set of hypotheses which it confirms and is simultaneously used to confirm or deny previously raised hypotheses.

Universal Quantification

A Universal Quantification $A \sqsubseteq AVF(P, C)$ stipulates that relationships P on individuals of class A must be with individuals of class C.

- **Formula**: $\forall i \{ A(i) \rightarrow \forall b \{ P(i, b) \rightarrow C(b) \} \}$
- **Selective Confirmation** ($\boldsymbol{E^+}$): i of class A that has relationships P only with individuals of class C and of class C only.
- **Counterexample** ($\boldsymbol{E^-}$): i that belongs to class A and has relationships P with at least one individual that does not belong to class C.
- **Assumptions:** CWA is necessary to draw counterexamples, as it may not be possible to determine the class of the individual in the range of the property (i.e. the individuals are not in *sw* simultaneously).

Existential Quantification

An Existential Quantification $A \sqsubseteq SVF(P, C)$ stipulates that individuals of class A must have at least one relationship P with individuals of class C.

- **Formula**: $\forall i \{ A(i) \rightarrow \exists b \{ P(i, b) \wedge C(b) \} \}$
- **Selective Confirmation** ($\boldsymbol{E^+}$): i of class A that has one or more relationships P with individuals of class C.
- **Counterexample** ($\boldsymbol{E^-}$): i that belongs to class A that has no relationships P to individuals belonging to C.
- **Assumptions:** CWA allows establishing that a relationship does not exist when it is not explicitly present, in order to be possible to draw counterexamples.

Individual Value Restriction

An Individual Value restriction $A \sqsubseteq HV(P, b)$ stipulates that individuals of class A must have at least one relationship P with a specific individual b.

- **Formula**: $\forall i \{ A(i) \rightarrow P(i, b) \}$
- **Selective Confirmation** ($\boldsymbol{E^+}$): i of class A that has a relationship P with the individual b.
- **Counterexample** ($\boldsymbol{E^-}$): i of class A that has no relationship P with the individual b.
- **Assumptions:** CWA allows establishing that a relationship does not exist when it is not explicitly present, and therefore makes it possible to draw counterexamples**.**

Self-Restriction

A Self-Restriction $A \sqsubseteq HS(P)$ stipulates that individuals of class A must have at least one relationship P with themselves.

- **Formula**: $\forall i \{ A(i) \rightarrow P(i, i) \}$
- **Selective Confirmation** ($\boldsymbol{E^+}$): i of class A that has a relationship P with itself.
- **Counterexample** ($\boldsymbol{E^-}$): i of class A that has no relationship P with itself.
- **Assumptions:** CWA allows establishing that a relationship does not exist when it is not explicitly present, in order to be possible to draw counterexamples**.**

Minimum Cardinality Restriction

A Minimum Cardinality restriction $A \sqsubseteq minC(P, n)$ stipulates that individuals of class A must have the relationship P at least n times.

- **Formula**: $\forall i \{ A(i) \rightarrow \#\{ b \mid P(i, b) \geq n \}\}$ [9]
- **Selective Confirmation** ($\boldsymbol{E^+}$): i of class A employing P more than or exactly n times.
- **Counterexample** ($\boldsymbol{E^-}$): i of class A employing P less than n times.

[9] Using a notation similar to that employed in [90] and [91].

- **Assumptions:** CWA allows establishing that a relationship does not exist when it is not explicitly present, making it possible to count the number of explicitly used properties and draw counterexamples**.**

Maximum Cardinality Restriction

A Maximum Cardinality restriction $A \sqsubseteq maxC(P, n)$ stipulates that individuals of class A must have the relationship P no more than n times.

- **Formula**: $\forall i, b \{ A(i) \rightarrow \#\{ b \mid P(i, b) \leq n\} \}$
- **Selective Confirmation** ($\boldsymbol{E^+}$): i of class A employing P less than or equal to n times.
- **Counterexample** ($\boldsymbol{E^-}$): i of class A employing P more than n times.
- **Assumptions:** CWA allows establishing that a relationship does not exist when it is not explicitly present, making it possible to count the number of explicitly used properties and draw counterexamples.

Exact Cardinality Restriction

An Exact Cardinality restriction $A \sqsubseteq exaC(P, n)$ stipulates that individuals of class A must have the relationship P exactly n times.

- **Formula**: $\forall i, b \{ A(i) \rightarrow \#\{ b \mid P(i, b) = n \} \}$
- **Selective Confirmation** ($\boldsymbol{E^+}$): i of class A employing P exactly n times.
- **Counterexample** ($\boldsymbol{E^-}$): i of class A employing P in any number that is not n times.
- **Assumptions:** CWA allows establishing that a relationship does not exist when it is not explicitly present, making it possible to count the number of explicitly used properties and draw counterexamples.

The process of classifying an individual as a selective confirmation or as a counterexample of a specific CeH follows the principles established in section **Erro! A origem da referência não foi encontrada.**. Consider again the hypothesis r aised by an individual with the triple <i_1, p_1, i_2> mentioned above, of class C_1. If a new individual with the triple <i_3, p_1, i_3 >, also of class C_1, was introduced, the following new hypotheses could be raised:

- $C_1 \sqsubseteq exaC(p_1, 1)$: as the property p_1 is used exactly one (1) time by i_3;
- $C_1 \sqsubseteq SVF(p_1, C_1)$: as p_1 has a range of class C_1 at least once;
- $C_1 \sqsubseteq AVF(p_1, C_1)$: as p_1 always has class C_1 as range;

- $C_1 \sqsubseteq HV(p_1, i_3)$: as p_1 always has the individual i_3 as range (for uses of p_1 by i_3), and
- $C_1 \sqsubseteq HS(p_1)$: as the subject and object of the triple are the same individual.

As for confirming or denying the previous hypothesis, the following would occur:

- $C_1 \sqsubseteq exaC(p_1, 1)$: i_3 is a confirmation;
- $C_1 \sqsubseteq SVF(p_1, C_2)$: i_3 is neutral, not considered;
- $C_1 \sqsubseteq AVF(p_1, C_2)$: i_3 is counterexample, since i_3 does not belong to C_2 under CWA;
- $C_1 \sqsubseteq HV(p_1, i_2)$: i_3 is a counterexample, as $i_2 \neq i_3$ under UNA.

5.2.1 Eliciting and Confirming Class Expression Hypotheses

Each new individual in the stream is used both to raise new hypotheses for potential new class expressions and compared with previously raised ones – to ascertain its status as confirmation, counterexample or otherwise irrelevant for each of them. Consider the generic task $evaluateClassRestrictionHypotheses$, which continuously adds new hypotheses to a list following each new individual and also categorizes that individual as a confirmation or counterexample of existing hypotheses. This process is described succinctly in Algorithm 3.

For each new individual i provided by the $\mathcal{S}$, the set of properties and classes associated with it are obtained (dP and dC, respectively). For each property P and class C in these sets (individuals may belong to more than one class, and therefore the hypotheses raised must be mirrored for all of them), the individuals in the range of the properties (iR) are obtained – this is necessary in order to establish if the $HS(C)$ and $HV(i, iR)$ hypotheses can be raised. The classes of these range individuals (rR) are also obtained, whenever possible (the individuals may not be in $\mathcal{Sw}$, and thus assessing their class may not be feasible), in order to raise $AVF(C, P, R)$ and $SVF(C, P, R)$ hypotheses. Finally, the number of times each property is used is counted in cdP, allowing the cardinality-based restrictions to be raised. At this point, the class expression hypothesis elicitation phase is complete, and all raised hypotheses are added to the $CeHList_{all}$.

Algorithm 3 – evaluate Class Expression Hypotheses

Input: $\mathcal{S}$, $\mathcal{Sw}$, C, P, $CeHList$
Output: $CeHList_{all}$

```
for each i in 𝒮 do
  if 𝒮𝓌 is full then
    dequeue 𝒮𝓌
  queue i to 𝒮𝓌;

  dP ← get distinct properties in i;
  dC ← get distinct classes in i;

  for P in dP do
    for C in dC do
      iR ← get ranges of P;
      add HasSelf(C) and HasValue(C,iR) to CeHList_all;
      rR ← get classes of iR;

      for R in rR do
        add SomeValuesFrom(C,P,R) and AllValuesFrom(C,P,R) to CeHList_all;

      cdP ← get number of P in dP;
      add minCardinality(C,P,cdP), maxCardinality(C,P,cdP) and
                                   exaCardinality(C,P,cdP) to CeHList_all;

  for CeH in CeHList_all do
    for C2, P2 in CeH do
      for C1, P1 in i do
        if P = P1 and C = C2 then
          if i confirms CeH then
            increase E+ of CeH;
          else
            increase E− of CeH;
```

In the second phase of the algorithm, the class expression hypotheses in $CeHList_{all}$ are iterated upon to be either confirmed or denied by i. For each CeH that is relevant for the individual (by concerning its classes and properties), the algorithm checks if the constraints of those CeH are followed by the individual. The specific validation methods differ according to the type of CeH, and will be described in the next section. Importantly, cardinality-based CeH regarding the class C and the property P can be updated in this stage: the minimum cardinality may be lowered, if necessary, the maximum cardinality raised, and the exact cardinality discarded if the minimum and maximum cardinalities are not the same. The output of the algorithm is the continuously updated list of class expression hypotheses $CeHList$. E^+ and E^- reflect, respectively, how many

individuals are categorized as confirmations and counterexamples of each CeH in $CeHList$ during t.

E^+ and E^- are used differently depending on the type of CeH they pertain to. When dealing with quantification or cardinality-based restrictions, a single counterexample is sufficient to repel the hypothesis permanently – these restrictions are useful to expand the definitions of existing classes.

On the other hand, ***value-based*** restrictions are particularly useful for indicating the need for new classes that describe groups of individuals with similar characteristics: potential new subclasses for existing classes. Therefore, it does not make sense to exclude these hypotheses on the bases of there being counterexamples for them: the goal is not to find a definition that suits all individuals of the existing class, but to find one that aggregates one or more subsets of them into new subclasses.

However, if all $HasValue$ restrictions are always allowed, it can easily result in a very high number of CeH being generated at any moment – consider the example given before, in which a single individual using only one property raises 5 CeH and confirms 4. This situation can easily be worsened – for example, if individuals have unique identifiers or timestamps – each individual would then raise its own $HasValue\ CeH$, not only resulting in a high number of them but also potentially leading to a case of overfitting, with each new class describing only one individual. Furthermore, since the subsequent processes require performing combinations of CeH, a higher number of them would lead to longer computational periods, which are unsuitable for dealing with near real-time information.

To counter this, a couple metrics have been defined for $HasValue$ hypotheses that consider the variability of the individuals in the range of the property being analysed, namely:

- ***Percentage with Value*** ($perc_{iv}$): We establish the total number of times a property is used by the individuals of a given class (tuc_p) as:

$$tuc_p(C, P) = |\{ (i, y) \mid C(i) \wedge P(i, y) \}|$$

and the number of times the property uses the specific individual value (n_{iv}) as:

$$n_{iv}\ (C, P, iR) = |\{ i \mid\ C(i) \wedge P(i, iR) \}|$$

Therefore, $perc_{iv}$ is the ratio between n_{iv} and tuc_p:

$$perc_{iv}\,(C, P, iR) = \frac{n_{iv}\,(C, P, iR)}{tuc_p(C, P)}$$

- ***Property Ratio*** (pr_{iv}): we define $distinct_{iv}$ as the total distinct individual values the property can take when used by individuals of a given class:

$$distinct_{iv}\,(C, P) = |\,\{\, iR \mid C(i) \wedge P(i, iR) \,\}\,|$$

the property ratio of a property P is therefore obtained by:

$$pr_{iv}\,(C, P) = \frac{1}{distinct_{iv}\,(i, C, P)}$$

Each metric has its own minimum and maximum thresholds that establish the desired variability of its individual values. A $HasValue$ class expression hypothesis is only valid if:

$$min\_perc_{iv} < perc_{iv} < max\_perc_{iv}$$

and, simultaneously, if:

$$min\_pr_{iv} < pr_{iv} < max\_pr_{iv}$$

i.e., the property must have several different individual values but the specific individual value under scrutiny needs to be sufficiently frequent. Should the property have the same individual value too often or too rarely, it is not considered relevant for aggregation purposes; if the individual value meets the frequency requirements, but there are too many or too few different possibilities of values, then it is also not statistically relevant and the hypothesis regarding it is not raised. The goal is to avoid creating new classes with very small numbers of individuals, and to discard $HasValue$ class expressions that apply to the vast majority of the individuals – avoiding the creation of a new class that describes most if not all individuals of an already existing class.

To make these metrics easier to understand, consider the set of the three individuals of class $Pokémon$ seen in Figure 48. Consider as well the threshold values of $min\_perc_{iv} = 0.25$, $max\_perc_{iv} = 0.6$, $min\_pr_{iv} = 0.2$ and $max\_pr_{iv} = 0.4$.

Bulbasaur	:learnsMove	"Toxic",
	:hasPkmnType	"Grass",
	:hasPkmnType	"Poison",
	:hasColor	"green".
Oddish	:learnsMove	"Toxic",
	:learnsMove	"Absorb",
	:hasPkmnType	"Grass",
	:hasPkmnType	"Poison",
	:hasColor	"blue".
Venonat	:learnsMove	"Toxic",
	:learnsMove	"Confusion",
	:hasPkmnType	"Bug",
	:hasPkmnType	"Poison".

Figure 48 – Three individuals and their property values[10]

The property $learnsMove$ has five uses, and three distinct values: "Toxic", "Absorb" and "Confusion". From this, we can calculate the following:

- $pr_{iv}\,(Pokémon, learnsMove) = \frac{1}{3} \cong 0.33$
- $tuc_p(Pokémon, learnsMove) = 5$
- For the range values of $learnsMove$:
  - For "Toxic":
    - $n_{iv}(Pokémon, learnsMove, Toxic) = 2$
    - $perc_{iv}\,(Pokémon, learnsMove, Toxic) = \frac{2}{5} = 0.4$
  - For "Confusion":
    - $n_{iv}(Pokémon, learnsMove, Confusion) = 1$
    - $perc_{iv}\,(Pokémon, learnsMove, Confusion) = \frac{1}{5} = 0.2$
  - For "Absorb":
    - $n_{iv}(Pokémon, learnsMove, Absorb) = 1$
    - $perc_{iv}\,(Pokémon, learnsMove, Confusion) = \frac{1}{5} = 0.2$

Of these, the property value "Toxic" is the only within the defined ranges for $perc_{iv}$ and pr_{iv}, and the remaining property values do not reach the minimum threshold required for $perc_{iv}$. As such, only the $Pokémon \sqsubseteq HasValue(learnsMove, Toxic)$ class expression hypothesis is raised, and the other two are dismissed.

[10] The images are from the *Bulbagarden Archives*, and all under a Fair Use license:
https://archives.bulbagarden.net/wiki/Category:Venonat
https://archives.bulbagarden.net/wiki/Category:Bulbasaur
https://archives.bulbagarden.net/wiki/Category:Oddish
https://archives.bulbagarden.net/wiki/Category:Game_fair_use_images

For the property $hasColor$, there are 2 different uses and 2 distinct property values: "green" and "blue". Similar to the previous example, the following can be computed:

- $pr_{iv}\ (Pokémon, hasColor) = \frac{1}{2} \cong 0.5$
- $tuc_p(Pokémon, hasColor) = 2$
- For the range values of $hasColor$:
  - For "green":
    - $n_{iv}(Pokémon, learnsMove, green) = 1$
    - $perc_{iv}\ (Pokémon, learnsMove, Grass) = \frac{1}{2} = 0.5$
  - For "blue":
    - $n_{iv}(Pokémon, learnsMove, blue) = 1$
    - $perc_{iv}\ (Pokémon, learnsMove, blue) = \frac{1}{2} = 0.5$

In this case, both property values represent half of the uses of the property. While the $perc_{iv}$ for both "green" and "blue" property values are within the allowed range, their pr_{iv} is not, and the $HasValue$ class expressions regarding them are not raised.

5.2.2 Using Formal Concept Analysis to create Class Expression Hypotheses Lattices

Following the reasoning employed in [83], the intents corresponds to the set of simple Ce (combined through $IntersectionOf$ and including the asserted and entailed classes) confirmed by an individual, while the extents are the individual themselves, as depicted in Figure 49. Individuals whose class expressions are equivalent are grouped under the same intent. For this, the AddIntent algorithm was modified: instead of checking for intent inclusion when looking for a generator, it checks if the new intent can be entailed by the intents of existing nodes using a reasoner[11]. Furthermore, it also assesses if an intent is self-redundant: i.e., the intent can be reduced to a minimum set of Ce that cannot be derived from each other. However, it is important to note that these

[11] The logical entailment of a class expression depends on the expressivity level the reasoner exploits. As such, different reasoners may select the generator nodes differently, resulting in different lattices.

modifications substantially affect the performance of the algorithm, as reasoning is quite computationally expensive.

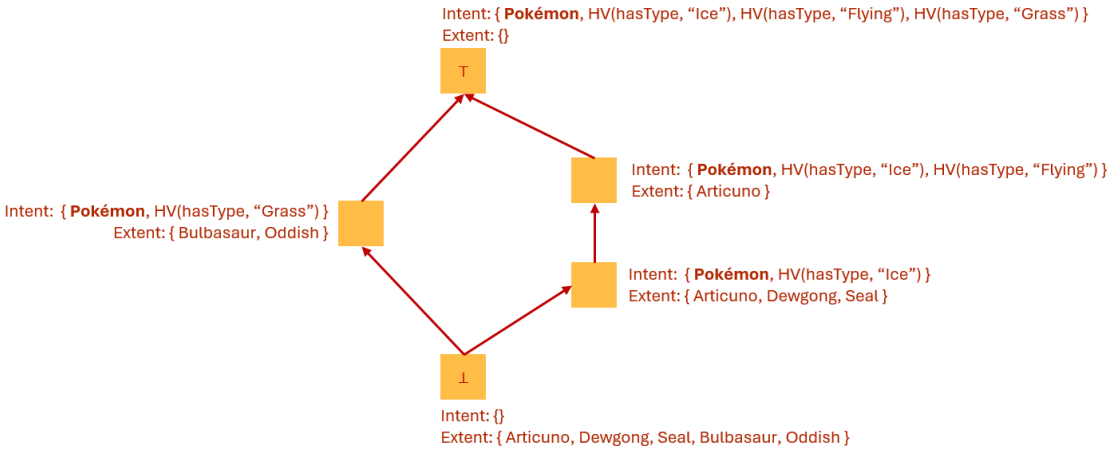

Figure 49 – Lattice formed using confirmed Ce as the intents and individuals as the extents

To prove the feasibility of the solution and avoid excessively long computational periods, only $AllValuesFrom$, $SomeValuesFrom$ and $HasValue$ class expression hypothesis will be considered in the experiments that follow. The other class expressions previously mentioned are processed fundamentally in the same way – the only change being the algorithm to ascertain their status as selective confirmations or counterexamples – and therefore can be omitted to streamline the remainder of this chapter.

A lattice can then be used to derive implications with the form $Antecedent \rightarrow Consequent$, meaning that the candidate GCI $Antecedent \sqsubseteq Consequent$ is supported by the data. Considering the ordered nature of the lattice, this means that a given node implies, recursively, the concepts above it. This can be exploited to infer potential new axioms to add to the ontology in different ways, as will be described next. Each implication also has a support, which is the size of the extent of the antecedent, which can be used to determine the value (as in, the relevance) of the implication and if it is worth materializing.

The three main steps of the solution are illustrated in Figure 50.

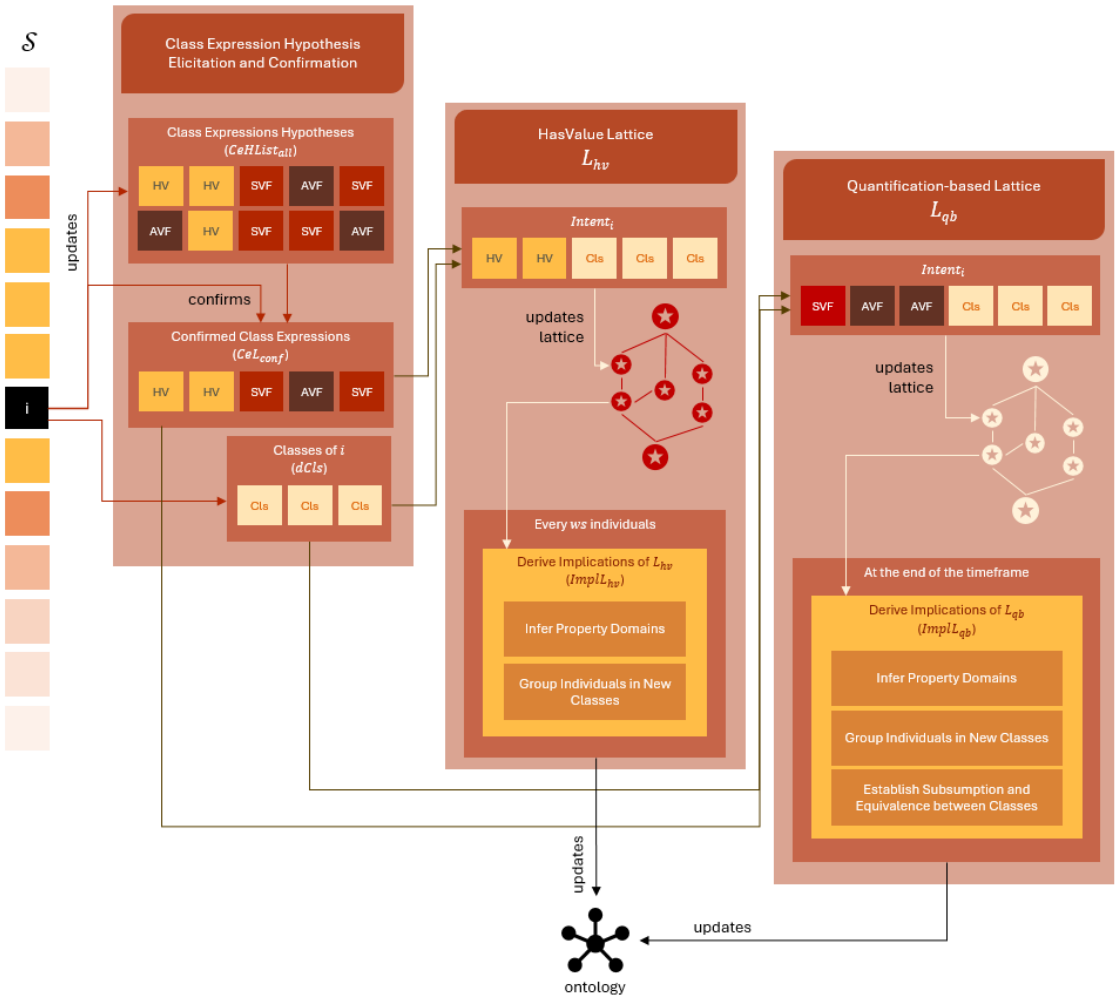

Figure 50 – The three steps of the solution: the CeH elicitation and confirmation, using $HasValue$ type Ce to populate the L_{hv} lattice and using SVF and AVF Ce to populate the L_{qb} lattice

The arrival of a new individual triggers the CeH elicitation processes and evaluates the subset confirmed by the individual. The list of classes of that individual is computed and added to the confirmed Ce to form the new intents to be added to the lattices. Since the different types of Ce will be used for different ends, the solution uses one lattice for each purpose:

1. The first lattice, L_{hv}, uses only class expressions of the type $HasValue$. This lattice is used to derive implications periodically that may be used for:
   i. Ascertaining new classes based on groups of individuals with the same $HasValue$ restriction
   ii. Search for potential domain and range restrictions for properties

2. The second lattice, L_{qb}, uses Quantification-based Ce – $AllValuesFrom$ and $SomeValuesFrom$ – to enrich the definition of existing classes by adding new $SubClassOf$ axioms to their Ce.

The difference in the periodicity in which implications are derived (and materialized) from each lattice ensures that the partial results obtained by L_{hv} – i.e., the new classes – can be used and enriched the L_{qb}as part of more complex dynamics between classes.

5.2.2.1 L_{hv}, the Has Value Lattice

Even though it is possible to derive implications from the lattices after each individual addition and its respective node found, this leads to an excessive computational overhead – and many of the implications would not have significant support, since not enough individuals would have been analysed. As such, implications are not derived every time a new individual arrives from $\mathcal{S}$, but periodically after a number of individuals has been analysed. In the following experiments, this is done once every w_s individuals: a number equal to that of the sliding window size previously used, as seen in Algorithm 4.

Upon the arrival of a new i, the individual count ($total_{inds}$) is raised. This number is relevant to assess when the implications should be derived from the lattice and materialized into new ontology axioms. Then, a reasoner is used to obtain the updated list of classes that describe the individual, $dCls$. For each class C in this list, the algorithm searches $CeHList_{all}$ for the CeH that are both valid and relevant for this case (the $HasValue$ types confirmed by i). The algorithm that executes this task is described in Algorithm 5. The intent to be added to the lattice is comprised of the intersection of all confirmed class expression hypothesis and the current classes of i. This conjunction is added to the L_{hv} lattice using the AddIntent algorithm. Finally, if $total_{inds}$ is divisible by w_s, implications are derived from the lattice and materialized into new ontology axioms following the procedures in Algorithm 6 and Algorithm 7.

Algorithm 4 – Processing and adding a new individual to L_{hv}

```
Input: ontology, S, Sw, w_s, L_hv, CeHList_all
Output: ImplL_hv, ontology (evolved)

for each i in S do
  increase total_inds

  if Sw is full then
   dequeue Sw
  queue i to Sw

  dCls ← get all classes that can currently describe i
  intent_i ← obtain CeH from CeHList_all and confirm HasValue Ce with i
  intent_i ← IntersectionOf(dCls, intent_i)
  addIntent( intent_i , L_hv )

  if total_inds % w_s = 0 then
   ImplL_hv ← get implications from L_hv
   materialize property domain axioms from ImplL_hv
   derive new classes from ImplL_hv
```

Algorithm 5 – Confirming *HasValue* type hypotheses

```
Input: i, dCls, CeHList_all
Output: CeL_conf

dCls ← Reduce(dCls)[12]

for each cls in dCls do
  ListPN ← get properties used by i and their values
  CeHList_cls ← get all possible CeH for cls from CeHList_all

 for each CeH in CeHList_cls do

  Type ← get the type of CeH

  if Type one of {DataHasValue, ObjectHasValue} then
      for each Prop, NodeVal in ListPN do
        CeH_P ← get the property in CeH's signature
        CeH_nv ← get the node or individual value in CeH's signature
        if CeH_P = Prop and CeH_nv = NodeVal then
          add CeH to CeL_conf
```

Algorithm 5 describes the process of confirming/denying a $HasValue$ type CeH by an individual i. To do this, it is first necessary to obtain the set of classes that describe i ($dCls$), both asserted and entailed – this is done using a reasoner,

[12] The Reduce algorithm simplifies a class expression. The description of this algorithm is provided in Annex 2 – Reducing.

ensuring that even the new classes that may have been added to the ontology in previous iterations are identified. Additionally, the list of property-value pairs $(Prop, NodeVal)$ in i is gathered in $ListPN$. Using the list of valid class expressions hypothesis $CeHList_{all}$ as input (the output of Algorithm 3), the subset of class expressions that relate to each class in $dCls$ is obtained, and further filtered so that only CeH of type $HasValue$ are evaluated. Then, the signatures of these class expressions are compared with each $(Prop, NodeVal)$ pair in $ListPN$: all elements of that are of the $HasValue$ type (either $DataHasValue$ or $ObjectHasValue$) and have identical property-value pairs to those of the individual are added to the list of confirmations (CeL_{conf}). The output of the algorithm, therefore, is a list of Ce of type $HasValue$ that have been confirmed by the individual and can thus be added to L_{hv}.

As previously mentioned, the lattice L_{hv} is periodically checked for new implications. These can be materialized into different ontology axioms in two different processes: ***inferring domains of properties*** and ***creating new classes*** pertaining to groups of individuals with similar characteristics.

Inferring Property Domains

As for inferring domains of properties, the set of implications derived by the lattice, $ImplL_{hv}$, is checked for hints that certain properties should have specific domains: i.e., if the same property is always implied by a particular class. Consider, for example, that the lattice generates the following implications:

$$DataHasValue(hasContestCategory, \text{"tough"}) \rightarrow PokemonMove$$
$$DataHasValue(hasContestCategory, \text{"cool"}) \rightarrow PokemonMove$$
$$DataHasValue(hasContestCategory, \text{"smart"}) \rightarrow PokemonMove$$

Since all uses of the $hasContestCategory$ property come from individuals of class $PokemonMove$, it is possible to conclude the property should have that class as its domain, and therefore the axiom $Domain(hasContestCategory, PokemonMove)$ could be added to the ontology if it does not introduce any inconsistencies. The idea is described in Algorithm 6. To make the algorithm easier to understand, no distinction is made between $ObjectProperties$ and $DatatypeProperties$, as the reasoning is similar. Here, some conditions must be met for the implication triple $(support, antecedent, consequent)$ to be considered:

- The *support* must be above a minimum threshold, the min_{sup}
- Both *antecedent* and *consequent* need to be simple class expressions (i.e., using only one atomic Ce each).
  - Furthermore, the *consequent* must be a simple class expression of type $Class$
- Both the *antecedent* and *consequent* must be nodes of the lattice with non-empty extents.

Should these conditions be met, the property P is obtained from the *antecedent*'s signature and the class Cls from the *consequent*. This process is done for all triples in $ImplL_{hv}$ as long as there is only one Cls for each P: if no other values of Cls are found, the class is considered the domain of P and the axiom $Domain(P, C)$ is added to the ontology.

Algorithm 6 – Obtaining potential property domains from Implications

Input: min_{sup}, $ImplL_{hv}$, $ontology$
Output: $ontology$ (evolved)

for each $support, antecedent, consequent$ **in** $ImplL_{hv}$ **do**

  $consequent_{size}$ ← get number of Ce in $consequent$
  $antecedent_{size}$ ← get number of Ce in $antecedent$
  $Type$ ← get the type of first Ce in $antecedent$

  if $support \geq min_{sup}$ **and** $consequent_{size} = 1$
    and $antecedent_{size} = 1$ **and** $Type$ **is** $Class$ **then**

    P ← get property in $antecedent$ signature

    if $consequent$ is the same for all $ImplL_{hv}$ with P in the $antecedent$ **do**
      Cls ← get class in $consequent$ signature

      if $Domain(P, Cls)$ **not entailed by** $ontology$ **do**
        add $Domain(P, Cls)$ axiom to $ontology$

Grouping Individuals under new Classes

Finally, and perhaps more importantly, the implications in $ImplL_{hv}$ are scrutinized for groups of potential individuals that have the same L_{hv} implications, suggesting that they may appear together or be classes of their own. Algorithm 7 describes the process of ascertaining the potential new classes and their subsumption relationships. Much like in Algorithm 6, an implication triple $(support, antecedent, consequent)$ must reach a minimum support min_{sup} to be

considered. Furthermore, implications are materialized differently depending on their characteristics:

- If both *consequent* and the *antecedent* are of type *Class*, add the axiom $SubClassOf(antecendent, consequent)$ to the ontology
- If the *antecendent* is not of type *Class*, a new class must be created that is equivalent to the *antecendent*. This new class, *newAnteCls*, can then be added as a subclass of the *consequent*.
- If the *consequent* is not of type *Class*, but the *antecendent* is, create a new equivalent class that describes the *consequent*, *newConsCls*, and add the axiom $SubClassOf(antecendent, newConsCls)$
- If both *antecendent* and *consequent* are not of type *Class*, create new classes equivalent to them (*newAnteCls* and *newConsCls*, respectively). A new $SubClassOf(newAnteCls, newConsCls)$ axiom can then be added to the ontology.

Algorithm 7 – Materializing antecedent and consequent subclasses

Input: min_{sup}, $ImplL_{hv}$, *ontology*
Output: *ontology* (evolved)

for each $support, antecedent, consequent$ **in** $ImplL_{hv}$ **do**
  $consequent \leftarrow Reduce(consequent)$
  $antecedent \leftarrow Reduce(antecedent)$

  $consequent_{size} \leftarrow$ get number of *Ce* in *consequent*
  $antecedent_{size} \leftarrow$ get number of *Ce* in *antecedent*

  $Type_{cons} \leftarrow$ get the type of first *Ce* in *consequent*
  $Type_{ante} \leftarrow$ get the type of first *Ce* in *antecedent*

  if $support \geq min_{sup}$ **and** $consequent_{size} = 1$ **and** $Type_{cons}$ **is** *Class* **then**
   if $antecedent_{size} = 1$ **and** $Type_{ante}$ **is** *Class* **do**
     add $SubClassOf(antecendent, consequent)$ axiom to *ontology*
   else
     create new class *newAnteCls* in *ontology*
     add $EquivalentClass(antecendent, newAnteCls)$ axiom to *ontology*
     add $SubClassOf(newAnteCls, consequent)$ axiom to *ontology*
  else
     create new class *newConsCls* in *ontology*
     add $EquivalentClass(consequent, newConsCls)$ axiom to *ontology*

     if $antecedent_{size} = 1$ **and** $Type_{ante}$ **is** *Class* **do**
       add $SubClassOf(antecendent, newConsClass)$ axiom to *ontology*
     else
       create new class *newqAnteCls* in *ontology*
       add $EquivalentClass(antecendent, newAnteCls)$ axiom to *ontology*
       add $SubClassOf(newAnteCls, newConsClass)$ axiom to *ontology*

To make Algorithm 7 easier to understand, consider the following example. The implication:

$$DataHasValue(hasDamageCategory, "physical") \rightarrow PokemonMove$$

shows that all individuals with the value *"physical"* for the datatype property *hasDamageCategory* belong to the class $PokemonMove$. If this implication has sufficient support, it proposes the creation of a new class – and that class would be described by the $DataHasValue\ (hasDamageCategory, "physical")$ class expression.

As such, three new axioms can be added to the ontology, namely:

Axiom 1: $Class(dmgCat_{physical})$

Axiom 2: $EquivalentClasses(dmgCat_{physical},$
$DataHasValue(hasDamageCategory, "physical"))$

Axiom 3: $SubClassOf(dmgCat_{physical}, PokemonMove)$

in which: Axiom 1 declares the new named class $dmgCat_{physical}$, Axiom 2 declares that $dmgCat_{physical}$ is equivalent to the $antecendent$ of the implication, and Axiom 3 states the new class is a subclass of the $consequent$.

If neither the $antecendent$ or $consequent$ are of type $Class$, it is necessary first to create new classes equivalent to each expression and then establish the $subClassOf$ relationship between them. For example, in the implication:

$$ObjectHasValue(hasMoveType, Fire) \rightarrow DataHasValue(hasDamageCategory, "special")$$

results in more axioms being added to the ontology, namely:

Axiom 1: $Class(MvType_{Fire})$

Axiom 2: $EquivClass(MvType_{Fire},\ ObjectHasValue(hasMoveType, Fire))$

Axiom 3: $Class(dmgCat_{spec})$

Axiom 4: $EquivClass(dmgCat_{spec},$
$DataHasValue(hasDamageCategory, "special"))$

Axiom 5: $SubClassOf(MvType_{Fire}, dmgCat_{spec})$

5.2.2.2 L_{qb}, the Quantification Lattice

The L_{qb}, built with the Quantification-based CeH and unlike L_{hv}, is only used to derive implications at the end of the timeframe. On top of applying the algorithms to generate new axioms described before, its purpose is also to enrich

existing classes – including those generated by L_{hv} – and to find potential subsumption relationships between them. First, to obtain the intents related to each new individual i, it is necessary to get its set of confirmed class expressions, CeL_{conf}. As seen in Algorithm 8, for each class C in the set of classes that describe i (the $dCls$), the list of property-value pairs of i is condensed in $ListPN$. $CeHList_{cls}$ is the list of CeH pertaining to cls.

From here on, the process differs depending on the type of each CeH:

- If the type of the CeH is $SomeValuesFrom$, then at least one of the property-value pairs of i must match the property and range class in the signature of the CeH
- If the type of the CeH is $AllValuesFrom$, then all the property-value pairs of i must match the property and range class in the signature of the CeH

If these conditions are met, the CeH is added to the list of confirmed class expressions, CeL_{conf}, the output of the algorithm.

Algorithm 8 – Confirming AVF and SVF hypotheses

Input: $i, dCls, CeHList_{all}$
Output: CeL_{conf}

$dCls \leftarrow Reduce(dCls)$
for each C **in** $dCls$ **do**
  $ListPN \leftarrow$ get properties used by i and their values
  $CeHList_{cls} \leftarrow$ get all possible CeH for C from $CeHList_{all}$

 for each CeH **in** $CeHList_{cls}$ **do**
  $Type \leftarrow$ get the type of CeH

  if $Type$ **is** $SomeValuesFrom$ **then**
    for each $Prop, NodeVal$ **in** $ListPN$ **do**
      $CeH_P \leftarrow$ get the property in CeH's signature
      $CeH_{rcls} \leftarrow$ get the range class in CeH's signature
      if at least one $CeH_P = Prop$ **and** $CeH_{nv} = NodeVal$ **then**
        add CeH to CeL_{conf}

  if $Type$ **is** $AllValuesFrom$ **then**
    for each $Prop, NodeVal$ **in** $ListPN$ **do**
      $CeH_P \leftarrow$ get the property in CeH's signature
      $CeH_{rcls} \leftarrow$ get the range class in CeH's signature
      if all $CeH_P = Prop$ **and** $CeH_{nv} = NodeVal$ **then**
        add CeH to CeL_{conf}

As for enriching existing classes with new Ce, the process is relatively simple. The reasoning is formalized in Algorithm 9: if the $antecedent$ is a simple Ce of type $Class$, a $SubClassOf(antecedent, consequent)$ axiom can be added to the ontology, regardless of the complexity of the $consequent$. However, if the

opposite implication is also present (i.e., $consequent \rightarrow antecedent$ and $antecedent \rightarrow consequent$ are both in the list of implications $ImplL_{qb}$, then the $SubClassOf$ axiom is replaced with an $EquivalentClass$ one.

Consider the implication $AlternativeForm \rightarrow Pokemon$. In this case, since all individuals of class $AlternativeForm$ are also of class $Pokemon$ (but not all $Pokemon$ are necessarily of class $AlternativeForm$), it is possible to establish that the axiom $SubClassOf(AlternativeForm, Pokemon)$ should be added to the ontology. If $Pokemon \rightarrow AlternativeForm$ also happened to be present, then the classes would, in fact, be equivalent, as they both imply each other. Note that only simple antecedents are considered whenever the antecedent is not of type $Class$. Otherwise, creating new classes equivalent to the complex Ce would be required, and the goal of L_{qb} is only to extend existing classes.

Algorithm 9 – Enriching class definitions

Input: min_{sup}, $ImplL_{qb}$, $ontology$
Output: $ontology$ (evolved)

for each $support, antecedent, consequent$ **in** $ImplL_{qb}$ **do**

  $consequent \leftarrow Reduce(consequent)$
  $antecedent \leftarrow Reduce(antecedent)$
  $Type_{ante} \leftarrow$ get the type of first Ce in $antecedent$

  if $antecedent_{size} = 1$ **and** $Type_{ante}$ **is** $Class$ **then**

    if $consequent \rightarrow antecedent$ **in** $ImplL_{qb}$ **do**
      add $EquivalentClass(antecendent, consequent)$ axiom to $ontology$
    else
      add $SubClassOf(antecedent, consequent)$ axiom to $ontology$

5.3 Experiments and Results

There are some notions to consider regarding the experimentation process for CeL using FCA that make it harder to compare the results with the classes and class expressions of existing ontologies. First, consider that there is only a limited set of class expressions being evaluated ($HasValue$, $AllValuesFrom$ and $SomeValuesFrom$), which do not represent the full set of class expressions commonly used in more formal ontologies. Secondly, only conjunctions of class expressions are being considered, which further limits the potential descriptions that a class can have. As such, the results obtained by *TICO* are not comparable with class expressions from existing axiom-rich ontologies and, therefore, information retrieval metrics cannot be assessed. With this in mind, the experiments described next were executed over a set of RDF individuals following the same ontology as those in section Experiment II: Accommodating for errors in data and the effects of previous knowledge in axiom-inclusion decisions; namely Gen II. This generation was chosen as it contains a relatively small number of Pokémons (251) and Moves (250), while also adding a significant number of properties to the domain in relation to Gen I (e.g. relationships about evolution groups and egg groups, and two different Pokédexes with a significant overlap). This brings the properties to a total of 33, between $DataType$ and $Object$ properties. Among all classes, the ontology features a total of 1311 individuals.

One possible organization of the concepts of the Gen II domain is represented in Figure 51, featuring a total of 14 classes. Consider the dashed line a $subClassOf$ relationship, and the full lines as depicting relationships between classes using $ObjectProperties$. The $DatatypeProperties$ associated with each class are shown within the class, although they may be used by several (e.g. the $hasName$ property is used by all classes, but is omitted for simplicity).

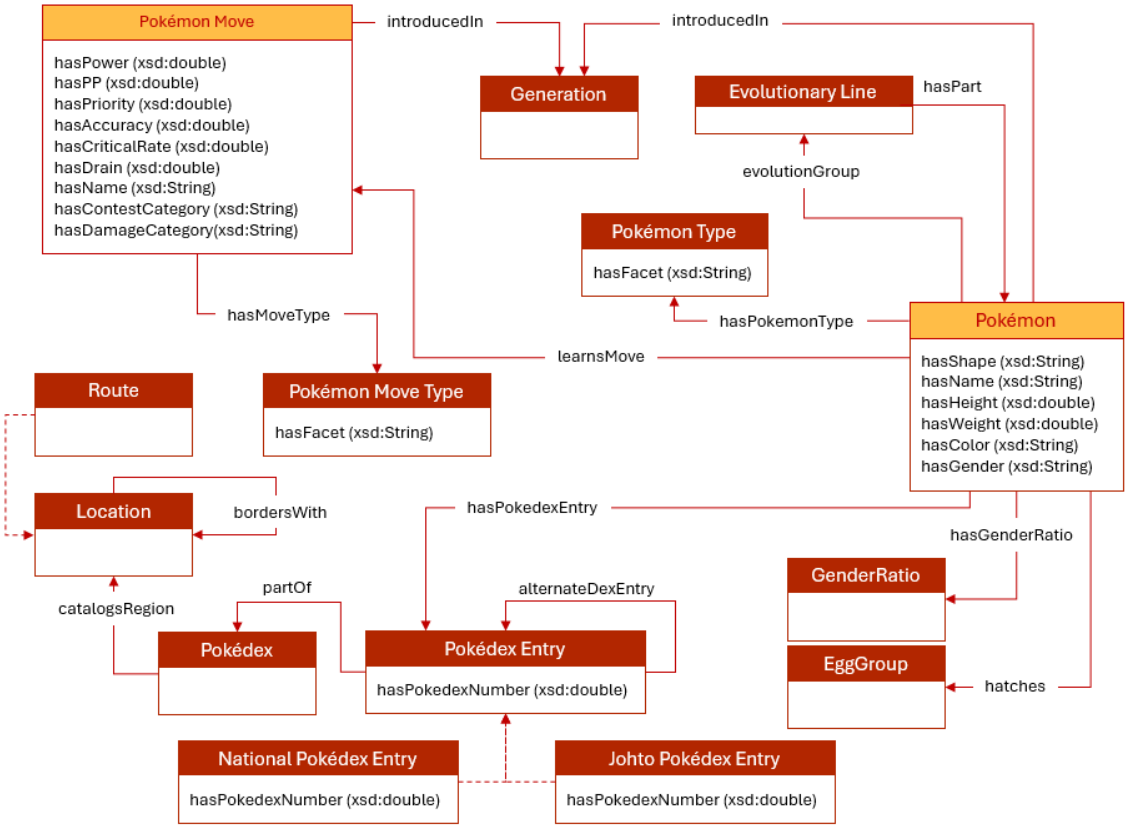

Figure 51 – Possible Gen II Pokémon domain ontology

The initial ontology, however, includes only named classes with no class expressions associated, and no hierarchy between them. Furthermore, the full list of *ObjectProperties* and *DataProperties* can be seen in Table 24. No information regarding their domains and ranges is known *a priori*.

Table 24 – Gen II Pokémon Ontology properties

OBJECT PROPERTIES (15 TOTAL)		DATA PROPERTIES (16 TOTAL)	
hasMoveType	introducedIn	hasPokédexNumber	hasColor
alternateDexEntry	bordersWith	hasAccuracy	hasPower
catalogsRegion	hasPart	hasContestCategory	hasPP
hasPokedexEntry	hasPokemonType	hasCriticalRate	hasPriority
hatches	evolutionGroup	hasDamageCategory	hasShape
genderRatio	learnsMove	hasDrain	hasWeight
locatedIn	partOf	hasFacet	value
presentIn		hasHeight	hasName

Employing the methods described before, it should be possible to not only infer most of the relationships figured, but also find potential new classes that are not yet known – corresponding, for example, to individuals of the class *Pokemon* with similar characteristics, such as a specific *PokemonType* or a value of *hasColor*. Ultimately, it should also be possible to identify even more specific relationships between these (e.g. Pokémon that have *hasColor* "blue" and learn the *Move* "Surf" is always of *PokémonType* "Water").

Experiments will be separated according to two groups:

- Running the lattices for specific classes only: Pokémon (with 251 individuals) and Move (with 250, of which 60 were analysed), as they make use of the biggest number of properties and individual values. Because the classes of this particular domain have little overlap in terms of the properties they use, there is no expectation that the lattices would find interesting new classes resulting from intersections of property use.
- Running the lattices for all classes, over a randomized small sample of individuals from Gen II.

The results obtained by the lattices will be compared to ascertain if there are significant distinctions in terms of performance and implications materialized. Partial results of the lattices will be shown and discussed in order to highlight the iterative nature of the process.

5.3.1 Experiment I – Studying classes independently

The values and thresholds employed in this experiment can be seen below, in Table 25. The lattice snapshot periodicity is used to determine how frequently the lattice, its implications and resulting changes to the ontology are documented.

Table 25 – Values and thresholds employed in the experiments

VARIABLE	DESCRIPTION	VALUE
min_perc_{iv}	Minimum threshold for Percentage with Value	0.05
max_perc_{iv}	Maximum threshold for Percentage with Value	0.45
min_pr_{iv}	Minimum threshold for Property Ratio	0.05
max_pr_{iv}	Maximum threshold for Property Ratio	0.70
min_sup_{impl}	Minimum implication support	3
w_s	Sliding window size	20
p_s	Lattice Snapshot periodicity	20
t	Number of individuals analysed	251/160/116

5.3.1.1 Class Pokémon

For the first experiment and to get a grasp of the capabilities of the combination of the lattices L_{hv} and L_{qb}, the RDF stream will provide only individuals of class Pokémon. Although this would reduce the number of interesting implications found, it should be able to show the ability to identify potential new subclasses of Pokémon and to enrich those and the original ones.

The evolution of the lattices can be seen in Figure 52 and Figure 53, representing the increase in the number of nodes of the L_{hv} and L_{qb} lattices, respectively.

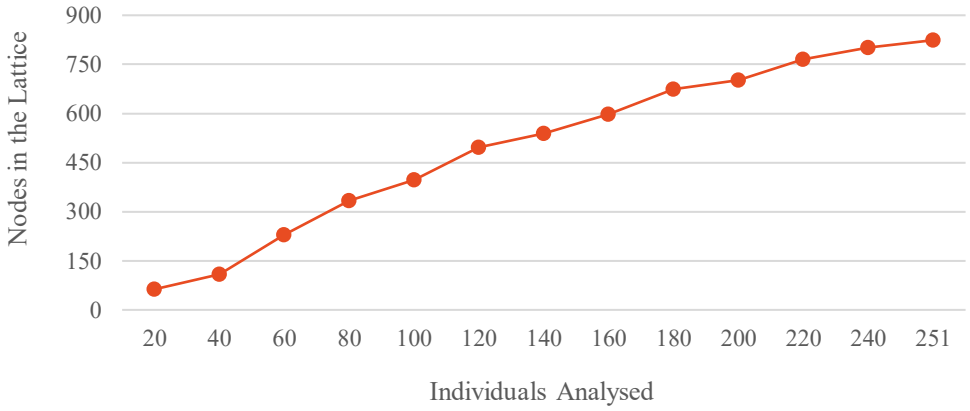

Figure 52 – Evolution of the number of nodes in the L_{hv} lattice for the *Pokemon* class over time

The number of nodes in L_{hv} sees a sharp increase early on, slowing towards the end of the timeframe. The biggest increases in the number of nodes occur

within the first 100 individuals. At the end of the timeframe, the lattice has a total of 824 nodes for 251 individuals analysed.

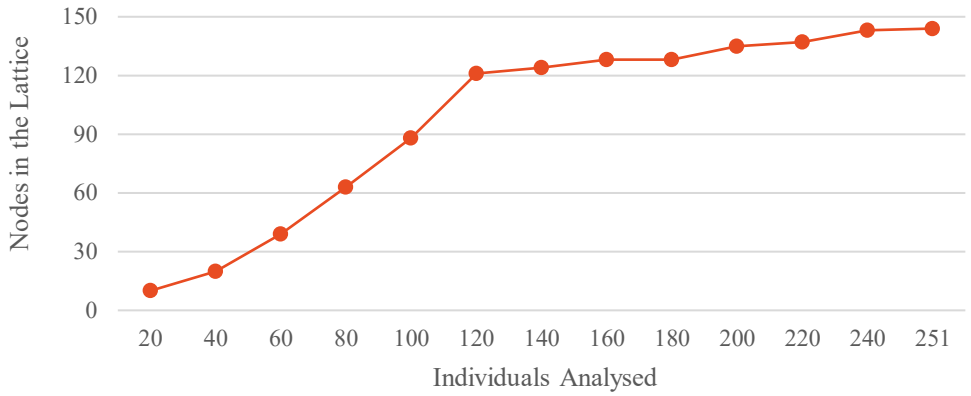

Figure 53 – Evolution of the number of nodes in the L_{qb} lattice for the *Pokemon* class over time

On the other hand, the L_{qb} has a much more modest evolution: starting slowly, increasing in between 10 to 20 nodes per snapshot until it slows down and almost stops growing. At the end of the timeframe, the L_{qb} has 144 nodes.

Figure 54 shows the evolution of number of axioms produced by the implications derived from L_{hv}.

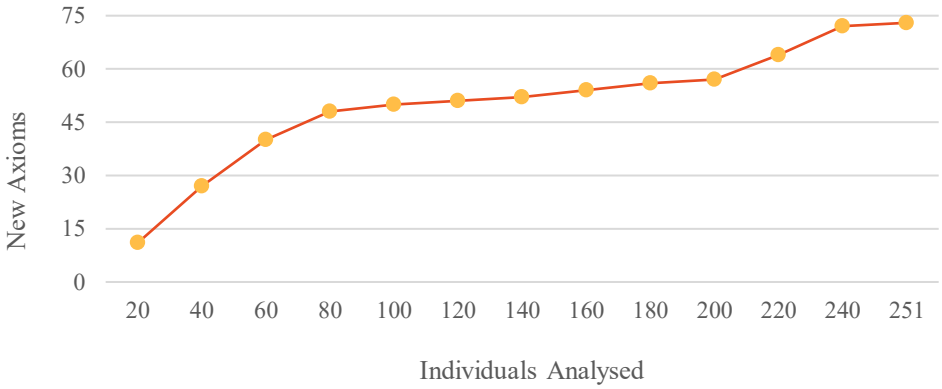

Figure 54 – Number of axioms derived from the implications of the L_{hv} lattice for the *Pokemon* class over time

A total of 73 new axioms are added to the ontology as the result of the implications generated by L_{hv}. After materializing the implications of L_{qb}, an additional 17 axioms were added to the ontology. At the end of the timeframe, 19 new subclasses of *Pokemon* are created. The class expressions associated with a sample of some of them can be seen in Table 26. Note that they are combined exclusively through the *IntersectionOf* operator and the names of the new classes are automatically generated.

Table 26 – Subset of the new *Pokemon* subclasses and their respective class expressions

CLASS EXPRESSIONS
Q3966183_hasColor_Blue
Equivalent to: DataHasValue(hasColor, "blue"), Pokemon
Subclass Of: Pokemon
hatches_Q26040471
Equivalent to: ObjectHasValue(hatches, 'Flying Egg Group')
Subclass Of: Pokemon, ObjectHasValue(hatches, 'Flying Egg Group') Q3966183_hasColor_brown, Pokemon
Q3966183_hasColor_blue_hatches_Q26040640
Equivalent to: Q3966183_hasColor_blue, hatches_Q26040640
Subclass Of: Pokemon, ObjectHasValue(hatches , 'Water 1 Egg Group') SomeValuesFrom(hatches, 'Pokemon Egg Group') Q3966183_hasColor_blue

From the analysis of this small sample, it is possible to see the solution is able to capture complex relationships between some of the original classes and groups identified by L_{hv}.

For example, the class $Q3966183\_hasColor\_blue\_hatches\_Q26040640$ is a new subclass of $Pokemon$ that is equivalent to the intersection of two other classes, describing thus individuals that simultaneously employs $ObjectHasValue(hatches, "Water\ 1\ Egg\ Group")$ and $DataHasValue(hasColor,$ "$blue$"). However, it is important to note that "Water 1 Egg Group" is an instance of the class $Pokemon\ Egg\ Group$, suggesting that the class expression could be further simplified. Very similarly, the class hatches_Q26040471 describes the individuals that hatch the "Flying Egg Group" and are of "brown" colour – which must also always occur simultaneously.

The property values "blue", "red", "pink" and "brown" for the property $hasColor$ have sufficient support to produce new classes. The values "green", "purple" and "yellow", on the other hand, did not meet the variability requirements to be considered. The value "purple", for example, appears early in implications (from the first snapshot onwards) but without sufficient support – and, since from then on it is no longer within the variability range, the support of the implications using it can no longer increase, and it is never materialized.

The domain of $hasColor$ is correctly assigned to the Pokémon class – effectively, the first axiom to be added to the ontology as a result of implications provided by L_{hv}. Interestingly, only the class generated for individuals with the value "blue" ($Q3966183\_hasColor\_blue$ in Table 26) intersects the property value with $Pokemon$. This is possibly because the property and the class appear together in the implication used to generate this class, namely:

$$ObjectHasValue(hatches, "Water\ 1\ Egg\ Group") \rightarrow Pokemon, DataHasValue(hasColor\ "blue")$$

Meanwhile, the remaining colour classes were generated by implications that do not involve an intersection between the $Pokemon$ class and another class expression in the consequent, such as:

$$DataHasValue(hasColor\ "red") \rightarrow Pokemon$$

Two values for the $hasShape$ property were used to materialize new classes, namely: "Pokémon with an insectoid body" and "Pokémon with a bipedal, tailed form". The first property value is reified into a class of its own after 240 individuals are analysed and sufficient support is gathered for the implication:

$$DataHasValue(hasShape, "Pokémon\ with\ a\ bipedal, tailed\ form") \rightarrow Pokemon, ObjectHasValue(pokemonType, FlyingTypePokemon)$$

The second value acquires sufficient variability to show in implications after 160 individuals, but not sufficient to be materialized. This only happens after the analysis of 220 individuals, from which point on there is sufficient support for the implication:

$$DataHasValue(hasShape, "Pokémon\ with\ an\ insectoid\ body") \rightarrow Q3966183\_hatches\_Q26040555$$

in which $3966183\_hatches\_Q26040555$ is equivalent to the class expression $Pokemon\ and\ ObjectHasValue(hatches, Bug\ Egg\ Group)$ – suggesting that all Pokémon with this shape also belong to the Bug Egg Group. Regarding this Egg

Group, it is also noteworthy that after 80 individuals are analysed, there is an implication establishing that all things of Bug Type also hatch the Bug Egg Group:

$$ObjectHasValue(pokemonType, BugTypePokemon) \rightarrow Pokemon, DataHasValue(hatches, "Bug\ Egg\ Group")$$

Since Egg Groups and Pokémon types are not necessarily related (in spite of the naming convention, Pokémon can have multiple types and hatch multiple Egg Groups), this is an interesting discovery, much like the one that can be inferred from the statements in Table 26: *Pokemon* that hatch the "Water 1 Egg Group" are always of colour "blue".

As for the domain axioms added to existing properties, a summary can be seen in Table 27.

Table 27 – Property domain axioms added to the ontology

PROPERTY	DOMAIN
hasShape	Q3966183_hatches_Q26040555
hasColor	Pokemon

While the properties *hatches* and *hasPokemonType* would also be good candidates for having the class *Pokemon* as their domain, they never met the conditions described in Algorithm 6 (either by being in complex class expressions, or subclasses of *Pokemon* been derived as the domain of the property in the meanwhile). Sadly, and because the order of the individuals analysed can affect the results, the property *hasShape* – which should also be of domain *Pokemon* – got its domain axiom introduced via one of *Pokemon*'s subclasses, which would be incorrect. However, by the time the counterexamples arrive, the axiom was already materialized – suggesting that some mechanism to update axioms previously introduced in the ontology should be integrated into the process.

The axioms introduced at the end of the timeframe by the implications of L_{qb} mostly create new subclasses of *Pokemon* where they should have enriched it – meaning that the requirements for class enrichment, as described in Algorithm 9 were not met. For example, the implication below, with a support of 51

individuals, has the $Pokemon$ class intersect with another class expression in the consequent:

$$Q3966183\_hasColor\_blue \rightarrow Pokemon, ObjectSomeValuesFrom(hatches, EggGroup)$$

The intersection in the consequent is then used to create a new class, while it could have been used to enrich $Pokemon$.

5.3.1.2 Class Move

The $Move$ class differs from the $Pokemon$ one in the way its properties can be used to aggregate its individuals. For the $Pokemon$ class, only a few properties were ever within the variability ranges for their corresponding class expression hypothesis to be valid – e.g. the previously mentioned hatches, $hasColor$, $hasShape$ and $hasPokemonType$. In the case of the $Move$ class, however, not only do moves have many $DataProperties$, but they mostly have a few specific values to choose from – for example, $hasPP$ is always a multiple of 5, and there are only three possible values for $hasDamageCategory$. For this reason, the lattices associated with this class are more complex than those of $Pokemon$, and only 160 individuals (of a total of 250) were analysed.

The evolution of the number of nodes of the L_{hv} can be seen in Figure 55. The complexity of this lattice, in comparison to the evolution seen previously in Figure 52, is evident from the start: for the Move class and after analysing 20 individuals only, the lattice already has 119 nodes. At the end of the timeframe of 160 individuals, the lattice has a total of 2054 nodes.

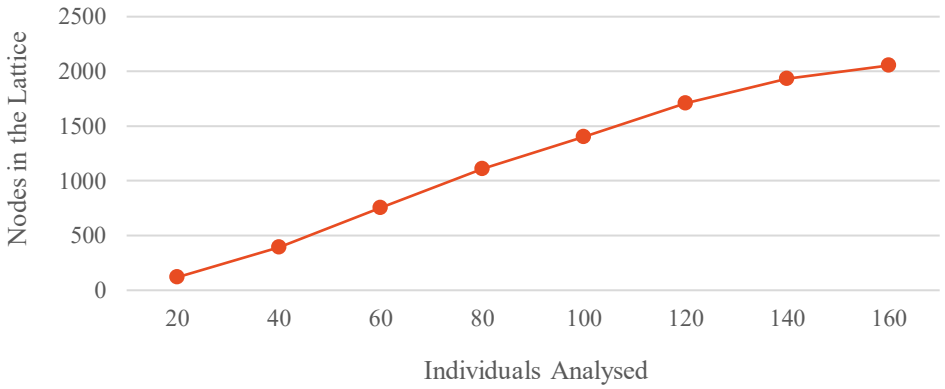

Figure 55 – Evolution of the number of nodes in the L_{hv} lattice for the $Move$ class over time

As for the evolution of the nodes in L_{qb}, Figure 56 shows that it starts slower than for the $Pokemon$ class seen in Figure 53, but quickly grows in complexity until it reaches a total of 436 nodes.

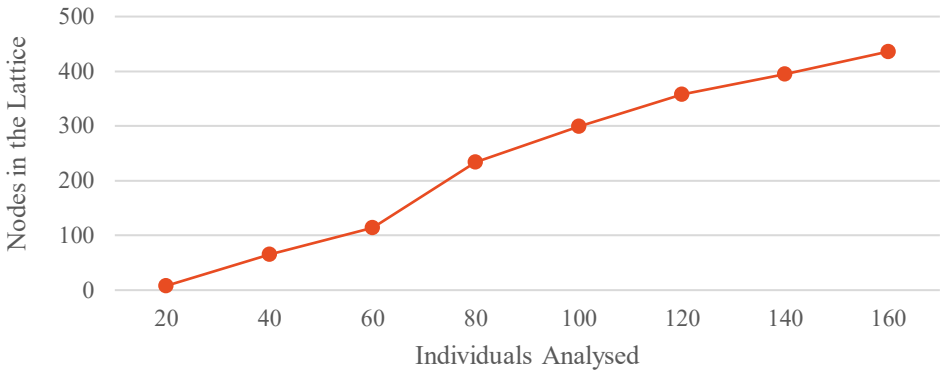

Figure 56 – Evolution of the number of nodes in the L_{qb} lattice for the $Move$ class over time

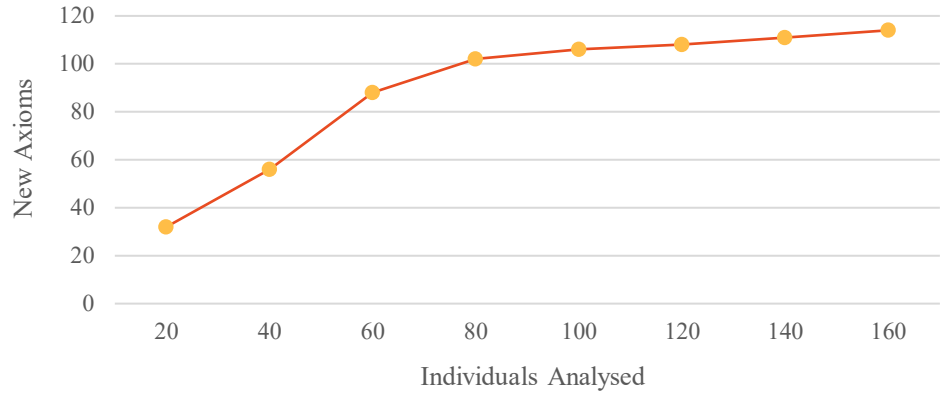

Figure 57 – Number of axioms derived from the implications of the L_{hv} lattice for the $Move$ class over time

Regarding the number of axioms created by the implications in the L_{hv} lattice, Figure 57 shows that the much larger number of nodes, in comparison to the lattice of class $Pokemon$, results in a higher number of axioms introduced. At the end of the timeframe, a total of 114 axioms are introduced. On top of these, the L_{qb} adds 14 more axioms to the ontology, for a total of 128 axioms. It is also clear that, in spite of the growth of the lattice, the number of axioms introduced every 20 individuals slows down considerably after the first 80 are analysed – suggesting that most of the relevant implications have already been materialized by that point.

A total of 27 new subclasses of $Move$ were added to the ontology, and no property domain axioms introduced. The new classes correctly describe groups of Moves with similar characteristics whose variability was within the desired range, as seen in Table 28.

Table 28 – Subset of the new *Move* subclasses and their respective class expressions

CLASS EXPRESSIONS
hasHealing_0
Equivalent to:
DataHasValue(hasHealing, "0")
Q15141195_hasContestCategory_cool
Equivalent to:
PokemonMove, DataHasValue(hasContestCategory, "cool")
HasPower_100
Equivalent to:
DataHasValue(hasPower, "100"), Pokemon
Subclass Of:
Q15141195_hasPP_15
Q15141195_hasPP_15
Equivalent to:
PokemonMove, DataHasValue(hasPP, "14")
hasCriticalRate_1
Equivalent to:
DataHasValue(hasCriticalRate, "1")
Subclass Of:
Q15141195_hasContestCategory_cool

This sample shows one complex and novel relationship, materialized in the class $hasCriticalRate\_1$, which asserts that individuals that have $DataHasValue(hasCriticalRate, "1")$ are always of class $PokemonMove$ and with "cool" as their $hasContestCategory$.

Four subclasses pertain to different values of the property $hasPower$, namely "120", "100", "50" and "40". Other possible values for the property, like "35" and "70", are apparently too rare to support a new class. Of the 18 possible move types, only "Psychic-type" and "Water-type" meet the criteria for being used in new classes.

One of the first implications to gather sufficient support in the L_{hv} provides an interesting insight about how moves are organized:

$$DataHasValue(hasContestCategory, "tough") \rightarrow PokemonMove,$$

$$DataHasValue(hasDamageCategory, \quad "physical")$$

suggesting that all "tough" moves are also "physical".

At 80 individuals, a class aggregating the moves with $hasDamageCategory$ with value "physical" had already been created, and it is possible to establish that these moves always have a $hasHealing$ of "0", which also had previously been aggregated under a new class:

$$Q15141195\_hasDamageCategory\_physical \rightarrow hasHealing\_0, PokemonMove$$

At 120 individuals, some of the implications are superfluous, featuring antecedents could already entail their consequents if the property domains had been asserted. Consider, for example, the following implication:

$$Q15141195\_hasPP\_20 \rightarrow hasPP\_20$$

in which both the antecedent and the consequent resulted from previous materializations of new classes, which again shows that more nuanced redundancy checks were necessary during intent inclusion; these could, nonetheless, be mitigated through the application of a reasoner before effectively creating new axioms or use a minimal implication basis. If one were to obtain the class descriptions of them, the implication would look like this:

$$MoveType, DataHasValue(hasPP, "20") \rightarrow DataHasValue(hasPP, "20")$$

This further suggests the current algorithm for asserting property domains lacks nuance, and that more checks should be in place when deciding which implications are worth materializing. The implication above should be considered a sign that the definition of $hasPP\_20$ could be expanded to include $MoveType$ in its set of necessary conditions, instead of creating a new subclass for it.

Finally, the 12 implications obtained by the L_{qb} lattice all pertain to new subsumption relationships. One such axiom is:

$$SubClassOf(hasContestCategory\_cute, Q15141195\_hasDamageCategory\_status)$$

from which can be said that all "cute" moves also affect "status".

5.3.2 Experiment II – Exploring Pokémon, Move and Egg Group Classes

While the relationships discovered in the previous experiment were interesting on their own and clearly show the strengths of using $HasValue$ type Ce to generate new classes, exploring only one class is evidently limiting. The remainder of this section is dedicated to understanding how well the two lattices can both identify new classes and establish new relationships between them. Because of the complexity of the lattices generated in the previous experiment, this one will be performed using three classes and a random subset of the individuals of each, namely:

- The $Pokemon$ and $Move$ classes, using only 50 individuals from each
- The $Egg\ Group$ class, with 16 individuals.

The other variables mentioned in Table 25 retain the same values.

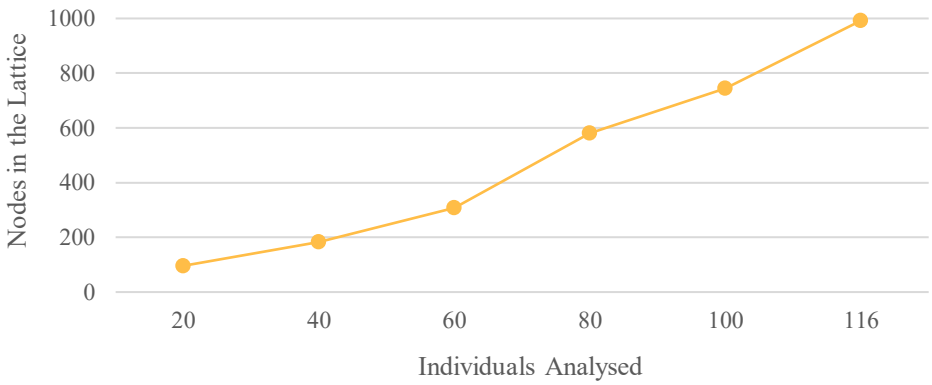

Figure 58 – Evolution of the number of nodes in the L_{hv} lattice over time

As seen in Figure 58, the L_{hv} lattice has a steady linear growth in nodes over time, which does not seem to ever slow down. At 116 individuals, it has a total of 992 nodes – less than the 1708 of the $Move$ class, but more than the 496 of the $Pokemon$ class after 120 individuals. This is possible due to the fact that, since less individuals of each class are analysed, the lattice is not able to find the same number of combinations of attributes – although the presence of both classes also introduces relationships between them that were not possible before.

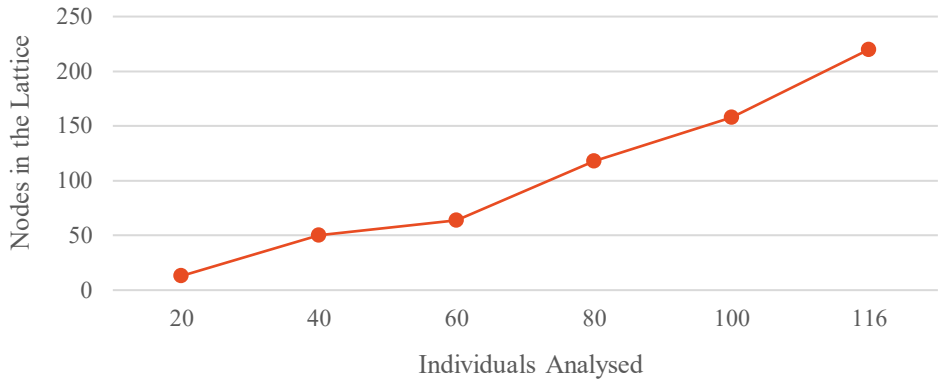

Figure 59 – Evolution of the number of nodes in the L_{qb} lattice over time

As for the L_{qb} lattice, it has 220 nodes at the end of the timeframe. As seen in Figure 59, with 120 individuals analysed, the *Pokemon* version of the same lattice had 121 nodes and the *Move*'s 358, reflecting an evolution similar to that of L_{hv}. 96 axioms had been added to the ontology at the end of the timeframe. Comparatively, at 120 individuals, the *Pokemon* lattice had produced only 50 axioms, while the *Move* lattice produced 108. The evolution of the number of axioms can be seen in Figure 60. After the materialization of the implications provided by L_{qb}, 139 more axioms are added to the ontology.

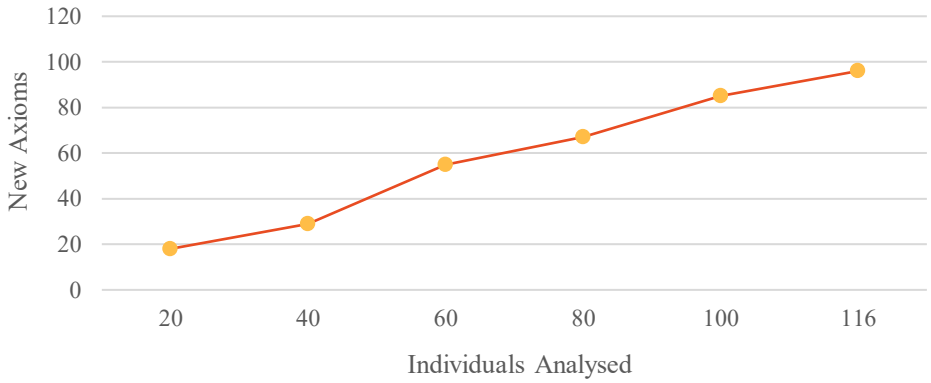

Figure 60 – Number of axioms derived from the implications of the L_{hv} lattice over time

At the end of the timeframe, a total of 35 new classes were added to the ontology. Of these, 5 are entirely new (not subclasses of any original class), 6 are new subclasses of *Pokemon* and 22 are subclasses of *Move*.

Only two property domain axioms are added, as Table 29 shows. Like before, *hasColor*'s domain was correctly identified as being the *Pokemon* class. However, all individuals of class Move use the property *hasPP* at least once, and not only those that also feature the property *hasDrain* with the value "0". This once again shows that the algorithm developed to derive property domains is not

suitable for the incremental changes in the lattices and in the ontology and should be modified to support the possibility of counterexamples and generalization.

Table 29 – Property domain axioms added to the ontology

PROPERTY	DOMAIN
hasPP	Q15141195_hasDrain_0_Q15141195
hasColor	Pokemon

Like before, new subclasses were created when the original class could just have been enriched. For example, the implication:

$$ObjectSomeValuesFrom(hasGenderRatio, GenderRatio) \rightarrow Pokemon, ObjectSomeValuesFrom(hatches, EggGroup), ObjectSomeValuesFrom(learnsMove, Move)$$

displays a set of properties that could be used to define the $Pokemon$ class – all $Pokemon$ have some $GenderRatio$, hatch from some $EggGroup$ and learn some $Move$. The more elegant solution would have been to add the class expressions to the $Pokemon$ class: however, since neither the antecedent nor the consequent are of type Class exclusively, the class creation algorithm can only identify the need to create a new class that is equivalent to the consequent. Because this new class has $Pokemon$ as one of its primitive class expressions, it is still considered a subclass of it.

Other classes reify sets of initially unknown constraints, like the ones in Table 30. For example, the class $hasContestCategory\_cool$ inherits from its superclasses the constraints $DataHasValue(hasMoveType, \text{“}normal\text{”})$ and $DataHasValue(hasDrain, \text{“}0\text{”})$. The implications regarding only $Pokemon$ or $Move$ classes are, for the most part, very similar to the ones described in the previous experiment. It is worth noting, however, that this time the solution created new classes without adding the superclass to the subclass' class expression, resulting in less Ce redundancy.

Table 30 – Subset of the new classes and their respective class expressions

CLASS EXPRESSIONS
hasContestCategory_cool
Equivalent to: DataHasValue(hasContestCategory, "cool")
SubClass of: Q15141195_hasMoveType_Q26001206
hasMoveType_Q26001206
Equivalent to: DataHasValue(hasMoveType, "normal")
Subclass Of: Q15141195_hasDrain_0
Q15141195_hasDrain_0
Equivalent to: PokemonMove, DataHasValue(hasDrain, "0")
hasMoveType_Q26001216
Equivalent to: DataHasValue(hasMoveType, "psychic")
Subclass Of: Q15141195_hasDamageCategory_status
Q15141195_hasDamageCategory_status
Equivalent to: DataHasValue(hasDamageCategory, "status") **Subclass Of:** Q15141195_hasDrain_0_ Q15141195
Q15141195_hasMoveType_Q26001206
Equivalent to: PokemonMove, DataHasValue(hasMoveType, "normal")
SubClass of: hasMoveType_Q26001206

For example, the classes *hasColor_blue*, *hasContestCategory_cute* and *hasPP*_30 result, respectively, from the following implications:

$$Pokemon \rightarrow DataHasValue(hasColor, "blue")$$
$$PokemonMove \rightarrow DataHasValue(hasContestCategory, "cute")$$
$$PokemonMove \rightarrow DataHasValue(hasPP, "30")$$

This is not the case for all new classes generated from groups of property-values, however. 6 of the new classes were identified through a consequent that intersected the property with what could have been considered its domain:

$$ObjectHasValue(hasMoveType, NormalType) \rightarrow DataHasValue(hasDrain, "0"), PokemonMove$$

Most $Move$ individuals will have a $hasDrain$ of "0", so it is interesting that it met the variability requirements to raise a class expression hypothesis. As discussed before, however, this is something that can more easily happen earlier on, when not enough property-value pairs have been counted. While in some cases, the variability may no longer be within the acceptable range and therefore the support of the implications that feature it cannot increase, it is still possible that such threshold had already been surpassed – in which case, the implications will continue to be considered.

A few novel relationships between $Pokemon$ and the types of $Move$ they can learn are also found. For example, with a support of 14 individuals, it is discovered that individuals (unfortunately, of uncertain class) that can learn moves with an accuracy of "100" are also able to learn moves that are simultaneously "psychic" and "cute":

$$ObjectSomeValuesFrom(learnsMove, Q15141195\_hasAccuracy\_100) \rightarrow ObjectSomeValuesFrom(learnsMove, hasContestCategory\_beauty), ObjectSomeValuesFrom(learnsMove, hasMoveType\_Q26001216)$$

All $Moves$ with a $hasDamageCategory$ with value "status" also have a $hasDrain$ of "0", which could have been used to simplify the implication below:

$$ObjectSomeValuesFrom(learnsMove, hasPP\_30) \rightarrow ObjectSomeValuesFrom(learnsMove, Q15141195\_hasDamageCategory\_status), ObjectSomeValuesFrom(learnsMove, Q15141195\_hasDrain\_0\_Q15141195\ )$$

Nonetheless, the resulting insight is the same: individuals that can learn things that have a $hasPP$ of "30" can also learn a $Move$ with "status" as its $hasDamageCategory$. Additionally, implications such as this show the potential of combining the two lattices for more than one class: while in Experiment I the property $learnsMove$ never reaches the variability requirements – for it has too many distinct values per individual – since those values were aggregated

beforehand, $learnsMove$ can now be used to establish relationships not only between $Pokemon$ and $Move$, but between $Pokemon$ and groups of $Move$s.

5.4 Discussion

While FCA can be effectively used to derive new class and class relationships by incrementally generating concept lattices, there are a number of factors that warrant discussion regarding the proposed solution.

The identification of property domains is particularly affected by the order in which the implications are materialized. A class is only considered for the domain of a property if it is the only class (and the only class expression) appearing in the $antecedent$ – however, as new subclasses are introduced, these will also often appear in the antecedent (and, in many cases, effectively replace the original class). As such, the property domain algorithm could tremendously benefit from generalizing the candidate class whenever possible. This is also the reason why $AllValuesFrom$ class expressions did not get materialized: if an individual at the range of a property has more than one class – as they often will, as the ontology evolves – they can no longer raise $AllValuesFrom$ class expression hypotheses, nor give more support to existing ones.

Regardless of the redundancy issues, the solution recognizes the need to create new classes by aggregating similar individuals and establishes novel relationships between these groups.

Using two lattices, one for each type of class expression being studied, ensures that each of them remains relatively scalable – when compared with the issues in scalability that arise from using all class expressions on the same lattice – but comes with the price of reducing the maximum expressivity the solution can achieve. Consider that, for the $Pokemon$ class only, the L_{hv} lattice had 824 nodes at the end of the timeframe, and the L_{qb} another 144. For the $Move$ class, 2054 and 436 nodes, respectively. While it is well possible that some of those nodes may, in fact, overlap, the more likely reality is that a single lattice would end up multiplying those numbers, as they grow exponentially w.r.t. the number of attributes. This design option, nonetheless, does not fully guarantee that the complexity of the lattices will not be excessive.

Furthermore, the modifications made to the AddIntent algorithm (making use of reasoners to assess the entailment of intents and reduce their redundancy)

introduce a severe computational drain. While the initial ontology was not particularly complex (no class axioms or property axioms), it grows increasingly more complex as more groups of individuals are reified into their own classes and as all classes, new and old, get more complex definitions. As such, the lattice building process starts relatively fast, only to grow progressively slower with time until it no longer produces results over long periods of time. This problem is exacerbated when, for each new individual, it is also necessary to use a reasoner to assess which of the new classes apply – in order to correctly elicit and classify the class expressions hypotheses. Because so many new classes are introduced that were not initially classifying the individual, such work must be performed by the reasoner – and because this work must be done not only for each new individual, but also for the individuals in the ranges of their properties – this step ends up introducing the vast majority of the computational time.

It is quite hard to strike a balance regarding the period between materializations of the implications from L_{hv}. If the period is too short, too many new classes may be produced, increasing the complexity of the ontology and making it harder to add new elements to both lattices. If the period is too long, new classes may be derived too late, and their actual support in L_{qb} underestimated.

The order of the individuals affects the construction of L_{hv}: and, therefore, the groups of individuals that can potentially become new classes. Because of this, the relationships between these groups of individuals are also affected, and therefore influence the results of L_{qb}. Relationships benefit from being discovered earlier, as they are added to the L_{qb} until counterexamples are found. As such, a Ce may be, temporarily, confirmed by all individuals analysed until a certain point, and used as the intent of some nodes in the lattice. Individuals showing up later on the stream can potentially render the Ce unviable, which would require removing it from the lattice – something to be considered in future work. As it currently stands, after a Ce is no longer valid, it will no longer be confirmed by any subsequent individuals, and implications that mention it will not see their support increase any longer – however, it is possible that the support had already reached the minimum threshold for implication consideration. A sufficiently large sliding window may mitigate this issue, but as previously discussed, such is not necessarily desirable. The design approach only considered incremental additions to the ontology; however, it is clear from the results that methods to remove axioms should be employed. While the process of elicitation,

confirmation and denial of class expression hypotheses guarantees that only confirmed class expressions are used, when a counterexample for them is found the ontology should equally be analysed for potential axioms to remove.

5.5 Conclusions

This chapter presented a Formal Concept Analysis-based solution for class learning and class enrichment using data provided by streams of RDF individuals. The goal of this chapter was to answer the research questions **RQ2.2:** *How can we identify new class expressions?* and **RQ2.3:** *Is it possible to identify the need to have super/subclass relationships that were not in the ontology?*

The proposal was based on the application of two concept lattices – the first using only $HasValue$ class expressions, and the second using $AllValuesFrom$ and $SomeValuesFrom$ class expressions – which are built iteratively as new individuals arrive using a modified version of the AddIntent algorithm. The lattices serve distinct purposes: learning new classes of individuals by their shared sets of properties and finding potential property domain axioms, and enriching existing class definitions, respectively.

To test the applicability of the solution, two experiments were conducted: (1) using individuals of a single class, in order to understand the types of implications and shortcomings of grouping individuals in new subclasses, and (2) using individuals of different classes simultaneously, and assessing what type of novel relationships could be uncovered.

Experiment I was executed for two different classes: the $Pokemon$ class, which has, on average, more properties per individual, but with high variability; and the $Move$ class, which with less properties, but that fall into the desired variability threshold. This discrepancy resulted in very different lattices: the higher variability resulted in less class expression hypotheses being raised, and therefore less being confirmed, and its lattices had a relatively smaller number of nodes. This resulted in the addition of 90 axioms to the ontology from the analysis of the $Pokemon$ class. For the $Move$ class, as more class expressions are confirmed and supported by larger groups of individuals, more subsets of them could be derived, resulting in a higher number of classes – and a total of 128 new axioms.

For performance reasons, Experiment II used individuals from both $Pokemon$ and $Move$ classes, but only a subset of 116 individuals in total. The combination

of two classes in the same stream produced lattices that sit in the middle ground compared to the previous two. Nonetheless, the resulting implications were more fruitful, and a total of 139 axioms were added to the ontology as a result.

Both experiments uncover novel relationships in the data, and clearly and properly identify the need for new classes. However, as it currently stands, the solution produces some redundant implications and, therefore, redundant classes. While using two lattices pushes the design closer to some RCA solutions, the patterns of redundancy seen in the results before could still be mitigated in subsequent designs. In inspecting the results, it is clear that more candidates for class enrichment could be considered: as it currently stands, the only way for a class to be enriched is to be the sole primitive class expression in the antecedent – but complex class expressions are much more frequent and potentially useful. A possible mitigation for this needs to address the intersection of a single class with class expressions (with properties in their signatures) in order to infer if those class expressions should be part of the single class's definition. This could result in a mechanism similar to that used to identify the domain of properties.

Future steps would involve mitigating the redundancy in the materialized classes through a careful selection process that investigates the effective novelty of each potential axiom. The redundancy between the implications provided by the lattices can also be mitigated by exploring more RCA-inspired processes and existing algorithms, which would allow for more general conclusions (e.g. regarding the property domain axioms). Furthermore, the current solution could be expanded to derive association rules from the lattices: potentially uncovering even more interesting relationships in the data while allowing for some individuals to contradict them. Future work should also see improving the existing Reduce method to remove not only the redundancy in between the primitives of a complex class expression, but also between subgroups of them.

Ultimately, while the potential future improvements to the solution could help reduce redundancy, the tool currently generates classes and relationships that are useful and insightful, and could prove particularly valuable in guiding an ontologist about how to describe previously unknown facts about the domain. However, as it currently stands and because it is highly dependent on a reasoner to obtain the full list of classes – ever expanding – that describe each individual multiple times, the computational time of the algorithm has a tendency to increase beyond usability.

CHAPTER 6

Conclusions and Future Work

6. Conclusions and Future Work

This thesis focused on the development of *TICO*, a tool for instance-guided ontology evolution. The work presented in this thesis stemmed from the original idea of developing a tool that could automatically update an ontology in response to changes in the individuals provided by an RDF stream, with applications for predictive maintenance solutions. *TICO*, nonetheless, is agnostic to its application scenario, and can be used to evolve ontologies on any domain. In fact, the evolutionary problem turned out to be much more enticing and interesting than that original application scenario: the change in the datasets and application domain allowed for a deeper exploration of the intricacies of ontology evolution, particularly class learning, class enrichment and property axiom induction – which proved fantastically more interesting.

The remainder of this chapter will summarize and discuss the contributions of this thesis and propose venues for future research.

6.1 Main Contributions

This thesis begins by introducing *TICO*, the TIme-Constrained Ontology evolution tool. It was originally developed with predictive maintenance in mind: data would be obtained from streams provided by sensors and transformed into individuals described by a PdM ontology. Subsequentially, further transformation processes would be made on the data, following the experimental needs of a data scientist: as new features are introduced in the domain to respond to these necessities, their timely identification and inclusion in the ontology proved an interesting challenge.

Returning to the working hypothesis of this work:

> *It is possible to establish an Ontology Evolution/Learning methodology to generate expressive, formal ontologies from semi-structured sources, encompassing both user and automatic methods of classification and assessment of possible evolution points.*

the goal was to obtain an axiomatically-rich ontology that closely described the data from a specific period of analysis, doing so incrementally and over fundamentally incomplete datasets. This problem was broken down into three

main steps, addressing the research questions presented in section 1.1.3. Furthermore, each step corresponds to a part of the solution as described by one of the main chapters of this thesis:

1. Transforming sensor data (in the JSON format) into individuals described by a known ontology, the identification new classes and properties and accurate representation of their evolution over time by reifying *TimeSlices* of classes. This step addresses **RQ1**.
2. Learning property axioms by exploring positive and negative evidence against each property axiom pattern using a sliding window to partially close the world. This step addresses **RQ2.1**.
3. Learning new classes and enriching class definitions by building concept lattices over the class expression hypotheses confirmed by individuals and exploring the implications they generate. This step addresses **RQ2.2** and **RQ2.3**.

6.1.1 Proposed Architecture and Time Representation

Regarding the first step, in ***Chapter 3, Time-Constrained Ontology Evolution,*** the similarly-named tool is introduced in its original PdM context. Here, the conceptual design of the solution is presented, and the main components are described in detail. These include *J2OIM*, a data transformation tool that obtains data from the sensors in JSON format and transforms it into RDF individuals, and *TICO*, which analyses the RDF stream of individuals to evolve the ontology.

To answer the research question **RQ1:** *To which extent is it possible to identify changes in classes and properties used by individuals provided via stream and reflect them in an ontology that represents those individuals during a specific period?* the 4D-Fluents pattern was chosen for the representation of changes in the ontologies and the time dimension of the different ontological entities. The same approach is applied to the evolution of classes: all classes can be "extended" with a temporal dimension (i.e., a new subclass for it is created) that specifies the class expressions that are valid only during that *Interval*. This new subclass is called a *TimeSlice* of the original class.

The data used in this chapter is obtained from 15 different sensors in 3 extrusion machines, representing a total of 24 variables that describe the machines, sensors, occurrences and manufacturing operations. Of the 20 webservice endpoints, 15 correspond to sensor data and 5 to the ERP software.

To integrate the information obtained from the sensors in the machines and that from the ERP, a domain ontology was designed, interconnecting classes from existing ontologies.

Regarding *J2OIM*, special attention was given to the reification of the time dimension: a distinction was made between temporal and non-temporal entities, which are processed differently by the tool. Aggregation methods were employed to reduce the number of the generated individuals: repeated sensor readings are represented using only one individual associated with a larger *TimeInterval* (reducing the number of sensor data by ≅10%), and sensor readings that occur simultaneously share Instants and Intervals.

Regarding *TICO*, this chapter explores the creation of sequential *TimeSlices* as changes are detected in the individuals on the stream. To do so, it employs a series of comparison operators, which are responsible for analysing whether the original version of an individual's class matches with the individual's definition (regarding the number and type of properties). The result of the application of these operators is a set of Evolutionary Actions, which can be executed to produce changes in the ontology by adding new *TimeSlices* in an incremental fashion – by using the same Instant as the starting date for the new *TimeSlice* and the ending date for the previous one, the solution ensures an individual cannot belong to more than one *TimeSlice*, preventing potential inconsistencies.

This setup is tested against a stream of RDF individuals over a set of three timeframes. The individuals are described through means of a simple seed ontology that describes only *Sensors*, *Events* and their respective time dimensions. *TICO* correctly identifies the need for new classes and the cardinalities of the new properties associated with them. It also identifies a set of necessary and sufficient conditions for the class *Event*, creating new *TimeSlices* accordingly. The results show that *TICO* is able to properly identify and represent changes in an ontology in an incremental fashion, correctly ordering the *TimeSlices* of each class. However, the methods through which it assesses the relevance of the class expressions used are relatively modest and do not consider the weight of the individuals that do not uphold them. These shortcomings are explored in the two subsequent chapters.

6.1.2 Identifying Property Axioms

In ***Chapter 4, Property Axiom Learning***, the focus is shifted towards identifying the need for and changes in property axioms. In this chapter, closer

attention is given to the incompleteness of the data provided by RDF streams, and how this can be maneuvered to make it possible to infer new axioms and answer the research question **RQ2.1:** *How can we identify changes in property axioms?*. A sliding window is used to maintain some memory of analysed individuals; as properties are used to, in most cases, establish relationships between distinct individuals that may not be sequential, it is necessary to be able to access more than one to establish if the property follows certain patterns.

Property axioms are described through means of an implication: an individual can then be considered a selective confirmation, a counterexample, or otherwise be irrelevant. Following the selective confirmation principle, only individuals that effectively use the property are considered for these metrics. Here, a distinction is made between weak and strong counterexamples, depending on the assumptions made when analysing the individual: Functionality, Inverse Functionality, Irreflexivity and Asymmetry require the UNA in order to generate strong counterexamples, while Symmetry and Transitivity require CWA (in this case, the world being the individuals within the sliding window) in order to elevate non-confirmations to weak counterexamples.

The implications for each property axiom are enforced through means of SPARQL queries, which are executed over the sliding window of the latest individuals and do not require the use of reasoners. Following the possibilistic approach to axiom scoring, for each property axiom it is possible to calculate its necessity – the degree to which the axiom is corroborated by the data while not being contradicted by it – and its possibility – the degree to which it is not contradicted by the data. The acceptation/rejection index (ARI) of each property axiom is computed as a function of possibility and necessity, with positive values suggesting the inclusion of the axiom, while negatives suggesting rejection. For this, a value of 0 indicates ignorance. Two forms of calculating necessity and possibility are presented in literature: here, they are recontextualized and applied according to strength of the counterexamples that can be obtained per each property axiom. Additionally, an evolving form of ARI is introduced, which aims to use the knowledge from the ontology – i.e., if the property was already present in a previous timeframe and if it had the axiom – in the decision to include or reject a property axiom. The pre-existence of the axiom is given a weight that influences that decision.

Two different experiments are executed to validate the applicability of the chosen approach to property axiom scoring. In the first one, a comparison is made between the performance of the possibilistic approach and traditional information retrieval metrics. For this purpose, ontologies for which the property axioms are known *a priori* are used. This experiment also assesses the effect of the sliding window in the classification of individuals as selective confirmations or counterexamples. The results show the possibilistic approach can correctly identify the need for a property axiom using a relatively small sample of the data, and performs better than traditional information retrieval metrics. The results also suggest that some axioms benefit from the application of a higher threshold for ARI than others.

The second experiment uses an ontology automatically generated from data obtained from public sources (Wikidata) and for which no property axioms are known. The goal of this experiment was to evaluate which inconsistencies are introduced if some counterexamples are allowed – by combining ARI with a minimum percentage of support of 90% – over a series of nine timeframes. After adding the axioms to the ontology at the end of each timeframe, a reasoner is employed to check for inconsistencies. This, in fact, shows that approach proves very effective at identifying property axioms that do not generate inconsistencies. Most of the inconsistencies generated concern errors in data – which highlights the potential of the tool for data validation. Only in one case it is shown that a property axiom is incorrectly added, which does not generate inconsistencies but wrong inferences instead: however, such evaluation cannot be made automatically, and only the careful examination of the results and knowledge of the domain would detect it.

The second part of the second experiment aimed to evaluate the performance of the evolving form of ARI over a series of nine timeframes. In this case, the goal of the experiment was to assess if evolving ARI performed better than the combination of the percentage of support with ARI. The results show that evolving ARI is not as effective in countering errors in data as the combined approach: whenever using the strong form of ARI, evolving ARI had difficulties in changing the decisions reached earlier on, even in the face of overwhelming evidence and often introducing more inconsistencies over the timeframes. On the other hand, for the axioms using the weak form of ARI, the result was the opposite: by allowing axioms to be introduced even in the face of missing information, they are favoured earlier and allowed to continue even if the

number of selective confirmations did not increase – and without introducing inconsistencies.

Ultimately, the experimental results of this chapter show that ARI is very suitable for applications with RDF streams but can be made more resistant to errors in data if combined with other metrics.

6.1.3 Class Learning and Enrichment

Finally, ***Chapter 5, Identifying New Classes and Enriching Class Definitions***, returns to the problem of class learning and class enrichment. This chapter restructures the way *TICO* learns class expressions, doing so incrementally as more individuals from the stream are analysed, and classifying them as selective confirmations or counterexamples according to the principles established before. The confirmed class expressions are then used to build two concept lattices, from which implications can be derived that will evolve the ontology.

To answer the research question **RQ2.2:** *How can we identify new class expressions?*, the possible class expressions are aggregated into a list of class expression hypothesis, which can then be confirmed/denied by incoming individuals. A subset of the possible primitive class expressions is explored, combined exclusively using the $IntersectionOf$ operator to form more complex class expressions. A class expression hypothesis is valid for as long as it is not contradicted by any individual. However, because of their applicability to describe groups of individuals, this is not the case for the $HasValue$ types of hypotheses, which are considered valid depending on two variability metrics: the Percentage with Value and the Property Ratio. The variability of each property-value pair must fall within specific ranges, making the solution less prone to cases of over and underfitting when creating new classes. As these are aggregated metrics, the more individuals are analysed, the more accurate to reality they become.

To answer the research question **RQ2.3:** *Is it possible to identify the need to have super/subclass relationships that were not in the ontology?*, Formal Concept Analysis is used to build two lattices of closed conjunctions of class expressions, from which implications can be derived. As such, one lattice deals exclusively with this aggregation process, using only confirmed $HasValue$ class expressions. The implications from this lattice were used to derive new classes and establish potential domains of properties. Using the $AllValuesFrom$ and $SomeValuesFrom$

confirmed class expressions, the second lattice focuses on enriching the classes obtained by the first one and also those already in the ontology and identifying subsumption relationships between classes. If sufficient evidence for it is found, equivalence between classes can also be added. Cardinality-based class expressions were not included for a matter of efficiency. The lattices are built with a modified form of the AddIntent algorithm, which checks for the entailment of the intents and attempts to remove redundancy between them.

To test the applicability of this solution, two experiments were conducted: first, the lattices were built using only elements of only one class, and a second one using three classes. In identifying the subsets of classes that should be reified into relevant new classes, the solution proved fruitful. Novel relationships, including between classes created by the first lattice, are uncovered and reified into new classes. These new classes manifest implicit knowledge that would be otherwise difficult to discover. On the other hand, some class expressions for the new classes are redundant, and the discovery of property domains could use more finesse. While the results of the experiments are solid, the methods used to derive implications from the lattice and the algorithms that materialize them into axioms have room for improvement. Existing algorithms and tools for obtaining implications from concept lattices and formal contexts could be combined with *TICO* to improve the current solution. Further studies should also address the limitations of building a lattice whose intents are not updatable and that cannot be pruned (even in the face of counterexamples), often resulting in implications that combine equivalent class expressions in redundant ways.

6.2 Final Considerations

While the proposed solution successfully answers all the research questions raised, as previously mentioned, there is a computational drain associated with an ever-increasing number of axioms and the need to use a reasoner to check to which classes an individual may belong. This makes the solution not very suitable for obtaining useful results in near-real time if the ontologies are rather complex, either in the number of classes or properties. Similarly, the number of axioms produced is always greater than the number which may actually be needed. As it currently stands, the solution is much more suited as a support tool for an ontologist analysing streams of data in their attempts to closely describe it: supplying and suggesting a set of axioms for the ontologist to choose from,

including potential descriptions of new classes, which may or may not need to be improved – but nonetheless provides a solid starting point. Suggesting new property axioms and class expressions also has the added value of allowing for more consistency checks to be performed on the data and make it easier to identify errors and noise. Furthermore, it is also important to note that the datasets used in chapters 4 and 5 were chosen precisely because they brought in complexity in the data that was not present in the original PdM dataset – allowing for an in depth evaluation of the solution and to exploit its strengths and weaknesses, but do not represent the average PdM scenario – in which evolution can be found, but rarely on the degree and frequency analysed.

Finally, and although it was out of the scope of the experiments performed for this work, it is also important to consider the possibility of over-characterization of the properties and classes in an ontology. The results show that the current solutions will almost always propose one or more axioms for each property, and class expressions that could easily be made redundant if property domains are asserted – which may or may not make sense, depending on the application context and purpose of the ontology; upper-level ontologies, by definition and application, benefit from excluding superfluous axioms. Even for lower-level ontologies, there is always an argument to be done in favour of simplicity of design, which should not be more complex than necessary to achieve its goals. In the future, the suitability of the suggested axioms should not rely solely on the fact that their addition to the ontology does not introduce inconsistencies, but be motived too by their capacity to allow for richer inference processes to happen in order to unlock the data's potential.

Conclusiones y trabajo futuro

Esta tesis se centró en el desarrollo de TICO, una herramienta para la evolución de ontologías guiada por instancias. El trabajo presentado en esta tesis partió de la idea original de desarrollar una herramienta que pudiera actualizar automáticamente una ontología en respuesta a cambios en los individuos proporcionados por un flujo RDF, con aplicaciones para soluciones de mantenimiento predictivo. TICO, sin embargo, es agnóstico a su escenario de aplicación, y puede ser utilizado para evolucionar ontologías sobre cualquier dominio. De hecho, el problema evolutivo resultó ser mucho más atractivo e interesante que el escenario de aplicación original: el cambio en los conjuntos de datos y el dominio de aplicación permitió una exploración más profunda de las complejidades de la evolución de ontologías, en particular el aprendizaje de clases, el enriquecimiento de clases y la inducción de axiomas de propiedades, lo que resultó ser fantásticamente más interesante.

En el resto de este capítulo se resumen y discuten las aportaciones de esta tesis y se proponen líneas de investigación para el futuro.

Contribuciones principales

Esta tesis comienza presentando *TICO*, Time-Constrained Ontology Evolution, una herramienta de evolución de ontologías. Se desarrolló originalmente pensando en el mantenimiento predictivo (PdM): los datos se obtendrían de flujos proporcionados por sensores y se transformarían en individuos descritos por una ontología afecta al campo del PdM. Posteriormente, se realizarían otros procesos de transformación sobre los datos, siguiendo las necesidades experimentales de un científico de datos: a medida que se introducen nuevas características en el dominio para responder a estas necesidades, su oportuna identificación e inclusión en la ontología resultó ser un reto interesante.

Volviendo a la hipótesis principal de este trabajo:

> *Es posible establecer una metodología de evolución/aprendizaje de ontologías para generar ontologías expresivas y formales a partir de fuentes semiestructuradas, englobando tanto el usuario como métodos automáticos de clasificación y evaluación de posibles puntos de evolución.*

el objetivo era obtener una ontología rica en axiomas que describiera fielmente los datos de un periodo concreto de análisis, haciéndolo de forma incremental y sobre conjuntos de datos fundamentalmente incompletos. Este problema se desglosó en tres pasos principales, que abordaban las preguntas de investigación presentadas en la sección 1.1.3. Además, cada paso corresponde a una parte de la solución descrita en uno de los capítulos principales de esta tesis:

1. La transformación de los datos de los sensores (en formato JSON) en individuos descritos por una ontología conocida, la identificación de nuevas clases y propiedades y la representación precisa de su evolución en el tiempo mediante la reificación *TimeSlices* de las clases. Este paso aborda la cuestión de investigación ***RQ1***.
2. Aprendizaje de axiomas de propiedades mediante la exploración de pruebas positivas y negativas contra cada patrón de axioma de propiedad utilizando una ventana deslizante para cerrar parcialmente el mundo. Este paso aborda la cuestión de investigación ***RQ2.1***.
3. Aprender nuevas clases y enriquecer las definiciones de clase construyendo redes de conceptos sobre las hipótesis de expresión de clase confirmadas por los individuos y explorando las implicaciones que generan. Este paso aborda las cuestiones de investigación ***RQ2.2*** y ***RQ2.3***.

Arquitectura propuesta y representación temporal

En cuanto al primer paso, en el ***Capítulo 3, Time-Constrained Ontology Evolution*** – Evolución ontológica con restricciones temporales – se presenta la herramienta *TICO* en su contexto original de PdM. Aquí se presenta el diseño conceptual de la solución y se describen en detalle sus principales componentes. Estos incluyen *J2OIM*, una herramienta de transformación de datos que obtiene los datos de los sensores en formato JSON y los transforma en individuos RDF, y *TICO*, que analiza el flujo RDF de los individuos para evolucionar la ontología.

Para responder a la pregunta de investigación ***RQ1:*** *¿Hasta qué punto es posible identificar los cambios en las clases y propiedades utilizadas por los individuos proporcionados vía stream y reflejarlos en una ontología que represente a dichos individuos durante un periodo específico?* se eligió el patrón 4D-Fluents para la representación de los cambios en las ontologías y la dimensión temporal de las diferentes entidades ontológicas. El mismo enfoque se aplica a la evolución de

las clases: todas las clases pueden «ampliarse» con una dimensión temporal (es decir, se crea una nueva subclase para ella) que especifica las expresiones de clase que sólo son válidas durante ese intervalo. Esta nueva subclase se designa como un *TimeSlice* de la clase original.

Los datos utilizados en este capítulo se obtienen de 15 sensores diferentes en 3 máquinas de extrusión, lo que representa un total de 24 variables que describen las máquinas, los sensores, las incidencias y las operaciones de fabricación. De los 20 puntos finales de *webservice*, 15 corresponden a los datos de los sensores y 5 al software ERP. Para integrar la información obtenida de los sensores de las máquinas y la del ERP, se diseñó una ontología de dominio, interconectando clases de ontologías existentes.

En cuanto a *J2OIM*, se prestó especial atención a la reificación de la dimensión temporal: se distinguió entre entidades temporales y no temporales, que la herramienta procesa de forma diferente. Se emplearon métodos de agregación para reducir el número de individuos generados: las lecturas repetidas de los sensores se representan utilizando un solo individuo asociado a un *TimeInterval* mayor (reduciendo el número de datos de los sensores en unos ≅10%), y las lecturas de los sensores que ocurren simultáneamente comparten sus *Instants* e *Intervals*.

En cuanto a *TICO*, este capítulo explora la creación de *TimeSlices* secuenciales a medida que se detectan cambios en los individuos del flujo. Para ello, emplea una serie de operadores de comparación, que se encargan de analizar si la versión original de la clase de un individuo coincide con el individuo (en cuanto al número y tipo de propiedades). El resultado de la aplicación de estos operadores es un conjunto de Acciones Evolutivas (originalmente *Evolutionary Actions*), que pueden ser ejecutadas para producir cambios en la ontología añadiendo nuevos *TimeSlices* de forma incremental - al utilizar el mismo Instante como fecha de inicio del nuevo *TimeSlice* y de finalización del anterior, la solución garantiza que un individuo no pueda pertenecer a más de un *TimeSlice*, evitando posibles inconsistencias.

Esta configuración se prueba con un flujo de individuos RDF en un conjunto de tres marcos temporales. Los individuos se describen mediante una simple ontología semilla que describe únicamente *Sensors*, *Events* y sus respectivas dimensiones temporales. *TICO* identifica correctamente la necesidad de nuevas clases y las cardinalidades de las nuevas propiedades asociadas a ellas. También identifica un conjunto de condiciones necesarias y suficientes para la clase *Event*,

creando nuevos *TimeSlices* en consecuencia. Los resultados muestran que *TICO* es capaz de identificar y representar adecuadamente los cambios en una ontología de forma incremental, ordenando correctamente los *TimeSlices* de cada clase. Sin embargo, los métodos a través de los cuales evalúa la relevancia de las expresiones de clase utilizadas son relativamente modestos y no consideran el peso de los individuos que no las sostienen. Estas deficiencias se analizan en los dos capítulos siguientes.

Identificación de Axiomas de Propiedad

En el ***Capítulo 4, Property Axiom Learning*** – Aprendizaje de axiomas de propiedades – , la atención se desplaza hacia la identificación de la necesidad y los cambios en los axiomas de propiedades. En este capítulo, se presta más atención al carácter incompleto de los datos proporcionados por los flujos RDF, y cómo se los puede maniobrar para hacer posible inferir nuevos axiomas y responder a la pregunta de investigación ***RQ2.1:*** *¿Cómo podemos identificar cambios en los axiomas de propiedades?* Se utiliza una ventana deslizante para mantener una cierta memoria de los individuos analizados; como las propiedades se utilizan para, en la mayoría de los casos, establecer relaciones entre individuos distintos que pueden no ser secuenciales, es necesario poder acceder a más de una para establecer si la propiedad sigue determinados patrones.

Los axiomas de las propiedades se describen mediante una implicación: un individuo puede considerarse entonces una confirmación selectiva, un contraejemplo o ser irrelevante. Siguiendo el principio de confirmación selectiva, para estas métricas sólo se tienen en cuenta los individuos que utilizan efectivamente la propiedad. Aquí se distingue entre contraejemplos débiles y fuertes, en función de las suposiciones que se hagan al analizar al individuo: La Funcionalidad, la Funcionalidad Inversa, la Irreflexividad y la Asimetría requieren la Suposición de Nombre Único para generar contraejemplos fuertes, mientras que la Simetría y la Transitividad requieren una Suposición de Mundo Cerrado (en este caso, el mundo son los individuos dentro de la ventana deslizante) para elevar las no confirmaciones a contraejemplos suaves.

Las implicaciones de cada axioma de propiedad se aplican mediante consultas SPARQL, que se ejecutan sobre la ventana deslizante de los últimos individuos y no requieren el uso de razonadores. Siguiendo el enfoque posibilista de la cualificación de axiomas, para cada axioma de propiedad es posible calcular su

necesidad – el grado en que el axioma es corroborado por los datos sin ser contradicho por ellos – y su posibilidad -el grado en que no es contradicho por los datos-. El índice de aceptación/rechazo (ARI) de cada axioma de propiedad se calcula en función de la posibilidad y la necesidad; los valores positivos sugieren la inclusión del axioma, mientras que los negativos sugieren su rechazo. Para ello, un valor de 0 indica ignorancia. En la literatura se presentan dos formas de calcular la necesidad y la posibilidad: aquí se recontextualizan y se aplican según la fuerza de los contraejemplos que pueden obtenerse por cada axioma de propiedad. Además, se introduce una forma evolutiva de ARI, cuyo objetivo es utilizar el conocimiento de la ontología – si la propiedad ya estaba presente en un marco temporal anterior y si tenía el axioma – en la decisión de incluir o rechazar un axioma de propiedad. La preexistencia del axioma recibe un peso que influye en esa decisión.

Se ejecutan dos experimentos diferentes para validar la aplicabilidad del enfoque elegido a la cualificación de axiomas de propiedades. En el primero, se realiza una comparación entre el rendimiento del enfoque posibilista y las métricas tradicionales de recuperación de información. Para ello, se utilizan ontologías cuyos axiomas de propiedades se conocen *a priori*. Este experimento también evalúa el efecto de la ventana deslizante en la clasificación de individuos como confirmaciones selectivas o contraejemplos. Los resultados muestran que el enfoque posibilista permite identificar correctamente la necesidad de un axioma de propiedad utilizando una muestra relativamente pequeña de los datos, y obtiene mejores resultados que las métricas tradicionales de recuperación de información. Los resultados también sugieren que algunos axiomas se benefician de la aplicación de un umbral más alto para ARI que otros.

El segundo experimento utiliza una ontología generada automáticamente a partir de datos obtenidos de fuentes públicas (Wikidata) y para la que no se conocen axiomas de propiedades. El objetivo de este experimento era evaluar qué incoherencias se introducen si se permiten algunos contraejemplos – combinando ARI con un porcentaje mínimo de apoyo del 90% – a lo largo de una serie de nueve *timeframes* (marco temporal). Tras añadir los axiomas a la ontología al final de cada marco temporal, se emplea un razonador para comprobar si hay incoherencias. De hecho, este enfoque resulta muy eficaz para identificar los axiomas de propiedades que no generan incoherencias. La mayoría de las incoherencias generadas se refieren a errores en los datos, lo que pone de manifiesto el potencial de la herramienta para la validación de datos. Sólo en un

caso se demuestra que un axioma de propiedad está incorrectamente añadido, lo que no genera incoherencias sino inferencias erróneas: sin embargo, tal evaluación no puede hacerse automáticamente, y sólo el examen cuidadoso de los resultados y el conocimiento del dominio lo detectarían.

La segunda parte del segundo experimento tenía por objeto evaluar el rendimiento de la forma evolutiva del ARI a lo largo de una serie de nueve marcos temporales. En este caso, el objetivo del experimento era evaluar si el ARI evolutivo funcionaba mejor que la combinación del porcentaje de apoyo con el ARI. Los resultados muestran que el ARI evolutivo no es tan eficaz para contrarrestar los errores en los datos como el enfoque combinado: siempre que se utilizaba la forma fuerte de ARI, el ARI evolutivo tenía dificultades para cambiar las decisiones alcanzadas anteriormente, incluso ante pruebas abrumadoras y a menudo introduciendo más incoherencias a lo largo de los plazos. En cambio, en el caso de los axiomas que utilizan la forma débil de ARI, el resultado fue el contrario: al permitir introducir axiomas incluso ante la falta de información, se les favorece antes y se les permite continuar aunque el número de confirmaciones selectivas no aumente, y sin introducir incoherencias.

En última instancia, los resultados experimentales de este capítulo muestran que ARI es muy adecuado para aplicaciones con flujos RDF, pero puede hacerse más resistente a los errores en los datos si se combina con otras métricas.

Aprendizaje y Enriquecimiento de Clases

Por último, el ***Capítulo 5, Identifying New Classes and Enriching Class Definitions*** – Identificación de nuevas clases y enriquecimiento de las definiciones de clase –, vuelve al problema del aprendizaje y enriquecimiento de clases. Este capítulo reestructura la forma en que *TICO* aprende las expresiones de clase, haciéndolo de forma incremental a medida que se analizan más individuos del flujo, y clasificándolos como confirmaciones selectivas o contraejemplos de acuerdo con los principios establecidos anteriormente. Las expresiones de clase confirmadas se utilizan entonces para construir dos redes conceptuales (*concept lattices*), de las que se pueden derivar implicaciones que harán evolucionar la ontología.

Para responder a la pregunta de investigación ***RQ2.2:*** *¿Cómo podemos identificar nuevas expresiones de clase?*, las posibles expresiones de clase se agregan en una lista de hipótesis de expresiones de clase, que luego pueden ser

confirmadas/negadas por los individuos entrantes del flujo. Se explora un subconjunto de las posibles expresiones de clase primitivas, combinadas exclusivamente mediante el operador $IntersectionOf$ para formar expresiones de clase más complejas. Una hipótesis de expresión de clase es válida mientras no sea contradicha por ningún individuo. Sin embargo, debido a su aplicabilidad para describir grupos de individuos, éste no es el caso de las hipótesis de tipo $HasValue$, que se consideran válidas en función de dos métricas de variabilidad: el Porcentaje con Valor (*Percentage with Value*) y el Ratio de Propiedad (*Property Ratio*). La variabilidad de cada par propiedad-valor debe situarse dentro de unos límites específicos, lo que hace que la solución sea menos propensa a casos de sobreajuste y de infraajuste al crear nuevas clases. Al tratarse de métricas agregadas, cuantos más individuos se analicen, más se ajustarán a la realidad.

Para responder a la pregunta de investigación ***RQ2.3:*** *¿Es posible identificar la necesidad de tener relaciones de super/subclase que no estaban en la ontología?*, se utiliza el Análisis Conceptual Formal para construir dos celosías de conjunciones cerradas de expresiones de clase, de las que se pueden derivar implicaciones. Como tal, una red conceptual se ocupa exclusivamente de este proceso de agregación, utilizando únicamente expresiones de clase $HasValue$ confirmadas. Las implicaciones de esta red conceptual se utilizaron para derivar nuevas clases y establecer posibles dominios de propiedades. Utilizando las expresiones de clase confirmadas $AllValuesFrom$ y $SomeValuesFrom$, la segunda red conceptual se centra en enriquecer las clases obtenidas por la primera y también las que ya están en la ontología e identificar relaciones de subsunción entre clases. Si se encuentran pruebas suficientes para ello, también se pueden añadir equivalencias entre clases. No se han incluido expresiones de clases basadas en la cardinalidad por una cuestión de eficiencia. Las redes conceptuales se construyen con una forma modificada del algoritmo *AddIntent*, que comprueba la vinculación de los *intents* e intenta eliminar la redundancia entre ellos.

Para comprobar la aplicabilidad de esta solución, se realizaron dos experimentos: en el primero, se construyeron las redes conceptuales utilizando únicamente elementos de una sola clase, y en un segundo, utilizando tres clases. A la hora de identificar los subconjuntos de clases que debían reificarse en nuevas clases relevantes, la solución resultó fructífera. Se descubren nuevas relaciones, incluso entre las clases creadas por la primera red conceptual, y se reifican en nuevas clases. Estas nuevas clases ponen de manifiesto conocimientos implícitos que, de otro modo, serían difíciles de descubrir. Por otra parte, algunas

expresiones de clase para las nuevas clases son redundantes, y el descubrimiento de dominios de propiedades podría ser más consistente. Aunque los resultados de los experimentos son sólidos, los métodos utilizados para derivar implicaciones de las redes y los algoritmos que las materializan en axiomas tienen margen de mejora. Los algoritmos y herramientas existentes para obtener implicaciones a partir de redes conceptuales y contextos formales podrían combinarse con *TICO* para mejorar la solución actual. Estudios posteriores también deberían abordar las limitaciones de construir una red conceptual cuyos *intentos* no son actualizables y que no se pueden podar (ni siquiera ante contraejemplos), lo que a menudo da lugar a implicaciones que combinan expresiones de clase equivalentes de forma redundante.

Consideraciones finales

Aunque la solución propuesta responde satisfactoriamente a todas las preguntas de investigación planteadas, como ya se ha mencionado, existe un desgaste computacional asociado a un número cada vez mayor de axiomas y a la necesidad de utilizar un razonador para comprobar a qué clases puede pertenecer un individuo. Esto hace que la solución no sea muy adecuada para obtener resultados útiles en tiempo casi real si las ontologías son bastante complejas, ya sea por el número de clases o de propiedades. Del mismo modo, el número de axiomas producidos es siempre superior al que puede ser realmente necesario. En su estado actual, la solución es mucho más adecuada como herramienta de apoyo para un ontólogo que analiza flujos de datos en sus intentos de describirlos de cerca: suministrar y sugerir un conjunto de axiomas para que el ontólogo elija, incluyendo descripciones potenciales de nuevas clases, que pueden o no necesitar ser mejoradas - pero que, no obstante, proporcionan un punto de partida sólido. Sugerir nuevos axiomas de propiedades y expresiones de clases también tiene el valor añadido de permitir realizar más comprobaciones de coherencia en los datos y facilitar la identificación de errores y ruidos. Además, también es importante señalar que los conjuntos de datos utilizados en los capítulos 4 y 5 se eligieron precisamente porque aportaban una complejidad a los datos que no estaba presente en el conjunto de datos original de PdM, lo que permitía una evaluación en profundidad de la solución y explotar sus puntos fuertes y débiles, pero no representan el escenario medio de PdM, en

el que se puede encontrar evolución, pero rara vez en el grado y la frecuencia analizados.

Por último, y aunque estaba fuera del alcance de los experimentos realizados para este trabajo, también es importante considerar la posibilidad de sobrecaracterizar las propiedades y clases de una ontología. Los resultados muestran que las soluciones actuales proponen casi siempre uno o varios axiomas para cada propiedad, y expresiones de clase que podrían fácilmente resultar redundantes si se afirman los dominios de las propiedades – lo que puede tener o no sentido, según el contexto de aplicación y la finalidad de la ontología; las ontologías de nivel general, por definición y aplicación, se benefician de la exclusión de axiomas superfluos. Incluso en el caso de las ontologías de nivel más concreto, siempre hay argumentos a favor de la simplicidad del diseño, que no debe ser más complejo de lo necesario para alcanzar sus objetivos. En el futuro, la aptitud de los axiomas sugeridos no debería basarse únicamente en el hecho de que su adición a la ontología no introduzca incoherencias, sino estar motivada también por su capacidad para permitir procesos de inferencia más ricos con el fin de liberar el potencial de los datos.

References

[1] F. Ansari, R. Glawar, and T. Nemeth, PriMa: a prescriptive maintenance model for cyber-physical production systems, *Int J Comput Integr Manuf*. **32** (2019) 482–503. doi:10.1080/0951192X.2019.1571236.

[2] N.T.M. Saeed, C. Weber, A. Mallak, M. Fathi, R. Obermaisser, and K. Kuhnert, ADISTES Ontology for Active Diagnosis of Sensors and Actuators in Distributed Embedded Systems, in: Proceedings of 2019 IEEE International Conference on Electro Information Technology (EIT), IEEE, 2019: pp. 572–577. doi:10.1109/eit.2019.8834013.

[3] M. Klusch, A. Meshram, A. Schuetze, and N. Helwig, ICM-Hydraulic: Semantics-Empowered Condition Monitoring of Hydraulic Machines, in: Proceedings of the 11th International Conference on Semantic Systems, Association for Computing Machinery, New York, NY, USA, 2015: pp. 81–88. doi:10.1145/2814864.2814865.

[4] M. Steinegger, M. Melik-Merkumians, J. Zajc, and G. Schitter, A framework for automatic knowledge-based fault detection in industrial conveyor systems, in: Proceedings of 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, 2017: pp. 1–6. doi:10.1109/ETFA.2017.8247705.

[5] H. Dibowski, O. Holub, and J. Rojícek, Knowledge-Based Fault Propagation in Building Automation Systems, in: Proceedings of 2016 International Conference on Systems Informatics, Modelling and Simulation (SIMS), 2016: pp. 124–132. doi:10.1109/SIMS.2016.22.

[6] T.M. Smoker, T. French, W. Liu, and M.R. Hodkiewicz, Applying cognitive computing to maintainer-collected data, in: Proceedings of 2017 2nd International Conference on System Reliability and Safety (ICSRS), 2017: pp. 543–551. doi:10.1109/ICSRS.2017.8272880.

[7] P. Delgoshaei, M.A. Austin, and D. Veronica, Semantic Models and Rule-based Reasoning for Fault Detection and Diagnostics: Applications in Heating, Ventilating and Air Conditioning Systems, in: Proceedings of 12th International Conference on Systems (ICONS 2017), 2017: pp. 48–53.

[8] R. Ferrari, H. Dibowski, and S. Baldi, A Message Passing Algorithm for Automatic Synthesis of Probabilistic Fault Detectors from Building Automation Ontologies, *IFAC PAPERSONLINE*. **50** (2017) 4184–4190. doi:10.1016/j.ifacol.2017.08.809.

[9] J.D. Hamilton, Time series analysis, Princeton University Press, 1994.

[10] P. Esling, and C. Agon, Time-series data mining, *ACM Comput Surv*. **45** (2012) 12. doi:10.1145/2379776.2379788.

[11] N.K. Ahmed, A.F. Atiya, N. el Gayar, and H. El-Shishiny, An Empirical Comparison of Machine Learning Models for Time Series Forecasting, *Econom Rev*. **29** (2010) 594–621. doi:10.1080/07474938.2010.481556.

[12] L. Stojanovic, Methods and tools for ontology evolution, M.Sc., Universitaet Fridericiana zu Karlsruhe, 2004.

[13] A. Shaban-Nejad, and V. Haarslev, Managing Changes in Distributed Biomedical Ontologies Using Hierarchical Distributed Graph Transformation, *Int. J. Data Min. Bioinformatics*. **11** (2015) 53–83. doi:10.1504/IJDMB.2015.066334.

[14] A. Polleres, R. Pernisch, A. Bonifati, D. Dell'Aglio, D. Dobriy, S. Dumbrava, L. Etcheverry, N. Ferranti, K. Hose, E. Jiménez-Ruiz, M. Lissandrini, A. Scherp, R. Tommasini, and J. Wachs, How Does Knowledge Evolve in Open Knowledge Graphs?, *Transactions on Graph Data and Knowledge (TGDK)*. **1** (2023) 1–59. doi:10.4230/TGDK.1.1.11.

[15] A.C. Khadir, H. Aliane, and A. Guessoum, Ontology learning: Grand tour and challenges, *Comput Sci Rev*. **39** (2021). doi:10.1016/J.COSREV.2020.100339.

[16] A. Canito, J. Corchado, and G. Marreiros, A systematic review on time-constrained ontology evolution in predictive maintenance, *Artif Intell Rev*. **55** (2022) 3183–3211. doi:10.1007/S10462-021-10079-Z/.

[17] P. Nunes, J. Santos, and E. Rocha, Challenges in predictive maintenance – A review, *CIRP J Manuf Sci Technol*. **40** (2023) 53–67. doi:10.1016/j.cirpj.2022.11.004.

[18] T.R. Gruber, A translation approach to portable ontology specifications, *Knowledge Acquisition*. **5** (1993) 199–220. doi:10.1006/knac.1993.1008.

[19] R. Studer, V.R. Benjamins, and D. Fensel, Knowledge engineering: Principles and methods, *Data Knowl Eng*. **25** (1998) 161–197. doi:10.1016/S0169-023X(97)00056-6.

[20] W.N. Borst, Construction of Engineering Ontologies for Knowledge Sharing and Reuse, 1997. https://research.utwente.nl/en/publications/construction-of-engineering-ontologies-for-knowledge-sharing-and- (accessed May 28, 2025).

[21] S.D. Cardoso, C. Pruski, and M. Da Silveira, Supporting biomedical ontology evolution by identifying outdated concepts and the required type of change, *J Biomed Inform*. **87** (2018) 1–11. doi:10.1016/j.jbi.2018.08.013.

[22] Z. Li, Z. Feng, X. Wang, Y. Li, and G. Rao, Analyzing the Evolution of Ontology Versioning Using Metrics, in: Proceedings of 2015 12th Web Information System and Application Conference (WISA), 2015: pp. 112–115. doi:10.1109/WISA.2015.70.

[23] L. Bayoudhi, N. Sassi, and W. Jaziri, Efficient management and storage of a multiversion OWL 2 DL domain ontology, *Expert Syst*. **36** (2019). doi:10.1111/exsy.12355.

[24] A.A. Algosaibi, and A.C. Melton Jr., Three Dimensions Ontology Modification Matrix, in: Proceedings of 2016 2nd International Conference on Information Management (ICIM2016), IEEE, 2016.

[25] F. Grandi, Dynamic Class Hierarchy Management for Multi-Version Ontology-Based Personalization, *J. Comput. Syst. Sci.* **82** (2016) 69–90. doi:10.1016/j.jcss.2015.06.001.

[26] G. Antoniou, and F. V. Harmelen, A semantic web primer, *Choice Reviews Online*. **46** (2004) 46-1523-46–1523. doi:10.5860/choice.46-1523.

[27] B. DuCharme, Learning SPARQL: Querying and Updating with SPARQL 1.1, O'Reilly Media, Inc., 2013.

[28] OWL Working Group, OWL - Semantic Web Standards, *World Wide Web Consortium*. (2019). https://www.w3.org/OWL/ (accessed May 28, 2025).

[29] S. Sbai, M. Reda, C. Louhdi, H. Behja, and R. Chakhmoune, JsonToOnto: Building Owl2 Ontologies from Json Documents, *International Journal of Advanced Computer Science and Applications*. **10** (2019). www.ijacsa.thesai.org (accessed July 18, 2022).

[30] H. Cheong, Translating JSON Schema logics into OWL axioms for unified data validation on a digital manufacturing platform, *Procedia Manuf*. **28** (2019) 183–188. doi:10.1016/j.promfg.2018.12.030.

[31] M. Kuhn, K.J.B. Raton, B. Butcher, and B.J. Smith, Feature Engineering and Selection: A Practical Approach for Predictive Models, Taylor & Francis, 2020. doi:10.1080/00031305.2020.1790217.

[32] A. Zheng, and A. Casari, Feature engineering for machine learning: principles and techniques for data scientists, O'Reilly Media, Inc., 2018.

[33] Pianism | Predictive and Prescriptive Automation in Smart Manufacturing, *ITEA3 CALL 4*. (2018). https://www.pianism.eu/ (accessed November 21, 2020).

[34] J.P. Amezquita-Sanchez, M. Valtierra-Rodriguez, and H. Adeli, Wireless smart sensors for monitoring the health condition of civil infrastructure, *Scientia Iranica*. **25** (2018) 2913–2925. doi:10.24200/sci.2018.21136.

[35] A. Canito, J. Corchado, and G. Marreiros, Bridging the gap between domain ontologies for predictive maintenance with machine learning, in: Proceedings of WorldCist'21 - 9th World Conference on Information Systems and Technologies, Springer, Terceira Island, Azores, Portugal, 2021.

[36] C. Sousa, D. Teixeira, D. Carneiro, D. Nunes, and P. Novais, Knowledge-based decision intelligence in street lighting management, *Integr Comput Aided Eng*. **29** (2022) 189–207. doi:10.3233/ica-210671.

[37] J.R. Hobbs, and F. Pan, Time ontology in OWL, *W3C Working Draft 27*. (2006).

[38] P. Burek, N. Scherf, and H. Herre, Ontology patterns for the representation of quality changes of cells in time, *J Biomed Semantics*. **10** (2019). doi:10.1186/s13326-019-0206-4.

[39] A. Preventis, E.G.M. Petrakis, and S. Batsakis, CHRONOS Ed: A Tool for Handling Temporal Ontologies in Protégé, *International Journal on Artificial Intelligent Tools*. **23** (2014). doi:10.1142/S0218213014600185.

[40] A. Canito, J. Corchado, and G. Marreiros, A systematic review on time-constrained ontology evolution in predictive maintenance, *Artificial Intelligence Review 2021 55:4*. **55** (2021) 3183–3211. doi:10.1007/S10462-021-10079-Z.

[41] E. Anagnostopoulos, S. Batsakis, and E.G.M. Petrakis, CHRONOS: A Reasoning Engine for Qualitative Temporal Information in OWL, *Procedia Comput Sci*. **22** (2013) 70–77. doi:10.1016/j.procs.2013.09.082.

[42] H. Kondylakis, and N. Papadakis, EvoRDF: evolving the exploration of ontology evolution, *Knowl Eng Rev*. **33** (2018). doi:10.1017/S0269888918000140.

[43] R. Peixoto, C. Cruz, and N. Silva, Adaptive learning process for the evolution of ontology-described classification model in big data context, in:

Proceedings of the 2016 SAI Computing Conference (SAI'2016), IEEE, 2016: pp. 532–540. doi:10.1109/SAI.2016.7556031.

[44] A. Kozierkiewicz, and M. Pietranik, A Formal Framework for the Ontology Evolution, in: Intelligent Information and Database Systems, ACIIDS 2019, PT I, Springer, 2019: pp. 16–27. doi:10.1007/978-3-030-14799-0_2.

[45] A. Lourenço, M. Fernandes, A. Canito, A. Almeida, and G. Marreiros, Using simulation to evaluate a concept drift detector for condition-based maintenance, in: Proceedings of 2022 48th Annual Conference of the IEEE Industrial Electronics Society (IECON'22), IEEE, 2022.

[46] A.G.B. Tettamanzi, C. Faron-Zucker, and F. Gandon, Possibilistic testing of OWL axioms against RDF data, *International Journal of Approximate Reasoning*. **91** (2017) 114–130. doi:10.1016/j.ijar.2017.08.012.

[47] R. Peixoto, C. Cruz, and N. Silva, Semantic HMC: Ontology-Described Hierarchy Maintenance in Big Data Context, in: Proceedings of On the Move to Meaningful Internet Systems: OTM 2015 Workshops, Springer, 2015: pp. 492–501. doi:https://doi.org/10.1007/978-3-319-26138-6_53.

[48] S. Benomrane, Z. Sellami, and M. Ben Ayed, An ontologist feedback driven ontology evolution with an adaptive multi-agent system, *Advanced Engineering Informatics*. **30** (2016) 337–353. doi:10.1016/j.aei.2016.05.002.

[49] A.E. Cano-Basave, F. Osborne, and A.A. Salatino, Ontology Forecasting in Scientific Literature: Semantic Concepts Prediction Based on Innovation-Adoption Priors, in: Proceedings of Knowledge Engineering and Knowledge Management (EKAW 2016), Springer, 2016: pp. 51–67. doi:10.1007/978-3-319-49004-5_4.

[50] S. Ghidalia, O.L. Narsis, A. Bertaux, and C. Nicolle, Combining Machine Learning and Ontology: A Systematic Literature Review, (2024). doi:10.48550/arXiv.2401.07744.

[51] T.H. Nguyen, Mining the semantic Web for OWL axioms, Université Côte d'Azur, 2021.

[52] F.N. Al-Aswadi, H.Y. Chan, and K.H. Gan, Automatic ontology construction from text: a review from shallow to deep learning trend, *Artif Intell Rev*. **53** (2020) 3901–3928. doi:10.1007/S10462-019-09782-9/FIGURES/3.

[53] S.H. Venu, V. Mohan, K. Urkalan, and T. Geetha V, Unsupervised Domain Ontology Learning from Text, in: Proceedings of Mining Intelligence and Knowledge Exploration (MIKE 2016), Springer, 2017: pp. 132–143. doi:10.1007/978-3-319-58130-9_13.

[54] K. Belhoucine, and M. Mourchid, A Survey of Ontology Learning from Text, in: Proceedings of 12th International Conference on Advances in Semantic Processing (SEMAPRO 2018), 2018: pp. 14–21.

[55] P. Buitelaar, P. Cimiano, and B. Magnini, Ontology Learning from Text: An Overview, (2005). https://api.semanticscholar.org/CorpusID:18638602 (accessed February 26, 2024).

[56] I. Scheffler, and N. Goodman, Selective confirmation and the ravens: A reply to Foster, *J Philos*. **69** (1972) 78. doi:10.2307/2024647.

[57] L. Bühmann, and J. Lehmann, Universal OWL axiom enrichment for large knowledge bases, *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. **7603 LNAI** (2012) 57–71. doi:10.1007/978-3-642-33876-2_8/.

[58] L. Bühmann, J. Lehmann, P. Westphal, L. Buehmann, J. Lehmann, and P. Westphal, DL-Learner-A framework for inductive learning on the Semantic Web, *Web Semant.* **39** (2016) 15–24. doi:10.1016/j.websem.2016.06.001.

[59] F. Sais, C. Pruski, and M. Da Silveira, Inferring the evolution of ontology axioms from RDF data dynamics, *International Conference on Knowledge Capture (K-CAP 2017)*. (2017). doi:10.1145/3148011.3154472.

[60] D. Malchiodi, and A.G.B. Tettamanzi, Predicting the possibilistic score of OWL axioms through modified support vector clustering, *Proceedings of the ACM Symposium on Applied Computing*. (2018) 1984–1991. doi:10.1145/3167132.3167345.

[61] T.H. Nguyen, and A.G.B. Tettamanzi, An evolutionary approach to class disjointness axiom discovery, *IEEE/WIC/ACM International Conference on Web Intelligence (WI 2019)*. (2019) 68–75. doi:10.1145/3350546.3352502.

[62] A.G.B. Tettamanzi, C.F. Zucker, and F. Gandon, Dynamically time-capped possibilistic testing of SubClassOf axioms against RDF data to enrich schemas, in: Proceedings of the 8th International Conference on Knowledge Capture (K-CAP 2015), Association for Computing Machinery, Inc, 2015. doi:10.1145/2815833.2815835.

[63] A.G.B. Tettamanzi, C. Faron-zucker, and F. Gandon, Testing owl axioms against RDF facts: A possibilistic approach, *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture*

Notes in Bioinformatics). **8876** (2014) 519–530. doi:10.1007/978-3-319-13704-9_39.

[64] Ontology Alignment Evaluation Initiative 2023, (2023). http://oaei.ontologymatching.org/2023/ (accessed February 26, 2024).

[65] How does it work? (OWL Example Wine Agent), *Standford University*. (2003). http://ksl.stanford.edu/projects/wine/explanation.html (accessed March 8, 2024).

[66] Oregon State University, Plant Ontology, (2014). https://planteome.org/ (accessed March 8, 2024).

[67] R. Bruskiewich, E.H. Coe, P. Jaiswal, S. McCouch, M. Polacco, L. Stein, L. Vincent, and D. Ware, The Plant Ontology™ Consortium and Plant Ontologies, *Comp Funct Genomics*. **3** (2002) 137–142. doi:10.1002/cfg.154.

[68] Wikidata, (n.d.). https://www.wikidata.org/wiki/Wikidata:Main_Page (accessed April 8, 2024).

[69] B. Ganter, and R. Wille, Formal Concept Analysis, (2024). doi:10.1007/978-3-031-63422-2.

[70] S. Jabbari, and K. Stoffel, Ontology Extraction from MongoDB Using Formal Concept Analysis, in: Proceedings of 2017 2nd International Conference on Knowledge Engineering and Applications (ICKEA), 2017: pp. 178–182.

[71] M. Rouane-Hacene, M. Huchard, A. Napoli, and P. Valtchev, Using Formal Concept Analysis for discovering knowledge patterns, in: Proceedings of the 7th International Conference on Concept Lattices and Their Applications (CLA'10), 2010.

[72] M.H. Rouane, M. Huchard, A. Napoli, and P. Valtchev, A Proposal for Combining Formal Concept Analysis and Description Logics for Mining Relational Data, *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. **4390 LNAI** (2007) 51–65. doi:10.1007/978-3-540-70901-5_4.

[73] F. Baader, B. Ganter, B. Sertkaya, and U. Sattler, Completing Description Logic Knowledge Bases using Formal Concept Analysis, in: Proceedings of the 20th International Joint on Artificial Intelligence (IJCAI'07), Morgan Kaufmann Publishers Inc., 2007: pp. 230–235.

[74] H. Jia, J. Newman, and H. Tianfield, A new Formal Concept Analysis based learning approach to Ontology building, in: Proceedings of Metadata and Semantics, Springer, 2009: pp. 433–444. doi:10.1007/978-0-387-77745-0_42.

[75] J. Poelmans, D.I. Ignatov, S.O. Kuznetsov, and G. Dedene, Formal concept analysis in knowledge processing: A survey on applications, *Expert Syst Appl*. **40** (2013) 6538–6560. doi:10.1016/j.eswa.2013.05.009.

[76] L. Bühmann, and J. Lehmann, Universal OWL Axiom Enrichment for Large Knowledge Bases, *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. **7603 LNAI** (2012) 57–71. doi:10.1007/978-3-642-33876-2_8.

[77] P. Cimiano, A. Hotho, G. Stumme, and J. Tane, Conceptual Knowledge Processing with Formal Concept Analysis and Ontologies, *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*. **2961** (2004) 189–207. doi:10.1007/978-3-540-24651-0_18.

[78] J. Lehmann, S. Auer, L. Bühmann, and S. Tramp, Class expression learning for ontology engineering, *Journal of Web Semantics*. **9** (2011) 71–81. doi:10.1016/j.websem.2011.01.001.

[79] D. Jean Kouagou, S. Heindorf, C. Demir, and A.-C. Ngonga Ngomo, ROCES: Robust Class Expression Synthesis in Description Logics via Iterative Sampling, (2024). https://github.com/dice-group/ROCES (accessed June 12, 2025).

[80] J. Lehmann, and P. Hitzler, Foundations of Refinement Operators for Description Logics, *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. **4894 LNAI** (2008) 161–174. doi:10.1007/978-3-540-78469-2_18.

[81] J. Lehmann, and P. Hitzler, Foundations of Refinement Operators for Description Logics, *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. **4894 LNAI** (2008) 161–174. doi:10.1007/978-3-540-78469-2_18.

[82] N. Fanizzi, G. Rizzo, C. D'amato, and F. Esposito, DLFoil: Class Expression Learning Revisited, *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. **11313** (2018) 98–113. doi:10.1007/978-3-030-03667-6_7.

[83] P. Monnin, M. Lezoche, A. Napoli, and A. Coulet, Using Formal Concept Analysis for Checking the Structure of an Ontology in LOD: The Example of DBpedia, *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. **10352 LNAI** (2017) 674–683. doi:10.1007/978-3-319-60438-1_66.

[84] S.O. Kuznetsov, and S.A. Obiedkov, Comparing performance of algorithms for generating concept lattices, *Journal of Experimental & Theoretical Artificial Intelligence*. **14** (2002) 189–216. doi:10.1080/09528130210164170.

[85] D. Van Der Merwe, S. Obiedkov, and D. Kourie, AddIntent: A New Incremental Algorithm for Constructing Concept Lattices, *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*. **2961** (2004) 372–385. doi:10.1007/978-3-540-24651-0_31.

[86] L. Mazzola, P. Kapahnke, M. Vujic, and M. Klusch, CDM-Core: A Manufacturing Domain Ontology in OWL2 for Production and Maintenance, in: Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, SCITEPRESS - Science and and Technology Publications, 2016: pp. 136–143. doi:10.5220/0006056301360143.

[87] M. Compton, P. Barnaghi, L. Bermudez, R. Garcca-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W.D. Kelsey, D. Le-Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, and K. Taylor, The SSN Ontology of the W3C Semantic Sensor Network Incubator Group, *SSRN Electronic Journal*. (2012). doi:10.2139/ssrn.3198991.

[88] P. Panov, L. Soldatova, and S. Džeroski, Ontology of core data mining entities, *Data Min Knowl Discov*. **28** (2014) 1222–1265. doi:10.1007/s10618-014-0363-0.

[89] D. Thakker, P. Patel, M. Intizar Ali, T. Shah, V.J. Ramírez-Durán, I. Berges, and A. Illarramendi, ExtruOnt: An ontology for describing a type of manufacturing machine for Industry 4.0 systems, *Semant Web*. **11** (2020) 887–909. doi:10.3233/sw-200376.

[90] F. Baader, I. Horrocks, C. Lutz, and U. Sattler, An Introduction to Description Logic, Cambridge University Press, 2017. doi:10.1017/9781139025355.

[91] OWL 2 Web Ontology Language Direct Semantics, *World Wide Web Consortium*. (2012). https://www.w3.org/TR/owl2-direct-semantics/ (accessed June 25, 2025).

ANNEXES

1. Annex 1 – Ontologies for Predictive Maintenance

In order for the PdM module to generate insights, predictions and potentially diagnoses, it will be necessary to select and properly apply diverse machine learning and data mining algorithms – the application of which will depend not only on the nature of the data, but on the necessary outputs. These algorithms must additionally consider the sensitive temporal requirements of the problem: incoming data will be arriving in near-real time, and the results of these algorithms will have a certain validity period depending on their purpose, and must also be provided to the end users in a timely fashion. Furthermore, while a particular fault may happen at a specific time, the equipment statuses that precede it may be classified as abnormal, i.e., for a particular interval of time, a certain equipment condition may be classified as abnormal.

The different selected ontologies can be interconnected in order to model several of the requirements of the use-case in study. It is important to note that only some of the concepts pertaining to each ontology are presented for the sake of simplicity, and only the most pertinent to the scenario and part of the new relationships are shown. Furthermore, while most relationships between these ontologies will be provided by either new properties or changes in domain/range of existing ones, new concepts may also be required for particular cases.

1.1 Combining and Extending Existing Ontologies

Several classes of CDM-Core [86] and SSN [87] are particularly useful for their ability to describe how different Systems are composed, how Sensors observe specific Properties and how these Properties can be used to understand Symptoms, Faults, Fault States and Component Conditions. Of these, Properties can be analysed by any type of machine learning or data mining algorithm, the execution of which is represented under Onto-DM [88] as Algorithm Execution. For the sake of brevity, attention is brought specifically only to two of its subclasses: Predictive Modelling Algorithm Execution and Pattern Discovery Algorithm Execution. The predictive algorithms will make use of information from Properties, Symptoms, Raw Materials and Extruders in use and System Conditions in order to infer Faults and the probability of new Fault States.

The pattern discovery algorithms, on the other hand, will be used to assess whether a particular Component Condition is Normal or Abnormal, using once again information about Properties and Symptoms. These two types of Component Condition are not specified in the CDM-Core ontology and would therefore need to be added; other classes that can represent the outputs of different algorithms will also be considered.

The CDM-Core ontology does not currently describe the manufacturing domain in the detail required for the use-case. As such, information pertaining to the extruder machines themselves and their internal behaviour can be attributed to ExtrudOnt [89]. Information obtained from the ERP in use, which will include data about manufacturing orders, tools in use and raw materials (among others), will have to be modelled anew.

Figure 61, below, shows existing ontologies in white, while new concepts being considered are represented within the grey packages, which also represent the new ontologies they should belong to.

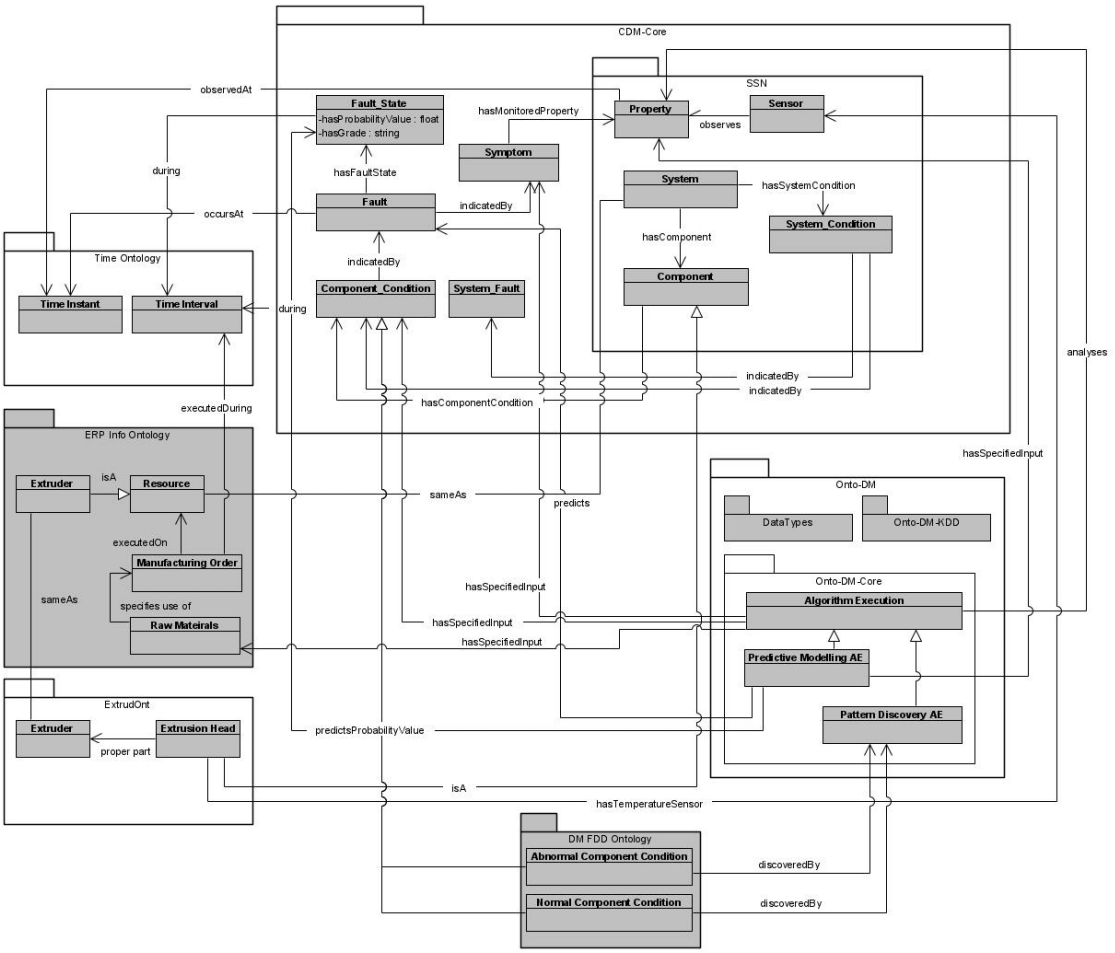

Figure 61 – Interconnecting the chosen ontologies

1.2 Temporal Representation and Reasoning

The present use-case deals with the evolution of equipment behaviour over time, and the ability of machine learning and data mining algorithms to classify this behaviour regarding its deviation from the normal and prediction of failure states. As such, being able to adequately represent the changes the equipment goes through is of supreme importance, as so as keeping a history of previous states. This means that the modelling provided by the ontology must be able to establish that specific quality assignments (e.g. Fault State) are only valid for a given period and may be overlapped totally or partially by other assignments, preferably with the least amount of data duplication as possible. An overview of common ontology patterns for the representation of time-constrained quality assignments is provided in [38], detailing the reasoning behind the different approaches and the tendency of these towards more ontologies and difficulties in maintainability. The paper concludes that reifying only the temporal relationships of concepts (by considering them states of those concepts) has a relatively high complexity and high maintainability for temporally equal qualities (overlapping quality assignments, however, may result in a higher number of quality assignments, and thus should only be applied when strictly necessary). For the purposes of the use-case presented here, two distinct situations arise, namely: (1) representing the time-based aspects of incoming data, which is acquired continuously through different sources and suffers pre-processing and aggregation tasks before being consolidated into a single stream and (2) the results of the machine learning and data mining algorithms applied to this stream, which will deliver time-sensitive results and will also classify periods of the stream (e.g. according to whether they manifest normal or abnormal behaviour).

On the data streams, it is important to note that while these are acquired through different sources (such as the previously mentioned sensors, ERP contextual information and machine protocols) and therefore possibly acquired at different rates, it must suffer a downsampling process – usually based on aggregation – guaranteeing that for each interval data from all sources is equally represented. As such, we can assume that stream data comes at precise intervals and data from one source does not partially overlap that from other source, but does, in fact, occur simultaneously. Furthermore, many algorithms that can be applied to stream processing rely not on the timestamps assigned to each data

point, but the order in which they arrive. The timestamps, however, may still be necessary to assess potential overlaps between other possible resource states and the monitored processes and, therefore, will also be represented.

This sequentiality can be represented by establishing a sequence with a relationship between the data points, as seen in Figure 62:

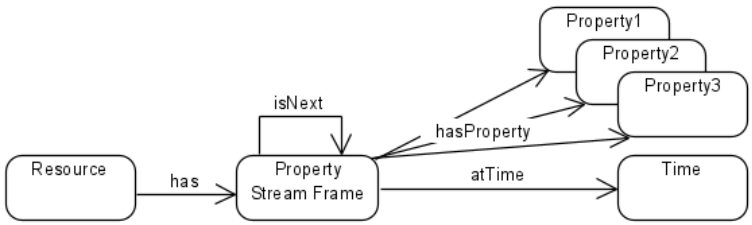

Figure 62 – Relationship between Resources and stream data about monitored Properties

Applying a reified approach to the results of algorithms is already partially covered by the modelling provided by CDM-Core, which separates Component Condition from Component Fault and Fault States and therefore allows the usage of Component Condition as a temporal part of Component. Considering that Component Conditions and Component Faults happen not at an instant but during a period of time, the only requirement is the addition of a new relationship, called "atTimeInterval" in Figure 63.

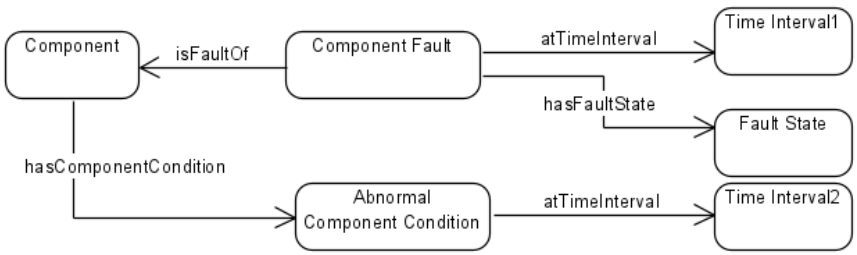

Figure 63 – Representing Component Condition evolution over time

As such, while different Components of the same System may be exhibiting different conditions, a single Component cannot exhibit, at the same time, both normal and abnormal conditions, i.e. they cannot overlap. Any Component Condition can, however, overlap completely or partially with Component Faults and any other quality assignments of Component.

2. Annex 2 – Reducing

A class expression can be composed of a number of other class expressions through means of an operator, namely: Intersection Of (⊓), Union Of (⊔), Complement Of (¬) and One Of (∈). If one looks at the class expression as a list of other class expressions, it is possible that some elements of that list are equivalent or could be derived from one another. To avoid this type of redundancy and ensuring that only the minimum required elements compose a class expression, one can employ a mechanism known as reducing.

The reducing algorithm used by *TICO* is described below, in Algorithm 10. Since $IntersectionOf$ operator is the only one considered in the solution, the Ce will be split into a list of Ces through that operator.

Algorithm 10 – Class Expression Reduction

Input: $Conjunct$
Output: $Conjunct$ (reduced)

con_{list} ← get all Ce in $Conjunct$

for each Ce_1 **in** con_{list} **do**
  for each Ce_2 **in** con_{list} **do**

    if Ce_1 **entails** Ce_2 **then**
      remove Ce_2 from $Conjunct$

    if $SubClassOf(Ce_1, Ce_2)$ is entailed **then**
      remove Ce_1 from $Conjunct$
    if $SubClassOf(Ce_2, Ce_1)$ is entailed **then**
      remove Ce_2 from $Conjunct$

In the context of intents and extents of a lattice, reducing is used to describe the process through which the intents and extents are reduced to the set of elements introduced in a particular node – i.e., elements that are not present in other nodes of the lattice. To do so, Algorithm 11 is used instead.

Algorithm 11 – compute Reduced Intent/Extent

Input: $node$
Output: $elements$ (reduced)

$elements_o$ ← get intent or extent of $node$
$directSupConcepts$ ← get the concepts directly above $node$

for each p **in** $directSupConcepts$ **do**
  $elements$ ← get the intent or extent of p
  remove all $elements$ from $elements_o$