MEDINA

Deliverable D2.3

Specification of the Cloud Certification Language – v1

Editor(s):	Marinella Petrocchi
Responsible Partner:	CNR
Status-Version:	Final - v1.1
Date:	30.09.2022
Distribution level (CO, PU):	PU

Project Number:	952633
Project Title:	MEDINA

Title of Deliverable:	Specification of the Cloud Certification Language – v1
Due Date of Delivery to the EC	31.10.2021

Workpackage responsible for the Deliverable:	WP2 - Certification Metrics and Specification Languages
Editor(s):	Marinella Petrocchi, CNR
Contributor(s):	Marinella Petrocchi, CNR; Michela Fazzolari, CNR; Franz Berger, Fabasoft; Patrizia Ciampoli, HPE; Immanuel Kunz, FgH
Reviewer(s):	Juncal Alonso, Cristina Martinez (TECNALIA)
Approved by:	All Partners
Recommended/mandatory readers:	WP2, WP3, WP4

Abstract:	This is the first of the three deliverables resulting from Task 2.3, Task 2.4 and Task 2.5. This set of deliverables will present the definition and implementation of the Cloud Certification Language which encompasses three major phases: 1) the encoding of requirements of cloud certification schemas – written in natural language -- in a Controlled Natural Language (CNL), so called MEDINA CNL; 2) the editing of the requirements in MEDINA CNL through an editor tool; 3) the mapping of the CNL requirements to a domain specific language DSL.
Keyword List:	MEDINA Cloud Certification Language, MEDINA CNL, CNL Editor, Domain Specific Language, MEDINA Ontology
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/
Disclaimer	This document reflects only the author's views and neither Agency nor the Commission are responsible for any use that may be made of the information contained therein.

Document Description

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	27.01.2021	TOC	Marinella Petrocchi CNR
v0.2	01.09.2021	Revised version of ToC to reflect implementation choices	Marinella Petrocchi CNR
V0.3	28.09.2021	Draft of Sections 3 and 4, Executive summary	Marinella Petrocchi CNR Franz Berger Fabasoft
v.04	04/10/2021	Section 5.3.2 `Cloud Resource Ontology' and Section 6 added	Immanuel Kunz, FgH
v.05	04/10/2021	Draft of Section 2, Section 3 completed, Section 4 revised	Marinella Petrocchi CNR
v.06	07/10/2021	Section 5 and 6.2 added	Patrizia Ciampoli HPE
v.07	11/10/2021	Review of the whole document	Marinella Petrocchi CNR
v0.8	12/10/2021	Review of the whole document	All contributors
v0.9	15/10/2021	Revised version submitted for internal review	Marinella Petrocchi CNR
v0.91	22/10/2021	QA review	Conchi Cortés TECNALIA
v0.99	26/10/2021	Final version after addressing QA review comments	Marinella Petrocchi CNR
v1.0	27/10/2021	Ready for submission	Leire Orue-Echevarria TECNALIA
v1.01	22/07/2022	Revised version of Executive Summary and Introduction	Michela Fazzolari CNR
v1.02	25/07/2022	Revised version of Section 4	Michela Fazzolari CNR
v1.03	28/07/2022	Revised version of Sections 5 and 6	Immanuel Kunz FgH
v1.04	01/08/2022	Revised version of Section 5	Patrizia Ciampoli HPE
v1.05	01/08/2022	Revised version of Section 3	Marinella Petrocchi CNR
v1.06	23/08/2022	Merged changes done by Patrizia Ciampoli at 08/08/2022 and review	Michela Fazzolari CNR
v1.07	29/08/2022	Addressed all comments received in the internal QA review	Marinella Petrocchi CNR
v1.1	30/09/2022	Ready for submission	Cristina Martínez (TECNALIA)

Table of contents

List of figures

Terms and abbreviations

API	Application Programming Interface
BNF	Backus-Naur Form
CCL	Cloud Certification Language
CNL	Controlled Natural Language
CNL4DSA	Controlled Natural Language for Data Sharing Agreement
CRSO	Cloud Resource Security Ontology
CSA or EU CSA	Cybersecurity Act
DCG	Discounted Cumulative Gain
DoA	Description of Action
DSL	Domain Specific Language
EC	European Commission
EUCS	European Cybersecurity Certification Scheme for Cloud Services
GA	Grant Agreement to the project
GUI	Graphical User Interface
ICT	Information Communications Technology
IDCG	Ideal Discounted Cumulative Gain
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
MTS	Modal Transition System
nDCG	Normalized Discounted Cumulative Gain
NL	Natural Language
NLP	Natural Language Processing
OPA	Open Policy Engine
P@k	Precision at k
PCA	Principal Component Analysis
REO	Requirement & Obligations
RS	Requirements Specifications
SVD	Singular Value Decomposition
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege
SW	Software
TOM	Technical and Organizational Measure
TSNE	T-distributed Stochastic Neighborhood Embedding
TSVD	Truncated Singular Value Decomposition
UI	User Interface
UML	Unified Modelling Language
XACML	eXtensible Access Control Markup Language
XML	Extensible Markup Language

Executive Summary

This deliverable (D2.3) is a public report that describes the progress made and the results obtained in the first year of the MEDINA project, regarding Work Package 2 `Certification Metrics and Languages', and in particular tasks 2.3, 2.4 and 2.5, so called `Cloud Certification Language' tasks. This is the first deliverable on this topic, the next deliverables will be D2.4 and D2.5, respectively `Specification of the Cloud Security Certification Language v2' and `Specification of the Cloud Security Certification Language Final', due in months 24 and 30 of the project.

The other tasks included in WP2 are 2.1, 2.2 and 2.6. The first two tasks are described in deliverable D2.1 [2], while the latter will be described in deliverable D2.6 [3] at month M15.

The final aim of the MEDINA project is the development of a framework for achieving a continuous audit-based certification for Cloud Service Providers based on cybersecurity certification schemes.

As reported in deliverable D2.1, a preliminary comparative analysis of four schemes has been carried out, together with a mapping of the controls of the schemes. The goal of this mapping is to provide guidance in the transitioning toward the EUCS candidate scheme (European Cybersecurity Certification Scheme for Cloud Services) [3].

For this reason, from now on, WP2 will focus on the security controls and requirements included in the EUCS. Furthermore, D2.1, in Section 3.1, defines the scope of the certification requirements for MEDINA. Such requirements have been selected according to two requisites: i) their 'assurance level' is 'high'; and ii) they include the wording `automatically monitor' or variations thereof. Thus, WP2 is aligned with this definition and, at least for the first year of the project, the focus is on the 33 EUCS requirements identified in D2.1.

The main aim of this deliverable is to investigate how to render the security requirements of the chosen certification schema, which are expressed in Natural Language (NL), into a language that can be automatically `executed' by a machine. To achieve this, two sequential translations are made: the first from NL to Controlled Natural Language (CNL), the second from CNL to Domain Specific Language (DSL).

The first translation from NL to CNL allows a security requirement to be expressed in a formal intermediate language, still understandable for a human user. The translation of a requirement is done automatically in two steps, and it is the aim of task 2.3.

The first step involves associating a set of metrics with a requirement. This is necessary because a requirement expressed in natural language is not objectively measurable, whereas metrics can be measured and then assessed automatically. Similar to requirements, metrics present a description in NL, so this first step relies on NLP techniques to associate metrics to requirements according to their textual similarity. The component dedicated to this step has been named Metric Recommender.

The second step translates the pair requirement-metrics into the MEDINA CNL, which expresses the possibility, or the necessity, that the metrics associated to a requirement assume certain values. The Medina CNL thus expresses policies, which may take the form of permissions or obligations, as we shall see later in this document.

Since the first of the two steps is complex, the output of the association should be checked by an experienced user. This check is indeed performed in task 2.4, where the output of the previous task can be inspected through a dedicated tool, named CNL editor, and an expert user is involved to confirm or change the association of metrics to a requirement. Through this tool,

it is also possible to modify some parameters of the obtained CNL. The CNL editor relies on a vocabulary and an ontology to describe the terms used in the CNL representation. Once the user is satisfied with the CNL representation of the policies, the second translation can take place. This second translation is called 'mapping', to distinguish it from the previous one that translated requirements and metrics from NL to CNL and its aim is to map policies from CNL into DSL. The mapping phase is the main output of task 2.5, which provide policies in DSL, a formal language that is not easy to be understood by humans but can be processed automatically. The output of this task will later be sent to the assessment tools, described in deliverable D3.4 [4].

In conclusion, the main results of D2.3 are the following:

- The definition of a strategy to associate a requirement with metrics that can be used to objectively measure it.
- The study and the definition of the MEDINA CNL for defining a unique, easy-to-read, yet rigorous format for the policies associated to a security requirement and associated metrics.
- The definition of the CNL editor tool that allows an expert user to validate the outcome of the association between a requirement, a set of metrics, and related policies.
- The definition of the first version of a Vocabulary and Ontology for MEDINA.
- The presentation of a suitable DSL i.e., the machine readable language expressing policies that will be used as input for the assessment tools.

1 Introduction

1.1 About this deliverable

This document describes the procedural steps and the results obtained during the first year of the MEDINA project to translate the requirements of the EUCS draft candidate cloud certification scheme [3] from natural language to a machine-readable language. The EUCS requirements, together with the associated metrics, can be seen as policies, i.e., rules that the Cloud Provider may/must fulfil in order to obtain the certification. The expression of these rules in the machine-readable language, so called MEDINA Domain Specific Language (MEDINA DSL), will allow the MEDINA assessment tools (presented in Deliverable 3.4 [4]) to automatically process them.

This is the first deliverable on this topic, the next deliverables will be D2.4 `Specification of the Cloud Security Certification Language v2' (m24) and D2.5 `Specification of the Cloud Security Certification Language Final' (m30).

1.2 Document structure

The document is structured as follows.

- Section 1 is the current section.
- Section 2 explains the motivations for the adoption of a Cloud Certification Language in MEDINA and describes the methodology followed for its definition and for representing Cloud Certification requirements and metrics into this language.
- Section 3 introduces the reader to the concept of Controlled Natural Languages and gives examples of their applications and advantages of their adoption. After this introduction, the section presents the MEDINA Controlled Natural Language.
- Section 4 describes the approach to pass from the Cloud Certification requirements and metrics, originally written in natural language, to the MEDINA Controlled Natural Language policies. In particular, the section illustrates how to automatically associate metrics to a requirement, via Natural Language Processing techniques.
- Section 5 presents the CNL editor tool that allows to modify the MEDINA Controlled Natural Language policies.
- Section 6 introduces the Rego code and related playground, motivating its adoption in the project as the ultimate MEDINA Cloud Certification Language.
- Finally, Section 7 gives final remarks.

2 The Cloud Certification Language

2.1 Motivation

Cloud certification schemes consist of a set of rules, technical requirements, standards and procedures to strengthen the cybersecurity of ICT services and products offered to citizens, and their terms and conditions are usually published in natural language. This is the case, e.g., of the EUCS draft candidate Cloud Certification Scheme [3][1]. Thus, a rigorous translation procedure is required to produce a machine-readable format out of textual NL requirements. This translation should minimise as much as possible human intervention - which is prone to errors and time consuming.

The Cloud Certification language is the language that the MEDINA consortium will develop in the course of the project, a language which will permit to express rules for cloud certification, in a uniform way and without the ambiguity of natural language (the latter being natively more complex). In addition, this language will be machine readable and will be the input of the MEDINA Assessment Tools. The methodology followed by the consortium to achieve the ultimate Cloud Certification Language is defined in Section 2.2.

2.2 Methodology

For a lean and seamless *trait d'union* between what is dictated by official documents of the European Commission in terms of certification and the definition of the MEDINA DSL, we intend to proceed as follows:

1) Semi-automatically translate Natural Language (NL) certifications terms and conditions, as they appear on official documents like the EUCS scheme, into policies expressed in a Controlled Natural Language (CNL) [6].
2) Visualize and possibly revise the generated CNL via a CNL editor tool, to verify the generated policy statements before proceeding with the mapping to the MEDINA DSL.
3) Map the CNL to a runtime-enforceable DSL language that can be used by the MEDINA assessment tools to check the compliance status of the certification terms and conditions.

Figure 1 highlights the components of the MEDINA architecture that constitute the building blocks to achieve the Cloud Certification Language.

- The CNL translator translates EUCS NL requirements into their MEDINA CNL representation. Part of the translation from NL to CNL will be done via NLP techniques (see Section 4). The CNL translator is the main output of MEDINA task 2.3.
- The CNL editor is the user interface that allows users to visualize and possibly revise the translation of the requirements into the MEDINA CNL. The editor architecture is introduced in Section 5. The CNL editor is the main output of task 2.4.
- The DSL mapper is the MEDINA component that maps the yet not executable MEDINA CNL into the MEDINA Domain Specific Language (DSL), whose statements are instead machine-readable. The DSL is the Cloud Certification Language. The DSL mapper is the main output of task 2.5.

[1] At the time of writing this deliverable, the public version of the EUCS is the one published in December 2020.

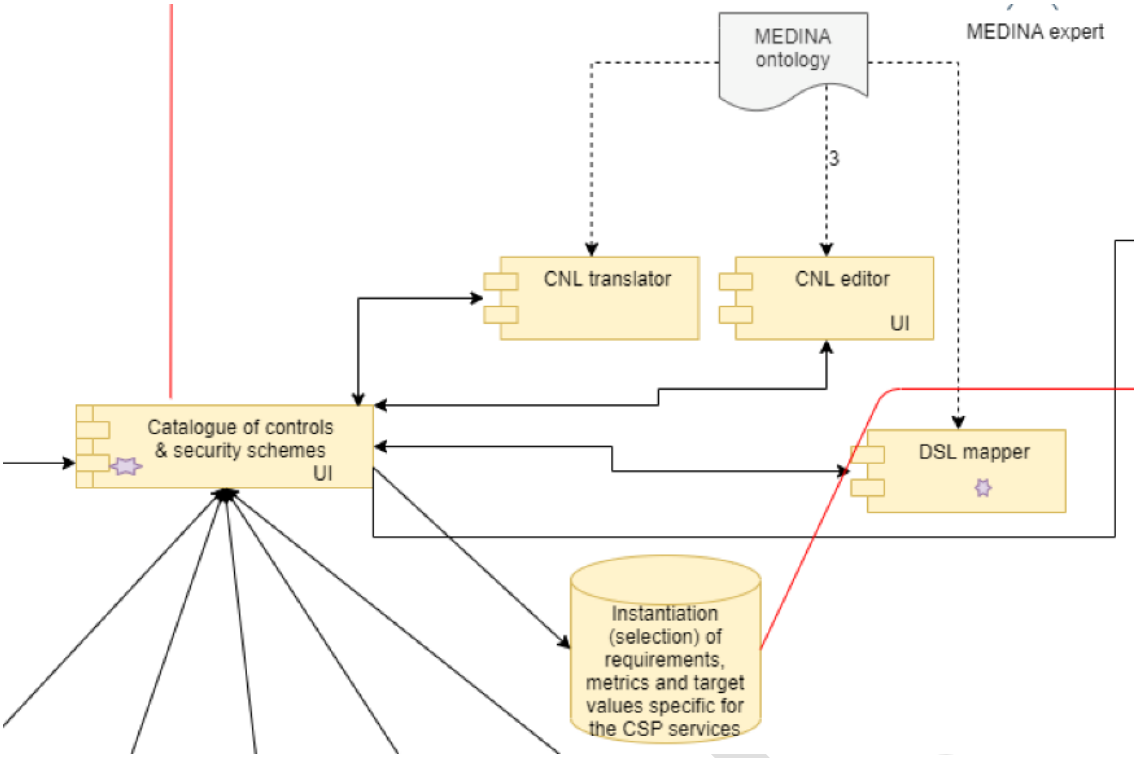

Figure 1. Excerpt of MEDINA architectural workflow, related to the building blocks of the MEDINA Cloud Certification Language (excerpt from source: D5.1 [7])

All the components interface with the catalogue of controls and security schemes, described in Deliverable D2.1 [2]. For the representation of the requirements, all the components rely on the MEDINA ontology, which will be presented in Section 5.3.

In the following, for each of the three steps/components listed above, we show the results of the first iteration of their implementation and how we achieved it. Before, we introduce the reader to CNLs and to the MEDINA CNL.

3 Patterns and Controlled Natural Languages for Requirements specifications

This section provides an introduction to Controlled Natural Languages (CNLs) and their benefits in various fields of applications, including the MEDINA scenario.

Quoting from [8], `*natural language (NL) is pervasive in [...] requirements specifications (RSs). However, despite its popularity and widespread use, NL is highly prone to quality issues such as vagueness, ambiguity, and incompleteness*.' Work in [9] identifies common issues affecting usability of NL requirements, like, e.g., the adoption of an unclear – or a too complex, or too poor – terminology, the missing of a requirements specification template, the duplication of requirements that simply use a diverse wordiness, the omission of significant descriptions, the vagueness and the ambiguity of terms and meaning.

With the aim of improving the quality of NL requirements, by, e.g., reducing its ambiguity and making them more precise and complete, Pohl proposes the following three approaches [11]:

- **Glossaries**. Requirements glossaries are lists of specific words that make explicit and provide definitions for the salient terms in a RS. Glossaries may also provide information about the synonyms, related terms, and example usages of the salient terms [12].
- **Patterns**. They are pre-defined sentence structures with optional and mandatory components. Patterns guide stakeholders in writing more standardized NL requirements by restricting their syntax, and thus possibly avoiding omissions and/or over-specifications.
- **Controlled natural languages**. Controlled natural languages (CNLs) are a subset of natural languages, specifically conceived to make language processing simpler. A CNL is, in essence, a developed language that is based on natural language, but it is more restrictive in terms of lexicon, syntax, semantics, while at the same time retaining most of its natural properties. CNLs prevent quality problems in requirements documents, while maintaining the flexibility to write and communicate requirements in an intuitive and universally understood manner [6]. CNLs are considered an extension of the pattern category which, in addition to restricting the syntax (the grammatical structures), also provide language constructs with which it is possible to precisely define the semantics of NL requirements. By adopting a contrived representation, in terms of grammar and vocabulary, CNLs may reduce the ambiguity and complexity of a complete language [13], e.g., English, Spanish, French, Swedish, Mandarin, etc. [14].

In the following, we will concentrate on the last two approaches (patterns and CNLs), being language representations the target of this section.

3.1 Patterns

Many patterns have been proposed in the literature. In [15], Phol and Rupp present a single pattern to specify functional software requirements, while Mavin et al., in [16], consider the aviation domain and propose a set of fine patterns to describe functionalities of the domain. Requirements writing according to these patterns has been proved to be easier to understand and less ambiguous.

Both functional and not functional requirements have been presented in [17] by Withall, all related to the business domain, while Riaz et al. in [18] define a set of patterns expressing security requirements. The latter comes with an assistant tool that helps the user in selecting the appropriate pattern according to the security aspects relevant in the NL requirement.

Among others, we cite here two other works specifying patterns for the description of embedded systems (Denger et al. [19]) and performance requirements (Eckhardt et al. [20]).

From this non-exhaustive review of existing patterns for drafting NL requirements, we can see that many of them are domain-dependent, that is, laced with terms specific to a certain domain.

3.2 Controlled Natural Languages

CNLs have been proved to be effective in mitigating linguistic ambiguity challenges, as they can easily be translated into a formal language such as first-order logic or different version of description logic, automatically and mostly deterministically (Schwitter, [13]). CNLs are formal *per se*, being born with an associated formal semantics. In Section 3.3, we will give an example of the semantics for a language ideated by some members of the MEDINA consortium and conceived for expressing privacy regulations (Matteucci et al, [21]). In general, CNLs can conveniently express the kind of information that occurs in, e.g., software specifications, formal ontologies, business rules, legal and medical regulations.

One interesting feature of CNLs is that they usually maintain a readability that is not so different from that of pure natural languages. This makes them easier to write and understand by people than pure formal languages. Furthermore, they precisely define subsets of natural languages, have a formal foundation and can therefore be adopted for automated reasoning (Schwitter, [13]).

CNLs have been proposed and used in many application domains and for different purposes. For example, the Attempto Controlled English (ACE)[2] [22] is a CNL that defines a subset of the English language intended to be used in different domains, such as software specification and the Semantic Web. Attempto can be automatically translated into first-order logic and it is supported by a number of tools, like a Parsing Engine to translate ACE texts into a variant of first-order logic, or a plug-in for the Protége ontology editor[3].

In [23], Hart et al. propose Rabbit as a way to overcome the impediment to the creation and adoption of ontologies. Rabbit is a CNL that can be translated into the Ontology Web Language[4] (OWL) in a way that achieves both comprehension by domain experts and computational preciseness. In a sense, Rabbit can be defined as complementary to OWL, supporting the need to author and understand domain ontologies but overcoming the difficult comprehension of descriptions logics.

Similarly, the idea behind the Sidney OWL Syntax SOS [24] is to propose a new syntax that can be used to write and read OWL ontologies in CNLs. SOS enables the generation of grammatically correct full English sentences to and from OWL syntax. This enables users to write an OWL ontology in a defined subset of English, also improving readability and understanding of OWL statements.

Regarding CNLs developed for specific domains, as an example Konrad and Cheng in [25] consider the automotive domain and propose a language to express real-time properties of systems under specification. As a follow-up of the same language, Post et al in [26] enlarge its expressivity to express other requirements in the same domain.

Just like other types of requirements, EUCS controls and TOMs are expressed in natural language. Although the use of natural language enables users (e.g., CSPs) to read and

[2] http://attempto.ifi.uzh.ch/site/
[3] https://protege.stanford.edu/
[4] https://www.w3.org/OWL/

understand the requirements specific for Cloud Security Certification, a key issue relies in the fact that NLs are not machine readable, and automatic measurements on whether controls and TOMs are actually going to be processed and fulfilled are not feasible. In particular, NL cannot be used as the input language for a rule-assessment software infrastructure to be used for automated, continuous evaluation of TOMs' fulfilment. In fact, such an evaluation requires inputs in a machine-readable form, like, e.g., the de facto standard XACML moving to the field of access control rules.

Like other (cloud) certification schemes, the EUCS scheme controls are groups of statements – the TOMs indeed - that can be likened to policies. There are several examples of CNLs coined by Academia and Industry to express such policies. In the following, we introduce the reader to languages expressing policies for data management, including the one that has inspired our MEDINA CNL.

3.3 CNLs for expressing policies for secure data management

In the last two decades, data protection has been discussed in many scenarios, from critical infrastructures to social networks, just to cite two fields of interest. Informally, regulations for data protection, data storage and data sharing are written in Natural Language[5,6]. A proper CNL is a flexible mean to fill the gap between a traditional legal contract regulating the sharing of data among different domains, and the software architecture supporting it.

Some examples of languages and associated tools for secure data management are as follows.

- Binder [27] is an open logic-based security language that encodes security authorizations among components of communicating distributed systems.
- The Rodin platform provides an animation and model-checking toolset, for analyzing properties of specifications based on the Event-B language[7], able to express security data policies. In [28], the developers of Rodin and Event-B presented a formalization of data management clauses in Event-B, and a model checker is exploited to verify that a system behaves according to its associated clauses.
- Also, work in [29] proposes a comprehensive framework for expressing highly complex privacy-related policies, featuring purposes and obligations. The Klaim family of process calculi [30] provides a high-level model for distributed systems, for programming and controlling access and usage of resources. Also, work in [31] considers policies that moderate the use and replication of information, e.g., imposing that a certain information may only be used or copied a certain number of times. The analysis tool is a static analyser for a variant of Klaim.

We will now describe a language defined by some members of the MEDINA consortium in a previous work. The *Controlled Natural Language* for *Data Sharing Agreement* (CNL4DSA) was introduced with the purpose to reduce the barrier of adoption of data policies in terms of security and privacy as well as to ensure policies mapping to formal languages that allow the automatic verification of the policies [21]. A data sharing agreement is essentially a contract between two or more parties to agree on some terms and conditions with respect to data

[5] Justice Information Sharing, U.S. Dept. of Justice. Online: https://bja.ojp.gov/program/it
[6] North American Electronic Reliability Corporation. Electricity Information Sharing and Analysis Center. Online: https://www.nerc.com/pa/CI/ESISAC/Pages/default.aspx
[7] http://www.event-b.org/platform.html

sharing, storage and usage. This language, called for the sake of brevity CNL4DSA, permits simple, yet formal, specifications of different classes of privacy policies, as listed below:

- *authorizations*, expressing the permission for subjects to perform actions on objects (e.g., data), under specific contextual conditions;
- *obligations*, defining that subjects are obliged to perform actions on objects, under specific contextual conditions.

Central to CNL4DSA is the notion of *fragment*, i.e., a tuple ***f =<s, a, o>*** where *s* is the subject, *a* is the action, *o* is the object. A fragment simply says that `subject *s* performs action *a* on object *o*'.

By adding the can/must constructs to the basic fragment, a fragment becomes an *authorization* or an *obligation*.

Fragments are evaluated within a specific context *c*, i.e., a predicate that characterizes factors such as user's roles, data categories, time, geographical locations, etc. Contexts are evaluated either as true or false. Simple examples of contexts are `subject hasRole CSP', or `object hasCategory CloudResource'. In order to describe complex policies, contexts are composable. A *composite context C* is defined inductively as follows

$$C = c \mid \text{C } \textbf{and } \text{C} \mid \text{C } \textbf{or } \text{C} \mid \textbf{not } \text{C}$$

Where **and**, **or**, and **not** are Boolean connectors.

The syntax of a composite fragment F_M is described by the following Backus-Naur Form BNF-like syntax:

$$F_M = nil \mid \textbf{mod } f \mid F_M; F_M \mid \textit{if } C \textit{ then } F_M$$

where the modality **mod** ranges over {**can**, **must**} and the subscript *M* ranges over {*A*, *O}* where *A* stands for Authorization and *O* stands for Obligation. By changing the modality and the subscript, we get two different types of policies, namely we have authorizations and obligations. F_A is a composite authorization fragment and F_O is a composite obligation fragment.

Let us now comment on the individual policy constructors:

- *nil* does nothing.
- **mod** *f* is the atomic authorization/obligation fragment that expresses that f = <s,a,o> is allowed/obliged. Its informal meaning is that "*subject s can/must perform action a on object o"*.
- F_M ; F_M is a list of composite fragments. (Subscript *M* takes either only *O* or only *A*).
- *If C then* F_M expresses the logical implication between a composite context *C* and a composite fragment F_M: if *C* holds, then F_M is permitted/obliged (according to the value of *M*).

CNL4DSA has an operational semantics based on a Modal Transition System (MTS), able to express *admissible* and *necessary* requirements to the behaviour of the CNL4DSA specification. To model the behaviour of the specifications, modal transition systems are used. In its original version [32], MTS is a structure

$$(\mathcal{A}, \mathcal{S}, \rightarrow\Diamond, \rightarrow\boxdot)$$

where $\mathcal{S}$ is a set of specifications, like for example processes in the context of Process Algebras,

$\mathcal{A}$ is the set of actions which specifications may perform, and $\rightarrow\Diamond, \rightarrow\boxdot \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ are the two modal transition relations expressing admissible and necessary requirements to the behavior of the specifications.

In particular, $S \xrightarrow{a} \Diamond S'$ with S, S' $\in \mathcal{S}$ and $a \in \mathcal{A}$ means that it is admissible that the implementation of *S* performs a and then behaves like S'. Dually, $S \xrightarrow{a} \boxdot S'$ represents a transition in which the implementation of *S* is required to perform a and then behaves like *S'*.

This works under the assumption that all the required transitions are admissible transitions.

Figure 2 shows the operational semantics of F_A in terms of a modified label transition system $MTS_{Auth} = (\mathcal{AUT}, \mathcal{F}, \rightarrow\Diamond, \mathcal{C})$. As usual, rules are expressed in terms of a set of premises, possibly empty (above the line) and a conclusion (below the line).

Let $\rightarrow_\Diamond$ be the smallest subset of $\mathcal{AUTH} \times \mathcal{F} \times \mathcal{AUTH}$, closed under the following rules:

$$(can)\ \frac{}{can\, f \xrightarrow{f}_\Diamond nil} \qquad (if)\ \frac{C = true \quad F_A \xrightarrow{f}_\Diamond F'_A}{if\ C\ then\ F_A \xrightarrow{f}_\Diamond F'_A}$$

$$(;)\ \frac{F1_A \xrightarrow{f}_\Diamond F1'_A}{F1_A; F2_A \xrightarrow{f}_\Diamond F1'_A; F2_A}$$

Figure 2. Operational Semantics for the composite authorization fragment, where the symmetric rule for (;) is omitted. Source: unpublished manuscript, Petrocchi M and Matteucci I.

MTS_{Auth} deals with authorized transitions only and it considers also the set of contexts because the transitions may depend also on the value of such contexts, see rule (*if)* in Figure 2.

The introduction of $\mathcal{C}$ (= a set of predicates) in a labelled transition system is a standard practice [33]. We observe that the *if* operator implies the binding of variable appearing in the context *C.*

The operational semantics of F_O is expressed in terms of the modal transition system $MTS_{Obl} = (OBL, \mathcal{F}, \rightarrow\boxdot, \mathcal{C})$. The axioms and rules are similar to the ones presented for F_A apart from changing the transition relation, that becomes $\rightarrow\boxdot$.

3.4 MEDINA CNL

In this section, we introduce the definition of the MEDINA CNL, along with the motivations leading to this format.

We remind the reader that the scope of the certification requirements for MEDINA was defined in D2.1 (Section 3.1) [2] where we focused on a subset of EUCS requirements that were identified from the draft version of the EUCS scheme released on December 2020. These requirements were selected based on two requisites: i) their 'assurance level' is 'high'; and ii) they include the wording "automatically monitor" or variations thereof.

Inspired by the presence of the authorization and obligation modalities in CNL4DSA, presented in Section 3.3, we further analysed the EUCS draft candidate certification scheme and were able to summarize the requirements, being either technical or organizational [5, p. Appendix B], in the following general textual formula:

The value assumed by the metric x on the resource of type y can/must be equal/major to/minor to the target value z/must fall in the range of values {z, ...w}.

Paraphrasing part of CNL4DSA, we define a MEDINA *fragment* as a tuple

$$f = <rt, m, type(op,tv)>$$

where *rt* is a resource type, *m* is a metric, *type(op,tv)* specifies the type of the metric, the designed target value *tv* for that metric, and the operator *op* which relates the value of the metric to *tv* (e.g., *=*, *>*, *<*, etc.).

A MEDINA fragment says that "the metric *m*, measured on the resource type *rt,* has a specific relation with the value *tv* of type *type*, based on the operator *op"*.

This leads to the following syntax for the MEDINA CNL:

$$F_M = nil \mid \textbf{mod}\ f \mid F_M; F_M$$

where *M* ranges over {*A*, *O*} (Authorization/Obligation) and

- nil does nothing.
- **mod** *f* is the atomic authorization/obligation MEDINA fragment that expresses that *f = <rt, m, type(op,tv)>* is allowed/obliged. Its informal meaning is that *"the metric m, measured on the resource type rt, can/must have a specific relation with the value tv of type type, based on the operator op".*
- F_M ; F_M is a list of composite MEDINA fragments.

Analysis of the EUCS certification scheme and connected metrics, set forth in D2.1 [2], have highlighted how the TOMs identified at month 12 can be viewed as *obligations*. In particular, the CSP must fulfil specific obligations regarding, e.g., the security configurations of cloud resources.

For this reason, at month 12, and therefore hereafter in this document, the format for expressing in a more formal, but always readable way, the NL TOMs, is represented as follows:

must *<rt, m, type(op,tv)>*

With a little abuse of notation, we will express the MEDINA obligation in this format too:

RT* must *M type(op,tv)

where, as introduced above, ***RT*** is the resource type, ***M*** is a metric associated to the TOM, ***tv*** is a target value, ***op*** is the comparison operator, which indicates how to compare the target value with the value measured on the resource, with respect to metric ***M***; finally, ***type*** indicates the unit of measure of the target value and the measured value.

It is worth noting that, having included both the *must* and the *can* modalities in the MEDINA CNL, it will be possible – when and if necessary – to express not only obligation requirements but also authorization requirements. Also, it will be possible to express a list of composite MEDINA fragments.

Section 4.2 will show some concrete examples of TOMs rendered into the MEDINA obligations.

As a final note, it is fair to point out that other projects have developed more complex languages, and also equipped them with tools devoted to analysing properties of the system that such languages describe. This is the case, for example, of the AMASS (Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems)[8] European project. The target of AMASS is the specification of safety and security requirements for Cyber-

[8] https://www.amass-ecsel.eu/

Physical Systems, which probably feature many variabilities in their specification than requirements for cloud security certification.

AMASS includes multiple modelling languages according to the level of user expertise. They can be based on UML, associated with editor tools with many plug-ins or have formal foundation and as such can be given as specification to property verification tools (see AMASS Deliverable 3.6 `Prototype for architecture-driven assurance' [36]).

The language chosen to represent MEDINA's requirements is a much simpler language, with no correlated analysis tools. However, we argue that the language is suitable for MEDINA's purpose, which is to create a close-to-standard language for the representation of cloud certification requirements, and which is then made machine readable and input to the assessment tools specified in D3.1 [5] (see Section 4).

4 From NL to CNL TOMs

The main goal of passing from a NL representation to a CNL representation of the security requirements and associated metrics is to help achieving the automatic and continuous monitoring of the EUCS scheme.

This process is carried out through two consecutive steps. The first step consists in associating each requirement with one or more predefined metrics. In fact, to get a compliance status, each requirement needs to be objectively evaluated and thus it should be associated with metrics that reflect the technical details of the requirement itself. The second step is represented by the translation of each requirement / associated metric into a policy, expressed in CNL. The two steps will be detailed in the following sections. In particular Section 4.1 describes the proposal of a system to automatically associate metrics to requirements, whereas Section 4.2 reports the actual translation of requirements/metrics into policies. We remind the reader that, for the scope of the certification at month 12, the policies will consist in obligations. This does not preclude the fact that it is possible to express also permissions since the MEDINA CNL has both modalities, *must* and *can* (see Section 3.4).

Figure 3 depicts the simplified workflow of task 2.3: first, for every requirement, a set of metrics is recommended/predicted, then the set is translated into the defined CNL. The output is stored in a dedicated database managed by the CNL editor (see Section 5), since it will be used by the Editor itself to visualize/refine the result of the recommendation.

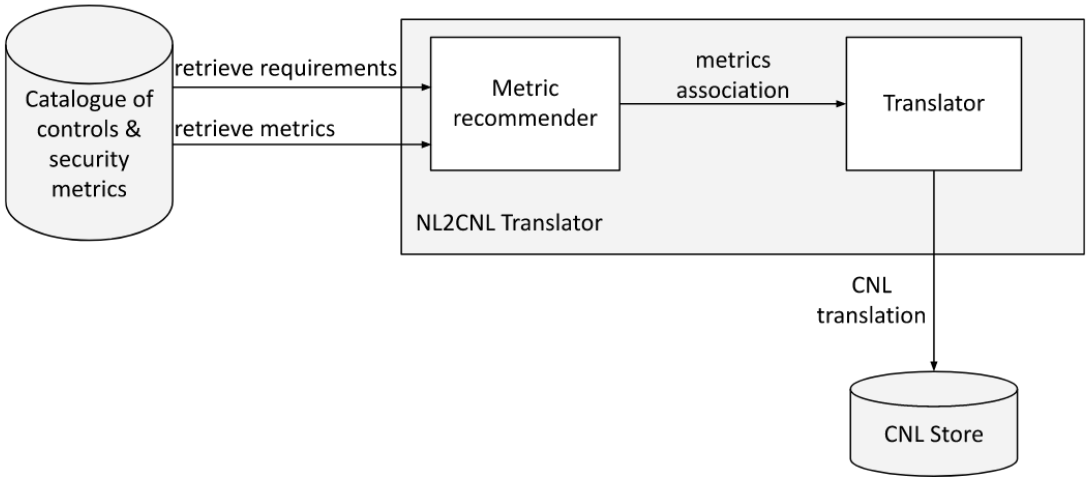

Figure 3. From Natural Language to Controlled Natural Language: Simplified overview (source: MEDINA's own contribution)

4.1 Metric association

This section explains the process carried out to automatically associate metrics to requirements. The idea behind this proposal comes from the fact that the metrics should be reusable when adding new requirements and the manual association between metrics and requirements is a time-consuming process. For these reasons, a tool for automatic association is proposed, namely a "metric recommender" system. This step is constructed as a research project and therefore single parts might change in the future. This means that the input data, or to be more exact the computed features, will be selected based on what produces the best results, which is an iterative research process.

However, the basic workflow should be fairly consistent over time. To this aim, input data (requirements and metrics, both expressed in NL) are selected from the MEDINA catalogue database and fed into the recommender system. The idea is to associate metrics to a requirement according to the similarity of their descriptions. To do so, both requirements and metrics descriptions are represented into vectors of numerical features by relying on NLP techniques, with the hypothesis that similar requirements and features descriptions will reside in the same local area in the feature space. The quality of the process is actively supervised in visual analysis and experiment results are computed using state of the art metrics. The following subsections will give more details about this process.

4.1.1 Data and features

The input of the metric recommender system is taken from the catalogue, which currently includes the requirements available in the EUCS scheme [3]. In particular, the first input is represented by the natural language description of the requirements. The first experiment will focus only on security requirements with assurance level "high". The natural language description of the metrics is also used as a second input for the metric recommender system.

4.1.1.1 Requirement input data example

The following is an example for a high-level security requirement description in natural language (ReqID=*OIS-02.3H*, category=*Organisation of Information Security*):

> The CSP introduces and maintains a manually managed inventory of conflicting roles and enforces the segregation of duties during the assignment or modification of roles as part of the role management process. OIS-02.3H – EUCS 2020 Version

4.1.1.2 Metric input data example

The following is an example for a metric description in natural language:

> "This metric is used to assess if access monitoring is enabled" M237 – MEDINA metric proposal

4.1.2 Experimental setup

First, the input texts are processed using a pre-trained network to produce a high dimensional feature vector. This feature vector represents the requirement or metric in a mathematically comparable way, instead of their original textual description. The basis for the recommender system is the hypothesis that requirements having similar descriptions are expected to be closer in the feature space. Since the same features are also computed for the metrics, this should also hold for the metrics. Which means, that, if the features are selected/fine-tuned to this purpose, the metrics, that should be associated with a requirement, are in the same local area in this high dimensional feature space.

4.1.2.1 Features

As described above, the input for the metric recommender system, i.e., the model that associates metrics with a requirement, is the computed feature vectors. The features can be computed using knowledge of the domain (e.g., for audio, spectrograms can be used). Alternatively, the features can be learned using machine learning. To train such a model, usually a lot of data is needed (in our case text), which is not available. Therefore, we use the output of pre-trained models that have worked well on state-of-the-art Natural Language Processing tasks.

For our use case, two types of features are considered applicable – the ones related to context-aware models (e.g. BERT [41] and derivatives [42]) and context-free word embeddings (e.g. word2vec [43] or fastText [44]). In the first experiments, fastText features (on metric/requirement description text) have shown the most promising results so far. Feature selection is of uttermost importance for the final outcome and, since this task is ongoing, it will be probably refined in the future.

The quality of the selected features directly influences the final result; therefore, some data cleaning is needed. In our case, this is mostly based on removing stop words [45]. This means that we remove words that appear often, but do not add useful information to the text.

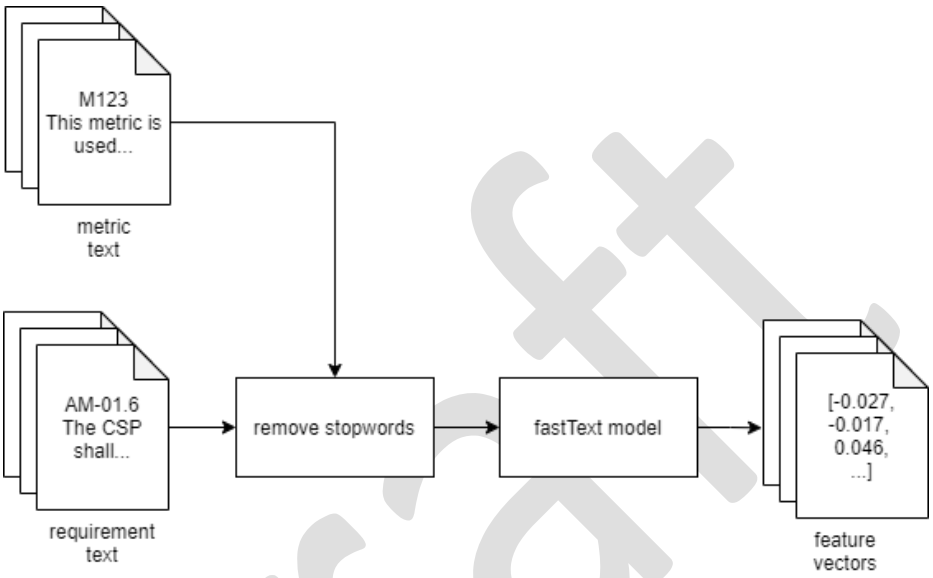

Figure 4. Feature computation workflow

4.1.2.2 Current approach

Features are computed using the fastText model *cc.en.300* which returns a 300-dimensional vector per requirement/metric (see Figure 4.). The fastText model is pre-trained on English texts from Wikipedia and Common Crawl [46]. For visual analysis, this high dimensional vector is reduced to two dimensions using the principal component analysis (PCA) [47] or the T-distributed Stochastic Neighbour Embedding (TSNE) [48] or the Truncated SVD [49].

A K-d tree is computed on the feature vectors of the metrics, which can be used to select the k closest neighbours of a query vector, based on the shortest Euclidean distance. As a query, we use the feature vector of a requirement. The workflow is depicted in Figure 5, and result examples are provided in Section 4.1.3.2.

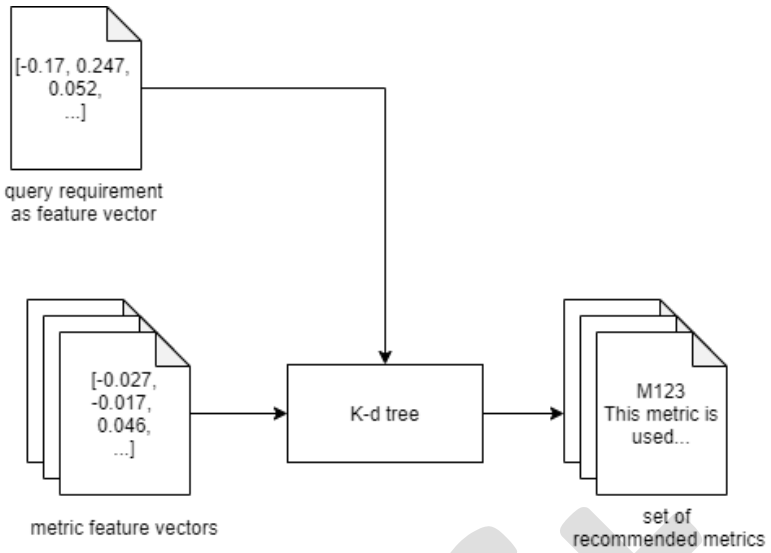

Figure 5. Recommender system workflow (source: MEDINA's own contribution)

4.1.2.3 Performance indicators

To analyse the performance of our system, basic indicators can be used. These indicators allow us to compare different approaches and help choosing the most promising one. Precision@K is typically the metric of choice for evaluating the performance of a recommender systems. However, additional diagnostic metrics and visualizations can be used, since they can offer deeper insights into a model's performance. First experiments results are based on the following indicators.

4.1.2.3.1 Precision@k

A quality metric to evaluate model's performance could be *precision@k* [50], e.g., *precision@5* reflects how well the system performs in the top 5 recommendation results. However, this score does not consider the rank of the relevant metrics only whether the relevant metrics are in the set of retrieved metrics. Equation 1shows the adjusted formula to calculate the precision@k score for this task. The numerator in Equation 1is the amount of metrics in the intersection of *relevant metrics* (=metrics associated with a requirement) and *k* is the number of *retrieved metrics* (=recommended metrics for a requirement), the denominator is the amount of *relevant metrics*.

$$precision@k = \frac{|\{relevant\ metrics\} \cap \{k\ retrieved\ metrics\}|}{|\{relevant\ metrics\}|} \qquad \textit{Equation 1}$$

4.1.2.3.2 Normalised Discounted Cumulative Gain nDCG

An alternative quality metric to *precision@*k is the Discounted Cumulative Gain [51] (DCG), which is an indicator that can be used to measure the quality of our recommender system results, including the rank of relevant documents. The resulting metric list is ranked, and the metrics associated with the query requirement (=relevant documents) receive a relevance score $rel_i=1$, while metrics not relevant are set to $rel_i=0$. The position of the document/metric is denoted by *i*. The DCG is calculated as defined in Equation 2. To make the DCG metric comparable to other results it needs to be normalized – therefore an ideal DCG (Equation 3) is calculated, reflecting the ideal result – all associated metrics are in the top results of the

recommender. The DCG is normalized to a score between 0 and 1 using the ratio of *DCG* to *IDCG* (Equation 4).

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{log_2(i+1)} \quad \textit{Equation 2}$$

$$IDCG_p = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{log_2(i+1)} \quad \textit{Equation 3}$$

$$nDCG_p = \frac{DCG_p}{IDCG_p} \quad \textit{Equation 4}$$

4.1.3 First results

For the purpose of testing the **performances** of the metric **recommender**, and only for **this scope**, **at time of writing** we considered not only the 33 **EUCS** requirements mentioned at the beginning of this document, but also others, resulting in a total of 44. Such 44 requirements have been manually associated with 108 metrics in total. This means that these can be actively used for testing this recommender system prototype. So far, the quality of the approach was visually analysed using interactive plots and filtering of the results and measured using the above defined nDCG.

4.1.3.1 Visual analysis

Figure 6 depicts three plots using the first two components of the down-projected features. The reduction has been done using TSNE, PCA and Truncated SVD, from left to right. Each coloured dot represents either a requirement or a metric. Visual analysis indicates that the features are mostly good, as the distribution of different schemes and metrics are clustered on top of each other. This empirically supports the hypothesis the recommender system is built on.

Especially in the PCA plot, it is noticeable that some metrics are skewed. Using the interactive analysis tool built for examining the data, anomalies such as this one can be directly investigated.

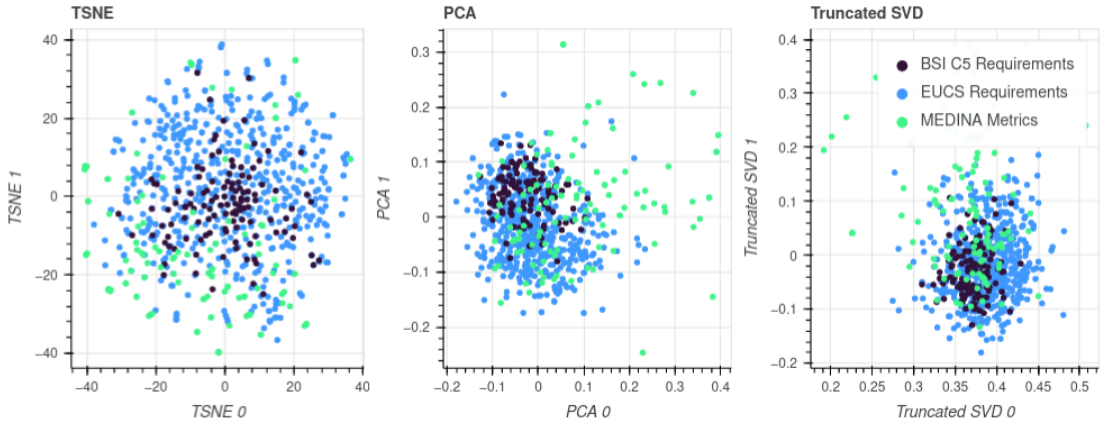

Figure 6. Plot of requirements and metrics using the first two components of the feature vectors, down-projected using TSNE, PCA and Truncated SVD respectively (source: MEDINA's own contribution)

4.1.3.2 Results

For 27 out of 44 requirements at least a subset of the linked metrics could be retrieved within the top 10 results (see Figure 7 and Figure 8). For the rest 17 requirements, no metrics could be retrieved (for an example see Figure 9). In total the mean of the nDCG is 0.41.

Disregarding the 17 requirements, for which no metrics could be retrieved yet, the "corrected" mean nDCG is 0.66. For 12 requirements 100% of the associated metrics were retrieved in the top results. An example of the latter can be seen in Figure 7. The nDCG was computed as described in Section 4.1.2.3.1.

Query:

	ID	ReqID	EUCS Control ID	Description
85	AM-01.6	NaN	AM-01	The CSP shall automatically monitor the inventory of assets to guarantee it is up-to-date

Mapped metrics:

	ID	ReqID	EUCS Control ID	Description	relevance	rank
68	M212	AM-01.6	AM-01	This metric is used to assess if the inventory of assets is regularly monitored	1	1
69	M213	AM-01.6	AM-01	This metric is used to assess if the inventory if assets are regularly monitored against policies	1	2

Recommended:

	ID	ReqID	EUCS Control ID	Description	relevance	rank
69	M213	AM-01.6	AM-01	This metric is used to assess if the inventory if assets are regularly monitored against policies	1	1
68	M212	AM-01.6	AM-01	This metric is used to assess if the inventory of assets is regularly monitored	1	2
7	M155	HR-03.5	HR-03	Check if there is a possibility to monitor the verification of acknowledgement of security policies automatically	0	3
8	M156	HR-04.7	HR-04	Check if exists a possibility to monitor the completion of the security awareness and training program automatically	0	4
78	M222	PM-04.7	PM-04	The check that exists an automatic functionality to monitor compliance	0	5
19	M164	PSS-04.3	PSS-04	Are integrity checks of VM and container images automatically monitored?	0	6
87	M231	ISP-03.7	ISP-03	Check if security approvals and exceptions are automatically monitored	0	7
79	M223	PM-04.7	PM-04	The check that the results of the monitoring automatically use in the listed procedures: • Configuration of system components;\n• Performance and availability of system components;\n• Response time to malfunctions and security incidents; and\n• Recovery time (time until completion of error handling).	0	8
40	M184	OPS-07.2	OPS-07	This metric is used to assess if a self service portal for data backup monitoring is available.	0	9
109	M253	IAM-03.11	IAM-03	Monitoring for log events produced by automated mechanisms to check if they are working properly	0	10

DCG: 2.35. IDCG: 2.35 nDCG: 1.0

Figure 7. Prototypical results for EUCS requirement AM-01.6, optimal results on rank 1 and 2 (source: MEDINA's own contribution)

Query:

	ID	ReqID	EUCS Control ID	Description
94	AM-03.6	NaN	AM-03	The approval of the commissioning and decommissioning of hardware shall be digitally documented and automatically monitored.

Mapped metrics:

	ID	ReqID	EUCS Control ID	Description	relevance	rank
70	M214	AM-03.6	AM-03	This metric is used to assess the existence of digital record of the commissioning requests including the approval or denial	1	1
71	M215	AM-03.6	AM-03	This metric is used to assess the existence of digital record of the decommissioning requests including the approval or denial	1	2

Recommended:

	ID	ReqID	EUCS Control ID	Description	relevance	rank
79	M223	PM-04.7	PM-04	The check that the results of the monitoring automatically use in the listed procedures: • Configuration of system components;\n• Performance and availability of system components;\n• Response time to malfunctions and security incidents; and\n• Recovery time (time until completion of error handling).	0	1
34	M178b	IM-03.4	IM-03	This metric is used to assess if the automated incident remediation mechanism requires user approvals.	0	2
74	M218	AM-04.4	AM-04	This metric is used to assess the existence of the information related to the verification of the secure configuration of the mechanisms for error handling, logging, encryption, authentication and authorisation according to the intended use and based on the applicable policies, before authorization to commission the asset can be granted	0	3
109	M253	IAM-03.11	IAM-03	Monitoring for log events produced by automated mechanisms to check if they are working properly	0	4
19	M164	PSS-04.3	PSS-04	Are integrity checks of VM and container images automatically monitored?	0	5
87	M231	ISP-03.7	ISP-03	Check if security approvals and exceptions are automatically monitored	0	6
71	M215	AM-03.6	AM-03	This metric is used to assess the existence of digital record of the decommissioning requests including the approval or denial	1	7
70	M214	AM-03.6	AM-03	This metric is used to assess the existence of digital record of the commissioning requests including the approval or denial	1	8
108	M252	AM-04.4	AM-04	No. of alerts raised for employees without or outdated acknowledgment record	0	9
10	M157b	HR-05.4	HR-05	Check if exist external employees with accesses granted after termination or change of employment, which should have been revoked according to the outcomes of the decision-making procedure	0	10

DCG: 0.94. IDCG: 2.35 nDCG: 0.4

Figure 8. Prototypical results for EUCS requirement AM-03.6, results on rank 7 and 8 (source: MEDINA's own contribution)

Query:

	ID	ReqID	EUCS Control ID	Description
441	IM-03.4	NaN	IM-03	The CSP shall allow customers to actively approve the solution before automatically approving it after a certain period

Mapped metrics:

	ID	ReqID	EUCS Control ID	Description	relevance	rank
33	M178a	IM-03.4	IM-03	This metric is used to assess if automated incident management (detection, response) and SIEM has been enabled on a cloud service / asset	1	1
34	M178b	IM-03.4	IM-03	This metric is used to assess if the automated incident remediation mechanism requires user approvals.	1	2
89	M233	IM-03.4	IM-03	(BSI-C5 / Sim-04) Check if customers have the ability to review security incident solutions.	1	3
90	M234	IM-03.4	IM-03	(BSI-C5 / Sim-04) Check if security incident solutions are up to date.	1	4

Recommended:

	ID	ReqID	EUCS Control ID	Description	relevance	rank
13	M158	HR-05.4	HR-05	Check if access rights are revoked on contract termination or change according to the decision making procedure automatically	0	1
74	M218	AM-04.4	AM-04	This metric is used to assess the existence of the information related to the verification of the secure configuration of the mechanisms for error handling, logging, encryption, authentication and authorisation according to the intended use and based on the applicable policies, before authorization to commission the asset can be granted	0	2
10	M157b	HR-05.4	HR-05	Check if exist external employees with accesses granted after termination or change of employment, which should have been revoked according to the outcomes of the decision-making procedure	0	3
9	M157a	HR-05.4	HR-05	Check if exist internal employees with accesses granted after termination or change of employment, which should have been revoked according to the outcomes of the decision-making procedure	0	4
7	M155	HR-03.5	HR-03	Check if there is a possibility to monitor the verification of acknowledgement of security policies automatically	0	5
79	M223	PM-04.7	PM-04	The check that the results of the monitoring automatically use in the listed procedures: • Configuration of system components;\n• Performance and availability of system components;\n• Response time to malfunctions and security incidents; and\n• Recovery time (time until completion of error handling).	0	6
8	M156	HR-04.7	HR-04	Check if exists a possibility to monitor the completion of the security awareness and training program automatically	0	7
2	M92	NaN	-	percentage of relevant employees who have received training on the privacy program and policies in place.	0	8
4	M105	NaN	-	the percentage of employees who have certified their acceptance of responsibilities for activities that involve handling of private data.	0	9
87	M231	ISP-03.7	ISP-03	Check if security approvals and exceptions are automatically monitored	0	10

DCG: 0.0. IDCG: 3.7 nDCG: 0.0

Figure 9. Prototypical results for EUCS requirement IM-03.4, no results (source: MEDINA's own contribution)

4.1.4 Limitations of the Metric Recommender system

Selecting the correct metrics and associating them with different requirements based on their textual description is a highly complex task, and a recommendation system that solves this task can produce reliable results only up to a certain point. Another problem is caused by the fact that the available data for training are limited, and a pre-trained model may not generalize well to the actual scenario.

One possibility to overcome the scarcity of data is to expand the number of requirements by using other security requirement datasets available in the research literature. Even then, however, there is the issue of whether or not the association obtained between requirements and available metrics is reliable. This association could be made once and for all by an expert on a subset of data (annotation), and then used to be integrated as a-priori knowledge into the learning process. In this way, it would be possible to use additional algorithms, particularly those for supervised or semi-supervised learning, which generally lead to results with better performance.

In any case, the purpose of the metric recommender is not to replace security experts, but to aid them in the tedious process of finding and associating metrics with different requirements, while also taking into account possible future updates.

For this reason, the output derived from the automatic association must be post-processed through the CNL Editor so that an experienced user can remove associations between a requirement and undesirable metrics.

4.2 CNL translations

Once the list of metrics associated to a requirement is obtained, it is possible to translate each pair requirement/metric in CNL. Here, we give some examples of translation from NL to CNL. For the time being, the translation is done manually, since, for the first part of the project, we mainly focused on the metric recommender system in task 2.3. Thus, the automatic translation from NL to CNL must still be implemented. This will be done in the next months.

Let the reader consider the following requirement:

If we consider EUCS requirement ReqID = OPS-21.3, category Operational Security [3]

> The CSP shall automatically monitor the service components under its responsibility for compliance with hardening specifications.

The metric recommender associates this requirement with the metric metricID=JavaVersion, whose description is: "This metric is used to assess the Java Runtime version used by the cloud service/asset". The metric data type is Integer in the range of [< 11; 11]. The target value is 11, and the operator that compares the target value with the measured value is "=". This metric can thus be expressed as the following obligation:

```
"Application" MUST "JavaVersion", Integer(=,11)
```

For the sake of clarity, we give another translation example:

If we consider EUCS requirement ReqID = PM-04.8, category Procurement Management [3]:

> The CSP shall automatically monitor Identified violations and discrepancies, and these shall be automatically reported to the responsible personnel or system components of the Cloud Service Provider for prompt assessment and action.

The metric recommender associates this requirement with the metric metricID=AutomaticallyDetectedViolationsDiscrepancies, whose description is: "The percentage of violations and discrepancies which can be automatically detected". The metric data type is Integer, ranged over [0;100]. The target value is 100, and the operator that compares the target value with the measured value is "=". Then, this metric can be expressed as the following obligation:

```
"ComplianceMonitoringSoftware" MUST
"AutomaticallyDetectedViolationsDiscrepancies", Integer(=,100)
```

Finally, we report the example of a requirement associated with two different metrics, i.e., the requirement ReqID = OPS-5.3, category Operational Security:

> The CSP shall automatically monitor the systems covered by the malware protection and the configuration of the corresponding mechanisms to guarantee fulfilment of OPS-05.1.

The metric recommender associates this requirement with metrics MetricID=MalwareProtectionEnabled and MetricID=MalwareProtectionOutput, respectively.

The description of the metric MalwareProtectionEnabled is “This metric is used to assess if the antimalware solution is enabled on the respective resource”, its data type is Boolean and the values it can assume are [true, false]. The default target value is “true” and the operator that compares the target value with the measured value is “=”. Then, this metric is translated into the following obligation:

```
“Compute.VirtualMachine” MUST “MalwareProtectionEnabled”,
                        Boolean(=,true)
```

Similarly, the description of the metric MalwareProtectionOutput is “This metric states whether automatic notifications are enabled (e.g. e-mail) about malware threats. This relates to EUCS definition of continuous monitoring”. The metric data type is Boolean and the values it can assume are [true, false]. The default target value is “true” and the operator that compares the target value with the measured value is “=”. Then, this metric is translated into the following obligation:

```
“Compute.VirtualMachine” MUST “MalwareProtectionOutput”,
                        Boolean(=,true)
```

It is worth highlighting that the example obligations shown so far are reported in the CNL format, whereas in the actual implementation they will be rendered in XML, according to the language expected by the CNL Editor.

5 The CNL Editor component

In this section we describe the CNL editor tool, which is an asset that HPE has developed starting from a tool built in past EU funded projects (CocoCloud, C3ISP[10]) and that will be customized and enriched with new specific implementations for MEDINA.

The CNL Editor has the objective to refine metrics assignments for a requirement, by visualizing and possibly modifying the obligations that come as output of the CNL translation described in Section 4. With the CNL editor tool, a user has the possibility to: 1) review the association between requirement and metrics, done by the Metric Recommender; and 2) change some values (i.e., operator and target value). Furthermore, in the event that the user does not wish to send one or more obligations to the assessment tools, they can be deleted via the Editor.

The CNL Editor is a Web GUI application where a user can login, view and edit the requirements and obligations saved in the CNL Store, an internal repository that keeps all the authored obligations of the CNL Editor. Hereafter, we describe the prototype of the Editor with its main functionalities.

5.1 Operational Flow and functionalities

Figure 10 represents the flow operated by the CNL Editor, the steps are described in detail below.

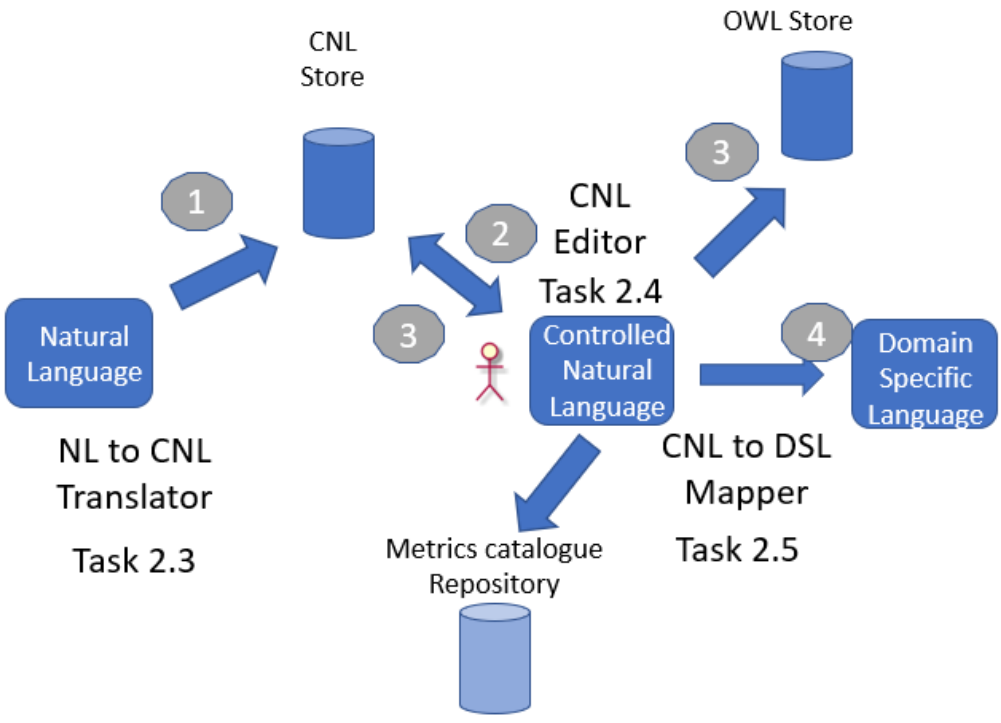

Figure 10. CNL Editor flow (source: MEDINA's own contribution)

1 As output of the NLP process and CNL translation (see Section 4), an object called REO (Requirement & Obligations) is created and saved in the CNL Store as an XML file (a CNL document compliant with CNL Editor XML format). The XML file includes the

[10] https://cordis.europa.eu/project/id/610853; https://www.c3isp.eu/

requirement's metadata and the metrics associated to the requirement by the Metric Recommender, translated into obligations expressed in CNL.

2. A user with specific access to the CNL Editor can view the list of REOs and select one of them to perform four available operations (Show, Edit, Map, Delete). We will give an explanation for each of them in the rest of this section.
3. If the Edit function is selected, the user can, for each requirement, change some items in the CNL obligation, like the operator and the target value, which are based on vocabulary and ontology built for MEDINA use cases and the CNL Editor. After completion of user activity, the CNL Editor writes the modified XML into the CNL Store, making it available for the DSL mapping process.
4. The final step is about the DSL mapping process (invoked by the user via the Map operation) that transforms the CNL document in a usable Domain Specific Language DSL file.

Thus, the CNL Editor gives the possibility to visualize and refine, for each requirement, the obligations the requirement is associated with.

In the following, we describe the main functionalities of the CNL Editor.

List of Requirements: When a user accesses the CNL Editor from the MEDINA UI, she/he can see all REO objects previously created (see Figure 11).

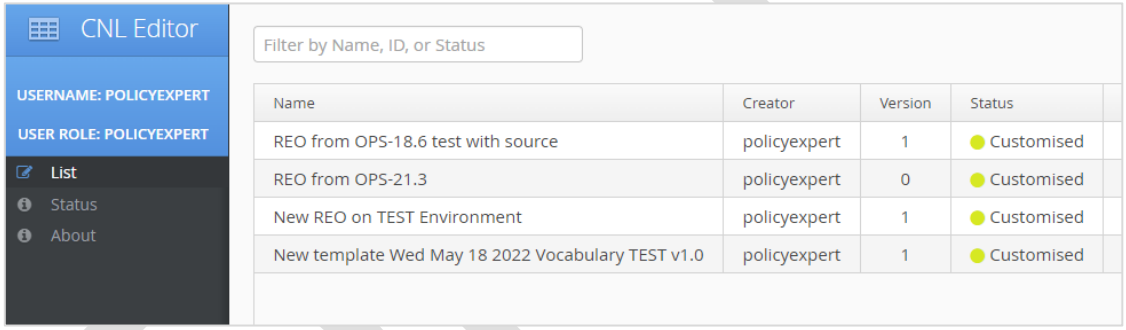

Figure 11. CNL Editor: List of REOs (source: MEDINA's own contribution)

Requirement selection: the user can select a REO and then choose an operation on it: Show, Edit, Delete, Map.

Show -- **Visualization of a specific Requirement**: when a user chooses "Show" for a specific requirement, a page appears showing the requirement's metadata and the list of associated obligations (see Figure 12).

Edit/Delete -- **Modification of a specific Requirement**: when a user chooses "Edit", a page appears, showing the requirement's metadata and the associated obligations. The user can perform some actions on each obligation, such as modify the operator, if available, change the value for the Target Value or delete the obligation itself. Wanting to change a target value or an operator, or to delete part of the obligations associated with a requirement, may be necessary for the specific needs of a particular Cloud Provider, for whom default values do not fit. Also, again, a particular Cloud Provider may not need to enforce certain obligations, and therefore deletes them before passing to the DSL Mapper phase.

Map -- **Map of a specific Requirement**: when the user is satisfied with the obligations associated to a requirement, she/he can invoke the "Map" functionality by clicking on

the correspondent button. The Requirement is then mapped, i.e. the CNL Editor calls the DSL Mapper that will convert this requirement into Rego Code (see Section 6).

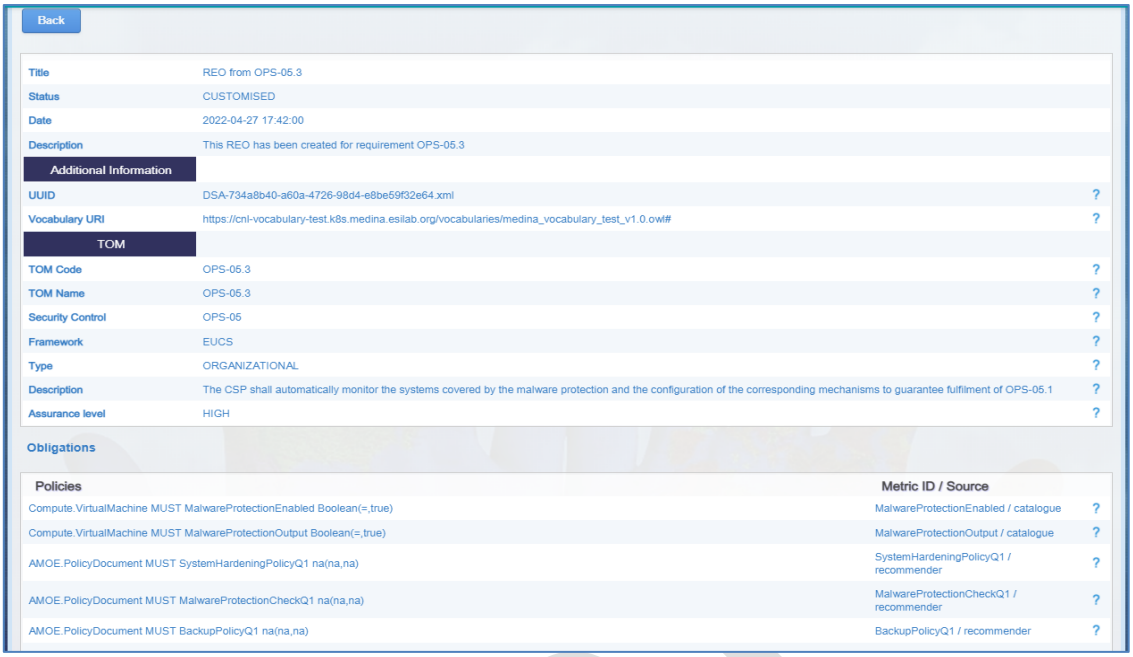

Figure 12. CNL Editor: Metadata and obligations of a requirement (source: MEDINA's own contribution)

5.2 CNL Editor architecture

The architecture of the CNL Editor, at the current state of the project, is depicted in Figure 13.

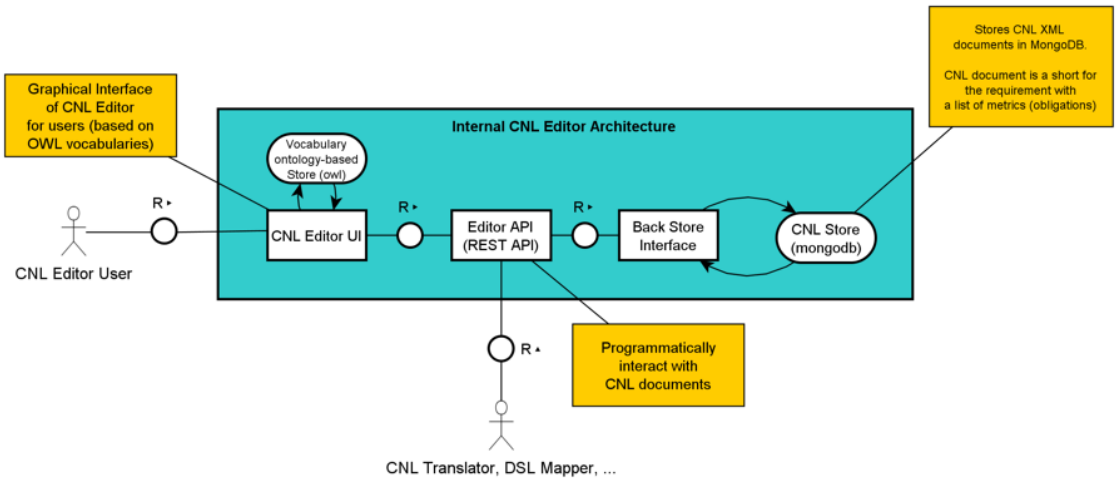

Figure 13. CNL Editor Internal Architecture (source: MEDINA's own contribution)

The internal components of the CNL Editor are:

- **UI**: the graphical User Interface is the front-end for the user to interact with the functionalities of the CNL Editor and it is based on a web application accessible via an Internet browser.
- **Vocabulary**: an ontology store based on OWL (Web Ontology Language by W3C), containing items needed for modifying CNL obligations. Ontology structure is

predefined and follows some CNL Editor constraints that are necessary to represent the terms on the UI.

- **Editor API**: a set of API that allows programmatically access to CNL documents, such as, e.g., to feed obligations to the CNL Editor following the MEDINA CNL syntax), or to allow the DSL Mapper to access CNL documents (in XML). The Editor API is available as a set of RESTful web services.
- **CNL Store**: a repository where the CNL documents (requirements metadata and obligations) are saved. It is accessed via a Back Store Interface, which is a set of API used to decouple the underlying CNL Store technology (i.e., MongoDB).

5.3 Vocabularies and ontologies

This section presents the MEDINA Ontologies that are at the base of the formation of CNL obligations in the CNL Editor tool, and of the rules in Rego Code (the latter presented in Section 6).

Regarding the relation of the two ontologies, they are mostly independent, but they both include a cloud resource taxonomy that will be aligned in a future iteration.

5.3.1 Background: Taxonomies and Ontologies

Taxonomies are classifications of arbitrary objects, usually in a hierarchical order. Their purpose is to create abstract classes to which any concrete instance can be assigned. This way, general methods and tools can be developed that apply to all instances of a certain class. Taxonomies exist in many different domains, for example in biology to classify animals and plants or in computer science to classify security flaws or cloud resources (as proposed in this deliverable).

Sometimes, taxonomies are not sufficient to describe classes and their relationships. Ontologies can integrate several taxonomies and describe their relations in arbitrary ways. In MEDINA, for instance, we use an ontology to subsume two taxonomies which describe cloud resources and security features, and add relationships between the two taxonomies to describe which cloud resource generally has which security feature.

5.3.2 Editor Ontology

The CNL Editor functionality of managing requirements, as described in Section 5.1 , is based on a specific vocabulary, saved in a .owl file (RDF format). CNL Editor allows users, for a selected requirement, to change the Target Value choosing from a predefined list saved in the vocabulary. The use of free text is also allowed in special situations (e.g., where there is a need of a value that cannot be enumerated in the ontology, like an integer number). This way, the user is interactively bound to make changes as defined in the vocabulary.

The CNL Editor vocabulary consists of entities (items) that are defined under *Action* and *Term* classes. Additional actions and terms must be defined as subclasses of these existing classes in any vocabulary that will be created.

In the CNL Editor, the *Term* (and its subclasses) is used to instantiate resource types and target values, while the *Action* (and its subclasses) is used to instantiate metrics. Figure 14 shows an example of the vocabulary, with *Action* and *Term* examples highlighted.

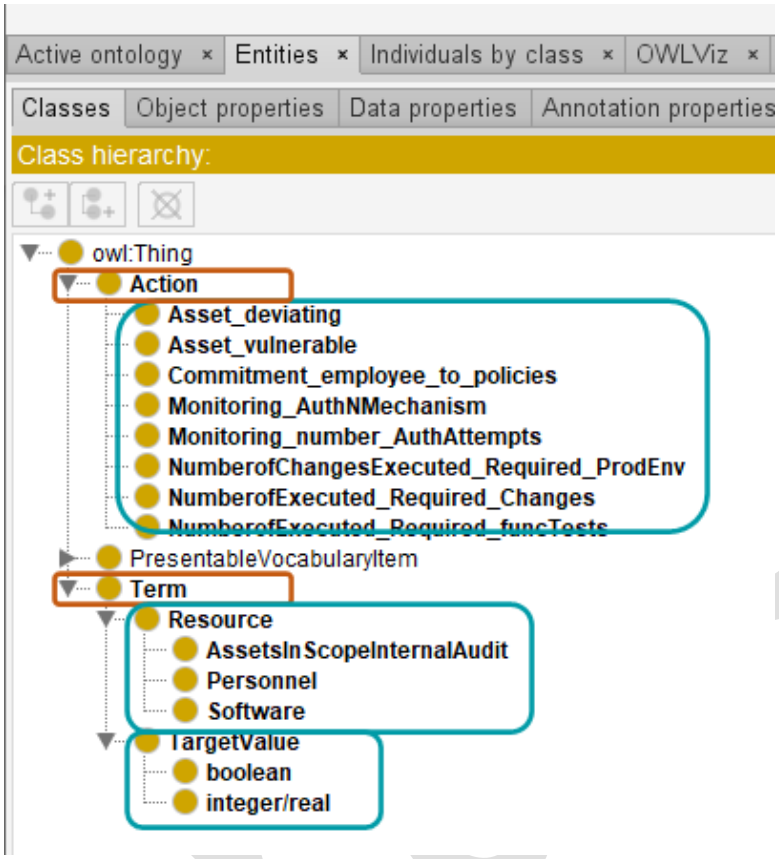

Figure 14. CNL Editor Vocabulary Structure Example (source: MEDINA's own contribution)

In the CNL Editor Vocabulary for MEDINA project, items come from the Catalogue of controls and security metrics. As anticipated, *Actions* are the Metrics Names and under the *Term* class we can find: i) Resource that contains ResourceTypes, and ii) TargetValue with the TargetValueTypes and values admitted.

For each Metric in the vocabulary, the type of resource and the type of target value associated are specified. The association can be retrieved from the Catalogue of controls and security metrics.

When editing an obligation, the user is allowed to choose only values defined in the vocabulary.

As an example, Figure 15 shows the association of ResourceType and TargetValueType to the Metric named BackupEncriptionEnabled.

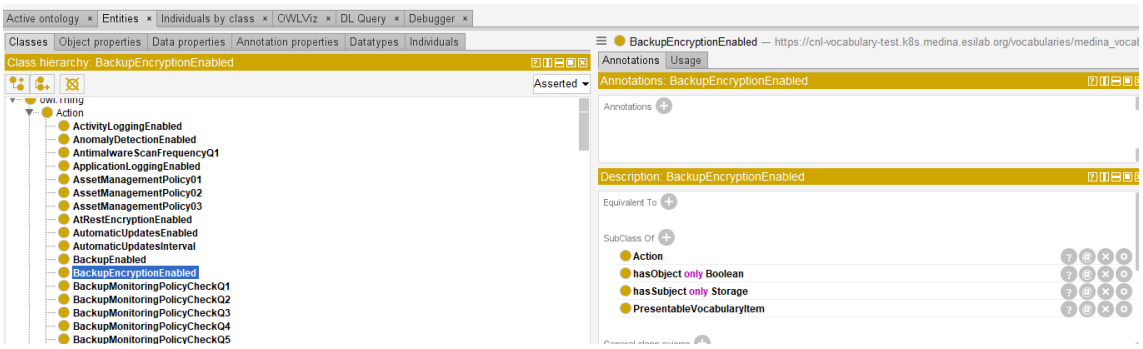

Figure 15. CNL Editor Vocabulary Metric "BackupEncriptionEnabled" (source: MEDINA's own contribution)

Figure 16 shows that the TargetValueType, is "Boolean".

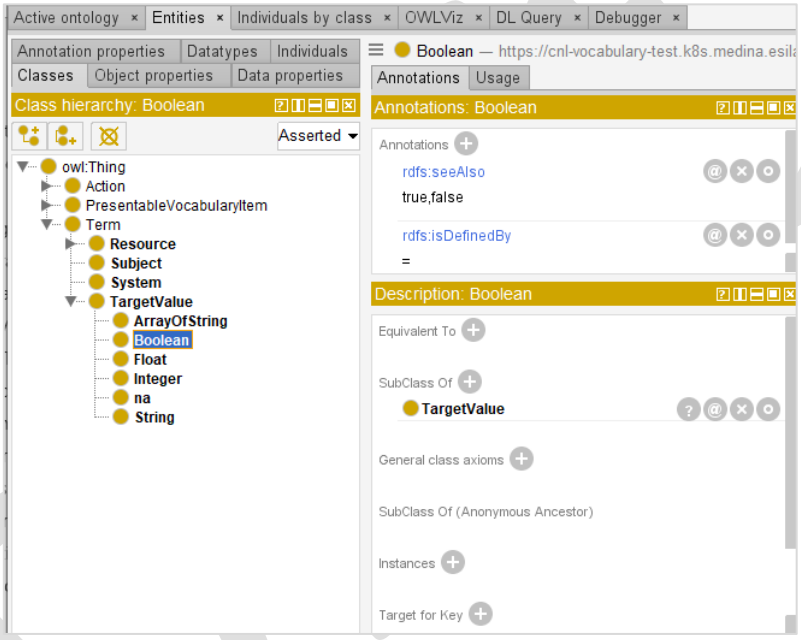

Figure 16. CNL Editor Vocabulary TargetValueType "Boolean" (source: MEDINA's own contribution)

For TargetValueType "Boolean", the vocabulary defines possible values (true or false) and possible operators that in this case is only "=".

The vocabulary must be defined considering as input what is specified in the Catalogue of controls and security metrics and must be aligned to it.

The MEDINA vocabulary is created and modified via the Protégé[11] tool, a free, open-source ontology editor, with an easy-to-use GUI.

5.3.3 Cloud Resource Security Ontology

The Cloud Resource Security Ontology (CRSO) is not directly related to the CNL Editor ontology described above, since these two ontologies serve two different purposes. They do, however, both include a classification of cloud resources which will be aligned in a future iteration.

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

The Cloud Resource Security Ontology has been developed to harmonise evidence gathering and assessment across certifications, cloud vendors, and resource types. It includes several taxonomies, including a taxonomy of cloud resources and a taxonomy of security properties. As an example, the cloud resource taxonomy includes computing resources which in turn can be virtual machines, containers or functions.

Figure 17 shows the current status of the cloud resource taxonomy. This taxonomy classifies cloud resources across all major cloud providers and architectures, like Microsoft Azure, Amazon Web Services, Google Cloud Platform, and OpenStack. It is ordered by cloud service categories which is the typical classification the major cloud providers also use. For instance, resources that can be used to execute code (virtual machines, serverless functions, etc.) are grouped in a *Compute* category, while resources that provide networking capabilities (virtual networks and subnetworks, IP addresses, routing rules, etc.) pertain to a *Networking* group. The higher-order classes in this taxonomy are the following:

- Compute
- Identity Management
- Container Orchestration
- Container Registry
- Continuous Integration/Continuous Delivery Service
- Logging
- Networking
- IoT
- Storage
- Account
- Image

The security properties taxonomy classifies security properties that can configured in a cloud service. We have ordered it by STRIDE categories. STRIDE is an acronym for the security threats Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege. The security properties taxonomy includes their respective counterparts, i.e., the following protection goals:

- Authentication
- Integrity
- Non-repudiation
- Confidentiality
- Availability
- Authorization

The Confidentiality category, for example, includes *encryption* as a security property, while the Integrity category includes the *immutability* property which may be implemented by some storage resources. A further example is presented in Figure 18 and the complete security properties taxonomy is shown in Figure 19.

In particular, Figure 18 shows an excerpt from the CRSO: A BlockStorage is a sub-type of Storage which in turn is a Service. Security properties that are offered on one level of this hierarchy are inherited by child nodes. For instance, Storage offers AtRestEncryption which is inherited by the BlockStorage. Furthermore, BlockStorage offers two specific types of AtRestEncryption, i.e., CustomerManagedKeyEncryption and CPManagedKeyEncryption

Without harmonizing evidence gathering across cloud vendors with the help of Cloud Resource Security Ontology and related taxonomies, processing evidence would be much more tedious, since, e.g., two resources hosted by different cloud providers but with similar security properties have to be parsed by dedicated pieces of code, and measured by dedicated metrics.

Figure 17. The cloud resource taxonomy which classifies cloud resources according to their functional purpose, like compute, storage, and networking

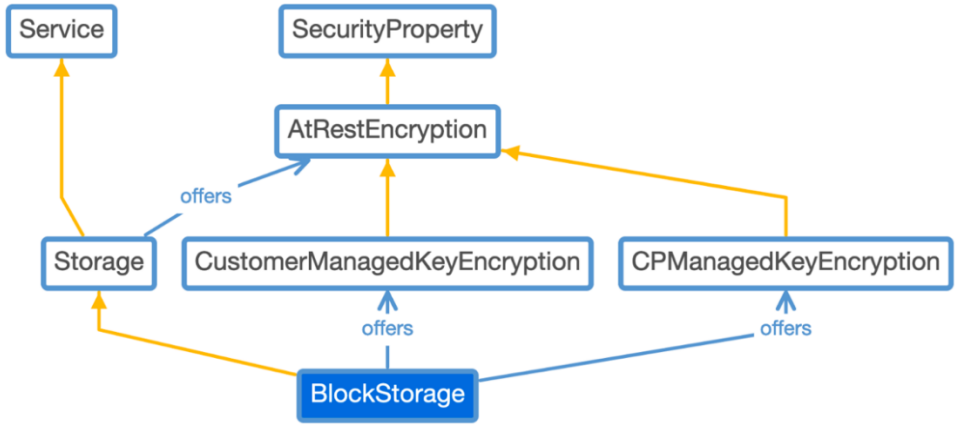

Figure 18. An excerpt from the CRSO

A further advantage of this harmonization is that requirements defined in different certifications or catalogues can be assigned to the ontological concepts they refer to. This assignment to a common ontological classification allows to assess evidence collectively and improves the assessment's reusability and modifiability. The ontological types can therefore be used in metric definitions related to the certifications and catalogues. This way, metrics are defined for abstract resource types, e.g., object storages, and their security properties; e.g., at-rest-encryption, rather than for cloud-provider-specific properties; e.g., an encryption property that is specifically defined for an AWS S3 bucket.

The ontology has been created using Protegé[12] and is published in the online Protegé platform webprotege[13]. Note that it has been described and used in the publication *Cloud Property Graph: Connecting Cloud Security Assessments with Static Code Analysis* (IEEE CLOUD 2021) as well, which has been funded by MEDINA [52]. In this publication, the ontology is used as a basis for a combined security analysis of cloud infrastructures and deployed software. Such an analysis similarly requires a harmonisation of security concepts, e.g., encryption or logging functionalities need to be recognised on the infrastructure level as well as on the source code level to allow for a comprehensive assessment of security requirements.

The ontology will be extended and may be further exploited, for instance it can be used to cover the resource and security properties specified in different certifications and control catalogues.

[12] https://protege.stanford.edu/
[13] https://webprotege.stanford.edu/#projects/8e5d391e-6436-41c0-b9a9-9226e90a38a9/edit/Classes

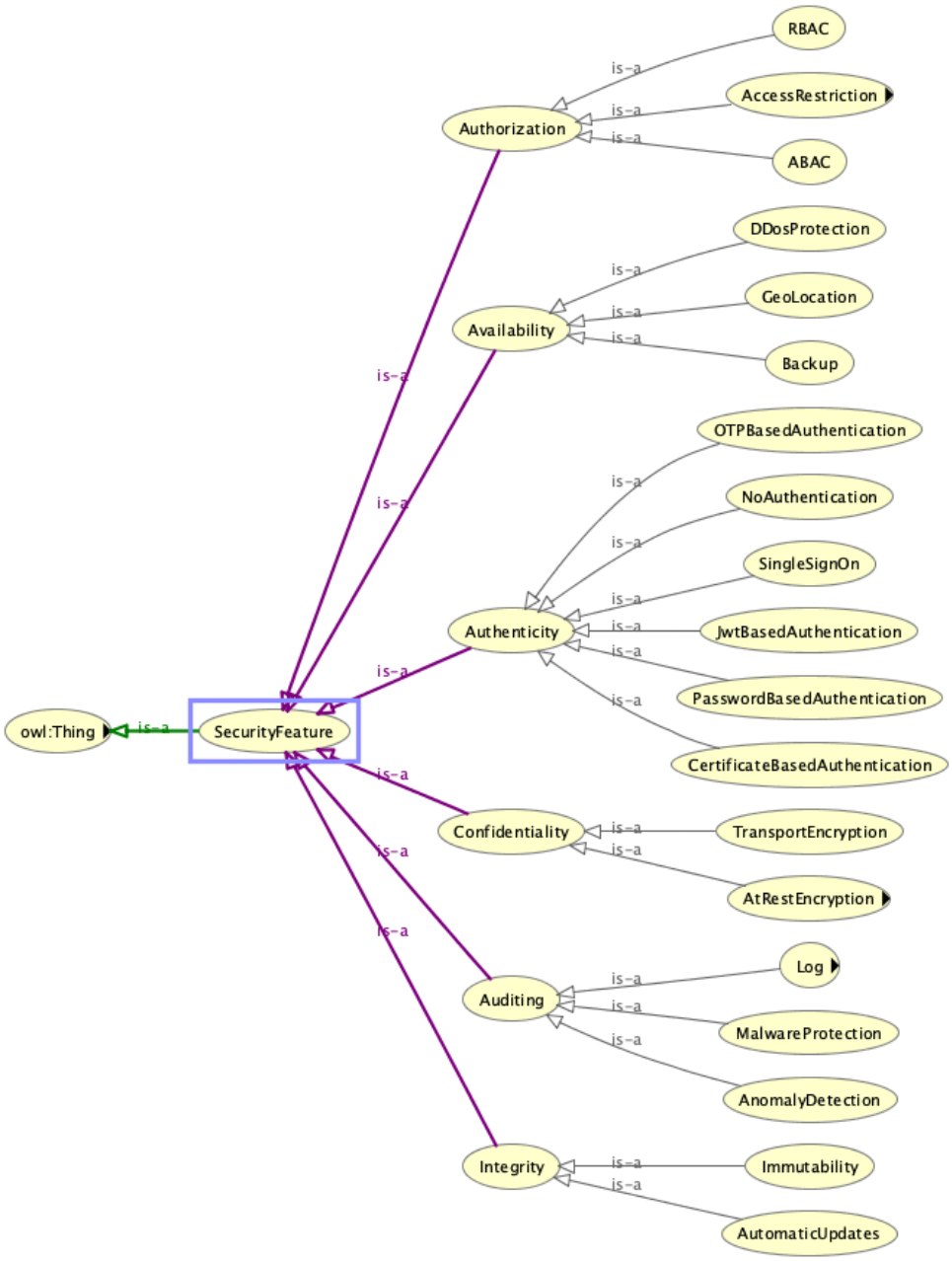

Figure 19. The security property taxonomy which classifies security properties according to their targeted STRIDE-based goal

6 The DSL mapper

The DSL Mapper is the MEDINA component responsible for passing from the CNL representation of the requirements/associated metrics to the machine-readable version input to the assessment tools. The definition and implementation of this component is the core of task 2.5. At time of writing, the Mapper has not yet been developed. However, in the next sections we describe Rego, the policy language chosen for implementing metrics as machine-readable policies and provide some examples of how these policies should be generated by the DSL Mapper, starting from the CNL obligations and the MEDINA metric definitions.

6.1 Choosing a language for policy evaluation

In MEDINA, the evidence collectors create evidence that need to be assessed using the MEDINA metrics. For example, evidence that describes an object storage may need to be assessed using a metric which defines that this storage needs to have a backup. This is a typical policy evaluation task: a request (the evidence describing the storage) is provided and needs to be checked for compliance with a pre-defined policy (the metric and its target value, i.e., backup enabled). To that end, it is practical to do the assessment using a policy engine and translate the metric to a format that can be processed in this engine.

In MEDINA, we use the Open Policy Engine (OPA)[14] to that end, i.e., to evaluate evidence against policies, and its native policy language Rego[15] (which is based on Datalog) for defining the policies. Note that for this particular task, we do not have special requirements for the policy language and policy engine. We require a policy language and policy engine that can evaluate simple JSON-formatted resource descriptions against potentially externally loaded metric data, e.g., loaded from the central catalogue. Also, we ideally want to use a technology that is intuitive to use and easy to integrate as a library. Rego and the Open Policy Agent fulfil these requirements and are also widely used, e.g., by CloudFlare, Atlassian, and Netflix[16].

Policy languages, engines, and respective architectures have been a topic of research for a long time. An alternative technology for Rego would be, for instance, the eXtensible Access Control Markup Language (XACML) [1] which describes policies in the XML format. It defines not only a policy language but also a reference architecture for evaluating policies. As such it provides similar features as OPA and Rego. Further policy languages include Ponder [52], KAoS [53], Rei [54], and EPAL [55], which differ in the engine they use for reasoning, their supportive tools like graphical editors, their expressiveness, and other criteria. The interested reader is referred to the comparative works by Tonti et al. [56] and Anderson [57].

Note also that in a previous version of Clouditor (see Deliverable 3.1 [5]) we used a custom DSL to describe policies, which, however, was a considerable overhead effort to maintain and extend to all necessary use cases.

Summarizing, we decided to use Rego and OPA as they present a modern and well-established, policy system that is easy to integrate and which allows us to focus on the core contributions of MEDINA.

6.2 Rego policies

When evaluating Rego policies, three elements are considered (see also the example below): an input (the request), policies (the pre-defined rules), and (optional) external data. The input is

[14] https://www.openpolicyagent.org/
[15] https://www.openpolicyagent.org/docs/latest/policy-language/
[16] see https://github.com/open-policy-agent/opa/blob/main/ADOPTERS.md

then checked against the policies to evaluate if it complies, possibly considering the external data.

In MEDINA, Rego policies are used in the assessment of evidence, i.e., the comparison of gathered evidence to the target values in pre-defined metrics. For instance, gathered evidence may present a firewall configuration (the input), while the Rego policies define which firewall configurations are allowed. Furthermore, external data can be queried, e.g., a specific target value for an allowed or disallowed port. Consider the following example:

Policies

```
compliant[m] {
   m := input.name
   data.operator = ">="
   input.atRestEncryption.algorithm >= data.target_value
}
```

The policy above is named *compliant* and evaluates to *true* or to *false,* and will also output the value of the variable *m* if the conditions are met. *m* is first given the name of the resource that the evidence describes (see below). Then, it is checked whether the operator specified in the metric is ">=" and applies it to the at-rest-encryption algorithm specified in the evidence, comparing it to the pre-defined target value from the metric. The *data* that is referenced in this policy may also be retrieved from an external source, such as the MEDINA catalogue of controls and security metrics.

Input

```
{
   "atRestEncryption": {
      "algorithm": 256,
      "enabled": true,
      "keyManager": "Microsoft.Storage"
   },
   "id":"/subscriptions/1234/resourceGroups/cloud-property-graphs-
examples/providers/Microsoft.Storage/storageAccounts/storage12",
   "name": "storage12",
   "type": [
      "ObjectStorage",
      "Storage",
      "Resource"
   ]
}
```

The Input that can be seen above presents an excerpt of an evidence that is collected by the Cloud Evidence Collector (developed in Work Package 3). It specifies a configuration of a Microsoft Azure Storage Account, including its name, id, and at-rest-encryption specification. The Cloud Evidence Collector also added the respective ontological terms that apply to a Microsoft Storage Account, i.e. (Cloud)Resource, Storage, ObjectStorage, which represents a path in the cloud resource taxonomy (see Figure 17).

Data

```
{
   "operator": ">=",
   "target_value": 256
}
```

The policy seen above, already referenced *data*. This data is specified in dedicated files and holds the data that should be variable and may be retrieved from external sources. In this case it specifies the operator to be used in the comparison and the desired target value which specifies the desired bit length of the encryption algorithm in this case.

This approach is straightforward and allows to modularise and easily integrate evidence gathering, evidence assessment and the definition of external data. The latter is usually defined individually by a CSP which is why this modularisation is so important in MEDINA.

The example can also be tested in the Rego Playground[19]. The Rego Playground is an in-browser environment for writing and testing Rego code.

6.3 Generating Rego Policies

To generate Rego policies, the Mapper will map the required necessary fields from the metric definitions to Rego code. For example, let the reader consider a metric that specifies the following:

- ID: BackupEnabled
- Description: This metric is used to assess if backups are enabled for a cloud service/asset
- Range: [true, false]
- Operator: ==
- Target value: true
- Target value data type: Boolean
- Resource type: Storage
- Security feature: backup.enabled

Using the specification above a Rego policy and Data can be generated as follows:

Rego Policy

```
compliant[m] {
   m := input.name
   data.operator = "=="
   input.backup.enabled == data.target_value
}
```

[19] https://play.openpolicyagent.org/p/oiitAt8acZ

Data

```
{
   "operator": "==",
   "target_value": true
}
```

Note that the code above is only an example, and may change in a future iteration. Still, it shows that certain fields specified in metrics can be mapped to Rego code to create valid policies automatically.

6.4 Interaction with the CNL Editor

The DSL mapper has a strict interaction with the CNL Editor. The next bullets represent the most important touchpoints between the two tools:

- The DSL mapper uses the CNL Editor output, which is an XML-based format with a strict schema definition. It is expected that the XML-based format will be slightly updated to match the project requirement, in particular in the metadata section.
- The DSL mapper uses the Editor API to read CNL documents (XML files) stored in the CNL Store to perform its mapping function.
- The DSL mapper is invoked by the CNL Editor UI. The CNL Editor uses the API (RESTful web services) provided by the DSL Mapper to invoke the mapping function when asked by the CNL Editor user.

7 Conclusions

This document describes the main results achieved during the first year of the project regarding the definition of the Cloud Certification Language, a machine readable language in which the policies derived from the EUCS Cloud Certification Requirements will be expressed and which will be input to the MEDINA assessment tools. The main achievements are as follows:

- The definition of a readable Controlled Natural Language to express in a uniform way the technical and organizational requirements (TOMs) to fulfil a cloud certification.
- The derivation of CNL obligations through the support of NLP techniques: to the best of our knowledge, this is the first time that NLP tools are used to synthesize cloud certification requirements and metrics in a Controlled Natural Language.
- The definition of the CNL Editor, to visualize and possibly revise CNL obligations.
- A first version of the MEDINA ontology.
- The introduction of Rego code as the MEDINA DSL.

A limitation in the process leading to the MEDINA Certification language concerns the automatic association of metrics to a requirement, due to the lack of annotated data to achieve good accuracy. Therefore, the intervention of an expert user who can validate the result of this task is required, at least at time of writing this document.

The plan for the next months is to: 1) bring forward the representation of requirements and associated metrics into CNL and the implementation of the CNL Editor, by continuing with experimentations on recommending metrics for requirements and by refining vocabulary and ontology for the editor; and 2) define the main steps for the translation from NL to DSL and from CNL to DSL (by possibly revising the DSL definition too).

8 Bibliography

[1] OASIS XACML Technical Comittee, "eXtensible Access Control Markup Language (XACML) Version 3.0. OASIS Standard," http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html, 2013.

[2] MEDINA Consortium, "D2.1 - Continuously certifiable technical and organizational measures and catalogue of cloud security metrics-v1," 2021.

[3] MEDINA Consortium, "D2.6 Risk-based techniques and tools for Cloud Security Certification-v1," 2022.

[4] ENISA, "EUCS – Cloud Services Scheme," [Online]. Available: https://www.enisa.europa.eu/publications/eucs-cloud-service-scheme.

[5] MEDINA Consortium, "D3.4 - Tools and techniques for collecting evidence of technical and organisational measures-v1," 2021.

[6] MEDINA Consortium, "D3.1 - Tools and techniques for the management of trustworthy evidence-v1," 2021.

[7] T. Khun, "A Survey and Classification of Controlled Natural Languages," *Computational Linguistics,* vol. 40, no. 1, pp. 121-170, 2014.

[8] MEDINA Consortium, "D5.1 - MEDINA Requirements, Detailed architecture, DevOps infrastructure and CI/CD and verification strategy-v1," 2021.

[9] A. Veizaga, M. Alferez, D. Torre, M. Sabetzadeh and L. Briand, "On systematically building a controlled natural language for functional requirements," *Empirical Software Engineering,* vol. 26, no. 79, 2021.

[10] D. Méndez Fernández and others, "Naming the Pain in Requirements Engineering: Contemporary Problems, Causes, and Effects in Practice," *Empirical Software Engineering,* vol. 22, pp. 2298-2338, 2017.

[11] D. Méndez Fernández and others, "Naming the Pain in Requirements Engineering: Contemporary Problems, Causes, and Effects in Practice," *Empirical Software Engineering,* pp. 2298-2338, 2017.

[12] K. Pohl, Requirements engineering - fundamentals, principles, and techniques, 2010.

[13] C. Arora, M. Sabetzadeh, L. Briand and F. Zimmer, "Automated extraction and clustering of requirements glossary," *IEEE Trans Software Eng,* vol. 43, no. 10, pp. 918-945, 2017.

[14] R. Schwitter, "Controlled Natural Language for Knowledge Representation," in *COLING* , 2010.

[15] A. Wyner, "On controlled natural languages: Properties and prospects," in *Controlled natural language*, 2009.

[16] K. Pohl and C. Rupp, Requirements engineering fundamentals - a study guide for the certified professional for requirements engineering exam: Foundation Level - IREB compliant, 2011.

[17] A. Mavin, P. Wilkinson, S. Gregory and E. Uusitalo, "Listens learned (8 lessons learned applying EARS).," in *In: 24th IEEE international requirements engineering conference*, 2016.

[18] S. Withall, Software requirement patterns, Pearson Education, London, 2007.

[19] M. Riaz, J. King, J. Slankas and L. Williams, "Hidden in plain sight: automatically identifying security requirements from natural language artifacts," in *IEEE 22nd international requirements engineering conference*, 2014.

[20] C. Denger, B. DM and E. Kamsties, "Higher quality requirements specifications through natural language patterns.," in *2003 IEEE International conference on software - science, technology and engineering (SwSTE 2003)*, 2003.

[21] J. Eckhardt, A. Vogelsang, H. Femmer and P. Mager, "Challenging incompleteness of performance requirements by sentence patterns," in *24th IEEE international requirements engineering conference*, 2016.

[22] I. Matteucci, M. Petrocchi and M. Sbodio, "CNL4DSA: a controlled natural language for data sharing agreements," in *Symposium of Applied Computing*, 2010.

[23] N. Fuchs, K. Kaljur and T. Kuhn, "Attempto Controlled English for knowledge representation," in *Reasoning Web — 4th International Summer School 2008, number 5224 in Lecture Notes in Computer Science*, 2008.

[24] G. Hart, M. Johnson and C. Dolbear, "Rabbit: Developing a Control Natural Language for Authoring Ontologies," in *European Semantic Web Conference*, 2008.

[25] A. Cregan, R. Schwitter and T. Meyer, "Sydney OWL syntax - Towards a controlled natural language syntax for OWL 1.1," in *CEUR Workshop Proceedings 258*, 2007.

[26] S. Konrad and B. Cheng, "Real-time specification patterns," in *27th International conference on software engineering (ICSE 2005)*, 2005.

[27] A. Post, I. Menzel and A. Podelski, "Applying restricted english grammar on automotive requirements - does it work? A case study.," in *Requirements engineering: foundation for software quality*, 2011.

[28] M. Abadi, "Logic in access control," in *LICS*, 2003.

[29] A. Arenas, B. Aziz, J. Bicarregui and M. Wilson, "An Event-B approach to data sharing agreements," in *IFM, LNCS 6396*, 2010.

[30] Q. Ni and e. al, "Privacy-aware Role-based Access Control," *ACM Transactions on Information and System Security,* 2010.

[31] R. De Nicola, G. Ferrari and R. Pugliese, "Programming Access Control: The KLAIM Experience," in *CONCUR 2000. LNCS, vol. 1877*, 2020.

[32] R. Hansen, F. Nielson, H. Nielson and C. Probst, “Static validation of licence conformance policies,” in *ARES*, 2008.

[33] K. Larsen and B. Thomsen, “A modal process logic,” in *Third Annual Symposium on Logic in Computer Science*, 1988.

[34] J. Bergstra, A. Ponse and S. Smolka, Handbook of Process Algebra, Elsevier, 2011.

[35] MEDINA Consortium;, “D2.1 – Continuously certifiable technical and organizational measures and catalogue of cloud security metrics-v1,” 2021.

[36] MEDINA Consortium, “D3.1 - Tools and techniques for the management of trustworthy evidence-v1”.

[37] Amass Consortium D3.6, “Prototype for Architecture-Driven Assurance (c),” 2018.

[38] I. Matteucci, M. Petrocchi and M. Sbodio, “CNL4DSA: a controlled natural language for data sharing agreements,” in *Symposium of Applied Computing*, 2010.

[39] G. Hart, C. Dolbear and J. Goodwin, “Lege Feliciter: Using Structured English to represent a Topographic Hydrology Ontology,” in *OWL: Experiences and Directions*, 2007.

[40] A. Cregan and T. M. R. Schwitter, “Sydney OWL syntax - Towards a controlled natural language syntax for OWL 1.1,” in *OWL: Experiences and Directions*, 2007.

[41] R. Schwitter, K. Kaljurand, A. Cregan, C. Dolbear and G. Hart, “A Comparison of three Controlled Natural Languages for OWL 1.1,” in *OWL Experiences and Directions*, 2008.

[42] “Wikipedia - BERT (language model),” October 2021. [Online]. Available: https://en.wikipedia.org/wiki/BERT_(language_model). [Accessed Octobre 2021].

[43] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,* 2018.

[44] “Wikipedia - Word2vec,” October 2021. [Online]. Available: https://en.wikipedia.org/wiki/Word2vec.

[45] “FastText,” October 2021. [Online]. Available: https://fasttext.cc/.

[46] “Wikipedia - Stop word,” [Online]. Available: https://en.wikipedia.org/wiki/Stop_word. [Accessed October 2021].

[47] “Common crawl,” [Online]. Available: https://commoncrawl.org/. [Accessed October 2021].

[48] “Wikipedia PCA,” [Online]. Available: https://en.wikipedia.org/wiki/Principal_component_analysis. [Accessed October 2021].

[49] “Wikipedia TSNE,” [Online]. Available: https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding. [Accessed October 2021].

[50] "scikit learn Truncated SVD," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html. [Accessed October 2021].

[51] "Wikipedia - Evaluation meaures (information retrieval)," [Online]. Available: https://en.wikipedia.org/wiki/Evaluation_measures_(information_retrieval). [Accessed October 2021].

[52] "Wikipedia DCG," [Online]. Available: https://en.wikipedia.org/wiki/Discounted_cumulative_gain. [Accessed October 2021].

[53] C. Banse, I. Kunz, A. Schneider and K. Weiss, "Cloud Property Graph: Connecting Cloud Security Assessments with Static Code Analysis," in *IEEE 14th International Conference on Cloud Computing (CLOUD)*, 2021.

[54] N. Damianou, N. Dulay, E. Lupu and M. Sloman, "The ponder policy specification language," *International Workshop on Policies for Distributed Systems and Networks,* pp. 18-38, 2001.

[55] A. Uszok, J. M. Bradshaw, M. Johnson, R. Jeffers, A. Tate, J. Dalton and S. Aitken, "KAoS policy management for semantic web services," *IEEE Intelligent Systems,* vol. 19, no. 4, pp. 32--41, 2004.

[56] L. Kagal, "Rei: A Policy Language for the Me-Centric Project," https://ebiquity.umbc.edu/_file_directory_/papers/57.pdf, 2002.

[57] A. Paul, S. Hada, G. Karjoth, C. Powers and M. Schunter, "Enterprise privacy authorization language (EPAL)," *IBM Research,* vol. 30, p. 31, 2003.

[58] G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri and A. Uszok, "Semantic Web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder," in *International Semantic Web Conference*, Springer, 2003, pp. 419--437.

[59] A. H. Anderson, "A comparison of two privacy policy languages: EPAL and XACML," *Proceedings of the 3rd ACM workshop on Secure web services,* pp. 53--60, 2006.

[60] P. Kruse, "Prototype for Architecture-Driven Assurance (c)," Amass Deliverable 3.6, 2018.

[61] C. Ardila, J. Patricia, B. Gallina and F. U. Muram, "Compliance checking of software processes: A systematic literature review," *Journal of Software: Evolution and Process,* vol. 34, no. 5, 2022.

[62] R. Hansen, F. Nielson, H. Nielson and C. Probst, "Static Validation of Licence Conformance Policies," in *ARES*, 2008.