

# GLUON: A Reasoning-based and Natural Language Generation-based System to Explicit Ontology Design Choices

Zakaria Mejdoul<sup>a</sup> and Gaëlle Lortal<sup>b</sup>

Thales Research & Technology, Palaiseau, France

**Keywords:** Ontology, Reasoner, Controlled Natural Language, Verbalization, Sentences Generation.

**Abstract:** The Semantic Web (*SW*) is an enhancement to the World Wide Web (*WWW*). It allows Humans to find, share and integrate information more easily. One of the Knowledge Representation technologies related to the *SW* standards is the ontology, which is an inventory of knowledge defining a universal or specific domain. Ontology construction requires expertise related to logics and to the expert domain the *SW* application is applied to. We aim to enable ontology wide adoption by bringing the end-users closer to their own ontology building choices, providing them with the possibility to build their ontology, to validate its consistency and to formally represent their knowledge without formal methods knowledge. In this paper, we detail our tool architecture combining *SW* technologies and Natural Language Generation (*NLG*) to support users in creating consistent ontologies.

## 1 INTRODUCTION

Knowledge sharing is key to support intelligent process automation and collaborative problem solving (Gruber, 1995) in large organizations. *THALES*, a Worldwide company, needs efficient critical data and knowledge management to support daily business applications in Air Traffic Management (*ATM*) and Avionics. An efficient tool to explicit information and allow its sharing (Wankhade and Raut, 2021) is ontology. Thus, the solution we consider to improve business application is ontology-based application appropriately managing domain knowledge (e.g. (Wu et al., 2020)). This process involves knowledge modeling as an ontology (Shimizu et al., 2020).


Despite, ontologies are expected to be widely used in *SW* applications to bridge the gap among machines and humans, their acceptance and use is hampered by their complexity and the need of skilled knowledge engineers to develop them. The human interaction aspects of these formalisms (ease of learning, reading, writing) have received little attention. Ontology editors such as *Protégé* (Musen, 2015), *Neon* (Suárez-Figueroa et al., 2015), etc. still require knowledge and training in ontology and formal logics domains, as well as a good grasp of the editor's plugins and


widgets, which makes the handling of these software difficult. Likewise, keeping track of the ontology design decisions is another missing aspect, blurring design rationale consequences and disorienting non-ontology-experts.

In section 2, we present the Air Mission domain we want to improve. In section 3, we present the use cases and the main scenarios we considered to evaluate the solution. In section 4, we detail existing systems of ontology, reasoners and verbalizers, then in section 5, the technical architecture including our conceptual design. Section 6 covers results of validation scenarios simulation as described in section 3. Then, we conclude and state the future work.

## 2 CONTEXT

With increasing airspace traffic congestion, *ATM* and Avionics need more advances in massive data processing, sophisticated edge avionics (coordination with weather updates, estimation of multiple uncertainty types,...). The complexity of the situation overwhelms the skills of pilots or *ATM* controllers. In order to provide automatic and Artificial Intelligence (*AI*) decision assistance for Air Missions, ontologies are an attractive knowledge technology. Air domains information management systems must be efficient with larger amounts of data, effective in combining

<sup>a</sup>  <https://orcid.org/0000-0003-4341-2139>

<sup>b</sup>  <https://orcid.org/0000-0001-5374-6584>

information from different sources, and relevant in reducing uncertainty in decision support systems (Insaurralde and Blasch, 2018).

An attractive approach to supporting decision making in advanced Air Mission systems is the implementation of ontologies for avionics systems (Palacios et al., 2016), (Palacios et al., 2018). Ontologies are intended to model cognitive processes by representing and reasoning about knowledge. In air Mission field, two ontologies are widely used: ATMONTO<sup>1</sup> (*The NASA Air Traffic Management Ontology*) which describes classes, properties and relationships related to ATM domain and represents information about a large and diverse set of interacting components in the U.S. and global airspace; AIRM<sup>2</sup> (*ATM Information Reference Model*) which is an OWL ontology derived from Air Traffic Management (ATM) and was developed in the framework of the BEST project of the Single European Sky ATM Research (SESAR) program (Standar, 2012).

To set up SW applications in air Mission field, we built scenarios answering realistic Air Mission use-cases (UCs) thanks to these two open-source ontologies (ATMONTO and AIRM). Our scenarios cover events that can occur during an air mission as: a weather event, a destination diversion, a systematic failure.

Table 1: Ontologies metrics extracted from Protégé.

Metrics	Ontologies	ATMONTO	AIRM
axiom		1644946	34576
logical axiom count		1386774	15735
declaration axioms count		2768	9605
class count		2461	1177
object property count		127	2727
data property count		189	1972
individual count		226516	3727
loading time (ms)		18966	837

The table 1 shows the ontologies metrics *ATMONTO* and *AIRM* which are composed of domains and concepts similar to our working context (air mission management). These ontologies metrics come from the Protégé ontology editor, giving a schema on the richness and density of the two: ATMONTO (with a total of 1644946 axioms) is richer and more complex compared to AIRM.

### 3 USE-CASES

#### 3.1 Purposes

To bring end-users closer to their ontology building choices and to provide them with formalization and validation means depends on end-users experience. We consider three main interaction levels in a SW application to fit with our end-users interaction needs:

- Average users with no computer science background, who want to contribute to the SW with knowledge about their domains of interest (Berners-Lee et al., 2001).
- Domain experts, who create formalized knowledge to be used in a SW application. Traditionally, these ontologies are created by ontology engineers, who interview domain experts in an iterative process. Allowing domain experts to create initial versions of these ontologies themselves lower the costs compared to the traditional method.
- Ontology engineers using our approach to rapid ontology prototyping.

We want to achieve our goal of increasing the ontology acceptance by enabling end-users to handle with full transparency the formalisms required to build them. As our aim is to support users, we will first propose the most widely used format of ontology to our users, namely *OWL2*. The bridge we build is between the OWL logical and formal format and an easy and natural interaction means: Natural Language. These end-users act in a domain and for our works we gathered Air Mission needs. In the next section, we present our scenarios to test the use of ontologies.

#### 3.2 Description of UCs

To be able to evaluate the usability and performances of our solution, we set up validation scenarios, based on various UCs given by Air Mission experts. The identified UCs are represented in our air mission ontology and are:

- System failure scenario;
- Diversion due to arrival runway closed scenario;
- Meteorological event scenario.

They are formally described below. Each scenario reflects an incident, which can occur during a flight or as we call it, air mission.

<sup>1</sup><https://data.nasa.gov/ontologies/atmonto/index.html>

<sup>2</sup><https://airm-o.github.io/airm-o/>

### 3.2.1 System Failure Scenario

This system failure UC, due to an air mission operated with an aircraft having a low fuel level (value of the level is equal to 0.0) can trigger an emergency. An emergency mission implies that the mission is unsatisfiable, therefore, the mission is classified as unsatisfiable mission, by semantic rules ('SWRL' rule). This System Failure scenario is represented as follows:

#### Property Assertions:

```
Individual: MissionSMAScenarioA
MissionSMAScenarioA Type Mission
MissionSMAScenarioA isFlownWith F-DEV1
F-DEV1 hasFuel FDEV1CurrentFuel
FDEV1CurrentFuel Type Fuel
FDEV1CurrentFuel hasFuelValue '0.0'
Mission DisjointWith UnsatisfiableMission
```

#### LowFuel SWRL Rule:

```
Mission(?m) ^ Fuel(?fcv) ^ IsFlownWith(?m, ?a)
^
hasFuel(?a, ?fcv) ^ hasFuelValue(?fcv, 0.0)
-> hasUrgency(?m, "3")
```

#### Urgency SWRL Rule:

```
Mission(?m) ^ hasUrgency(?m, ?urg) ->
UnsatisfiableMission(?m)
```

**Explanation:** F-DEV1 has fuel with fuel value 0.0

**Implies:** MissionSMAScenarioA Type UnsatisfiableMission

### 3.2.2 Diversion Due to Arrival Runway Closed Scenario

It reflects a diversion of the aircraft's destination during a mission, due to a closure of the runway intended for the destination, which also implies that the current mission is unsatisfiable, by an implication semantic rule. This Diversion scenario is represented as follows:

#### Property Assertions:

```
Individual: MissionSMAScenarioB
MissionSMAScenarioB Type Mission
MissionSMAScenarioB hasActiveAirportDeparture
LFPB
MissionSMAScenarioB hasActiveAirportArrival
KTEB
MissionSMAScenarioB hasActiveArrivalRunway
KTEB19
KTEB19 isOpen 0
Mission DisjointWith UnsatisfiableMission
```

#### ArrivalRunwayAvailability SWRL Rule:

```
Mission(?m) ^ hasActiveArrivalRunway(?m, ?arw)
^ isOpen(?arw, 0) -> UnsatisfiableMission(?m)
```

**Explanation:** KTEB19 is not open

**Implies:** MissionSMAScenarioB Type UnsatisfiableMission

### 3.2.3 Meteorological Event Scenario

It describes an air mission failure due to a weather event that can endanger the mission aircraft during its landing on its destination runway: the wind speed of the destination runway exceeds the speed limit that the aircraft can tolerate, which also trigger a pre-landing condition alert. This event classifies an air mission as unsatisfiable mission by a rule containing a conditional form (*swrlb:greaterThan*) and then a direct implication. The meteorological event scenario is represented as follows:

#### Property Assertions:

```
Individual: MissionSMAScenarioC
MissionSMAScenarioC Type Mission
MissionSMAScenarioC hasActiveArrivalRunway KTEB06
MissionSMAScenarioC isFlownWith F-DEV1
F-DEV1 hasLandingCapacity F-DEV1LandingCapacity
F-DEV1LandingCapacity hasCrosswindLimitation '5.0'
KTEB06 hasWeather Weather_Crosswind_KTEB
Weather_Crosswind_KTEB hasWindMagnitude '20.0'
Mission DisjointWith UnsatisfiableMission
```

#### CrosswindSatisfiability SWRL Rule:

```
Mission(?m) ^ IsFlownWith(?m, ?a) ^
hasLandingCapacity(?a, ?lc) ^
hasCrosswindLimitation(?lc, ?xwindLim) ^
hasActiveArrivalRunway(?m, ?arw) ^
hasWeather(?arw, ?w) ^ Crosswind(?w) ^
hasWindMagnitude(?w, ?xwm) ^
swrlb:greaterThan(?xwm, ?xwindLim) ->
UnsatisfiableMission(?m)
```

**Explanation:** KTEB06 has wind magnitude 20.0 greater than F-DEV1 limitation 5.0

**Implies:** MissionSMAScenarioC Type UnsatisfiableMission

The proposed scenarios lead to classify a mission into "missions that cannot be accomplished", through inconsistencies detection in the air missions ontology and declaring also an expert intervention. The inconsistency is triggered by a disjoint relation between the concepts "Mission" and "UnsatisfiableMission". For this function, we developed a solution, on which we can test scenarios based on the main UCs of air mission applications. Nevertheless, several solutions close to our needs exist. The next section presents them.

## 4 RELATED WORKS

Our basic needs are to 1. Create ontology, 2. Ensure users their modeling actions are consistent and 3. Ensure users their modeling actions answer their design choices. Existing systems answer partially these

needs using Ontologies engines, Semantic Reasoners and Verbaliser. We present a review below.

### 4.1 Ontology and Semantic Reasoning

**Ontology Overview.** In less than thirty years, the term "ontology" has gained considerable popularity in the field of computer science and information systems. This popularity is due to the ability of ontologies to achieve interoperability among multiple representations of reality (e.g., data or business process models) residing in computer systems, and among these representations and reality, i.e., human users and their perception of the domain. Surprisingly for a tool, which aims at reconciling people from various communities, the term ontology has different, even incompatible, definitions. It is something of a paradox that the starting term of a research field, which aims to reduce ambiguity about the meaning of symbols, is understood and used so inconsistently. From the early years of ontology research, Guarino and Garetta (Guarino et al., 1995) were concerned about the inconsistent use of the term "ontology". They found at least seven different notions attributed to the term. We consider this general definition for our works: An *ontology* is a structured set of insights in a particular domain of knowledge (Guarino et al., 1995). We also consider the use of a W3C recommendation as OWL2 (Hitzler et al., 2012) to ensure interoperability base on a valid and shared syntax and hence semantics.

To encapsulate OWL possibilities in a new application, the use of *OWL API* seems inescapable despite some lacks. Most of reasoners are accessible via the *OWL API*, which is advantageous for applications that want to access multiple reasoners via the same interface. The use of the *OWL API* greatly facilitated the execution of our experiments. The design of the *OWL API* is directly based on the OWL2 structural specification described in (Motik and al., 2008).

**Other Ontology Frameworks.** A number of other similar developments have been initiated to provide application interfaces for OWL: Jena toolkit (Carroll and al., 2004) provides practical ontological interfaces around RDF interfaces. Comparisons of the OWL API and Jena's triplet-based approach for some tasks are discussed in (Bechhofer and Carroll, 2004); The KAON toolkit (Bozsak and al., 2002) is an open source ontology management system for commercial applications. The KAON toolkit includes an API for RDF graph processing and differs from the OWL API in that KAON doesn't offer support for processing OWL ontologies, and the OWL API doesn't provide direct support for processing RDF graphs. Thus, OWL API is a good candidate for our works, yet our

system needs a reasoner to ensure consistency and explain users their modeling actions.

**Reasoner Overview.** A "reasoner" is a program that performs reasoning on an ontology or a knowledge base (KB). It relies on the KB, as well as on the set of rules of the ontology to infer (logically deduce) new knowledge updating the KB content.

However, rules, common to all reasoners, may not be sufficient for all application cases. Moreover, it is possible to define rules specific to a given application, in a language like *SWRL* (Semantic Web Rule Language). In this case, the reasoner assimilates these new rules and applies them to the KB in the same way they pre-established rules. Advanced verification techniques are imperative to have a consistent ontology. The reasoner then composes the core of the functional validation module.

The *logics* we rely on is *Description Logics* in general to ensure OWL adequacy with reasoning as well as termination of valid inferences. Description logics (*DLs*) are a family of knowledge representation formalisms based on classes (concepts). They are characterized by the use of various constructors to build complex concepts from simpler concepts, by the decidability of key reasoning tasks, and by providing correct, complete and feasible reasoning. *DLs* have been used in a range of applications, for example configuration and reasoning about schemas and database queries (Toman and Weddell, 2022).

**Reasoners Benchmark.** To specify the reasoners, we relied on a benchmark based on reasoning features: *Reasoning Algorithm*, *Expressivity*, *Rule Support*, *Justification* (provides explanations for inconsistencies that exist in ontologies) and *ABox reasoning task support* (reasoning with individuals for instance checking, answering queries and ABox consistency checking), the benchmark is also based on usability features: support for *OWL API* and reasoner *license* (open source/commercial).

Table 2: Reasoner Benchmark (*HermiT*, *Pellet* and *ELK*).

	HermiT	Pellet	ELK
Format	OWL2	OWL2	OWL2
	DL	DL/EL	EL
Reasoning Algorithm	Hyper-Tableau	Tableau	CB
Expressivity	<i>SROIQ(D)</i>	<i>SROIQ(D)</i>	<i>EL</i>
Rule Support	SWRL	SWRL	Own rule format
Justifications	-	+	+
Abox Reasoning	+	+	-
		(SPARQL)	
OWL API	+	+	+
License	Open-source	Open-source	Open-source

+ stands for yes and - for no.

After a detailed study of ontological reasoners and our benchmark results, the table 2 shows us a good compromise among speed, expressivity and precision: Pellet, based on the Tableau algorithm and supporting OWL2 DL. Nevertheless, for unit tests we use also ELK based on the Consequence-based algorithm, although it supports a less expressive profile of OWL, OWL2 EL. Now we set a reasoner to "understand", users should be able to understand too. We checked the different verbalizers and Controlled Natural Languages (CNLs) systems.

## 4.2 Verbalizers

**Overview.** Natural language is very expressive and doesn't require additional learning effort to present Human knowledge. To represent knowledge in computers, people use formal languages (Fuchs et al., 2008) as CNLs. To make ontologies easier to understand, several ontology verbalizers have been developed (Schwitter, 2010). The *verbalizers* typically translate the ontology axioms (here the OWL statements) one by one into CNLs statements. Often not quite fluent, they generally don't pay attention to the resulting texts consistency. There are different syntax possibilities to express a specific OWL axiom in CNL.

Several works were led to develop CNLs, mainly from English, which can be used as alternative OWL syntaxes (Khabarov et al., 2019). Sydney OWL Syntax (sos) (Cregan et al., 2007) is an English-like CNL with a bidirectional mapping to and from OWL syntax; sos is based on peng (Schwitter, 2010). A similar bidirectional mapping has been defined for ACE "*Attempto Controlled English*" (Fuchs et al., 2008).

The verbalization approach proposed by (Vidange et al., 2021) gave good results and verbalized beyond the level of the CNL. Its free configuration, being domain and schema independent and its overall accuracy results, close to 82%, is interesting. However, Semantic Level Refinement (Venugopal and Kumar, 2021), aiming at transforming knowledge represented as a combination of low expressive (and non-specific) logical expressions into high expressive and specific expressions is more precise. This technique uses a predefined set of rules that are repeatedly applied to the restrictions associated with individuals (and concepts) to obtain a refined set of restrictions. These refined restrictions sets are then verbalized to obtain concise descriptions of the ontology's entities. Nevertheless, SLR is not particularly adapted to OWL2 and needs heavy preprocessing.

ACE is particularly, well adapted to the verbalization of OWL ontologies, because the axioms are

expressed in compact ready-made sentences, which, besides, do not offer the user the possibility to express variable patterns that OWL would not be able to express. In other words, ACE makes explicit the sentences that are compatible with the expressivity of OWL and those that are not. ACE is formally described by a Definite Clause Grammar (DCG) which also provides a mapping of ACE to (a part of) OWL which can be a set of axioms (Fuchs et al., 2008). The main focus is on the transition from CNL to OWL.

**Evaluation of existing systems.** This section presents a comparison between existing systems and OWL verbalisation libraries.

Table 3: Comparisons of OWL-based verbalizers.

System	Tbox	Abox	Cove- rage	Aggre- gation	Lexi- con	Dom- ain
ACE	+	+	OWL2	-	Auto	Gen
SWOOP	+	-	OWL DL	-	Auto	Gen
MIAKT	-	+	RDF	+	Hand- crafted	Spec Spec
Natural- OWL	+	+	OWL DL	+	User- defined	Spec
SWAT	+	+	OWL2	+	Auto	Gen

+ stands for yes and - for no. Auto stands for automatic. Gen and Spec stands respectively for Generic and Specific.

The table 3 shows a comparison of the different OWL verbalizers, according to several axes: 'TBox' and 'ABox'. The coverage is approximate: for example, ACE and SWAT cover almost all of OWL2. The lexicon indicates the source of lexical entries for atomic entities. 'Domain' is generic if the system can produce text for any ontology (of the indicated OWL coverage) with English names, and specific if hand-crafted lexical entries or grammar rules are needed.

ACE (Fuchs et al., 2008), SWAT (Power, 2012) and SWOOP (Narasimha et al., 2022) are suitable for generic domains as lexicons are created automatically, which is good for a larger ontology with a general domain. Compared to MIAKT (Bontcheva and Wilks, 2004), 'TBox' is not supported, which is a crucial factor when generating text from an ontology.

Then, both *NaturalOWL* and *ACE* could be used for our UCs as they provide more functionality and are relevant for our topic. The exception is the integrability and the fact that both systems offer APIs exploitable by our system and adequate with the architecture and flows of our solution. For technical reasons, then, we chose the ACE-based OWL verbalizer.

## 5 SOLUTION ARCHITECTURE

This section discusses the proposed solution of GLUON (Generation of Links Unifying an ONtology) and in figure 1 shows the high level architecture of GLUON.

To solve this problematic, we have developed a module named *GLUON* which is the acronym of "Generation of Links Unifying an ONtology", a user-friendly ontology design and creation methodology assisted by software components, each one dedicated to a task of the scenario of iterative ontology creation and evaluation, with the aim of making ontologies usable by non-experts. We want to make everyone interested in ontology creation understand that ontology design does not have to be complex and that by using our guide-assisted methodology, the user will be more productive than with existing methodologies and tools. The novelty of the solution resides in the

structor, a reasoning system and a natural language generator or verbalizer.

### 5.1 Components

#### 5.1.1 Ontology Constructor

This component allows to create an ontology from scratch or to load an already designed ontology via the ontology handler, which is a group of instructions from an ontology engine.

This block allows the creation of concepts, relations and individuals to be added to the ontology in question, or even to apply CRUD (Create, Read, Update, Delete) operations on ontology's content to ensure what we can call persistence of ontology data (mechanism responsible for the backup and restoration of the ontology's data or content). The user also has the possibility to add rules (e.g. *SWRL rules*) to his ontology by adapting them to the domain and the business logic that organize his field of expertise. About ontology engine, we opted for *OWL API* as an ontology manipulation library. This module is present in many uses and has a large community that uses it, so documentation and references can also complete our need when using it.

#### 5.1.2 Reasoning System

This sub-module is a set of subsystems allowing the use of a standalone reasoner or several parallel reasoning processes. In addition to reasoning on an ontology (inferences, detection of inconsistencies and unsatisfiabilities), this module also includes the function of generating explanations in the form of justifications expressed in axioms (OWL axioms). This module also provides a function that manages the reasoner in use by using the ontology defined by the ontology constructor module and checks its consistency after each operation or change that the user applies to the ontology in use. Explanation is a function that also extends this module and generates explanations in the form of an owl axiom or a list of OWL axioms after each operation applied on the current ontology, also using the default reasoner (possibility to use more than one reasoner, e.g. *Pellet*, *ELK*, ...) for the generation of explanations. For the selection of reasoners, two reasoners were chosen that are aligned with two different types of ontology formats and reasoning algorithms: *Pellet* based on the Tableau algorithm and supports OWL2 DL and *ELK* based on the Consequence-based algorithm and supports OWL2 EL. This choice allows us to cover both DLs and ELs formats and also allows justification,

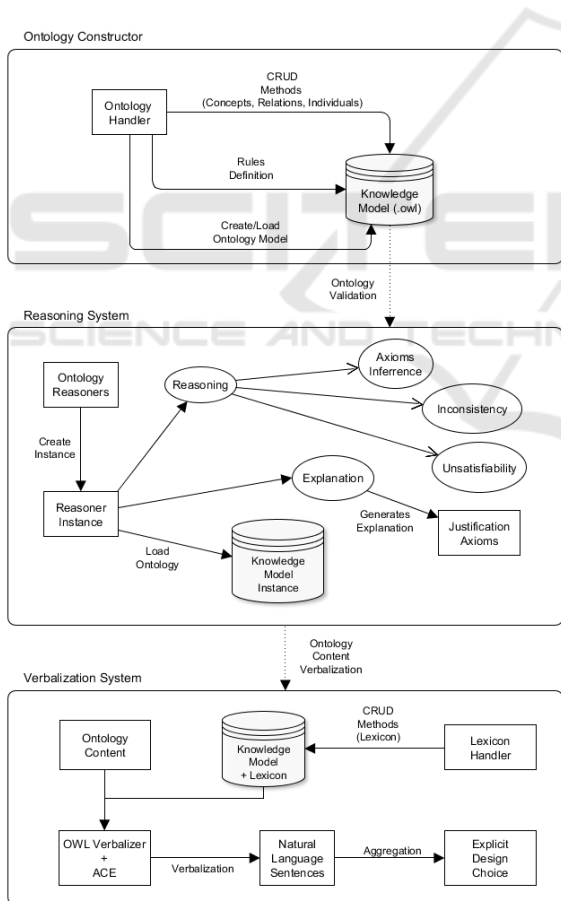


Figure 1: High level architecture of GLUON.

integration of an OWL verbalizer and in the fact that informative sentences are generated with the ontology itself by the system equipped with an ontology con-

SWRL rules support, Abox reasoning and the use of OWL API as part of reasoning processes.

### 5.1.3 Verbalisation System

The verbalization system is an integral component of the system where sentences are generated from this module with the content of the ontology in question. In view of the flexibility of creating NL resources in NaturalOWL, it is more appropriate to create an NLG system for small domains. Since the user has to define the NL resources, this type of library is not suitable for large-scale ontologies, where the author will have extra work to generate the texts. Furthermore, this library does not provide any support for integration with other systems. Considering a potential comparison, *NaturalOWL* and *ACE* could be used for our case because they provide more functionalities and are relevant for our subject. The exception is the integrability and the fact that both systems offer APIs that can be used by GLUON and that are adequate for the architecture and flow of our solution, technically viewed. In this project, we chose OWL Verbalizer as the ACE-based OWL verbalizer. For this project, we used the verbalizer as an HTTP server and at the same time exploited the module's Java methods for hybrid verbalization. The lexicons can either be generated from the names associated to concepts, relations and individuals (e.g. *#Aircraft*, *#flownWith*, *#MissionA*.etc) or can be defined by the user through the lexicon handler module by adapting it to the different grammatical forms it can take (singular form, plural form, past participle form for verbs .etc). Nouns, adjectives and verbs must be defined for better form of generation or have the possibility to import a predefined NL resource thanks to lexicon handler methods. The benefits of OWL Verbalizer include: expressing the content of OWL ontologies in a natural way; naturally adapts to expressing the content of rules and business queries; textualizing an OWL ontology in concise, understandable English, on condition to adopt a specific naming style for OWL individuals, classes and properties; allows people with no formal logic background to read and understand ontologies written by others, provided the readers are experts in the domain described by the ontology; uses the syntax of the ACE controlled sub-language; Provides interfaces (APIs) that can be used in other software. On disadvantages: difficulty in aggregating sentences in the case of verbalizing a list of axioms; generic and non-customized lexicon ( fixed with the lexicon handler).

## 6 EVALUATION

GLUON enables the outputs in CNL from OWL axioms presented below. Our evaluation plan is in two-folds. The first part is unit test for consistence checking of the scenarios based on avionics experts needs. Here we present the Alpha scenario i.e. the consistent ontology and then the 3 scenarios revealing failures. The second part will be to set up a full scenario for avionics experts, who will play an ontology building according to a new mission (as Sick Passenger On-board).

### 6.1 Scenario Alpha (Consistent Ontology)

#### Consistent Ontology Verbalization:

*F-DEV1 is an Aircraft.*  
*F-DEV1 has fuel FDEV1 current fuel.*  
*FDEV1 current fuel is a fuel.*  
*FDEV1 current fuel have fuel value 0.0.*  
*KTEB is an airport.*  
*KTEB06 is an operational runway.*  
*LFPB is an airport.*  
*Mission SMA A is a mission.*  
*Mission SMA A has active airport arrival KTEB.*  
 ...

### 6.2 Scenario A (System Failure)

**System Failure Inconsistency Explanation:** *You know that Mission SMA A is flown with F-DEV1 and FDEV1 current fuel has fuel value 0.0 and Mission SMA A is a mission and No mission is an unsatisfiable mission and F-DEV1 has fuel FDEV1 current fuel and FDEV1 current fuel is a fuel.*

### 6.3 Scenario B (Diversion for Closure)

**Diversion Inconsistency Explanation:** *You know that KTEB19 is open 0 and Mission SMA B has active arrival runway KTEB19 and No mission is an unsatisfiable mission and Mission SMA B is a mission.*

### 6.4 Scenario C (Meteorological Event)

**Meteorological Event Inconsistency Explanation:** *You know that Mission SMA C has active arrival runway KTEB06 and Mission SMA C is flown with F-DEV1 and KTEB06 has weather weather-crosswind of KTEB and F-DEV1 has landing capacity F-DEV1 landing-capacity and No mission is an unsatisfiable mission and weather-crosswind of KTEB has wind*

*magnitude 20.0 and Mission SMA C is a mission and F-DEVI landing-capacity has crosswind limitation 5.0.*

### 6.5 Discussion

The results show that despite the simplicity of most axioms in real-world ontologies, these ontologies can contain very complex axioms. Only a few of these axioms are so complex that the verbalization method proposed by OWL Verbalizer cannot handle them, the rest, however, can be presented in a way that is more readable than the standard syntax of the description logic and its variants.

By feeding the verbalizer a lexicon of morphological correspondences, we can further improve the verbalization. Note also that we have presented each verbalization as a simple sequence of words. Finally, we note that the integration of the verbalizer is easy to adapt to GLUON and can translate large currently available OWL ontologies in seconds.

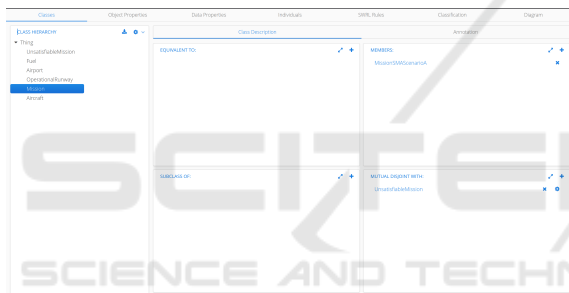


Figure 2: GLUON UI Prototype (Ontology construction).

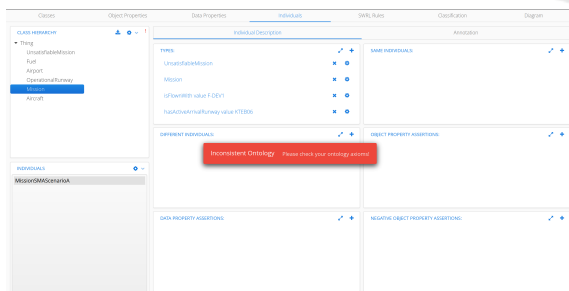


Figure 3: GLUON UI Prototype (Inconsistency check).

The semantics of the sentences that composes the generated text is identically fine-grained and related to the main causes of inconsistencies that the three scenarios from the aerial missions domain declare.

A first GUI of GLUON (Figure 2 and 3) has been developed to support the user in the ontology creation, what remains to be done is to adapt this interface to the ontology validation events through the verbaliza-

tion methods of GLUON, developed and tested with the defined aeromission scenarios.

## 7 CONCLUSIONS

In this paper, we promoted ontology broad acceptance by bringing the various formats closer to end-users, who may have no knowledge of formal methods. On the one hand, our tool allows user to create SW application backbone, i.e. ontology, in a user-friendly way. On the other hand, it provides an unambiguous language to explain (*verbalize*) the content of existing ontologies in CNL. In order to support ontology creation and to formally represent the consequences of each step of ontology design in a CNL, GLUON (Generation of Links Unifying ONtology) has been developed. The GLUON system, which supports the creation of ontologies and more specifically the verbalization of inferences in the ontology, is a system presenting a methodology for the creation of ontologies, a set of components necessary to ontology building and a principle for the validation of the ontology created. It has been evaluated through Air Mission UCs given by Air Mission experts. As future work, we intend to improve the developed GLUON HMI of section 6.5 to adapt it to scenarios of ontology creation and clarification of user’s design choices. Such a graphical interface should answer interaction levels and be integrated with the methods developed in GLUON, in order to be able to identify the reasons for inconsistencies or invalid actions, and to justify them in verbalized texts.

## REFERENCES

Bechhofer, S. and Carroll, J. J. (2004). Parsing owl dl: trees or triples? In *WWW '04*.

Berners-Lee, T., Hendler, J. A., and Lassila, O. (2001). The semantic web” in scientific american.

Bontcheva, K. and Wilks, Y. (2004). Automatic report generation from ontologies: The miakt approach. In *NLDB*.

Bozsak, E. and al. (2002). Kaon - towards a large scale semantic web. In *EC-Web*.

Carroll, J. J. and al. (2004). Jena: implementing the semantic web recommendations. In *WWW Alt. '04*.

Cregan, A., Schwitter, R., and al. (2007). Sydney owl syntax - towards a controlled natural language syntax for owl 1.1. In *OWLED*.

Fuchs, N. E., Kaljurand, K., and Kuhn, T. (2008). Attempto controlled english for knowledge representation. In *Reasoning Web*.



- Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing? *Int. J. Hum. Comput. Stud.*, 43:907–928.
- Guarino, N., Uniti, S., and Giaretta, P. (1995). Ontologies and knowledge bases. towards a terminological clarification.
- Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F., and Rudolph, S. (2012). Owl 2 web ontology language primer (second edition).
- Insaurralde, C. C. and Blasch, E. P. (2018). Uncertainty in avionics analytics ontology for decision-making support. *Journal of Advances in Information Fusion*, 13.
- Khabarov, V., Stepanov, I., and Serenko, A. (2019). Controlled natural language for ontology editing. *Science Bulletin of the Novosibirsk State Technical University*.
- Motik, B. and al. (2008). Owl 2 web ontology language: structural specification and functional-style syntax.
- Musen, M. A. (2015). The protégé project: a look back and a look forward. *AI matters*, 1 4:4–12.
- Narasimha, V. B., Sujatha, B. M., and al. (2022). An open-source web-based owl ontology editing and browsing tool: Swoop. *The Role of IoT and Blockchain*.
- Palacios, L., Lortal, G., and al. (2016). Avionics maintenance ontology building for failure diagnosis support. In *International Conference on Knowledge Engineering and Ontology Development*, volume 3, pages 204–209. SCITEPRESS.
- Palacios, L., Lortal, G., and al. (2018). Knowledge discovery for avionics maintenance support. In *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, pages 1–8. IEEE.
- Power, R. J. D. (2012). Owl simplified english: A finite-state language for ontology editing. In *CNL*.
- Schwitler, R. (2010). Controlled natural languages for knowledge representation. In *COLING*.
- Shimizu, C., Hitzler, P., and al. (2020). Modular ontology modeling: A tutorial. In *Applications and Practices in Ontology Design, Extraction, and Reasoning*.
- Standar, M. (2012). Sesar and the european atm master plan - cooperation with the us/nextgen. In *ICNS 2012*.
- Suárez-Figueroa, M. C., Gómez-Pérez, A., and al. (2015). The neon methodology framework: A scenario-based methodology for ontology development. *Appl. Ontology*, 10:107–145.
- Toman, D. and Weddell, G. E. (2022). First order rewritability in ontology-mediated querying in horn description logics. In *AAAI*.
- Venugopal, V. E. and Kumar, P. S. (2021). Verbalizing but not just verbatim translations of ontology axioms. In *BNAIC/BENELEARN*.
- Vidanage, K., Mohemad, R., and al. (2021). Fully automated ontology increment's user guide generation.
- Wankhade, S. R. and Raut, A. B. (2021). A review on ontology-based semantic web information retrieval: Techniques, weight functions. *Algorithms for Intelligent Systems*.
- Wu, Z., Shen, Y., and al. (2020). An ontology-based framework for heterogeneous data management and its application for urban flood disasters. *Earth Science Informatics*, 13:377–390.