

## Research Article

# An Algorithm Tool for Atom Decomposition and Interaction: AD Visualiser

**Xiaobo Liu** 

*School of Computer Science, University of Manchester, Manchester M13 9PL, UK*

Correspondence should be addressed to Xiaobo Liu; [xiaobo.liu@sxgkd.edu.cn](mailto:xiaobo.liu@sxgkd.edu.cn)

Received 25 October 2021; Accepted 8 December 2021; Published 28 January 2022

Academic Editor: Punit Gupta

Copyright © 2022 Xiaobo Liu. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

With the rapid development of software-defined network and network function virtualization technology, the scale of infrastructure and the number of available resources in cloud platforms continue to grow. It is also used in AD Visualiser. AD is a visualisation tool for displaying the atomic decomposition (AD) of OWL ontologies. As the size of ontologies increases, ontology engineers become more difficult to understand and reuse ontologies. Atomic decomposition (AD) is a modular structure to help ontology engineers modularly manage ontologies. It decomposes ontologies into sets of atoms, with dependency, based on modules that provide strong logical guarantees (such as locality-based modules). This paper describes the design and implementation process of AD Visualiser and discusses its usability for ontology engineers in their daily work. For example, using AD Visualiser, ontology engineers avoid choosing signatures and determining the extraction results. They can extract modules very simply and faster. Besides, the graph of AD's modular structure should be helpful for engineers to intuitively explore and comprehend ontologies.

## 1. Introduction

In recent decades, ontologies have progressively attracted the attention of researchers and engineers due to their unique knowledge expression in their field or industry [1]. Ontologies are widely used in many application fields, such as agent systems [2], knowledge management systems [3], and e-commerce platforms [4]. They can produce natural language, integrate intelligent information, and provide semantic-based access to the web. Additionally, they can also be used to extract data from texts in addition to many other applications to declare the knowledge embedded in them explicitly [5]. Besides casual ontology users, ontology experts even have been challenged to deal with the tasks of design, maintenance, reuse, and integration of complex ontologies. For example, in the medical industry, Systematized Nomenclature of Medicine – Clinical Terms (SNOMED CT) [6], Foundational Model of Anatomy (FMA) Ontology [7], and Gene Ontology (GO) are famous large ontologies. Their size also increases with the expansion of knowledge, which makes them difficult to comprehend, edit, and use. For

example, FMA contains a large amount of knowledge that is irrelevant to a particular application area, such as anatomy. In this case, an increasing number of methodologies and tools have been developed to support ontology-related work.

When creating ontologies, designers may be experts in one field but is not familiar with other fields. Especially for medium and large ontologies, such problems are more common. Take the FMA ontology as an example; when expanding the knowledge related to body structure, if the creator only knows the bones but not the skin, then the most straightforward and cheapest method is to obtain a subset of the skin from existing ontologies. In addition, when using FMA ontology, the dermatologist may not need orthopaedic-related knowledge. The fastest and direct method is to extract a subset of the dermatological knowledge from the original ontology. Therefore, both in the process of creating and using an ontology, it is helpful for ontology engineers to extract a subset of knowledge about a specific term from the existing ontology. In order to complete this task better, the module extraction of ontologies have been explored in recent years [8]. Syntactic locality-based module extraction

(ME) [9] is a module extraction algorithm. Its module extraction time linearly depends on the size of the ontology. However, for medium and large ontologies, there are some disadvantages in extracting modules directly from the ontology [10]. First, loading the file into the memory for subsequent ME will cause much delay. For example, GO's OWL file exceeds 200 MB, and pre-loading it into the main memory will cause a great burden on the memory. Second, the conventional ME algorithm checks the relevance of every piece of knowledge in the ontology, which wastes lots of time. Third, it is difficult to determine the content of the extracted module. For example, the user only uses the term bone to extract the module from FMA ontology through ME. The result is unpredictable and may be different from the user's wishes. One way to solve this problem is modularisation of ontologies.

Atomic decomposition (AD) is a fine-grained, well-connected, and easily computable modular structure based on modules that provide strong logical properties, such as locality-based modules (LBMs) [11]. In the case that AD of an ontology shows in text form, people uneasy directly discover the internal connection between the overall structure and the decomposition results. Intuitively, displaying AD in graphical forms can make it more straightforward and more accessible for people to understand the structure and internal relations of the results [12]. However, displaying only graphical AD is less likely to display complete information or directly help people extract modules. Therefore, in order to improve the usability of tools, it is essential to add features allowing users to explore and interact with images. So far, it has been challenging to find an AD visualisation tool that meets the assumptions aforementioned, so we decided to develop such software called AD Visualiser to fill the gap.

## 2. Background

*2.1. Description Logics.* Description logics (DLs) are a family of knowledge depiction languages that describe a specific domain's knowledge in a well-structured and easy-understood form [13]. Generally, a DL is a decidable fragment of first-order logic (FOL) [14]. FOL is a standard for the formalising of mathematics into first-order formulas (named axioms). Then, it can be said that the DL is a syntactically restricted subset of axioms in which truth is computable. From the perspective of knowledge representation, DLs typically contain two main parts of domain knowledge: a terminological part called the TBox (T) and the assertional part called the ABox (A). The union of these two is called a knowledge base (K) [13]. Among them, the TBox represents knowledge about the structure of the domain (similar to the schema in JSON, XML, or database), while the ABox is about a concrete scenario (akin to the data in JSON, XML, or database).

Example 1 shows the knowledge base of the juice domain (1–14 from ABox and 15 and 16 from TBox).

### Example 1

Juice = {  $\alpha 1$ : Apple  $\sqsubseteq$  Fruit,  
 $\alpha 2$ : Orange  $\sqsubseteq$  Fruit,

$\alpha 3$ : Adult  $\sqsubseteq$  Person,  
 $\alpha 4$ : Child  $\sqsubseteq$  Person,  
 $\alpha 5$ : Carrot  $\sqsubseteq$  Vegetable,  
 $\alpha 6$ : Tomato  $\sqsubseteq$  Vegetable,  
 $\alpha 7$ : NamedJuice  $\sqsubseteq$  Juice,  
 $\alpha 8$ : ChildJuice  $\sqsubseteq$  NamedJuice  $\exists$  hasTargetPerson.Child,  
 $\alpha 9$ : FruitJuice  $\sqsubseteq$  Juice  $\exists$  hasIngredient.Fruit,  
 $\alpha 10$ : AppleJuice  $\sqsubseteq$  FruitJuice  $\exists$  hasIngredient.Apple,  
 $\alpha 11$ : OrangeJuice  $\sqsubseteq$  FruitJuice  $\exists$  hasIngredient.Orange,  
 $\alpha 12$ : VegetableJuice  $\sqsubseteq$  Juice  $\exists$  hasIngredient.Vegetable,  
 $\alpha 13$ : CarrotJuice  $\sqsubseteq$  VegetableJuice  $\exists$  hasIngredient.Carrot,  
 $\alpha 14$ : TomatoJuice  $\sqsubseteq$  VegetableJuice  $\exists$  hasIngredient.Tomato,  
 $\alpha 15$ : Bobby: Child,  
 $\alpha 16$ : (Bobby, ChildJuice):: likes  
}

To the semantics of DL is defined in terms of an interpretation  $I = ("21600" \text{ o:spt} = "75" \text{ o:preferrelative} = "t" \text{ path} = "m@4@5l@4@11@9@11@9@5xe" \text{ filled} = "f" \text{ stroked} = "f"> \Delta^I, \cdot^I)$ . The interpretation domain  $\Delta^I$  is a nonempty set, and an interpretation function  $\cdot^I$  that maps each atomic concept  $A$  to a subset  $A^I$  of  $\Delta^I$ , each atomic role  $r$  to a binary relation  $r^I$  on  $\Delta^I \times \Delta^I$ , and each individual  $a$  to an element  $a^I \in \Delta^I$  [13].

Different DL languages use different sets of constructors, which distinguish and limit the expressive power of this DL. The two main DLs cited throughout this project are *ALC* and *SROIQ*. The constructors allowed in the language, their syntax, and their semantics are described in Table 1, where  $C^I$  is the extension of  $C$  in  $I$  and  $b \in \Delta^I$  is an  $r$ -filler of  $a$  in  $I$  if  $(a, b) \in r^I$ .

The logic-based semantics of DLs make each statement to be well-defined and easy to share, so it is easy to judge whether a knowledge base entails a piece of knowledge. DLs use the standard entailment symbol " $\models$ " because the semantics of entailment in DL coincides with FOL [13]. Entailment is a deduction or implication, that is, some axioms are logically derived from or implied by other axioms.

*2.2. Ontology.* In computer science, the term ontology typically represents a formal, explicit specification of a conceptual model specified using some ontology languages [15]. To be more specific, the ontology denotes a computer-processable and well-defined knowledge description form about concepts and their interrelationships. Previous ontology languages are generally based on semantic networks and frames. In contrast, recently, a majority of ontology languages is based on DLs [16]. An ontology can be viewed as a knowledge base. Therefore, an ontology can be regarded as a finite set of axioms. So we can call Example 1 juice ontology.

TABLE 1: DLs semantics.

Language	Name	Syntax	Semantics
<i>ALC</i>	Top	$\top$	$\Delta^I$
	Bottom	$\perp$	$\emptyset$
	Intersection	$\mathbf{C} \sqcap \mathbf{D}$	$\mathbf{C}^I \cap \mathbf{D}^I$
	Atomic negation	$\mathbf{A}$	$\Delta^I \setminus \mathbf{A}^I$
<i>SROIQ</i>	Limited		
	Existential		
	Quantification value	$\exists r$	$\{a \in \Delta^I \mid \exists b \cdot (a, b) \in r^I\}$
	Restriction	$\forall r.C$	$\{a \in \Delta^I \mid \forall b. (a, b) \in r^I \longrightarrow b \in C^I\}$
	Union	$\mathbf{C} \sqcup \mathbf{D}$	$\mathbf{C}^I \cup \mathbf{D}^I$
	Negation	$\mathbf{C}$	$\Delta^I \setminus \mathbf{C}^I$
	Role chain	$\circ$	$\mathbf{r} \circ \mathbf{s} \sqsubseteq \mathbf{t}$
	Nominal	$\{a\}$	$\{a\}^I \subseteq \Delta^I$ with $\#\{a\}^I = 1$
	Inverse		
	Role	$\mathbf{r}^-$	$\{(a, b) \in \Delta^I \wedge \Delta^I \mid (a, b) \in \mathbf{r}^I\}$
	Unqualified	$\geq nr$	$\{a \in \Delta^I \mid \#\{\text{bin} \Delta^I \mid (a, b) \in \mathbf{r}^I\} \geq n\}$
	Number	$\leq nr$	$\{a \in \Delta^I \mid \#\{\text{bin} \Delta^I \mid (a, b) \in \mathbf{r}^I\} \leq n\}$
	Restriction	$= nr$	$\{a \in \Delta^I \mid \#\{\text{bin} \Delta^I \mid (a, b) \in \mathbf{r}^I\} = n\}$
Qualified	$\geq nr.C$	$\{a \in \Delta^I \mid \#\{b \in C^I\} \geq n\}$	
Number	$\leq nr.C$	$\{a \in \Delta^I \mid \#\{b \in C^I\} \leq n\}$	
Restriction	$= nr.C$	$\{a \in \Delta^I \mid \#\{b \in C^I\} = n\}$	

The backbone of ontology entails a generalization/specialization hierarchy of concepts, such as taxonomy. This example can be described in lots of ontology languages. In particular, the most concerned language in this project is Web Ontology Language (OWL), a state-of-the-art semantic web language standardized by the World Wide Web Consortium (W3C). OWL uses its own grammar to explain the grammar in DL, but such a grammatical sentence is too long and complicated. In order to facilitate users to understand the meaning of the content, AD Visualiser uses Manchester syntax, which is a user-friendly compact syntax of OWL ontologies. It is frame-based, contrary to other axiom-based syntaxes of OWL. This project involves these three kinds of syntaxes in total. Example2.2.1 borrows the DL syntax. The OWL syntax is mostly used in OWL files. For the convenience of users to read and understand, the software uses the Manchester syntax on the user interface. The comparison of the DL, OWL, and Manchester syntaxes is shown in Table 2.

**2.3. Module.** Given a seed signature  $\Sigma$ , a module  $M$  is a subset of the ontology  $O$ . Therefore, for all axioms with terms only from the signature, we have  $M \models a$  if  $O \models a$ . As the usability of OWL ontology continues to improve, some of them already contain thousands of concepts. Medium to large ontologies generally contains more than 30,000 axioms (such as gene ontology, including 558,760 axioms [17]). As a result, these posed some major challenges to the entire development process of the ontology, such as understanding, editing, and debugging. As a subset of ontology, modules can be used to share and reuse parts of ontology. In recent years, several approaches to ontology module extraction and ontology modularisation have been explored. For these tasks, the most fundamental question is which module to choose as the basis. This project focuses on locality-based modules (LBMs), a family of logical modules

that provide strong logical guarantees. Compared with other module types, they are proved to be more suitable for module extraction [8]. The reasons are that they are easy to obtain, are computationally efficient, and has been implemented and used to extract modules [18]. Besides, LBMs are as expressive as SROIQ: they provide necessary and unique features (called logical guarantee), such as coverage, self-contained functions, and exhaustive functions, which make the axioms locality. In other words, for each axiom, whether it is included in the module or not, it must be independent of other axioms. They strike a perfect balance between the computable and the minimal. This means that given the seed signature, although the LBM extracted from the ontology possibly contains axioms that are not relevant to the signature, the extraction time is truly short.

**2.4. Atomic Decomposition.** Atomic decomposition (AD) is a method of decomposing ontology into modules and offering a modular structure. Using LBMs as the basis, AD divides an ontology into numerous portions, called atoms, which have a dependency relationship in pairs. An atom is a set of axioms that always cooccur in a module. Thus, any  $\Sigma$ -module either contains all axioms in the atom or does not contain any axioms. Dependency relation means that all modules containing atom  $a$  must also contain atom  $b$ , meaning that atom  $a$  depends on atom  $b$ .

All atoms of the ontology are represented as  $A(O)$ ; then each atom in  $A(O)$  is disjoint with any another one. Atoms are maximal subsets of axioms that are not separated by any  $\Sigma$ -module. The definition of the genuine module is that a  $\Sigma$ -module that cannot be decomposed as the union of two or more incomparable modules. Therefore, every module can be obtained as the union of suitable genuine modules. In this sense, atoms are genuine modules; thus, a new module can be obtained with the union of atoms.

TABLE 2: Comparison of syntaxes.

DL	OWL	Manchester
$T$	owl:Thing	owl:Thing
$\perp$ ConceptnameRole name	owl:Nothing Class	owl:Nothing Class
$C \neg C$	Object property	Object property
$D \text{ HD}$	ObjectComplementOf(C)	notC
$HD$	ObjectUnionOf(C D)	C or D
$\exists r . C$	ObjectIntersectionOf(C D)	C and D
$\forall r . C$	ObjectSomeValuesOf(C D)	r some D
$(\geq n \text{ r} . C)$	ObjectAllValuesFrom(r C)	r only C
$(\leq n \text{ r} . C)$	ObjectMinCardinality(n r C)	r min nC
$(= n \text{ r} . C)$	ObjectMaxCardinality(n r C)	r max nC
	ObjectExactCardinality(n r C)	r exactly n C

The definition of dependency is a binary relation between atoms in terms of cooccurrence in modules. The relation is a partial order: if both  $a$  depends on  $b$ , and  $b$  depends on  $a$ , consequently  $a = b$ . We use  $>$  for the strict partial order underlying  $\leq$ . We can use the standard notions of a principal ideal (a downwards closed subset of a partially ordered set) and an antichain:

- (1) The poset  $(A(O), >)$  is denoted as atomic decomposition (AD) of  $O$ .
- (2) The principal ideal of an atom  $a \in A(O)$  is the set  $\downarrow a = \{b \mid a \leq b\}$ .
- (3) An antichain of atoms is a set  $B \subseteq O$  such that  $a \not\leq b$  for any two distinct  $a, b \in B$ .

An ontology's modular structure is determined by all modules and interrelations of the ontology or at least a suitable subset thereof [11]. This modular structure is based on two fundamental notions: (a) an atom of the input ontology  $O$  is a maximal subset of axioms that are never separated by any module of  $O$  and (b) the dependency relation between atoms of  $O$  captures a further kind of cohesion and allows for a natural definition of basic modules of  $O$ . Consequently, the atoms of  $O$  are  $O$ 's highly cohesive and low-couple subsets, and the number of axioms in  $O$  bounds that of atoms. Because AD is based on LBMs, it also has three types corresponding to the notions of locality ( $\perp$  bot,  $\perp$  top, and  $T\perp^*$  nested).

**2.5. Existing Ontology Visualisers.** A number of software provides developers with a standard ontology development environment, such as Prote'ge' [19], SWOOP [20], and OntoTrack [21]. These tools can assist users in completing ontology-related work in the text form. For humans, however, the text is not as intuitive as images. Hence, with the rise of ontology, ontology visualisation software has received widespread attention as a method of giving information well-defined meaning.

In the last couple of years, with the popularity of ontologies, a variety of ontology visualisation tools have been developed. There are mainly two ways to realise them: a plugin of the ontology editor Prote'ge' and a standalone web application. But visual tools for AD of ontologies are still

scarce, one of which is a tiny visual tool for AD; it is a part of Nicolas Matentzogl'u's framework, named Katana [22]. It only illustrates the AD as a graph without any further operation or information. However, the image gives viewers an insight to consider the structure of ontologies. Since AD is a modular structure of an ontology, the characteristics of the visible results of the software and the overall design concept of the software can be learned and referenced in the AD visualisation tool. The following are three representative software.

**2.5.1. Graph Visualisation of Ontologies.** The graphs are typically laid out in force-directed, radial or hierarchical way, which usually produce appealing visualisations. However, only a few visualisations show complete ontology information. For example, KC-Viz [23], OWLViz [24], and OntoGraf [25] show merely the class hierarchy of ontologies. Numerous works provide more comprehensive graph visualisations that represent all critical elements of ontologies. For example, TGViz [26] and NavigOWL [27] use easy to understandable node-link diagrams where all nodes and links are in different colours. Other than that, 3D-graph visualisations for ontologies, such as OntoSphere and Onto3DViz, provide users with a multidimensional sight to view ontologies.

**2.5.2. UML-Based Ontology Visualisations.** Unified Modelling Language (UML) is not new to most software engineers. The benefits of presenting ontology information in that language form are obvious: engineers can easily understand and have familiarity with the tools. However, its disadvantages cannot be ignored: on the one hand, it is unfriendly to users who are not familiar with UML as some basic knowledge of UML is required. On the other hand, it is designed to associate objects in the field of software engineering, and there are some limitations when it comes to the knowledge presentation domain. For example, ontology focuses on the description of relationships between classes, while software engineering focuses on the description of properties and methods of objects themselves.

**2.5.3. Symbol-Based Ontology Visualisations.** The OWLGrEd Ontology Visualiser (OWLGrEd) [28] is an online visualiser for OWL ontologies using a compact UML-

based notation. OWLGrEd provides a “bird’s-eye view” of the ontology to help developers debug ontologies. The notation of OWLGrEd is UML-style diagrams that most software engineers are familiar with and easy to use. OWLGrEd map OWL features to UML concepts, that is, OWL classes to UML classes, data type properties to class attributes.

Using defined symbols to explicit splitting rules, symbol-based ontology visualiser specifies different elements in the graph. The visual representation of OWL ontology is a visual language for ontology representation. It defines the graphical descriptions of most elements of OWL, and these are represented as forced graph layouts of visual ontologies that replace text.

The Visual Notation for OWL Ontologies (VOWL) [29, 30] is a type of visual language for the user-oriented ontologies representation. It provides graphical representations for elements of the OWL Web Ontology Language. Not only domain experts but also beginners have the ability to understand the content of OWL ontology with clearly specified symbols.

### 3. Design and Implementation

*3.1. Design.* This project aims to create an atomic decomposition visualiser (AD Visualiser) for ontology engineers to express the AD of large- and medium-sized ontology intuitively and comprehensively. AD Visualiser focuses on two parts of the visual representation: one part is the hierarchical structure of classes and object attributes, and the other part is the atoms and their relationships. The functional design of AD Visualiser is mainly based on the three ontology-related tasks mentioned in Section 3, to help users complete tasks-related tasks. AD Visualiser can parse and display a graph for the AD of an OWL ontology in an interactive way. For medium and large ontology, the way to visualise AD is more direct and objective than using other edit tools such as Prote’ge’, and it offers a method to find the module related to a signature faster. Users can expeditiously browse the ontology in a modular structure to find and extract the modules they need. In the reasoning result, users can quickly see classes, object property hierarchy, and graphical AD. Users can detect the atom’s information efficiently in the tool. They can recognise the notion of the modular structure of an ontology, which has the potential to help users comprehend, share, and maintain ontology. Users can use it to guide their extraction choices, to understand its topicality, connectedness, structure, unnecessary parts, or differences between actual and intended modelling. For example, ontology designers can inspect the modular structure and observe unconnected parts that are intended to be connected and modelled parts of their ontology.

Figure 1 depicts the system flow chart. It illustrates the possible results and corresponding logic of all operations that the user may perform from the start of the software. In addition, it also reveals how the tool guides users to explore the ontology, and what functions the tool provides for users to interact with the ontology.

*3.2. Basic Building Blocks of AD Visualiser.* AD Visualiser uses the OWL API to underpin all ontology management tasks, including loading, decomposing, and saving ontologies. Then, it uses Gephi API to draw the graph.

*3.2.1. User Interface.* Figure 2 shows the user interface (UI) prototype diagram of AD visualiser that includes the menu bar, search bar, classes (and object property) hierarchy, and graphical AD.

The visualisation of AD is one of the most significant tasks of AD Visualiser. When displaying image-based AD, in order to retain and display as many AD functions as possible, we set the following information conversion method from text to image. The features of AD determine the structure of its graphical result. In the graph, each node corresponds to an atom in the AD, and each arrow line between a pair of nodes corresponds to a dependency relationship between two atoms. Moreover, the graph layout depends on the source data structure so that the graph of AD can be easily generated. The reason is that the result of AD is a partial set, so the graph of AD is the type of directed acyclic graph (DAG).

In an AD, an atom has many attributes. For example, an atom’s size is determined by the number of its axioms; an atom may be top or bottom atom according to its position (an atom only have dependents may be at the bottom, and the atom only have dependencies may be at the top); an atom may have many types of labels (positive Boolean formula (PBF) label is used in this project, which is a representation of all seed signatures of the atom’s module with the only union and intersection constructors). In the program of AD Visualiser, every node is an object with multiple attributes to represent the relevant atom’s attributes. This system defines a set of visual language systems to help users understand AD in a visual language way. For example, the node’s size is relevant to the atom’s size; the node’s position shows the atom’s position in the AD; the node’s label corresponds to the atom’s label; and much more. The features of nodes in a graph are more intuitive and easy to understand than the information of atoms in textual AD.

Algorithm 1 shows the process of drawing DAG in AD Visualiser.

Another important component in AD Visualiser is hierarchy trees of classes and object properties.

Algorithm 2 shows the process of building a classes tree.

*3.3. Colour Scheme.* Colours play a significant role in beautifying user interfaces. For example, in graphical AD, colours are important from differentiating nodes’ positions to finding the target node to identifying a module. Besides, elements in the background and foreground of a screen have to be different colours. If their colours are the same, it is difficult to identify the foreground elements immediately. Hence, colours that distinguish with each other are also meaningful.

Figure 3 illustrates that even the colours of the small square and background are different, it is still ugly, and people may feel uncomfortable when looking at them. The

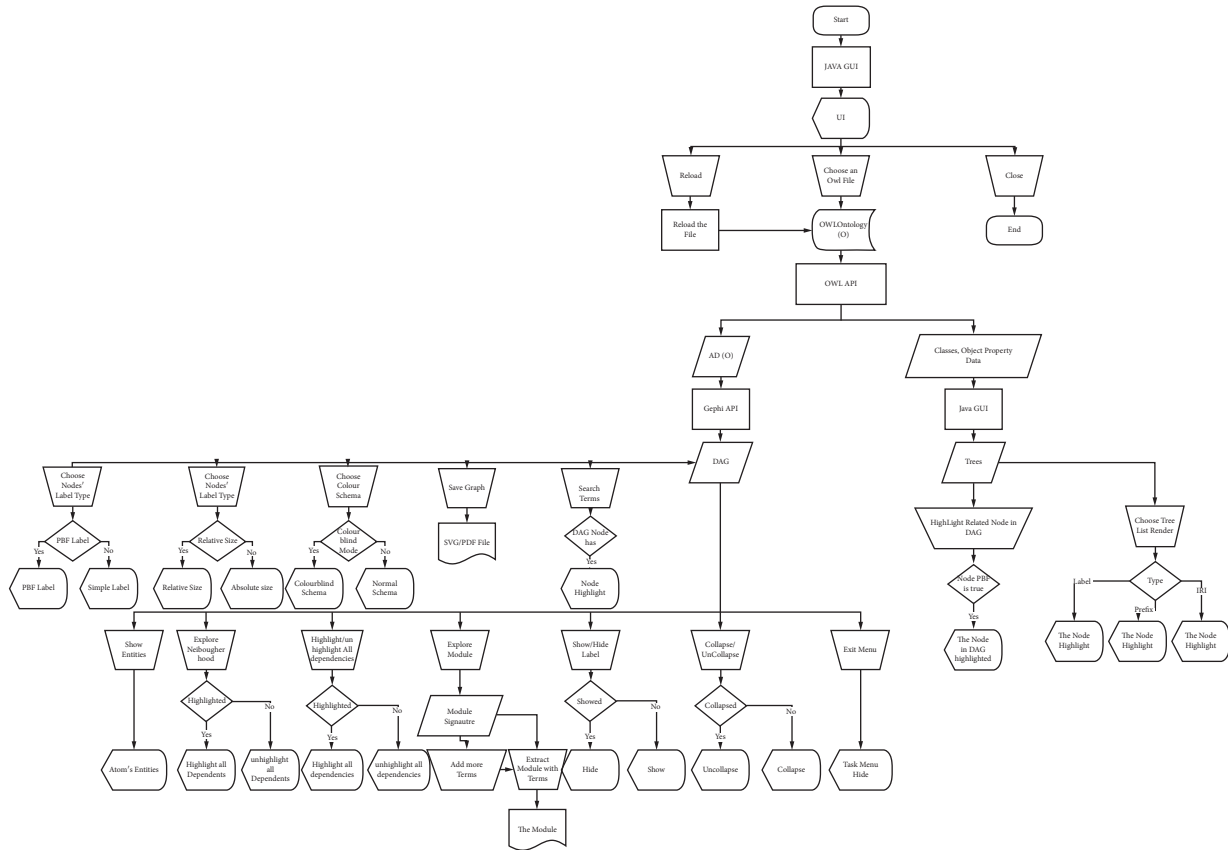


FIGURE 1: The system flow chart.

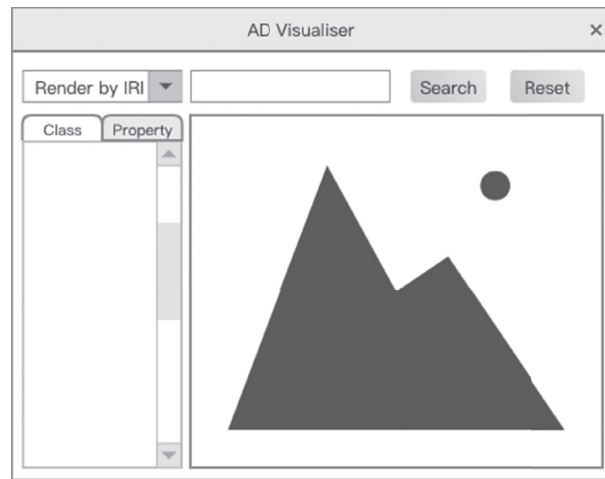


FIGURE 2: User interface design.

reason is that the contrast ratio for the two colours is smaller than 4.5:1 [31].

If the contrast ratio is higher, the screen looks more legible and readable. Moreover, human beings may have different experiences with the same colour. Some of us have defects in vision, called colour blindness, which is defined as the inability to distinguish the same colour differences (the most common ones are red and green or blue and yellow). It is

estimated that 1 in 12 men and 1 in 200 women have such an unusual colour experience [31]. In order to make these people have a better experience, AD Visualiser provides a colour blind mode for users to choose. The solution for these users is still to increase the contrast of colours. By using high-contrast colours, they can distinguish these colours no matter what actual colours are used. Following these colour matching rules, the colour scheme of this project is shown in Table 3.

```

Input: An Ontology  $O$ 
Output: A DAG  $g$ 
(1)  $ad \leftarrow AtomicDecomposition(O)$ ;
(2)  $g \leftarrow DirectedGraph(ad)$ ;
(3)  $graphNodes \leftarrow \emptyset$ ;
(4)  $topAtoms \leftarrow \{atoms \in ad\}$ 
(5)  $dep(atom) \leftarrow \{atoms \in dependenciesofatom\}$ 
(6) foreach  $atom \in topAtoms$  do
(7)  $graphNodes \leftarrow graphNodes \cup atom$ ;
(8) end for
(9) repeat
  10 if  $atom/graphNodes$  then
(11)  $graphNodes \leftarrow graphNodes \cup atom$ ;
(12) endif
(13) if  $dep(atom) = \emptyset$  then
(14) for  $eachatomChild \in dep(atom)$  do
(15) if  $atomChild/graphNodes$  then
(16)  $graphNodes \leftarrow graphNodes \cup atomChild$ ;
(17) endif
(18) endfor
(19) endif
(20) until add all the atoms in  $graphNodes$  of  $ad$ 
(21) repeat
(22) if  $ad.getDependencies(atom).size() \neq 0$  then
(23) foreach  $atomChild \in ad.getDependencies(atom)$  do
(24)  $g.addEdge(atom, atomChild)$ ;
(25) endfor
(26) endif
(27) until draw all the edges
(28) show the result  $g$ ;

```

ALGORITHM 1: DAG drawing algorithm.

```

Input: An Ontology  $O$ 
Output: A DAG graph
(1)  $superClass \leftarrow owl: Thing$ ;
(2)  $super \leftarrow Node\ superClass$ ;
(3) repeat
(4) if  $subClasses(superClass) \neq \emptyset \ \&\& \ owl: Nothing \notin subClasses(superClass)$ 
then
(5) foreach  $subClass \in subClasses(superClass)$  do
(6)  $subNode \leftarrow subClass$ 
(7)  $superNode.addSubNode(subNode)$ ;
(8)  $superClass \leftarrow subClass$ ;
(9) endfor
(10) endif
(11) until add all the classes as nodes in the tree
(12) show the result tree;

```

ALGORITHM 2: Tree building algorithm.

In this project, many types of highlights are used to distinguish the target nodes from others. The author shows people's possible different experiences with colours in the picture below. These include deuteranopia (Figure 4; lack of green affects about 5% of men), protanopia (Figure 5; red defects affect about 2.5% of men), tritanopia (Figure 6; blue defect affects about 0.5% of men), and grayscale (Figure 7; luminance-preserving grayscale simulation).

**3.4. Functionality Introduction.** Users can view a graphical AD of an OWL file and then explore the AD around or target to their interested module with a signature. As mentioned before, a module is a principal ideal of an atom  $a$ ; each atom is a unique set of axioms; and each node in the graph corresponds to an atom. To quickly find a, every atom has two types of labels,

The simple label containing all terms in the atom and positive Boolean formula (PBF) label. Given the terms, the

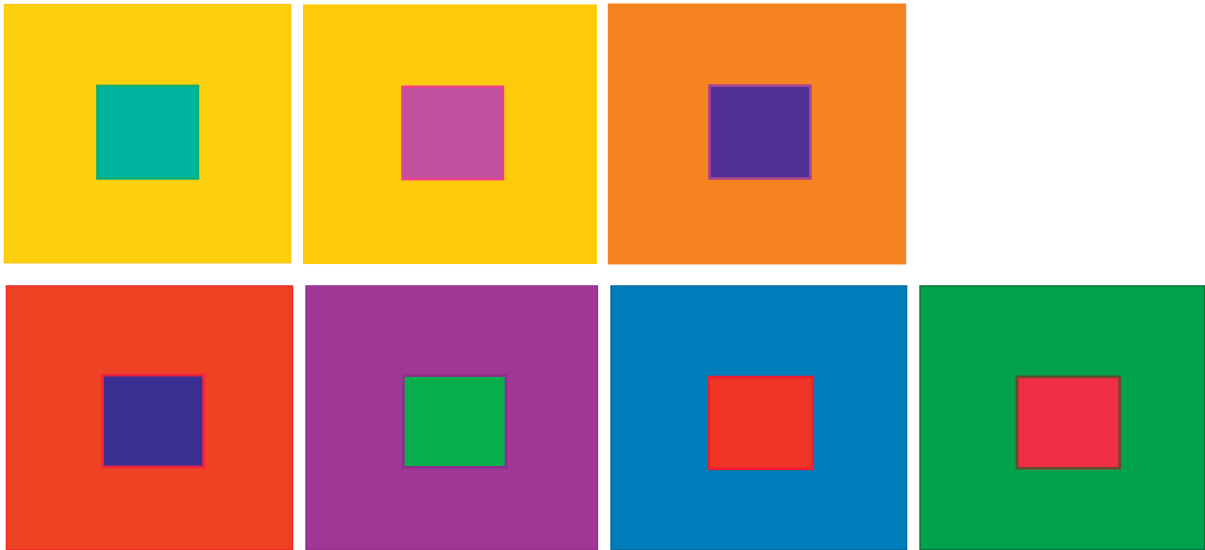


FIGURE 3: Colour contrast [31].

TABLE 3: Colour schema.

Name	Colour	Application
Green		Nodes below connected to a node
Cyan		Nodes direct connected to a node
Blue		Nodes have hidden subnodes
Yellow		Nodes are not bottom nodes
Grey		Nodes are bottom nodes
Red1		Nodes contain one term
Red2		Nodes contain two terms
Red3		Nodes contain three terms
Red4		Nodes contain four terms
Red5		Nodes contain five terms
Red6		Nodes contain more than five terms

tool calculates each atom's PBF label and then highlights true-result corresponding nodes. With a, users can explore the genuine module and extract it until they feel satisfied.

These are three solutions to tasks mentioned in Section 3.2:

- (i) Ontologists not only have a bird's-eye view of an ontology with a modular structure but also can focus on a small part of an ontology. Additionally, they have the opportunity to check each atom's information. Hence, we suppose this tool is helpful for them to refine ontologies.
- (ii) A feasible way to merge ontologies is to import other ontologies from the web. After merging, designers can find, and the term may be involved in many small ontologies to check whether or not they are intended in the final big ontology. Consequently, we suppose this tool should be useful for ontology merging.
- (iii) Developers have the opportunity to explore an ontology with terms and extract their interesting modules from an ontology in AD Visualiser.

That is pretty helpful for them to reuse the ontology flexibly.

### 3.5. Implementation

**3.5.1. User Interface.** Figure 8 depicts the layout of AD Visualiser's user interface. It depends on the GridBagLayout [32], a type of flexible layout manager provided by the Java platform.

Figure 9 displays the grid for the user interface. As shown in the screenshot, the grid has two rows and four columns. All the components are in the grid except the menu bar fixed at the top of the window. Particularly, the panel in the lower right corner spans three columns.

Using juice ontology as an example, Figure 10 shows how the DAG looks like in AD Visualiser in which grey nodes are bottom nodes and the others are yellow.














Name	Colour	Application
Green		Nodes below connected to a node
Cyan		Nodes direct connected to a node
Blue		Nodes have hidden subnodes
Yellow		Nodes are not bottom nodes
Grey		Nodes are bottom nodes
Red1		Nodes contain one term
Red2		Nodes contain two terms
Red3		Nodes contain three terms
Red4		Nodes contain four terms
Red5		Nodes contain five terms
Red6		Nodes contain more than five terms

FIGURE 4: Deuteranopia.












Name	Colour	Application
Green		Nodes below connected to a node
Cyan		Nodes direct connected to a node
Blue		Nodes have hidden subnodes
Yellow		Nodes are not bottom nodes
Grey		Nodes are bottom nodes
Red1		Nodes contain one term
Red2		Nodes contain two terms
Red3		Nodes contain three terms
Red4		Nodes contain four terms
Red5		Nodes contain five terms
Red6		Nodes contain more than five terms

FIGURE 5: Protanopia.












Name	Colour	Application
Green		Nodes below connected to a node
Cyan		Nodes direct connected to a node
Blue		Nodes have hidden subnodes
Yellow		Nodes are not bottom nodes
Grey		Nodes are bottom nodes
Red1		Nodes contain one term
Red2		Nodes contain two terms
Red3		Nodes contain three terms
Red4		Nodes contain four terms
Red5		Nodes contain five terms
Red6		Nodes contain more than five terms

FIGURE 6: Tritanopia.












Name	Colour	Application
Green		Nodes below connected to a node
Cyan		Nodes direct connected to a node
Blue		Nodes have hidden subnodes
Yellow		Nodes are not bottom nodes
Grey		Nodes are bottom nodes
Red1		Nodes contain one term
Red2		Nodes contain two terms
Red3		Nodes contain three terms
Red4		Nodes contain four terms
Red5		Nodes contain five terms
Red6		Nodes contain more than five terms

FIGURE 7: Grayscale.



FIGURE 8: UI of AD Visualiser.

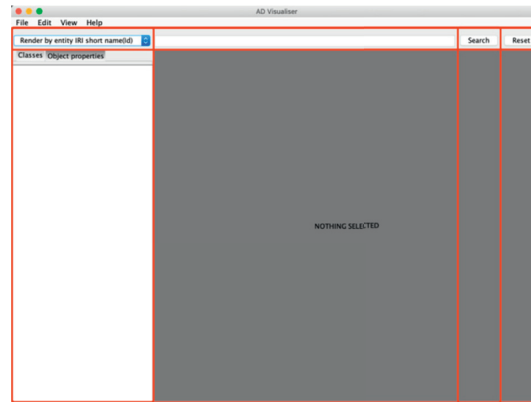


FIGURE 9: GridBagLayout.

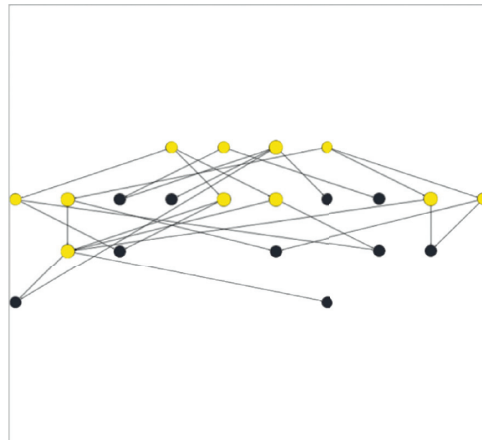


FIGURE 10: DAG of juice ontology.

3.5.2. *Search Terms.* Figure 11 shows the feedback from AD Visualiser to users when searching the term carrot.

3.5.3. *The Node Menu in Graphical AD.* As shown in Figure 12, the implementation of the functions in the menu bar is as follows:

- (i) *Show Entities.* Figure 13 shows the entities of the highlighted node.
- (ii) *Explore Neighbourhood.* As shown in Figure 14, the neighbours of the chosen node are cyan.
- (iii) *Explore All Dependencies.* As shown in Figure 15, the subnodes (all descendants) pointed to by the selected node are in green.
- (iv) *Explore Module.* As shown in Figure 16, the “Module Signature” is the signatures obtained

through the seed signature pre-extraction module. Related signature uses the characteristics of the directed graph to extract the signatures contained in the atoms corresponding to the node that is connected to the selected node and the arrow points to the selected node. When the user adds an atom to the module, the author uses the same method to add the signatures of its related atoms to a table, which is convenient for users to view and operate.

- (v) *Show or Hide a Node Label.* The label of a node shows or hides.
- (vi) *Collapse and Uncollapse a Node Chain through Exploring All Dependencies.* Nodes connected below the selected node are obtained. As shown in Figure 17, hiding these nodes can remove these nodes from the graph.

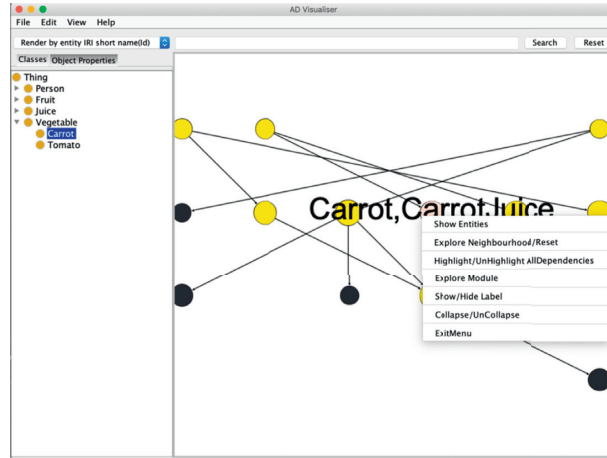


FIGURE 11: Search carrot.

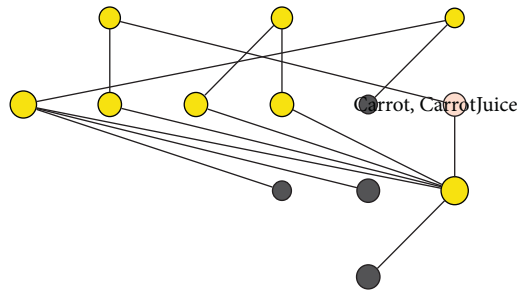


FIGURE 12: Node menu.

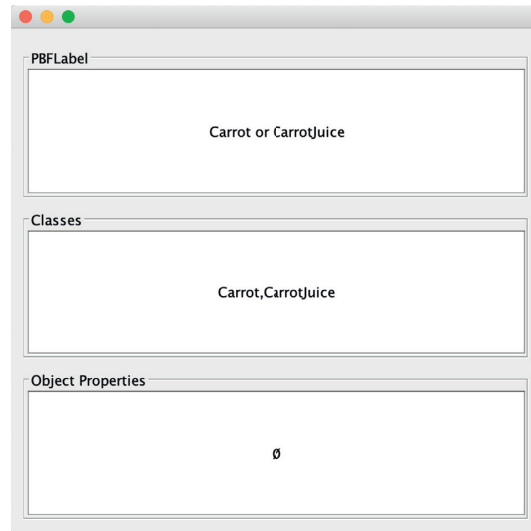


FIGURE 13: Show entities.

#### 4. Testing and Evaluation

In the later stages of program development, testing and evaluation are critical to measuring and improving system performance. For this project, software testing is divided

into two parts. The first part is the self-testing of the software performance: whether it meets the functional requirements mentioned in Section 3, the operating speed, and efficiency of the software. The second part is to invite experts to try the software and then give the trial experience and opinions.

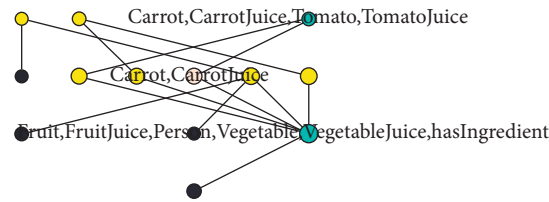


FIGURE 14: Explore neighbourhood.

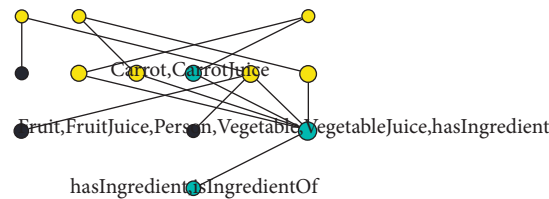


FIGURE 15: Highlight all dependencies.

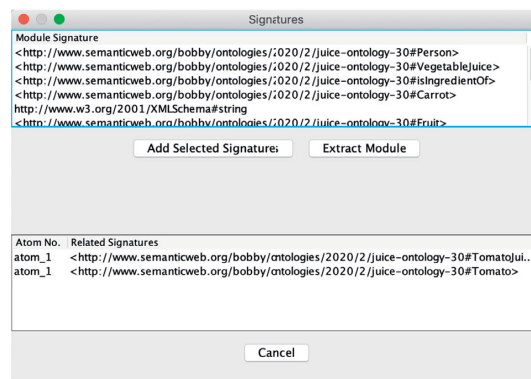


FIGURE 16: Explore module.

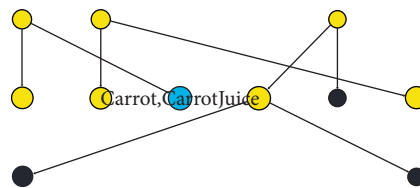


FIGURE 17: Collapse a node chain.

#### 4.1. Test Design and Implementation. Test Cases Selection.

Since AD is more dependent on logical axioms, the author deliberately selected some small (including 1,000 logical axioms), medium (including 1,000 to 20,000 logical axioms), and large ontologies (including more than 20,000 logical axioms) for testing. Through this method, it is possible to test the visual results and functional support of AD Visualiser for ADs of different sizes ontologies.

*Influence Factors.* The length of time to parse the advertisement depends on the OWL API. Chiara Del Vescovo, the inventor of AD theory, had tested 357 ontologies; the computing time for half of the ontologies spend less than 1 second; 95% spend within 2 minutes;

and 99% spend no more than half and 1 hour. Besides, the length of time for Gehpi API drawing graphical AD and JTree drawing hierarchy trees depends on Algorithms 1 and 2, respectively. The author tested six small ontologies, one medium ontology, and two large ontologies and individually recorded the time spent in each stage before the image was displayed. Table 4 illustrates the performance of the AD visualiser when processing these ten ontologies. Figure 18 shows the visualisation result of pizza ontology as a representative of small ontologies. Figure 19 presents the visualisation result of wbphenotype as a representative of medium ontologies, and Figure 20 displays the visualisation of gene ontology as a representative of large ontologies.

TABLE 4: Processing time.

Name	Axioms	TBox	Atoms	Type	AD (s)	DAG (s)	Tree (s)	Total (s)
juice	53	33	BOT	13	0.047	1.235	0.074	2.170
			TOP	1	0.008	1.001	0.12	1.062
			STAR	13	0.018	1.008	0.009	1.065
gfo-basic	479	212	BOT	75	0.080	1.159	0.124	2.183
			TOP	2	0.010	1.002	0.032	1.126
			STAR	90	1.090	0.039	1.019	1.290
Biblio	332	219	BOT	29	0.012	1.012	0.007	1.076
			TOP	197	0.17	1.014	0.009	1.233
			STAR	99	0.022	1.017	0.009	1.090
asdphenotype	1,434	283	BOT	283	0.010	2.019	0.022	2.127
			TOP	77	0.012	1.004	0.030	1.094
			STAR	283	0.010	1.007	1.078	
pizza	801	322	BOT	89	0.045	2.030	0.220	2.380
			TOP	1	0.011	1.002	0.145	1.215
			STAR	91	0.088	2.023	0.118	2.286
gist	664	384	BOT	154	0.135	2.051	0.469	2.821
			TOP	5	0.019	1.010	0.472	1.559
			STAR	159	0.165	2.075	0.645	2.968
wbphenotype	20,255	3,939	BOT	2581	6.355	31.417	5.471	45.149
			TOP	348	286.78	4.823	5.876	40.353
			STAR	2774	8.348	25.168	5.179	39.438
fission-yeast	35,968	27,602	BOT	8281	47.143	216.667	190.65	285.153
			TOP	20	6.282	1.975	30.7178	
			STAR	10464	70.385	475.906	182.37	565.964
cell-culture	53,297	33,235	BOT	20058	185.54	441.972	122.22	475.740
			TOP	5	7.416	1.606	378.45	48.226
			STAR	27481	38.209	435.993	42.804	518.325
gene	558,760	103,676	BOT	43652	1058.634	535.129	2004.321	
			TOP	1	8.766	1.390	441.792	465.306
			STAR	63311	669.733	948.899	478.616	2112.548

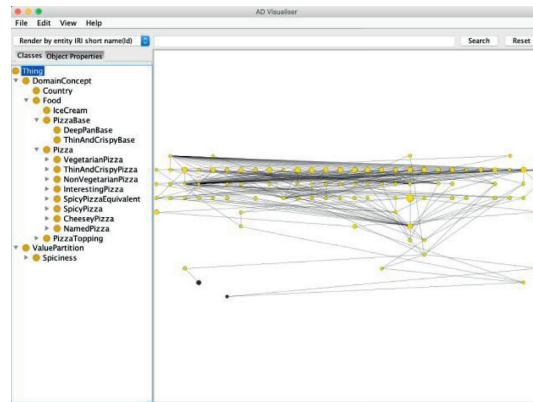


FIGURE 18: The test of pizza ontology.

### 5. Results and Analysis

AD Visualiser fully supports small ontologies; the visualisation results are clear; and various functions are fully supported. For medium-sized ontologies, the graphical ADs are relatively clear, and users may have a little difficulty in finding the highlighted nodes. However, users can view nodes' information and explore their neighbourhood at will, and AD Visualiser is also very smooth to use. Moreover, for large ontologies, the visualisation results are very poor. Users

may have a clear sense of lag in use, and users can hardly find the highlighted nodes.

Table 4 depicts that the processing time of AD Visualiser on the ontology increases as the number of logical axioms (TBox in Table 4 points at the number of ontology's logical axioms) in the ontology increases. They are approximately linearly related. Over the first 15 minutes, Uli Sattler introduced the origin and principle of the theory of AD. Then the author spent some time introducing and showing how to use AD Visualiser to the audiences. Dave McComb asked the

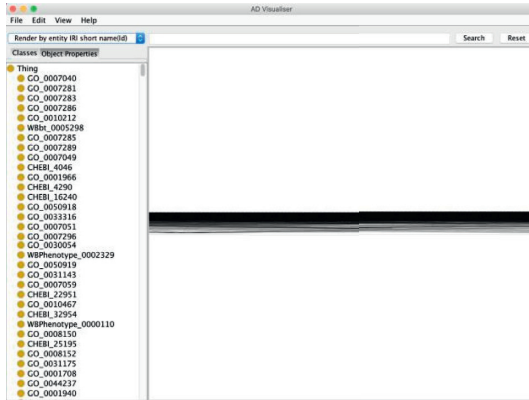


FIGURE 19: The test of wbphenotype on-tology.

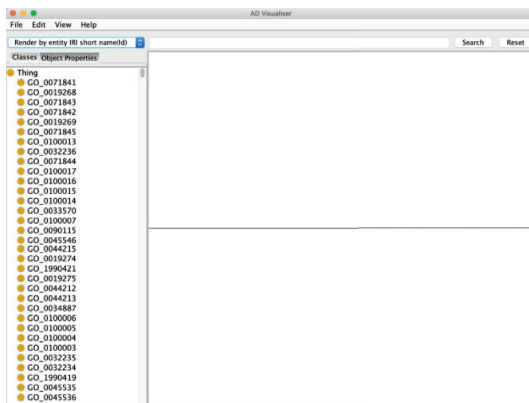


FIGURE 20: The test of gene ontology.

author to extract a module about the term Person. Then, we checked the person module in Protege. The details are shown in Figures 21 and 22.

The result of module extraction surprised all of the audiences. Even though they are designers and makers of gist ontology, they have never thought that so many terms are related to the term person, and then heated discussions began. To their surprise, they even wondered that if the author opened the correct module just extracted. Haoruo Zhao discovered that the module contains 41 disjoint axioms, which may be the possible reason for retaining some much knowledge perhaps, which is attributed to logical guarantees. They thought disjoint is a common and important relation between axioms, but they were not convinced that this relation has such a powerful influence on the term or the ontology.

Dave McComb, the president and cofounder of Semantic Arts, believes the visualisation of AD is less useful for him. Instead, the important thing is obtaining a suitable module from an ontology. He said “the presentation helped me think a lot in this respect, it did not occur to me until this that there would have to be some ‘explain’ function that could explain why a given concept or axiom got included.” In terms of the task of ontology reusing, he said most existing ontologies are not reusable. As every OWL ontology has a domain and range, once the module is extracted out from ontologies’

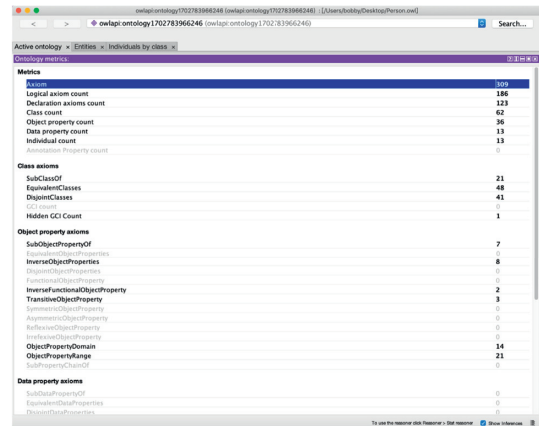


FIGURE 21: Ontology metrics of person module.

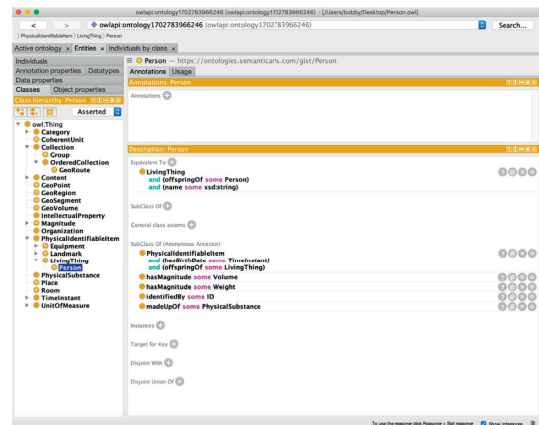


FIGURE 22: Classes hierarchy of person module.

original context, they are not compatible with others. Sometimes, ontologists even need to trim the module down to make it compatible with new ontologies. He supposed such a complex operation could not be accomplished by a tool or algorithm automated.

Peter Winstanley, an ontologist in the Semantic Arts and cochair of the W3C Dataset Exchange Working Group, said AD Visualiser had stimulated a more in-depth revision of an ontology, such as gist ontology. Its development tends to be organic and driven by many stakeholders. He thought AD Visualiser provides a new insight for his colleagues and him for their daily work.

Michael Uschold, a senior ontology consultant at Semantic Arts, is an internationally recognised expert with more than two decades of experience in developing and transitioning semantic technology from academia to industry. He pointed out that AD Visualiser cannot be used immediately in his daily work. This is because the supergranularity makes the graphics huge, and then it is not easy to see at a glance. Such a case is a significant obstacle to the effective use of images. In this regard, he suggested that images can be cut into small pieces so that users can see a magnified view of a specific area. Regarding the functional requirements of the tool, he hopes that the tool can achieve

excellent support for the input of a specific ontology with hundreds of classes and object properties. After proper processing, the output is a subset of the gist that should have everything they need, and nothing else is superfluous.

More than dozens of people attended this online conference, and they are all experienced engineers working on ontology. Most of them are rather interested in AD Visualiser and its functionality. For Dave's negative feedback, it may be because the author only focuses on introducing the functionality of the tool in a short period and neglected to introduce the characteristics of visualisation. For example, users can intuitively see the size of the atom and the relationship between atoms, quickly find the top and the bottom atoms and explore more simply and quickly to extract modules.

## 6. Conclusion and Discussion

This project is to develop a Java-based tool called AD Visualiser to visualise OWL ontology in a modular structure based on AD theory. Unlike other existing ontology visualisation tools that focus on displaying ontology content with image symbols, AD Visualiser pays more attention to assisting users in completing ontology-related tasks. AD Visualiser allows users to interact with graphical AD in a variety of ways, such as finding nodes related to terms, obtaining detailed information about the terms contained in a node, and viewing the dependency relationship of a node corresponding to the atom. Its functional design is based on three tasks related to ontology: ontology refinement, ontology merging, and original use. At present, it has achieved two specific functions. One of them is the visualisation of AD that better supports medium-sized ontology (including 1,000 to 20,000 axioms). The ultrafine granularity of AD causes that when the body is too large, the number of atoms in the imaged AD and the dependence between atoms are too much, so the current AD Visualiser's display effect of its imaged AD is not good. Another is the support of module extraction with some terms from the ontology. This module is encapsulated, which means that it contains all the knowledge about terms and is independent of the original body.

The limitations have been identified and discussed in Section 5 based on expert feedback. In the next version of AD Visualiser, modifications and upgrades will be made based on these issues.

First of all, it is most important to change the display mode of the graphical AD. On the one hand, the author plans to display AD containing less than 100 atoms directly. For larger AD, take bottom AD as an example. In the beginning, the graphical AD only displays bottom atoms and then displays related nodes (including terms and the PBF result is true), and all its dependencies based on the terms are searched by the user. Besides, they can also explore the neighbourhood of any node. In this way, AD Visualiser can support the visualisation of large ontology AD. Users can focus on the modules related to terms or freely explore AD from the bottom to up. At the same time, it can also greatly reduce the computer's memory consumption and computing time. On the other hand, in the graphical AD, the user will be able to hide all the nodes above and connect to a node.

In this case, users can hide information they do not care about. Additionally, a scale will be added to the lower left of the graphical AD to assist users in zooming in and out.

Next, optimise the colour matching of the software user interface to enhance the user's visual experience. The author plans to convey the information using as few colours as possible. The nodes in the graphical AD are in yellow except the bottom nodes that are in grey. When the user locates a node related to the terms, the related nodes are in red. The more the terms contained, the darker the red. When exploring the neighbourhood of a node, it and its neighbours are in blue. At the same time, nodes with unshown neighbourhoods are no longer in blue. When the node explored by the user has no neighbours, AD Visualiser will prompt with a symbol and a red text message next to it. In order to facilitate users to understand the meaning of different colours, the author plans to record the colour information in the colour schema item in the help column of the menu bar for users.

Then, optimise the details of the interface display during the user module extraction process. When the user selects terms in the hierarchy trees of classes or object properties or searches for terms in the search bar, the colour of the icons of the selected nodes in the hierarchy trees changes to red to show that they are selected. When the user explores the module according to any node, the user is allowed to go back and cancel the current operation after adding a piece of content.

The author also plans to improve the user experience of AD Visualiser. One of them is to display a loading bar between user operations to remind users of the estimated waiting time, for example, when the user opens an OWL file and waits for the image to load, when the user locates the relevant node in the map through terms to refresh the image content, when the user saves the explored module, and when the user saves the image of the graphical AD. Although many tasks will be completed within a few seconds, the flashing loading bar can also display a kind of feedback to the user's operation.

In addition to optimising the existing functions, AD Visualiser will add new functions to meet user needs (ontology refinement, ontology merging, and original use). AD Visualiser distinguishes the AD atoms of each ontology referenced by the four colour map theorem with different colours. Among them, the atoms belonging to multiple bodies are in colour as a result of colour mixing. This function can help users view the status of each component ontology in the merged ontology. In the future, AD Visualiser may be used as a plug-in for Protege to help more ontology engineers.

## Data Availability

All data, models, and code generated or used during the study appear in the submitted article.

## Conflicts of Interest

The author declares that there are no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

## References

- [1] N. F. Noy, *Tools for Mapping and Merging Ontologies*, Springer, Berlin, Germany, 2004.
- [2] M. Obitko and V. Marik, "Adding owl semantics to ontologies used in multi-agent systems for manufacturing," in *Proceedings of the International Conference on Industrial Applications of Holonic & Multi-Agent Systems*, Prague, Czech Republic, September 2003.
- [3] D. Fensel, "Ontology-Based Knowledge Management," *IEEE Computer Society Press*, vol. 35, no. 11, 2002.
- [4] C. C. Albrecht, D. L. Dean, and J. V. Hansen, "An ontological approach to evaluating standards in e-commerce platforms," *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 5, pp. 846–859, 2007.
- [5] S. Staab and R. Studer, "Handbook on ontologies," *International Handbooks on Information Systems*, Springer, vol. 2, pp. 227–255, Berlin Germany, 2004.
- [6] K. A. Spackman, K. E. Campbell, and R. A. Cote, "Snomed rt: a reference terminology for health care," *Proceedings A Conference of the American Medical Informatics Association*, vol. 4, pp. 640–644, 1997.
- [7] C. Rosse and J. L. V. Mejino, "A reference ontology for biomedical informatics: the foundational model of anatomy," *Journal of Biomedical Informatics*, vol. 36, no. 6, pp. 478–500, 2003.
- [8] B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler, "Modular reuse of ontologies: theory and practice," *Journal of Artificial Intelligence Research*, vol. 31, no. 1, pp. 273–318, 2008.
- [9] R. Kontchakov, L. Pulina, U. Sattler et al., "Minimal module extraction from dl-lite ontologies using qbf solvers," in *Proceedings of the 21st International Joint Conference on Artificial Intelligence IJCAI*, vol. 9, pp. 836–841, Pasadena, CA, USA, July 2019.
- [10] P. Klinov, C. Del Vescovo, and T. Schneider, *Incrementally Updateable and Persistent Decomposition of Owl Ontologies*, OWLED, 2012, [http://webont.org/owlled/2012/papers/paper\\_7.pdf](http://webont.org/owlled/2012/papers/paper_7.pdf).
- [11] C. Del Vescovo, B. Parsia, U. Sattler, and T. Schneider, "The modular structure of an ontology: atomic decomposition," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, Barcelona, Spain, July 2011.
- [12] V. Geroimenko and C. Chen, "Visualizing the Semantic Web," *Xml-Based Internet and Information Visualization*, Springer, London, UK, 2006.
- [13] F. Baader, I. Horrocks, C. Lutz, and U. Sattler, *Introduction to Description Logic*, Cambridge University Press, Cambridge, UK, 2017.
- [14] C. Del Vescovo, "The modular structure of an ontology: atomic decomposition and its applications," PhD Thesis, University of Manchester, Manchester, UK, 2013.
- [15] T. R. Gruber, "A Translation Approach to Portable Ontology Specifications," *Knowledge Acquisition*, vol. 5, no. 2, 1993.
- [16] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patelschneider, "The description logic handbook," *Theory, Implementation, and Applications*, Cambridge University Press, Cambridge, UK, 2nd edition, 2007.
- [17] M. A. Harris, J. Clark, A. Ireland et al., "The gene ontology (go) database and informatics resource," *WCB/McGraw-Hill*, vol. 32, 2004.
- [18] U. Sattler, T. Schneider, and M. Zakharyashev, "Which kind of module should i extract?" in *Proceedings of the International Workshop on Description Logics*, Oxford, UK, July 2009.
- [19] H. Knublauch, R. W. Ferguson, N. F. Noy, and M. A. Musen, *The Protege Owlplugin: An Open Development Environment for Semantic Web Applications*, Springer, Berlin, Germany, 2004.
- [20] A. Kalyanpur, B. Parsia, E. Sirin, B. C. Grau, and J. Hendler, "Swoop: A web ontology editing browser," *Journal of Web Semantics*, vol. 4, no. 2, pp. 144–153, 2014.
- [21] T. Liebig and O. N. Ontotrack, "A semantic approach for ontology authoring," *Web Semantics: Ence, Services and Agents on the World Wide Web*, vol. 3, no. 2, pp. 116–131, 2011.
- [22] N. Matentzoglou, "Module-based Classification of Owl Ontologies," Thesis, University of Manchester, Manchester, UK, 2016.
- [23] E. Motta, S. Peroni, J. M. Go´mez-Pe´rez, M. D’Aquin, and N. Li, *Visualizing and Navigating Ontologies with KC-Viz*, Springer, Berlin, Germany, 2012.
- [24] M. Horridge, "Owlviz," 2020, <https://protegewiki.stanford.edu/wiki/OWLviz>.
- [25] S. Falconer, "Ontograf," 2020, <https://protegewiki.stanford.edu/wiki/OntoGraf>.
- [26] H. Alani, "Tgviz," <https://protegewiki.stanford.edu/wiki/TGViz>, 2020.
- [27] K. L. Ajaz Hussain and A. T. Rextin, "Navigowl," 2020, <https://protegewiki.stanford.edu/wiki/NavigOWL>.
- [28] R. Liepins, M. Grasmanis, and U. Bojars, "Owlgred ontology visualizer," in *Proceedings of the 2014 International Conference on Developers*, vol. 1268, pp. 37–42, Riva del Garda, Italy, October 2014.
- [29] S. Lohmann, S. Negru, F. Haag et al., "Visualizing ontologies with vowl," *Semantic Web*, vol. 7, no. 4, pp. 399–419, 2016.
- [30] D. M. M. Uschold, "Introduction to gist," 2020, <https://iaoa.org/isc2014/uploads/Whitepaper-Uschold-IntroductionToGist.pdf>.
- [31] P. Pierce, "How to boost usability with intelligent color choices," 2020, <https://www.sitepoint.com/how-to-boost-usability-with-intelligent-color-choices/>.
- [32] J. Zukowski, *Java AWT Reference*, OR’eilly & Associates, Inc., Sebastopol, CA, USA, 1997.