Chowlk: from UML-based ontology conceptualizations to OWL*

Serge Chávez-Feria[0000−0002−7454−9202],
Raúl García-Castro[0000−0002−0421−452X], and
María Poveda-Villalón[0000−0003−3587−0367]

Ontology Engineering Group, Universidad Politécnica de Madrid, Spain
serge.chavez.feria@upm.es, {rgarcia, mpoveda}@fi.upm.es

Abstract. Ontology conceptualization is an ontology development task that consists in generating a preliminary model based on the requirements that the ontology should represent. This activity is often carried out by generating the models as diagrams in a blackboard, paper or digital tools. The generated models drive the ontology implementation activity, where the model is formalized and completed using an implementation language. Normally, the ontology conceptualization output serves as guidance for the ontology implementation; however, ontology implementation is usually done from scratch using ontology editors. The goal of this work is to consider ontology conceptualizations as first-order artifacts in ontology development in order to boost the ontology implementation activity. For doing so we present Chowlk, a framework to transform digital machine-processable ontology conceptualization diagrams into OWL. Domain experts and ontologists benefit from this approach in several ways: 1) reduce time generating the first versions of the OWL file that can be invested on 2) focusing on the conceptualization diagrams that can be used both for 3) improving communication between ontology users and developers and 4) be reused during the ontology documentation stage.

Keywords: Ontology engineering · ontology conceptualization · OWL

1 Introduction

Everyday more and more applications are being built on top of or in combination with semantic technologies. Ontologies play a crucial role in this development as they allow the representation of knowledge in a formal and structured way, being the OWL [4] language the default choice for their implementation because of its high level of expressiveness, reasoning capabilities and the fact that it has been designed for the web environment.

One of the first and most important steps in ontology development is the conceptualization one, during which the ontology development team defines a set of

* This work has been supported by the BIMERR funded from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 820621.

concepts and properties to represent the knowledge of a specific domain. Often, this conceptualization is materialized in a diagram that displays the relationships, attributes and axioms of the different concepts of an ontology. From this model, the ontology implementation is carried out normally using an ontology editor, such as Protégé [11], realizing the model into OWL code.

However, in this process the diagram is in most of the cases only used as a guideline to implement the ontology, translating the ontological elements and constructs to a formal syntax, being this process mostly manual and error-prone. Some tools have been proposed in the last years that allow the graphical creation or modification of ontologies following their respective visual notations [16, 2].

In our case, rather than building a graphical ontology editor, the effort is driven towards the goal of allowing a smoother transition from the conceptualization activity to a first version of the actual implementation by taking the conceptualization output as a first order artifact in ontology development projects. For doing so, the Chowlk framework has been designed. The framework, shown in Figure 1, consists of: 1) an UML-based visual notation; 2) a pair of diagrams.net templates implementing the visual notation; and 3) a converter from diagrams.net XML diagrams to OWL. It should be clarified that the resource presented in this paper is the converter that will be detailed in Section 3). However, for a better understanding of the converter, the visual notation is briefly presented in Section 2.

It should be clear at this stage that our goal is to fill the gap between the conceptualization and implementation of ontologies which is still a manual process, and as every manual procedure, it can be prone to errors. Even though, it is true that users can create ontologies directly in specialized editors such as Protégé [10] and avoid the creation of a diagram, our focus is on ontology users who follow developments where the conceptualization is the corner stone of the development process, and want to take full advantage of the effort made in the conceptualization step, for example to communicate and verify the model with users or clients as well as for documenting the ontology to publish or share it.

The validation of the Chowlk converter is described in Section 4 while a comparison with existing approaches is presented in Section 5. Future lines of work to evolve and improve the present work are proposed in Section 6.

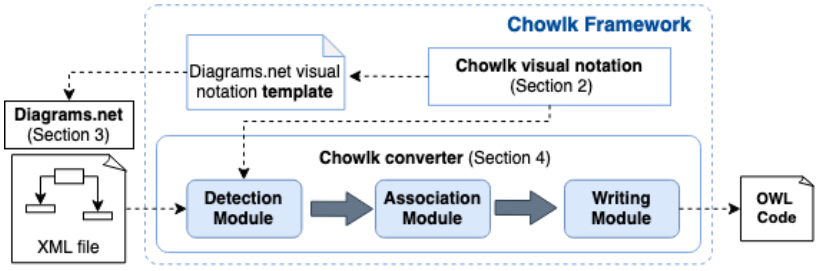

Fig. 1: Chowlk framework.

2 Visual notation

The converter presented in this paper is based on the Chowlk visual notation that extends the UML_Ont profile [7]. It should be mentioned that while the original UML_Ont profile utilizes custom stereotypes and dependencies to cover OWL 1 constructs, the Chowlk notation binds the stereotypes used in the profile to OWL and RDF(S) constructs. Also, the visual notation used in this work proposes compact alternatives for representing property characteristics and axioms.

Due to the fact that the notation is considered an input for the converter instead of part of the resource presented in this paper, and for space matters, in this section only the main characteristics of the notation are included. While the notation has been partially published in [6], a more complete and updated version, including examples and alternative notation elements for those presented in this paper, is provided in the notation website.[1]

Figure 2 provides an overview of the notation of the main OWL elements. Named **classes** are represented by labelled boxes. Unlabelled boxes or circles are used to represent anonymous classes and class intersections, unions, equivalences and disjoints. **Object properties** are represented by labelled arrows and **datatype properties** by labelled boxes attached to class boxes. Note that both types of properties can be represented by diamonds, notation needed in some cases, for example to represent equivalences or property hierarchies for datatype properties. For object properties, the relations between them (subproperty of, inverse or equivalent) can be represented both by arrows linking either the arrows representing the properties or the diamonds representing them.

Property characteristics (functional, inverse functional, transitive and symmetric) can be indicated before the property name or stating the characteristic construct in the diamond. **Class constraints** are represented between classes including the operator (universal, existential o cardinality) before the property over which the constraint is stated for subclass constraints. For equivalent class constraints or constraints in domains or ranges, unlabelled boxes are used in combination with the equivalent or domain/range indicator.

The Chowlk visual notation also allows to declare namespaces, for example to link entities from different ontology modules within a network or to indicate the reuse of other ontology elements. Finally, the notation includes a metadata block used not only for documenting the diagram but for ontology metadata generation during the conversion phase. The metadata is stated in a printed-document alike shape and makes use of the prefixes defined in the namespaces building block. Examples of namespaces and metadata blocks are shown in Figure 3.

[1] `https://chowlk.linkeddata.es/notation.html`

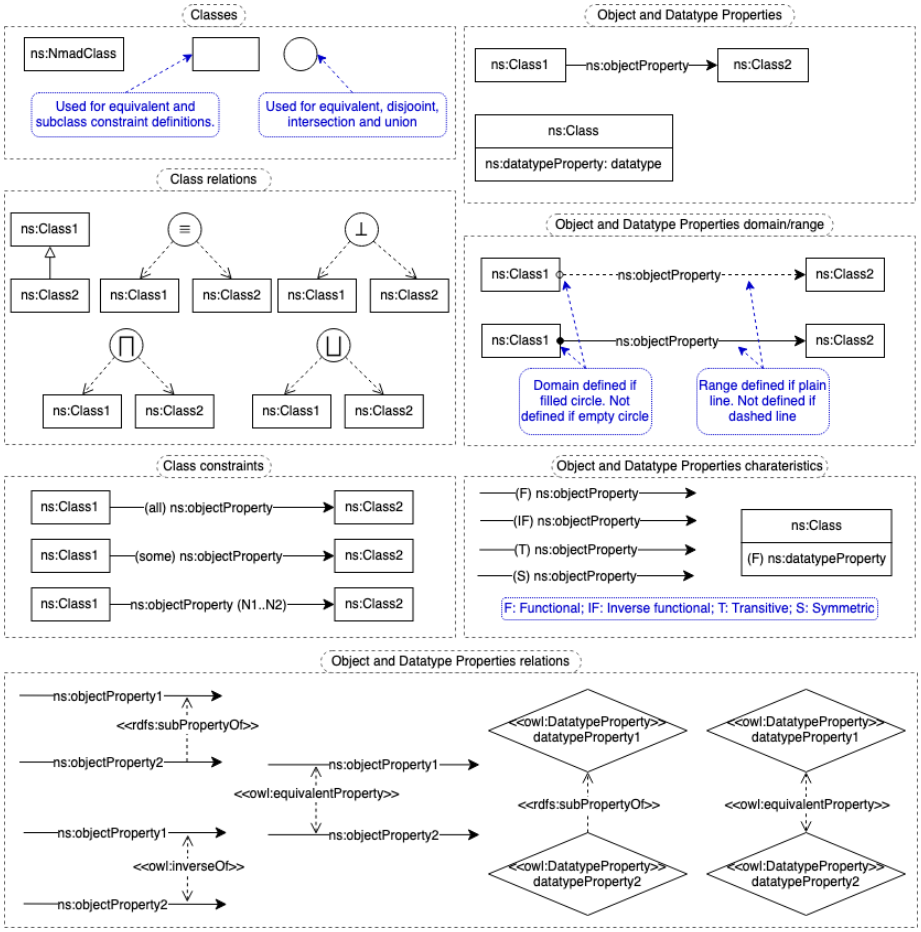

Fig. 2: Chowlk visual notation summary.

Figure 3 shows an excerpt of the BIMERR building ontology.[2] The figure shows basic elements such as classes, class hierarchies, object properties and datatype properties. Also, some more complex statements are represented as universal restrictions, for example between `building:Building` and `building:Storey` over the object property `bot:hasStorey`. Class cardinality constrains are shown for several datatype properties, for example the cardinality of the attribute `building:ifcIdentifier` for `building:Storey` is exactly 1.

Even though the presented visual notation is in some cases a one to one representation of the formalisms of the OWL language, it gives the freedom to

[2] `http://bimerr.iot.linkeddata.es/def/building#`

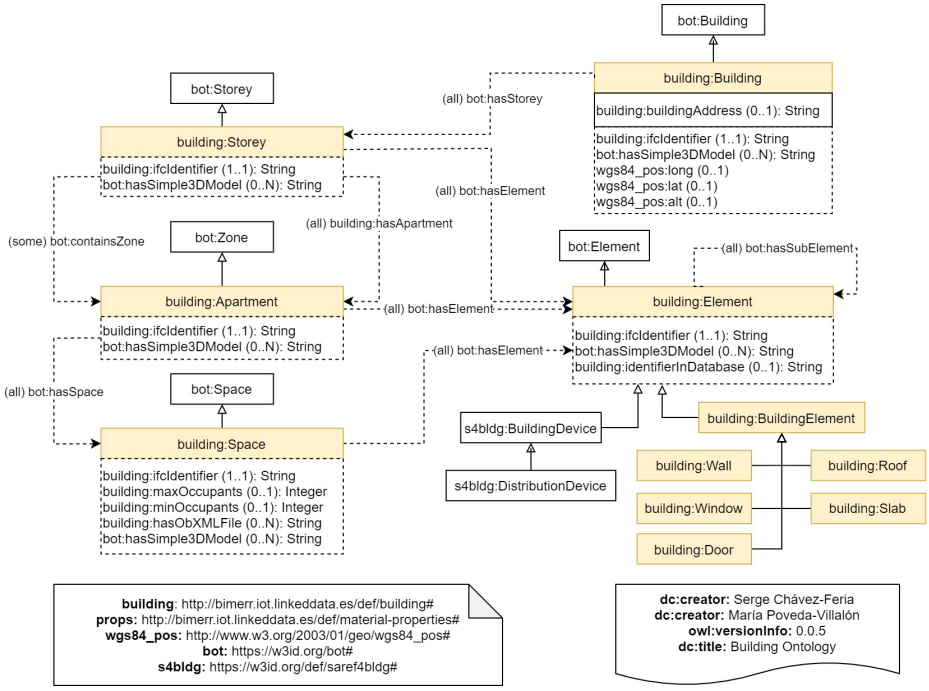

Fig. 3: Conceptualization example for an ontology using Chowlk.

develop lighter models. These less complex models can contain just boxes and plains arrows, without indicating restrictions or more complicated constructs, almost like a conceptual map, which is easier to develop and understand by non ontology experts. For this reason, the notation allows for different alternatives for representing most of the OWL constructs and the framework includes two flavours of the notation that are implemented in two different templates.

The first template is a complete version containing all the building blocks described in the visual notation. This version was designed for ontology engineers who are knowledgeable about OWL. The second template is a lightweight version containing just a subset of the blocks, such as rectangles, arrows, and Boolean operators without more complicated constructs like restrictions. This second version was intended for users which are not familiar with OWL. Users can upload the templates and start making their conceptualizations by dragging and dropping the building blocks of the template into the diagramming layout of diagrams.net. This procedure reduces the entry barrier to start using the notation and avoids visual syntax errors when constructing the conceptualizations by providing already predefined combinations of the blocks in order to represent the OWL constructs.

3 The Chowlk converter

Chowlk is a web application that takes as input an ontology conceptualization created with diagrams.net and generates the OWL implementation. The conceptualization is made following the Chowlk visual notation described in section 2. The web application is available through its URL,[3] and through its API.[4] The source code is shared in a GitHub repository[5] under the Apache 2.0 license. The software has a canonical citation using the DOI[6] provided by its Zenodo entry.[7]

Figure 1 shows the modules in which the system is decomposed, namely: the detection module, the association module, and the writing module. The input to the system is a diagram representing the conceptual model of an ontology in XML format. After the conversion process, the tool outputs the ontology implementation in Turtle that can be downloaded to continue with the remaining ontology engineering process.

It is worth mentioning that even though the workflow shown in Figure 1 has been defined within the Chowlk Framework, it can be reused to develop converters for other visual notations, just by adapting the detection stage which is in charge of detecting the underlying syntax of the blocks. Section 3.1 exposes the reasons to build the converter based on diagrams.net and the rest of the sub-sections cover in detail each of the modules in the transformation pipeline.

3.1 Selecting a diagramming tool

As already mentioned, the goal is not to produce a graphical ontology editor but to take advantages of conceptualizations that can be developed with a variety of diagramming tools. Indeed, the Chowlk notation is independent of the tool used to draw the diagram shapes or symbols and provides alternatives in case the diagramming tool does not support some symbols as the existential or universal operators. However, in order to use the converter to generate the OWL code from the conceptualization, diagrams.net should be used as the diagramming tool. The main reasons for choosing diagrams.net are:

1. It is flexible enough in terms of features and drawing options, so it allows to implement all the elements of the visual notation.
2. It supports synchronous collaborative diagram edition. In this sense, ontologists and domains experts, or other roles involved in ontology conceptualization, could be visualizing and/or editing the diagrams at the same time.
3. It is able to export diagrams in a structured format, such as an XML file. Figure 4 shows an example of the nested structure generated, where on the left side we have a very simple ontology excerpt composed by two classes and one object property, and on the right side the XML counterpart. Additionally,

[3] `https://chowlk.linkeddata.es`
[4] `https://chowlk.linkeddata.es/api`
[5] `https://github.com/oeg-upm/Chowlk`
[6] `https://doi.org/10.5281/zenodo.4312930`
[7] `https://zenodo.org/record/4312930#.X9yNt9hKiUk`

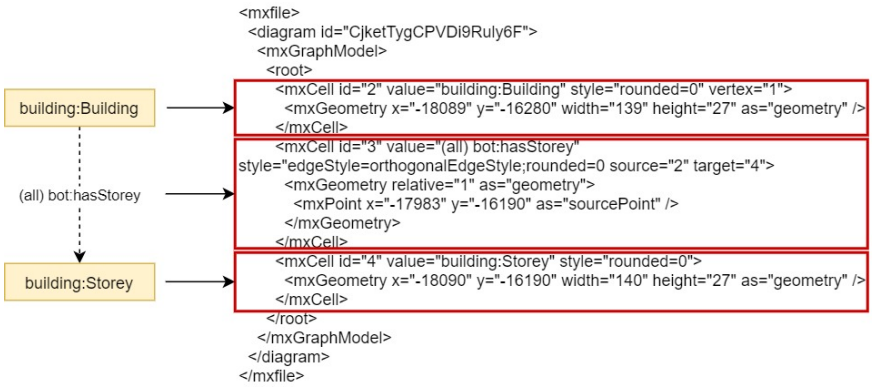

Fig. 4: Sample XML output of diagrams.net

each child element has a sequence of attributes that helps in the identification of each building block. Table 1 describes the fields used to describe the children elements. Some attributes apply to all the building blocks of the diagram such as the "id" field, while others only apply to specific shapes like the arrow blocks that should include a "target" and a "source" field.

4. It is a web-based open source platform. This feature lowers the barrier for its adoption, avoiding the process of having to download the software, install it and run it locally. The open source characteristic also opens the door to increase its functionalities whether through the extension of its source code or by means of plugins.

Table 1: XML diagrams.net data structure.

Block attribute	**Block type**	**Definition**
id	Classes, object properties, datatype properties.	Unique identifier of the block in the diagram.
value	Classes, object properties, datatype properties.	Text content assigned to the block. Used to represent the URIs of the elements of the ontology.
style	Classes, object properties,datatype properties.	Allows to give style to the blocks and make a differentiation between the elements of the ontology.
source	Object properties	Points to the block id that is connected to the source side of an arrow.
target	Object properties	Points to the block id that is connected to the target side of an arrow.

3.2 The detection module

Once the diagram is uploaded to the system, the transformation process triggers. The first step in the conversion procedure is performed by the detection module, where all the building blocks of the diagram are found.

The detection of ontology elements is performed for all the building blocks represented in the diagram that follow the Chowlk visual notation, discarding any shape that does not correspond to the notation ones. This detection is done by analyzing the attributes of the XML data structure mentioned in section 3.1. Specifically, the module searches for information in the "style" attribute of children elements to derive the type of shape it is dealing with. For instance, if the "style" attribute contains the keyword "edge", the module can interpret that the shape being analyzed is of type "arrow" that could represent an object property in the OWL language. Each element identified in the diagram populates a predefined data structure, where the fields change according to the type of ontology element. For example, in the case of an object property the data structure will store information regarding its prefix, the URI, if it is functional or symmetric, etc. These data structures facilitate later the querying of elements and searching for information during the subsequent stages.

In most of the cases the type of visual blocks used in the specification has a unique mapping to the OWL construct, like the namespace block. However, there are other situations in which the same type of building block is used to represent more than one OWL element. This is the case of concepts and attributes, where both use the rectangle definition, and it is needed to identify the geometry disposition of the blocks in the layout in order to disambiguate their meaning. In this particular example, if the algorithm detects a rectangle, then it searches for other rectangles above it in a close neighborhood. If they exist, the rectangle we are analyzing represents datatype properties, otherwise it represents a class.

In the current version of the converter, the source and target of arrows in a diagram must be anchored to other building blocks in order to identify the relationship. This characteristic in combination with the restriction that diagrams.net does not allow connections between arrows, impedes the creation of relationships between properties. This means that in order to represent `rdfs:subPropertyOf` relations between two object properties, the diamond option specified in the visual notation to represent object properties should be used. Diamond shapes can also be used as an optional alternative to state several other characteristics of the properties such as symmetry, functionality, range, domain, etc. If an object property is represented as an arrow in one part of the diagram and additional information is provided using the diamond shape, the definition of the property is generated by combining the information represented in both shapes.

Additionally, the converter is able to identify ontology metadata, and the namespaces and prefixes being used in the model, thanks to specific blocks dedicated to this type of information. Labels to each ontology element are added during the detection process.

Finally, the detector module also identifies any deviation from the visual notation and returns a report diagram indicating in which part of the diagram

the ontology engineer is not following the correct syntax. For instance, if the ontology engineer attempts to instantiate a property without a prefix, or a prefix was detected in the ontology elements that was not included in the namespace declaration block, the module detects those errors and outputs: the id of the block involved, the label if available, and a generic explanation of the error. This example can be seen in Figure 5.

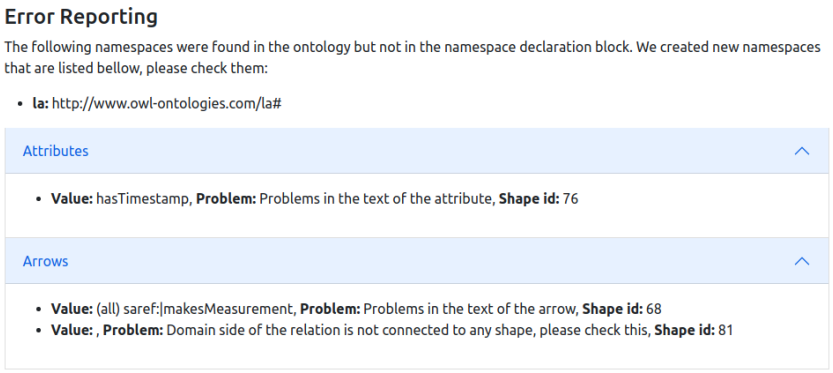

Fig. 5: Example of error report

3.3 The association module

The association module performs the connection between the classes, and the object and datatype properties instantiated in the diagram.

The correspondences are established following different procedures. In the case of associations between classes and object properties, the module checks if the identifier of the building block representing a class and the identifier in the "source" field of an object property is the same. For the case of association between classes and datatype properties, the module analyzes the location of the blocks representing them. If the datatype property block is below and close enough to a class block, it means those attributes are intended to be used with that class.

In a second step, the module analyzes if the object and datatype properties have a restriction with the class at hand. The module specifically searches for the following notation in the text of the properties: (some), (all), or (N1..N2), which indicates existential, universal, and cardinally restrictions respectively.

If the restrictions exist, the module maintains the connections previously created between the classes and the properties. Otherwise, the associations are eliminated because the properties have been diagrammed in that way only to give the potential user of the ontology an idea of how the properties are planned

to be used. However, there is no formal restriction that states that it can only be used with that specific class.

Finally, the output of this module is an array that contains the concepts, objects properties and datatype properties associated through restrictions. This will facilitate the serialization of the restrictions in the final Turtle file.

3.4 The writing module

The writing module takes all the ontological elements detected in the previous steps and writes them one by one in an RDF file. The process starts by taking as basis a template that already incorporates common namespaces (rdfs, owl, rdf, xml, dcterms and vann) and their prefixes to avoid the user to indicate them.

This default list is complemented with the namespaces and prefixes found in the metadata block. If there exist some prefixes detected on the elements of the ontology (e.g., concepts, relations, attributes) that were not declared on the namespace block, new namespaces invented by the tool are created automatically. Afterwards, the module writes high level information about the ontology declared in the ontology metadata block, such as title, authors, imports, etc. This information will be written in the `owl:Ontology` header.

Next, the module writes the definition of the object and datatype properties. The following information is included for both types of properties: English labels (which are automatically extracted from the URI), if they are functional, the domain, the range, and if they are sub-property or equivalent to another one. In the case of object properties additional characteristics are included if they are stated during the conceptualization: symmetric, transitive, inverse functional, or inverse of another property.

The process is similar for writing the classes, with the difference that in this case the module uses as input the output from the associations module. The writing module needs such data structure to know the type of restrictions that applies over each class with respect to the object and datatype properties. Relationships of the type `owl:disjointWith` and `owl:equivalentClass` with other concepts are also included. Instances and general axioms such as multiple disjoints between several classes are also added.

Once the writing process is finished, the converter provides the ontology in the Turtle format, and the file can be finally downloaded from the GUI of the web application.

Additionally, if the visual notation was not followed properly, a report is provided in the GUI of the application listing all the errors found during the parsing process. The report includes the id of the block containing the error, the label of the block for a rapid inspection in the diagram, and a generic explanation of the problem.

3.5 Current limitations

The diagrams.net tool is a general purpose diagramming software, not specific for the development of ontologies, so the user has to be very careful when con-

structing the conceptualizations in order to avoid deviations from the visual notation being used. Even though the current version of the model can generate reports about the errors detected in model and the user can make the appropriate changes in the diagram, the process of identifying the blocks in the diagram manually can be very complex for very large conceptualizations.

Also, because diagrams.net does not allow to anchor the extremes of arrows to other arrows, the system cannot detect the `rdfs:subpropertyOf`, `owl:inverseOf` and `owl:equivalentOf` relationships. For that we need to use the diamond options of the Chowlk visual notation. For instance, to express that the object property "hasSpace" has a relationship of type `owl:inverseOf` with the property "isSpaceOf" we need to use the diamond shapes for the converter to be able to detect this kind of construct.

4 Validation

In the following section we provide a series of examples that prove the usage of the tool. Additionally, we verified the correctness of the results obtained by the converter by transforming the visual OWL constructs listed in the visual notation. Because of the simplicity of the tool, we do not include user experience evaluation in this first version, but it is something that we plan to do for the next iterations.

4.1 Adoption and use

The service has been adopted in different projects from several institutions. For instance, Chowlk is being used as part of the ontology development pipeline in different H2020 European projects, such as BIMERR,[8] and COGITO,[9] within the research lab developing Chowlk, but also by external teams, for example in the BIM4EEB[10] and CosWot ANR projects.[11]

Additionally, the system is being used to support the development of ontologies in different domains such as agriculture,[12] public transport [14], time,[13] ethics,[14] material science,[15] and ICT infraestructure [3]. Furthermore, some ontologies developed by international communities such as the W3C[16] has also being implemented using Chowlk, such as the WoT discovery ontology.[17]

[8] `https://bimerr.iot.linkeddata.es/`
[9] `https://cogito.iot.linkeddata.es/`
[10] `https://digitalconstruction.github.io/v/0.5/index.html`
[11] `https://coswot.gitlab.io/`
[12] `http://www.elzeard.co/ontologies/c3po/plant#`
[13] `https://github.com/mnavasloro/ft3/blob/04c65c2b2ed2bd57f9ac6cfb32b7f4ebfda1f4c4/ft3.owl`
[14] `https://krnlet.github.io/#`
[15] `https://github.com/Mat-O-Lab/MSEO`
[16] `https://www.w3.org/`
[17] `https://github.com/w3c/wot-discovery/blob/24b2141e8e0cb74abd24cead0b4bbffb672e24c6/context/discovery-ontology.ttl`

Finally, the usage of the tool can be demonstrated by the issues, pull requests and forks made to the Github of the project. This demonstrate that Chowlk is being used not only to develop ontologies, but also being integrated in other ontology development softwares.[18]

4.2 Validation Tests

The service has been tested against a set of 49 diagrams, where all the results obtained were valid ontologies. Each diagram contains a set of building blocks representing the OWL constructs defined in the Chowlk visual notation. The diagrams constructed with their corresponding OWL ontologies are available in the GitHub repository of the project[19] for its verification. Figure 6 shows an example of an input diagram and the ontology generated by the converter.

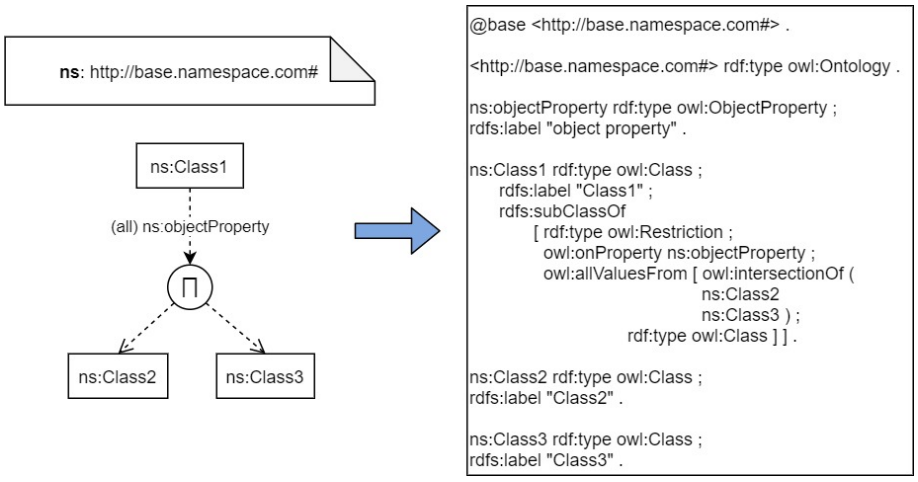

Fig. 6: Test Conversion Example.

As it was mentioned in section 4.4, since it is not possible for the system to detect the `owl:inverseOf`, `owl:equivalentProperty` and `rdfs:subPropertyOf` axioms between object properties when they are represented using arrows, we tested those axioms representing the object properties with the diamond shapes.

[18] `https://gitlab.com/kupferdigital/ontoflow`
[19] `https://github.com/oeg-upm/Chowlk/tree/webservice/tests/inputs`

5 Related work

Several approaches have been proposed in the recent years with respect to visual ontology edition tools. The work developed in [5] presents a good review of the state of the art regarding tools with edition and visualization capabilities. From the spectrum of tools analyzed, only six include the visual edition of ontologies as a feature. It is important to remark that the following review only considers tools that are free for its usage.

On the one hand, there is a set of applications that are implemented as a web service. WebVOWL [16] is an application that has as a principal feature the visualization of OWL ontologies, which are displayed following the VOWL visual notation that has a graph representation. Among other capabilities it allows the customization of the visualization and the modification of the ontology by directly manipulating the elements of the graph. On the same line, OWLGrEd [2] is a framework that offers visualization capabilities following an UML based notation. The tool allows the visual edition of ontologies but only in its desktop version. Even though, the graphical edition of the ontologies is possible in both applications, neither of them allow collaborative work. Graffoo[20] is an open source tool that can be used to represent OWL ontologies as easy-to-understand diagrams. Originally, it was developed as a standard library for the yEd diagram editor including a set of pallets to create ontology conceptualizations and afterwards using the Ditto[21] web service to generate the OWL implementation. Recently, a library for diagrams.net was created to develop ontology conceptualization using the Graffoo visual notations; however, in this case the conversion service is not available.

On the other hand, there is a set of applications that require the local installation of software. The following tools are described based on their publications because there is no evidence of their availability. CMap Ontology Editor [8] is a set of tools that allows the creation and visualization of ontologies as conceptual maps [12], which are general artifacts that serve for the representation of any kind of knowledge. Ontotrack [10] is a standalone application that supports graph based and hierarchical representations of ontologies. It includes instant reasoning capabilities that provide instant feedback about the modeling decisions made by the user. Triple20 [15] is a manipulation and visualization tool developed using Prolog. Some of its characteristics include the representation of the ontology following a graph based and hierarchical view and the ability to handle large ontologies because all the data is stored in RAM memory. Finally, GrOWL [9] is a standalone Java application that, apart from the basic visualization and edition features, also makes use of shape, color and shade to encode properties in the nodes of the graph.

One common characteristic among the tools described previously is that all require the learning of a new development environment, the local installation of the software, or do not allow collaborative work.

[20] `https://essepuntato.it/graffoo/`
[21] `https://essepuntato.it/ditto/`

The Chowlk converter eliminates the need for software installation by leveraging on existing popular diagramming tools that already provide collaborative edition features to generate the ontology conceptualizations. It could also be integrated with third party software. In addition, the proposed framework and converter are based on UML notation as it is commonly used in software engineering, and it is familiar to software engineers.

6 Conclusions and future work

This paper presents a system, Chowlk, to ease the ontology development process by leveraging the conceptualization activity outputs in order to transform the obtained diagrams into OWL code. Chowlk is implemented as a web application that allows the uploading of the diagram as an XML file and outputs the ontology in RDF/XML and Turtle formats speeding up the ontology developments.

The system was tested using a unit-test procedure with all the OWL constructs defined by the Chowlk visual notation, and also using it to generate the ontologies of the BIMERR ontology network.

We will explore the support for other visual notations for a broader adoption and testing of the tool. Additionally, further research should be carried out in order to support the updating of the conceptualizations. That is, how for a given ontology created by Chowlk, and then modified by an editor (Protégé), the changes can be appropriately represented in the diagram. The support for other standard formats such as SVG is also something to be explored in the next version. This could allow the converter to be independent of the diagramming tool to be used.

Finally, the sustainability plan for the system includes its continue use and evolution as part of current and future research projects and as part of the group ontology engineering tools suite roadmap. Some foreseen interactions exist between Chowlk and OnToology [1], by integrating the XML file as a resource in GitHub repositories from where OnToology can trigger Chowlk to generate the OWL code; and incorporating the pitfalls detection from OOPS! [13] within the conceptualization phase by the diagrams.net Chowlk plugin.

References

1. Alobaid, A., Garijo, D., Poveda-Villalón, M., Santana-Perez, I., Fernández-Izquierdo, A., Corcho, O.: Automating ontology engineering support activities with ontoology. Journal of Web Semantics (2018)
2. Barzdins, J., Barzdins, G., Cerans, K., Liepins, R., Sprogis, A.: UML style graphical notation and editor for OWL 2. In: Perspectives in Business Informatics Research - 9th International Conference, BIR 2010, Rostock Germany, September 29-October 1, 2010. Proceedings. Lecture Notes in Business Information Processing, vol. 64, pp. 102–114. Springer (2010). https://doi.org/10.1007/978-3-642-16101-8_9
3. Corcho, O., Chaves-Fraga, D., Toledo, J., Arenas-Guerrero, J., Badenes-Olmedo, C., Wang, M., Peng, H., Burret, N., Mora, J., Zhang, P.: A high-level ontology network for ict infrastructures. In The Semantic Web-ISWC (sep 2021)

4. Dean, M., Schreiber, A., Bechofer, S., van Harmelen, F., Hendler, J., Horrocks, I., MacGuinness, D., Patel-Schneider, P., Stein, L.: Owl web ontology language reference. w3c recommendation, world wide web consortium (2004), latest version: http://www.w3.org/TR/owl-ref/
5. Dudás, M., Lohmann, S., Svátek, V., Pavlov, D.: Ontology visualization methods and tools: a survey of the state of the art. The Knowledge Engineering Review p. 33 (2018)
6. Garijo, D., Poveda-Villalón, M.: Best practices for implementing FAIR vocabularies and ontologies on the Web (Nov 2020). https://doi.org/10.3233/SSW200034
7. Haase, P., Brockmans, S., Palma, R., Euzenat, J., d'Aquin, M.: D1.1.2 updated version of the networked ontology model. Tech. rep., Universität Karlsruhe (2009), NeOn Project. http://www. neon-project. org
8. Hayes, P., Eskridge, T., Saavedra, R., Reichherzer, T., Mehrotra, M., Bobrovnikoff, D.: Collaborative knowledge capture in ontologies. In Proceedings of the 3rd International Conference on Knowledge Capture (2005)
9. Krivov, S., Williams, R., Villa, F.: Ontotrack: a semantic approach for ontology authoring. GrOWL: A tool for visualization and editing of OWL ontologies (2007)
10. Liebig, T., Noppens, O.: Ontotrack: a semantic approach for ontology authoring. Web Semantics: Science, Services and Agents on the World Wide Web (2005)
11. Musen, M.A.: The protégé project: a look back and a look forward. AI Matters **1**(4), 4–12 (2015). https://doi.org/10.1145/2757001.2757003, `https://doi.org/10.1145/2757001.2757003`
12. Novak, J., Cañas, A.: The theory underlying concept maps and how to construct and use them. Tech. rep. (2006), technical Report IHMC CmapTools 2006-01, Institute for Human and Machine Cognition (IHMC)
13. Poveda-Villalón, M., Gómez-Pérez, A., Suárez-Figueroa, M.C.: Oops! (ontology pitfall scanner!): An on-line tool for ontology evaluation. Int. J. Semantic Web Inf. Syst. **10**(2), 7–34 (2014). https://doi.org/10.4018/ijswis.2014040102, `https://doi.org/10.4018/ijswis.2014040102`
14. Ruckhaus, E., Anton-Bravo, A., Scrocca, M., Corcho, O.: Applying the lot methodology to a public bus transport ontology aligned with transmodel: Challenges and results (nov 2021). https://doi.org/10.3233/SW-210451, `https://content.iospress.com/articles/semantic-web/sw210451`
15. Wielemaker, J., Schreiber, G., Wielinga, B.: Using triples for implementation: the triple20 ontology manipulation tool. In The Semantic Web-ISWC (2015)
16. Wiens, V., Lohmann, S., Auer, S.: Webvowl editor: Device-independent visual ontology modeling. International Semantic Web Conference (2018)