A Visual Rule Generation Tool for SWRL*

Alia El Bolock[1,2], Ibrahim Mohamed[1], Cornelia Herbert[2], and Slim Abdennadher[1]

[1] German University in Cairo, Cairo, Egypt
{alia.elbolock,ibrahim.mohamed,slim.abdennadher}@guc.edu.eg
[2] Ulm University, Ulm, Germany
cornelia.herbert@uni-ulm.de

Abstract. The number of applications relying on representing knowledge through ontologies and interactions through rules has grown significantly in the past years. The knowledge acquisition needed for ontology engineering requires input from domain experts, which can lack computing and knowledge modeling skills. In this paper, we present a generic visual programming tool for generating SWRL rules for OWL ontologies. The developed web application allows ontology developers and domain experts alike to easily create SWRL rules. By interacting with a simple graphical user interface, the users can specify the antecedents and consequents of SWRL rules by combining ontology entities and applying restrictions on them. Different control features, such as implicit variable handling, are implemented, which only enables rule creation options that maintain the correct SWRL syntax and maintain correct semantics. Consistency is ensured by allowing the users to review the generated rules and their effects before actually applying them to the ontology.

Keywords: SWRL · OWL · Ontology · Rule Generation · Visual · Web Application · Character Computing

1 Introduction

Ontologies are commonly used to represent knowledge in many domains, e.g. bio-medicine, psychology, and human computer interaction. Due to the complexity and size of the knowledge to be represented, it is often vital to include domain experts while authoring ontologies [8]. However, most domain experts lack knowledge modeling skills and find it hard to follow the logical notations of the semantic web languages used to develop ontologies, such as the Web Ontology Language (OWL) [23]. Accordingly, some kind of mediation between knowledge engineers and domain experts was needed. While the architecture of the ontologies themselves plays a vital role in capturing the essence of a specific domain, defining rules that detail how these concepts interact together is one main advantage of using ontologies. Some approaches even propose allowing including rules to ontology knowledge bases and reasoning on them [6].

* Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Rule-based languages, such as the Semantic Web Rule Language (SWRL) [22] with its high expressivity, became an integral part in most ontologies [26], due to the added reasoning capabilities and flexibility. However, SWRL, like OWL, requires expertise in logic-based knowledge representation and the syntax and semantics of the language. This might be relatively easy for ontology engineers and computer scientists but not for the domain experts that are usually required to produce and define the rules. Ontologies are also constantly revised and adapted based on newly arising information and data, which also requires changing the belonging rules. Accordingly, one of the main motivation behind this work is the need for solutions that simplify the development of ontologies and defining rules on them. Having a tool that enables non-ontology developers to add rules to knowledge bases and ontologies in an intuitive format is thus integral to simplifying the development and extension of ontologies and their rules. Such a tool is especially relevant when reasoning and defining rules over large and complex ontologies. There is large number of approaches aiming at improving the mediation process between knowledge engineers and domain experts and in turn the process of ontology engineering. One of them is by parsing and translating informal texts of experts to generate rules by using Natural Language Processing and other Machine Learning algorithms [5]. The disadvantage of this approach is that it is prone to imprecision and ambiguity [20]. Besides, the systems that apply these solutions are so complex and hard to develop and maintain. A lot of previous work promoted the idea of controlled natural language, this idea was used to create the so-called natural language interfaces (NLIs). These interfaces are developed to guide domain experts through the rule authoring process. The FluentEditor [31] and AceWiki [24] are two such tools. One major drawback of NLIs is that they show good results mostly when they are customized into a specific domain [7]. The efficiency and easy of use decreases, the more generic the tools become. Another drawback of NLIs is their adaptivity to new domains [3]. It is still possible to provide a more intuitive and user-friendly approach to increase the usability, especially for the novice users who come from disciplines lacking the needed skills. The approach we follow in this work, is the use of visual language interfaces to create an easier and more user-friendly experience for domain experts. Unlike NLIs, it is easy to build generic visual language interfaces that are independent of associated ontologies from different domains. Some work has been done in this area but often toward the visualization and paraphrasing of existing rules e.g., [19]. One tool used for visualizing rule creation process is [30]. The tool was the first to fully visualize SWRL rules creation process. The proposed platform fully covers all SWRL constructs and is designed in a manner that targets ontology experts as it requires understanding of the SWRL. Other tools, such as [25], restrict SWRL a bit to simplify the process but at the cost of limiting the users. In [2], a graph-based model to represent the relations between SWRL atoms is presented. It does not consider the order of atoms within a rule, which can complicate the modeling whenever the number of rule atoms increases. While these solutions might be a bit easier for non-experts to use, it still shows a lot for the lower-level implementation details and gives users a lot of control and decisions that can lead to mistakes in the rule creation. Thus, most existing solutions do not specifically target domain experts that have no ontology knowledge, while providing the complete expressivity of SWRL.

In this paper, we present a generic visual programming tool for generating SWRL rules for OWL ontologies. The developed web application allows ontology developers and domain experts alike to easily create SWRL rules to be added to specific ontologies, see their effect on the ontology, and decide whether or not to apply the rules. By interacting with a simple graphical user interface, the users can easily specify the antecedent and consequent of a SWRL rule by combining ontology entities. By relying on visual interaction techniques, the developed tool can be easily used by individuals without programming background. We add constraints to the rule creation process and guide the user through the process step by step to ensure the integrity and validity of the generated rule and increase the usability. This is further enabled by implementing different control features, such as implicit variable handling, which do not exist in other similar tools to date. The rule generation tool only enables choices that lead to syntactically correct rule, thus avoiding any syntax errors that could arise from the user's side. Correct semantics are ensured by allowing the users to review the generated rules and their effects before actually applying them to the ontology.

This work is part of a bigger project within the field of Character Computing [4, 15, 9, 16], where a group of computer scientists and psychologists are developing an ontology-based model for representing human character [10] and its interactions with behavior in different situations. This model and its ontology, CCOnto [14], is to be used by computer scientists, as well as psychologists, to develop semantic web solutions and applications that improve the user experience by sensing, adapting to, and guiding user interaction [11] e.g., [17, 13, 12]. This project heavily relies on developing solutions that ease the cooperation between computer scientists and psychologists, such as the application proposed in this paper and the work proposed in [1].

The rest of the paper is structured as follows. The application design and features are presented in Section 2, while the system architecture and implementation details are explained in Section 3. Finally, Section 4 concludes the paper as well as discusses the future work.

2 Application Design and Features

The design of the proposed rule generation application and its different components are shown in Fig. 1. The application tool is a generic tool for visually generating SWRL rules. SWRL rules are in the form $A \Rightarrow C$, where A is the antecedent (body) and C is the consequent (head) of the rule. This is interpreted as, if the antecedent A holds, then the consequent C must also hold i.e. it represents the classical implication. Both the antecedent and the consequent are conjunctions of atoms. Atoms are limited to unary or binary predicates i.e., class and property axioms, respectively. Variables, representing individuals or data, are allowed in the antecedent and the consequent. One applied safety restric-

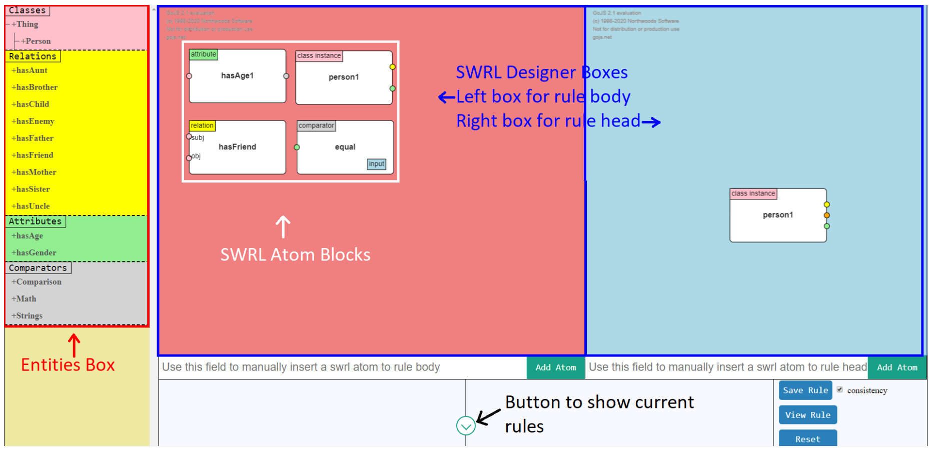


Fig. 1: Web application view and components

tion is that a variable can only be placed in the rule consequent if it appears in the rule antecedent. SWRL supports most of the OWL axioms such as classes, sub-classes, equivalent and disjoint classes, disjoint data types, data-values properties, and facts. These can be used to describe variables within a SWRL rule. SWRL also supports built-ins for performing and evaluating various operations and expressions e.g., math and string operations.

A SWRL rule is used to represent a piece of information. A simple example of such a rule would be to describe the concept of being an uncle using a basic ontology representing family relations. This is done by asserting the conjunction of the `hasParent` and `hasBrother` properties in the antecedent of the rule to imply the `hasUncle` property in the consequent of the rule.

```
Person(?person1) ^ Person(?person2) ^ Person(?person3)
^ DifferentFrom(?person2, ?person1) ^ DifferentFrom(?person3,
?person1) ^ DifferentFrom(?person3, ?person2) ^
hasFather(?person1, ?person2) ^ hasBrother(?person2, ?person3)
-> hasUncle(?person1, ?person3)
```

The rule states that if we have three different persons where the first is the father of the second and the second is the brother of the third then the first is the uncle of the third. Although the rule is trying to explain a very simple relationship, it can be quite hard to understand for someone without a background in logic. In this section, we will explain the different features implemented into our SWRL rule generation tool, which is already deployed[3]. The various features included in the tool can be divided into six main categories. Table 1 gives an overview of the main feature categories as well as some example features. These features alongside relying on visualization make authoring SWRL rules more intuitive even for novice users.

[3] `http://rules.us-east-1.elasticbeanstalk.com/`

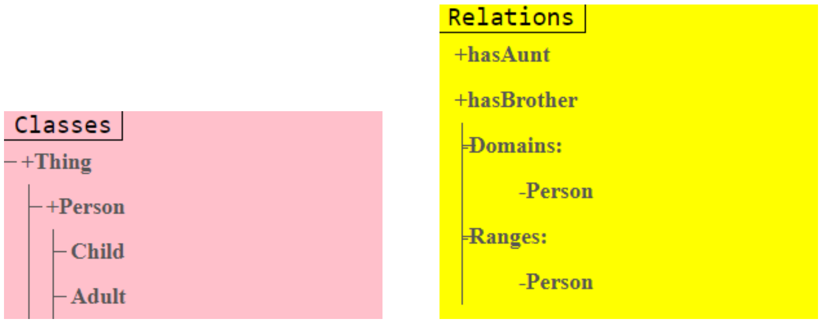
Table 1: Main categories of features included in the SWRL rule generation tool and belonging features. Only the main features are included in this table. Entities refer to all OWL entities usually browsable by Protégé e.g., classes, properties and individuals.

Feature Category	**Features**
Entity Exploration	viewing all entities with their hierarchy
Rule Management	creating rules, viewing and deleting existing rules
Abstraction	removing all unnecessary details, e.g. IRIs, entities renaming automatic variable handling, providing only upper-level entity details
Flexibility	viewing current rule, adding atoms to rules in SWRL notation
Constraints	entity linking validations (directions and types), cardinality control, consistency checking
Interactivity	sameAs/differentFrom atom control

2.1 Must-Have Features

The "must-have" features achieve the minimum basic requirements of the developed tool. The main requirement is to be able to create SWRL rules in a visualized way. The other requirements are deleting SWRL rules and the ability to view all current rules in an ontology. Thus, we deal with SWRL rules as resources and apply basic resource management operations (i.e., creating, viewing, deleting, and editing) on them. Editing a SWRL rule is not currently a feature of the application. However, it can easily be achieved by combining the other existing features. The Entities Box, a visual component of the web application, is used to view the current ontology and shows all the current entities in the ontology that can be used to make valid SWRL atoms, namely `Classes`, `Object Properties`, `Data Properties` and `SWRL Built-ins`. As Fig. 2 indicates, classes are shown in a tree structure that can be collapsed and expanded to give the user better readability. The tree structure reflects the actual inheritance relations between the classes of the ontology. The shown classes are not just the classes that are explicitly stated as members of `SubClassOf` OWL axioms, but also the implicit classes inferred by the reasoner. In our case we use HermiT [18], which is a reasoner based on Description Logic (DL) which aims to be efficient and implement a series of improvements that allow it to work with larger and more complex ontologies. Object properties are listed together with a given ability to view their domains and ranges if available (see Fig. 2). The same goes for data properties and their domains. SWRL built-ins are listed together, however, the currently available atoms do not represent all the built-ins specified in the SWRL language. All the entities in the Entities Box can be dragged and dropped into SWRL Designer Boxes. When they are dropped, they turn into SWRL Atom Blocks.

The SWRL Designer Boxes are used to contain the SWRL Atom Blocks. There are 2 boxes of these, the first one represents the body of an SWRL rule, while the second one represents the head of a SWRL rule (see Fig. 3). In SWRL,


Classes

−+Thing

  −+Person

    −Child

    −Adult

Relations

+hasAunt

+hasBrother

−Domains:

  -Person

−Ranges:

  -Person

(a) Class hierarchy (b) Object properties

Fig. 2: Entities Box

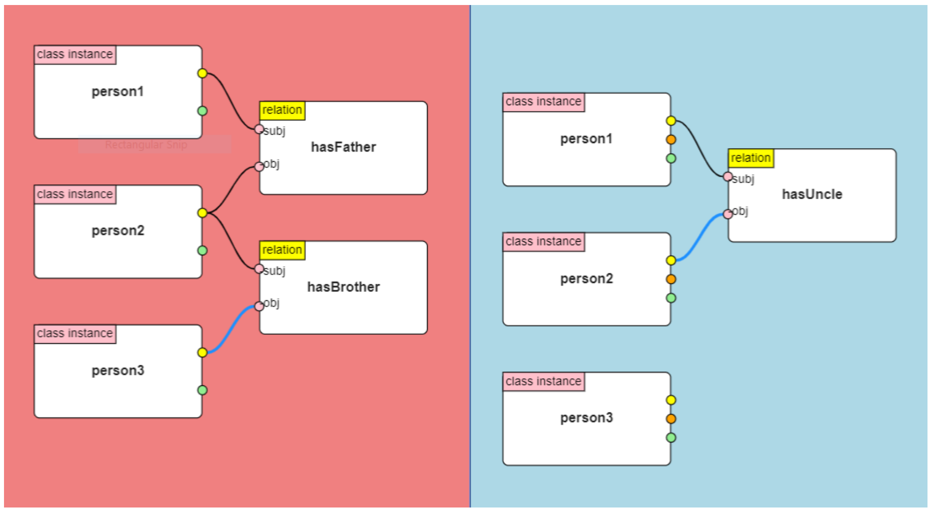
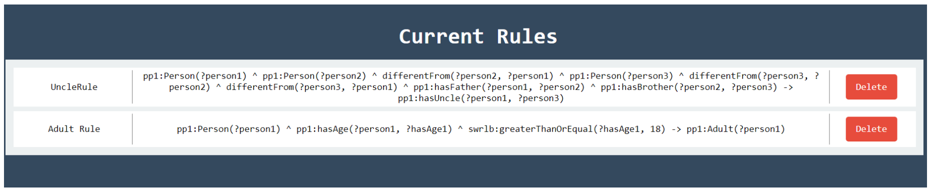


Fig. 3: The visualization for creating the `hasUncle` rule.

one can link between atoms using variables as shown in the `hasUncle` rule. As illustrated in Fig. 3, it is possible to represent those variables by actual links (a.k.a., wires) that can be extended between appropriate SWRL Atom Blocks.

As shown in Fig. 4, at the bottom of the application window, one can see all the previously created rules listed together. In front of every rule there is a `delete` button that can be used to remove that rule from the ontology.



Current Rules

UncleRule	pp1:Person(?person1) ^ pp1:Person(?person2) ^ differentFrom(?person2, ?person1) ^ pp1:Person(?person3) ^ differentFrom(?person3, ?person2) ^ differentFrom(?person3, ?person1) ^ pp1:hasFather(?person1, ?person2) ^ pp1:hasBrother(?person2, ?person3) -> pp1:hasUncle(?person1, ?person3)	Delete
Adult Rule	pp1:Person(?person1) ^ pp1:hasAge(?person1, ?hasAge1) ^ swrlb:greaterThanOrEqual(?hasAge1, 18) -> pp1:Adult(?person1)	Delete

Fig. 4: Listing of the generated SWRL rules.

2.2 Nice-To-Have Features

The "nice-to-have" features are control features which make the proposed tool unique. They enhance the experience of the users and the usability of the application. These features are responsible for enabling users without logic modelling experience to generate rules using the tool. These features also try to prevent the user from making mistakes during rule creation, as much as possible. Those features are either standalone features or additions to the basic "must have" features.

Our nice-to-have features can be divided into four categories, each category represents the purpose of the features within it.

In order to simplify the complexities of SWRL language and to hide some of its details, we implemented a meaningful color-coded scheme. Every SWRL entity has a specific color in the Entities Box and the labels of its corresponding SWRL Atom Blocks have the same color Fig. 5. Atom blocks are wired through ports. To wire two atom blocks, each atom block label has the same color as the other one's port. This feature imposes an order on the rule creation process, as well. A user has to use a SWRL Class Atom Block before being able to use an Object Property Atom Block or a data property atom block. In addition, there are no ports at the left of a SWRL Class Atom Block, which indicates that this type of blocks represent the start of the rule. Also, it is not possible to use a SWRL Built-in Atom Block unless it is preceded by a Data Property Atom Block. This imposed order is easier for users to think of and understand [19]. The underlying library that is used for diagramming automatically aligns the blocks in that same order. To further abstract the SWRL constructs, a modified naming of the SWRL entities is used. `Relations` denote SWRL object properties, `attributes` denote SWRL data properties, and `comparators` denote SWRL built-ins.

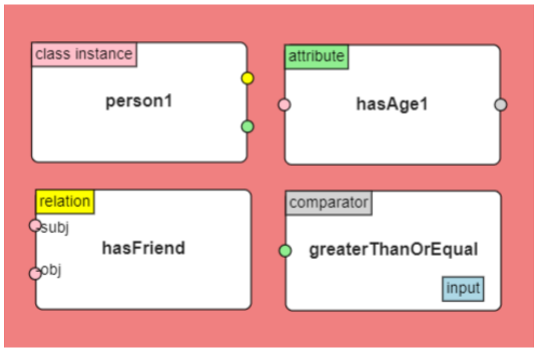


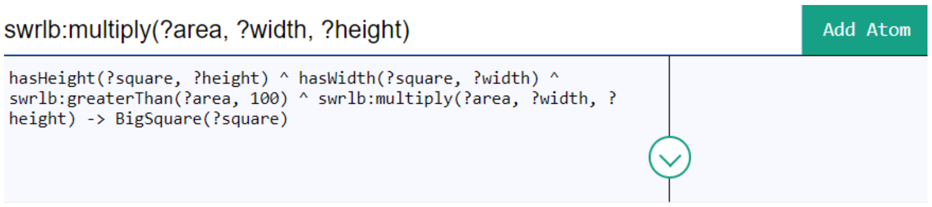
Fig. 5: SWRL Atom Blocks

We removed some constructs from our representation. As mentioned before, not all SWRL built-ins are represented as well as SWRL data ranges. Removing those constructs keeps our representation simple and helps reduce the complexity of the developed application and avoid unnecessary errors. These constructs are not often used and thus do not strongly affect the expressivity of SWRL. However, to ensure that the tool is complete in terms of generating SWRL rules, we enabled adding raw SWRL atoms in basic SWRL syntax. To integrate this feature with our proposed way of rule creating using diagrams, users can view the current SWRL rule in SWRL language syntax. This was the user can know which implicit variables are used and can manually add SWRL atoms. As an example, let us consider the following SWRL rule:

```
hasHeight(?square, ?height) ^ hasWidth(?square, ?width) ^
swrlb:greaterThan(?area, 100) ^
swrlb:multiply(?area, ?width, ?height) -> BigSquare(?square)
```

The rule states that if a square has area greater than 100 units, then it is a big square. The area is calculated using the SWRL built-in atom `swrlb:multiply`. Currently, this specific atom does not have a corresponding block representation. In this case, the previously mentioned feature can be useful. As shown in Fig. 6, we added the `swrlb:multiply` manually in the text-box. Nevertheless, these advanced constructs , e.g. `swrlb:multiply`, will usually only be added by users who will have some experience with knowledge representation and logic rules.

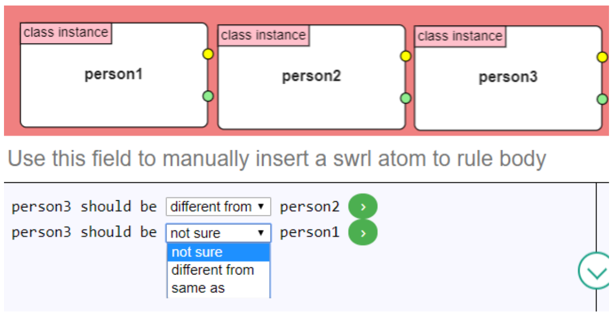
Putting constraints on the rule creation process will reduce the possible errors that can be done by users. Despite being visually appealing, the coloring scheme feature is mainly a validation mechanism. That is, if the user tries to wire blocks that mismatch colors, this action will be prevented and the wire will not be added. Another feature is controlling the cardinality of each SWRL Atom Block (i.e., how many wires can be extended from each port). This feature is used to reduce verbosity, making the user able to wire one block to multiple blocks and vice versa based on the entity type of those blocks being wired. We introduced a way to keep the underlying ontology consistent after adding a rule to it. Before



```
hasHeight(?square, ?height) ^ hasWidth(?square, ?width) ^
swrlb:greaterThan(?area, 100) ^ swrlb:multiply(?area, ?width, ?
height) -> BigSquare(?square)
```

Fig. 6: Adding an SWRL atom in SWRL syntax.

adding a rule to ontology, we use it to infer more ontology data. Then, we check if the result keeps the ontology consistent. If it does, we save that rule with the new ontology state. If not, we refuse that rule, keep the ontology as is, and notify the user of that problem. Even though this feature guarantees to keep the ontology consistent, in some use cases, users may want to allow inconsistencies in their ontologies. Therefore, we allow the user to turn this feature on and off.



```
person3 should be different from person2
person3 should be not sure person1
```

Fig. 7: Representation of the sameAs and differentFrom atoms.

The tool is designed to be interactive, thus control goes back and forth between the user and the web application and is not completely left to the user. One major example of this is how we represent `SameAs` and `DifferentFrom` SWRL atoms. SWRL adopts the Open World Assumption [4] i.e., it is not allowed to assume that two individuals are automatically distinct if they have different name. Therefore, we should be cautious when using `SameAs` and `DifferentFrom` SWRL atoms. Instead of representing those atoms in the diagrams and requiring the user to explicitly remember setting them, the application detects whenever a user reuses a Class Atom Block. Accordingly, it asks the user a question whether this new class instance is same as or different from the same previously declared class instances, or they do not know. In Fig. 7, we see how this feature got fired when we used more than one Person class instance. this same behaviour occurs while authoring hasUncle rule.

[4] https://github.com/protegeproject/swrlapi/wiki/SWRLLanguageFAQ

3 System Architecture

We propose a portable web application following a client-server architecture. The client-server architecture also allows for cooperation, that is because multiple clients can concurrently connect to the same server and edit the underlying ontology. The client application visible to the users gets the ontology entities from the ontology through the server. According to the fetched entities, the users can use the GUI to create syntactically correct SWRL rules. Once a rule is created and its effect on the ontology is approved by the user, it is added to the ontology.

Like most web applications, we used Javascript for the client-side application as it provides us with a wide variety of libraries. Flexible diagramming is one of the most complex and crucial features needed for the developed application. For this purpose, we used GoJS, a JavaScript and TypeScript library for building interactive diagrams and graphs [27]. GoJS is highly customizable and abstracts away a lot of HTML and Canvas machinery. We used JQuery [29], a multi-purpose Javascript library, for DOM manipulation and Apache Tomcat for the server. It is an open-source implementation of the Java Servlet, JavaServer Pages, and WebSocket technologies. It provides a Java HTTP web server environment in which Java code can be executed. To expose our core services to the client application, RESTEasy is used. RESTEasy is an implementation of JAX-RS specification to build RESTful web services in Java. In order to read entities from ontologies and have a general access to it, the OWL API is used. It is a Java API and reference implementation for creating, manipulating, and serializing OWL Ontologies [21]. However, our main concern is accessing and handling SWRL rules. Although a SWRL rule is an additional type of OWL axiom, OWL API does not provide enough tools to easily manage SWRL rules. We used the SWRL API to author and manage SWRL rules. The SWRL API provides both an authoring environment for developing rules and a set of application programming interfaces that support the building of rule-driven applications [28]. Finally, we use the HermiT reasoner to check the consistency of underlying ontologies after editing them through the generated rules. We also use it to extract and display class entities in a tree structure. The different features presented in Section 2 are implemented through different modules included in the client and server-side applications. Fig. 9 gives an overview of the different modules described above and the features they realize.

1. Entity exploration features: Extracting the ontology classes to be displayed in a tree-like structure is done using Hermit Reasoner and with the help of a Depth First Search algorithm. Hermit reasoner provides the method `getSubClasses(OWLClassExpression c, boolean directSubClasses)`, which can be used to traverse all OWL classes with depth-first search algorithm. After fetching all classes and other entities. The client is responsible of listing them to the user and providing the ability of collapse and expand the entities listing to enhance the readability, this task was easily achieved by JQuery library.

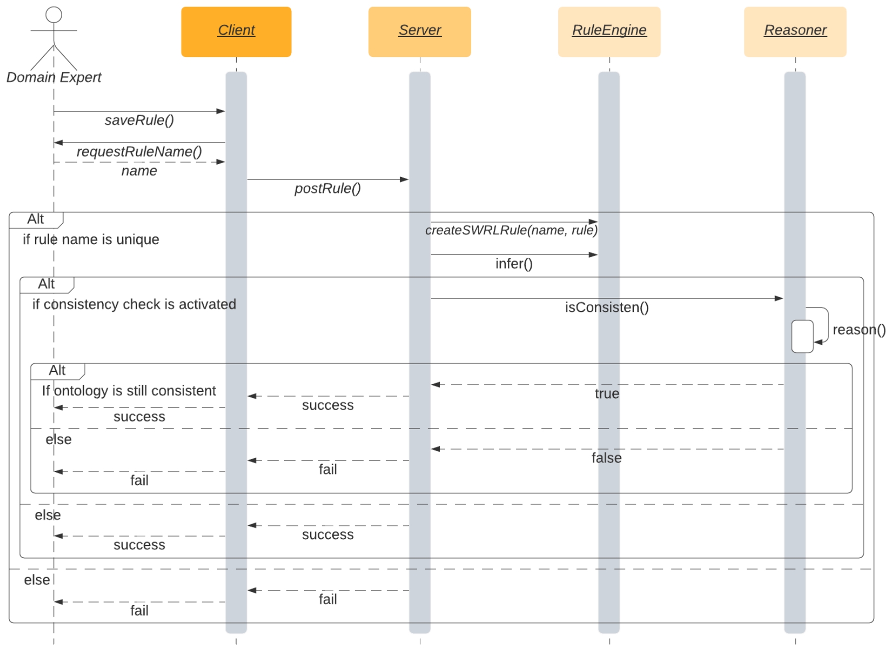


Fig. 8: Sequence diagram detailing the rule creation process.

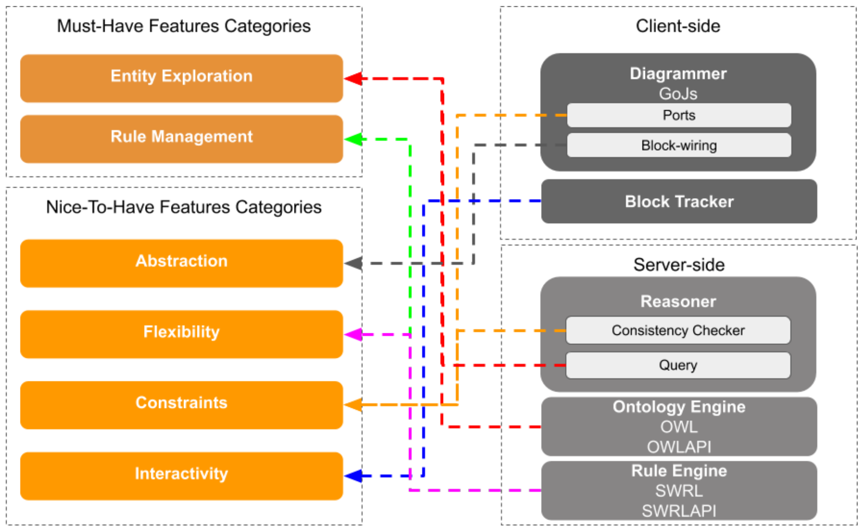


Fig. 9: Overview of the different modules responsible for realizing the features belonging to the different categories.

2. Rule management features: Rule management consists of two parts: rule generation and rule viewing and deletion. The first one is the main feature of the developed application. Fig. 8 shows the sequence diagram of the rule generation process. The main part of the rule generation is done in the client-side application with the help of the Diagrammer. A rule can be easily formed by wiring appropriate blocks before being submitted to the server. In the server, the Rule Engine is responsible for parsing, inferring, and saving the rule. The different checks done while creating a rule, such as reasoner consistency check, are also displayed in Fig. 8. The SWRLAPI is used to extract existing rules to be viewed by the users. Rules can also be displayed in a list-item pattern with the ability of deleting specific rules.
3. Abstractions: abstraction features are achieved with the help of GoJS Diagrammer. The ports feature of GoJS as well as the automatic layouting enable the implemented abstractions. these abstractions do not only affects the user by giving simple and visually appealing components, they also abstract a lot of programming, making our code more open for extension.
4. Flexibility: To guarantee not to lose the SWRL language expressivity, we added the features that allow users to deal with SWRL language syntax directly. The implementation of those features was well integrated with the rest of the features.
5. Constraints: The Diagrammer allows the programmer to set rules for block linking, checking these rules is fired whenever a user tries to link two blocks. Our rules do not check any block types, instead, the rules check the port colors to ensure correct linking between entities in the rule body and head. The cardinality restrictions on SWRL atoms is controlled by the Diagrammer as it allows the programmer to specify how many links can go in or out from a port. Hermit reasoner is responsible for the consistency checking whenever a rule is submitted, this feature can be toggled on and off by the user.
6. Interactivity: Whenever a new class instance is added, the Block Tracker checks whether instances of the same class were used before. If this occurs, the Block Tracker prompts the user with same as/ different from questions to determine if individuals refer to the same underlying individual or are distinct.

4 Conclusion

In this paper, we have introduced a tool for creating SWRL rules using a wizard-like graphical user interface. The tool enables domain experts without detailed knowledge of ontologies and rules to define syntactically and to the most part semantically correct SWRL rules. This is achieved by including six main feature categories in the tool. Entity exploration and rule management are the must-have features for a visual rule generation tool. The main advantage provided by our tool is the four nice-to-have feature categories that enable anyone to use the tool. These categories are abstraction, flexibility, constraints and interactivity.

A user study to evaluate the ease of use of prototype, in comparison to other rule generation tools, is being conducted. We are currently working together with our partners to incorporate the proposed tool into the ontology-based model. This would enable psychologists to see the effect and check the consistency of hypotheses on human behaviour. Comparing the rule effect on the ontology-based model with actual collected data would help improve our understanding of human behavior and the resulting models. We are also investigating how to maintain the ease of use of the tool for very large complex ontologies. Here we plan to combine different visualization and abstraction techniques to be able to easily navigate through the ontology entities and define rules on them.

References

1. Alaa, M., Bolock, A.E., Abas, M., Abdennadher, S., Herbert, C.: Appgen: a framework for automatic generation of data collection apps. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing. pp. 1906–1913 (2020)
2. Bak, J., Nowak, M., Jedrzejek, C.: Graph-based editor for swrl rule bases. In: RuleML (2). Citeseer (2013)
3. Bernstein, A., Kaufmann, E.: Gino – a guided input natural language ontology editor. pp. 144–157 (11 2006). https://doi.org/10.1007/11926078$_1$1
4. Bolock, A., Abdelrahman, Y., Abdennadher, S.: Character Computing. Human–Computer Interaction Series, Springer International Publishing (2020), `https://books.google.com.eg/books?id=VZXHDwAAQBAJ`
5. Boufrida, A., Boufaida, Z.: Automatic rules extraction from medical texts. In: 2014 International Workshop on Advanced Information Systems for Enterprises. pp. 29–33 (Nov 2014). https://doi.org/10.1109/IWAISE.2014.14
6. Costa, N., Knorr, M., Leite, J.: Next step for nohr: Owl 2 ql. In: International Semantic Web Conference. pp. 569–586. Springer (2015)
7. Damljanovic, D., Agatonovic, M., Cunningham, H.: Natural language interfaces to ontologies: Combining syntactic analysis and ontology-based lookup through the user interaction. pp. 106–120 (05 2010). https://doi.org/10.1007/978-3-642-13486-9$_8$
8. Dimitrova, V., Denaux, R., Hart, G., Dolbear, C., Holt, I., Cohn, A.G.: Involving domain experts in authoring owl ontologies. In: International Semantic Web Conference. pp. 1–16. Springer (2008)
9. El Bolock, A.: Defining character computing from the perspective of computer science and psychology. In: Proceedings of the 17th International Conference on Mobile and Ubiquitous Multimedia. pp. 567–572. ACM (2018)
10. El Bolock, A.: What is character computing? In: Character Computing, pp. 1–16. Springer, Cham (2020)
11. El Bolock, A., Abdennadher, S., Herbert, C.: Applications of character computing from psychology to computer science. In: Character Computing, pp. 53–71. Springer (2020)
12. El Bolock, A., El Kady, A., Herbert, C., Abdennadher, S.: Towards a character-based meta recommender for movies. In: Computational Science and Technology, pp. 627–638. Springer (2020)
13. El Bolock, A., Ghonaim, A., Herbert, C., Abdennadher, S.: Detecting impulsive behavior through agent-based games. In: International Conference on Intelligent Human Systems Integration. pp. 208–213. Springer (2020)
14. El Bolock, A., Herbert, C., Abdennadher, S.: Cconto: Towards an ontology-based model for character computing. In: 14th International Conference on Research Challenges in Information Science, RCIS 2020, Limassol, Cyprus, September 23-25, 2020. IEEE (2020)

15. El Bolock, A., Salah, J., Abdelrahman, Y., Herbert, C., Abdennadher, S.: Character computing: Computer science meets psychology. In: 17th International Conference on Mobile and Ubiquitous Multimedia. pp. 557–562. ACM (2018)
16. El Bolock, A., Salah, J., Abdennadher, S., Abdelrahman, Y.: Character computing: challenges and opportunities. In: Proceedings of the 16th International Conference on Mobile and Ubiquitous Multimedia. pp. 555–559. ACM (2017)
17. El Nashar, Z., El Bolock, A., Salah, J., Herbert, C., Abdennadher, S.: Investigating the effect of personality traits on performance under frustration. In: International Conference on Games and Learning Alliance. pp. 595–604. Springer (2019)
18. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: Hermit: An owl 2 reasoner. J. Autom. Reason. **53**(3), 245–269 (Oct 2014). https://doi.org/10.1007/s10817-014-9305-1, `https://doi.org/10.1007/s10817-014-9305-1`
19. Hassanpour, S., O'Connor, M., Das, A.: Exploration of swrl rule bases through visualization, paraphrasing, and categorization of rules. vol. 5858, pp. 246–261 (11 2009). https://doi.org/10.1007/978-3-642-04985-9$_2$3
20. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies (01 2009). https://doi.org/10.1201/9781420090512
21. Horridge, M., Bechhofer, S.: The owl api: A java api for owl ontologies. Semant. Web **2**(1), 11–21 (Jan 2011)
22. Horrocks, Ian, Patel-Schneider, F, P., Boley, Harold, Tabet, S., Said, Grossof, Benjamin, Dean, M., Mike: Swrl: A semantic web rule language combining owl and ruleml. W3C Subm **21** (01 2004)
23. Horrocks, I., Patel-Schneider, P.F., Van Harmelen, F.: From shiq and rdf to owl: The making of a web ontology language. Journal of web semantics **1**(1), 7–26 (2003)
24. Kuhn, T.: Acewiki: Collaborative ontology management in controlled natural language. CoRR **abs/0807.4623** (2008), `http://arxiv.org/abs/0807.4623`
25. Leutgeb, A., Utz, W., Woitsch, R., Fill, H.G.: Adaptive processes in e-government-a field report about semantic-based approaches from the eu-project" fit". In: ICEIS (3). pp. 264–269 (2007)
26. MacLarty, I., Langevine, L., Bossche, M.V., Ross, P.: Using swrl for rule-driven applications. Accessed February **9** (2009)
27. Northwoods Software: Gojs, `https://hadoop.apache.org`
28. O'Connor, M., Shankar, R., Musen, M., Das, A., Nyulas, C.: The swrlapi: A development environment for working with swrl rules. (01 2008)
29. OpenJS Foundation: Jquery, `https://jquery.com/`
30. Pittl, B., Fill, H.G.: A visual modeling approach for the semantic web rule language. Semantic Web **11**(2), 361–389 (2020). https://doi.org/10.3233/SW-180340
31. Wróblewska, A., Kapłański, P., Zarzycki, P., Ługowska, I.: Semantic rules representation in controlled natural language in fluenteditor. In: 2013 6th International Conference on Human System Interactions (HSI). pp. 90–96 (June 2013). https://doi.org/10.1109/HSI.2013.6577807