An SKOS-Based Vocabulary on the Swift Programming Language

Christian Grévisse(✉) and Steffen Rothkugel

Department of Computer Science, University of Luxembourg,
2, avenue de l'Université, 4365 Esch-sur-Alzette, Luxembourg
{christian.grevisse,steffen.rothkugel}@uni.lu

Abstract. Domain ontologies about one or several programming languages have been created in various occasions, mostly in the context of Technology Enhanced Learning (TEL). Their benefits range from modeling learning outcomes, over organization and annotation of learning material, to providing scaffolding support in programming labs by integrating relevant learning resources. The Swift programming language, introduced in 2014, is currently gaining momentum in different fields of application. Both its powerful syntax as well as the provided type safety make it a good language for first-year computer science students. However, it has not yet been the subject of a domain ontology. In this paper, we present an SKOS-based vocabulary on the Swift programming language, aiming at enabling the benefits of previous research for this particular language. After reviewing existing ontologies on other programming languages, we present the modeling process of the Swift vocabulary, its integration into the LOD Cloud and list all of its resources available to the research community. Finally, we showcase how it is being used in different TEL tools.

Keywords: Swift · Vocabulary · SKOS · E-Learning

Resource Type: Ontology/Vocabulary

Permanent URL: http://purl.org/lu/uni/alma/swift

1 Introduction

Programming languages have been the subject of domain ontologies in several occasions. Such ontologies have typically covered both syntactic and semantic elements of one or several programming languages. The modeled domain knowledge was then used mostly in the context of Technology Enhanced Learning (TEL). The benefits have been manifold: modeling learning outcomes [16], creation of learning paths [8,13,16], semi-automatic annotation of learning resources [7,11,13], organization of learning objects (LOs) [14,19], scaffolding support in

J. Z. Pan et al. (Eds.): ISWC 2020, LNCS 12507, pp. 244–258, 2020.
https://doi.org/10.1007/978-3-030-62466-8_16

programming labs [7,12,21], reusable knowledge beyond the boundaries of systems [11,14,16] as well as visual and linguistic [14,20] aids. As some approaches realize a mapping between the abstract syntax tree (AST) returned by a parser and the elements of their respective ontologies [4,11,13], it became possible to perform static code analysis through SPARQL queries [4] or even retrieve information on a piece of source code through natural language questions [9].

One of the more recent programming languages is Swift, which was only introduced in 2014. First a proprietary language, it became open source and can nowadays be used within Apple's ecosystem, on Linux and even on Windows (using *Windows Subsystem for Linux*). Often considered a niche language for iOS development, it can be used for server-side development (with frameworks like *Vapor*[1]) and has more recently been considered for deep learning and differentiable computing in *Swift for TensorFlow*[2]. While its first versions often broke backward compatibility, the language has, in its 6 years of existence, reached a certain maturity, with Swift 5 guaranteeing ABI (application binary interface) stability. As of May 2020, Swift is among the top 9 programming languages according to the *PYPL (PopularitY of Programming Language)* Index[3] which measures how often language tutorials were searched for on Google, and among the top 11 on the *TIOBE* index[4]. Furthermore, the *Developer Survey 2019* carried out by the popular question-and-answer site *Stack Overflow* shows Swift among the top 6 most "loved" languages[5]. From an educational point of view, Swift is, similar to Python, less verbose than Java, but, as a statically-typed language, provides type safety, which can be beneficial to both new and experienced programmers.

In this paper, we present an SKOS-based vocabulary on the Swift programming language. Building on the growing popularity of the language, this vocabulary aims at enabling the previously mentioned benefits of modeling domain ontologies on programming languages for Swift. Currently, its main goal is to provide a controlled set of concepts used in the annotation of related learning material, as well as to enable scaffolding support in programming labs by integrating relevant resources.

The remainder of this paper is organized as follows. In Sect. 2, we present an overview of previous programming language ontologies. The modeling process and implementation of the Swift vocabulary is described in Sect. 3. Section 4 shows how the vocabulary is currently used. A general discussion is given in Sect. 5. We conclude in Sect. 6, along some ideas for future work.

[1] https://vapor.codes.
[2] https://www.tensorflow.org/swift/.
[3] https://pypl.github.io/PYPL.html.
[4] https://tiobe.com/tiobe-index/.
[5] https://insights.stackoverflow.com/survey/2019#most-loved-dreaded-and-wanted.

2 Related Work

There have been several contributions in research covering one or more programming languages in domain ontologies, of differing complexity and construction methodologies.

Most of these attempts focused on the Java programming language. In [16], a Java ontology is created to model learning outcomes of a distance education course, effectively determining learning paths. The concepts comprised in this ontology were collected from the *Java Tutorials*[6]. The *QBLS* intelligent tutoring system (ITS) presented in [7] uses a similar Java ontology to annotate and provide learning resources as a support in programming labs. While the initial set of concepts was retrieved again from the Java Tutorials, the authors needed to add further concepts while annotating learning material, which shows a gap between the initial domain ontology design and the intention of the annotating teacher. The *Protus* ITS also uses an ontology on Java to provide semantic recommendations of resources [21]. Ivanova built a bilingual Java ontology to help Bulgarian students understand English terminology [14]. This ontology goes beyond pure language features, but also includes application types, such as applets or servlets. The author states that the visualization of such an ontology provides students a quick overview of the concepts and terminology to be learned. The *JavaParser* [13] is a fine-grained concept indexing tool for Java problems. Mapping AST elements to entities in a custom Java ontology enables sequencing of code examples. The *CodeOntology* [4], while mainly tailored to Java, provides a formal representation of object-oriented programming (OOP) languages, and serializes Java source code to RDF triples. This ultimately makes it possible to perform static code analysis through SPARQL queries. Here, the Java language specification was used to retrieve the comprised concepts.

There has also been work on ontologies about the C programming language, used in University courses [18,20]. The ontology presented in [20] was constructed based on glossary terms from the reference book by Kernighan and Ritchie (often abbreviated as *K&R*) [15]. Similar to [14], the authors state that visualizing their ontology can give students a navigation interface across learning objects.

There is also work covering more than one concrete programming language. Pierrakeas et al. constructed two ontologies on Java and C in order to organize LOs [19]. They again used a glossary-based approach, using the Java Tutorials and the K&R book. The *ALMA Ontology* in [11] combines both languages under one SKOS-based umbrella ontology, separating language-specific concepts in different modules, while aligning common elements across the modules. A similar approach focusing on OOP languages is done in *OOC-O* [1], which realizes a consensual conceptualization between common elements from Smalltalk, Eiffel, C++, Java and Python to foster *polyglot programming*.

Finally, there have also been ontologies on other languages not yet mentioned. *CSCRO* is an ontology for semantic analysis of C# source code [9]. Based on the *Source Code Representation Ontology (SCRO)* [2], an ontology solution

[6] https://docs.oracle.com/javase/tutorial/.

for detecting programming patterns in Java code, concepts from MSDN references on C# language features were added. This ontology was used to retrieve information from source code through natural language questions. *PasOnto*, an ontology on the Pascal language, was created to describe prerequisites necessary to understand exercise solutions [8]. Again, a glossary approach was used, extracting terms from existing C ontologies and Pascal programming courses.

While mostly TEL applications were the target of the previously mentioned ontologies, a semantic representation of code can enable code analysis and queries [4,9]. The *Function ontology* [6] was created to declare and describe functions, problems and algorithms, independently of a concrete programming language. As opposed to a specification, which defines how to use a function (e.g., on a web service), using this ontology an unambiguous description of what a function does can be realized, while omitting any language-specific implementation details. Similarly, in model-driven development, platform ontologies can be used to define platform dependencies of model transformations and reason about their applicability to specific platforms [22].

3 Swift Vocabulary

The use of ontologies generally fosters the interoperability, discovery, reusability and integration of data and resources within and beyond the boundaries of heterogeneous systems in an unambiguous and machine-readable way. To enable the advantages related to these objectives as realized in the related work for other languages, our main requirement for the Swift vocabulary was to cover the major features of the language. The level of granularity should be reasonable, considering its main application domain, i.e., the annotation and retrieval of learning material. In fact, the resulting vocabulary shall not overwhelm a human annotator. A too fine-grained knowledge representation, as returned by a parser, would violate *Occam's razor*[7], whereas a too coarse-grained one would not allow to annotate learning material in the necessary detail.

For annotation purposes, we can stay in the spectrum of *lightweight* ontologies [3,18] that (i) comprise a controlled vocabulary of entities, (ii) organize these entities in a hierarchical structure, and (iii) enable relations among them. Similar to [7,11,18], we decided to use *SKOS (Simple Knowledge Organization System)*[8], a W3C recommendation for KOS using RDF. SKOS organizes *concepts* within *concept schemes*, provides *labels* to concepts and realizes both hierarchical (e.g., `skos:broader`) and associative (`skos:related`) links.

The vocabulary described in this paper is based on version *5.2* of the Swift programming language, released in March 2020. Nevertheless, upcoming releases of the language will hardly change a lot of the existing language features, rather than add new ones, making this initial version of the Swift vocabulary a solid starting point for future versions of the ontology.

[7] *"Pluralitas non est ponenda sine necessitate."*
[8] https://www.w3.org/2004/02/skos/.

3.1 Ontology Design

For the Swift Vocabulary, we decided to go for a glossary-based approach that was used most of the time in related work. Our previous experience described in [11], where we proceeded with a bottom-up approach by modeling the syntactic elements returned by a parser, resulted in too many purely parser-related (and parser-dependent) elements, requiring a significant filtering and renaming effort, as well as the addition of missing high-level concepts. While a glossary-based approach might still suffer from a gap between the resulting ontology and the concepts needed for annotations as mentioned in [7], this gap would be even bigger when relying on a parser-based entity extraction method.

As a main reference, we used *The Swift Programming Language*[9] (henceforth called the "Swift book"), the authoritative reference comprising a language guide and its formal specification. A vocabulary created based on the content of this reference book can reasonably be considered as a vocabulary about the language itself, comprising concepts that fit the purpose for our TEL use case. However, this vocabulary shall not be seen as a full-fledged, formal ontology covering the semantics of the language itself, which was not the goal of this work.

The Swift book is available for free, either in the browser or as an ePub. A Python script using the popular *Beautiful Soup*[10] library crawled and parsed the online version. Due to the lack of an index, our working assumption here is that relevant concepts are mentioned in the headings, i.e., the HTML tags `h1` to `h6`. The heading strings were then passed through *spaCy*[11], a natural language processing library. After stopword removal and lemmatization, the encountered noun phrases (if any) were considered as candidate concepts. In case no noun phrase was contained (e.g., headings simply stating the name of a keyword like `break` or `continue`), the verbatim text was taken as a candidate. All headings comprise a permalink anchor (`a` tag) to the respective section.

The resulting set contained 458 candidate concepts, mapped to their respective permalinks (some concepts were mentioned at several occasions). This high number is due to the fact that the book includes many in-depth discussions and examples, such that not all headings represent a new, dedicated concept. To filter such headings and avoid a too fine-grained representation, we manually selected a subset of 139 concepts, however added 31 additional ones representing important keywords and access level modifiers that were not subject of a dedicated heading, yet mentioned in the text. For the remaining 170 concepts, we cleansed the concept names (removing non-alphabetical characters, applying camel case), gave them a preferred label (`skos:prefLabel`) and selected the best-matching resource from the reference book. A first representation in RDF comprising only the concepts and their resource was created using the *RDFLib*[12] package. The `skos:definition` property was chosen to map each concept to the URL of the corresponding excerpt in the Swift book.

[9] https://docs.swift.org/swift-book/.
[10] https://www.crummy.com/software/BeautifulSoup/.
[11] https://spacy.io.
[12] https://rdflib.readthedocs.io/en/stable/.

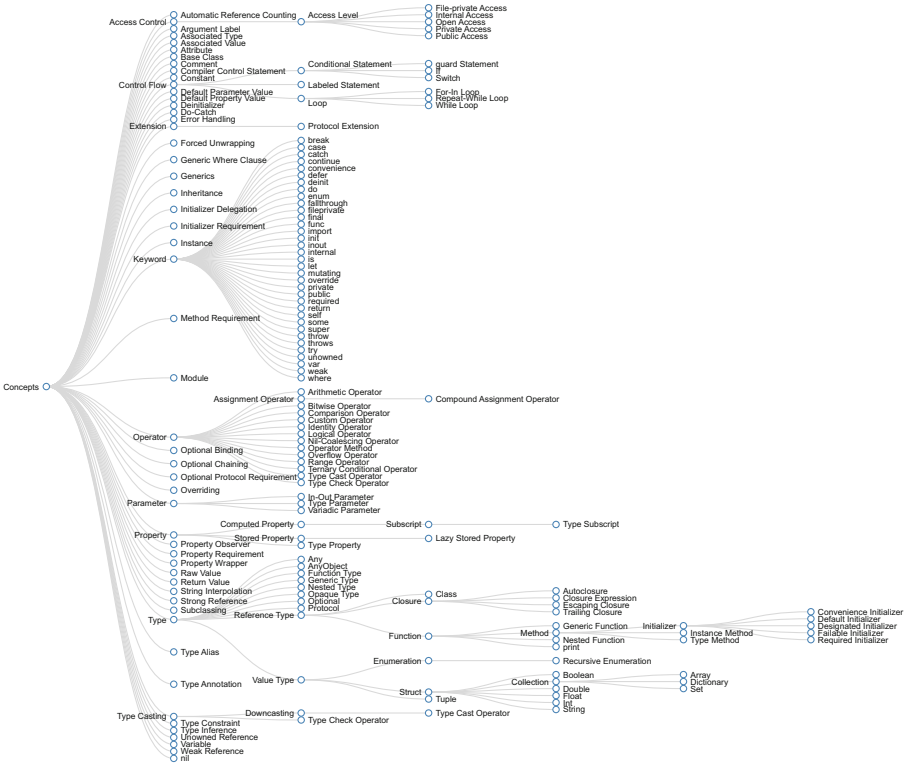

Fig. 1. Dendrogram of the Swift vocabulary

Using *Protégé*[13], we then created both hierarchical and associative links among the concepts. As the number of relevant concepts about a programming language is typically below 200, this manual process is feasible. An automatic approach relying on cross-references in the text to establish associative links would have been possible, but would have been incomplete (not enough cross-references) and yielded false positives (e.g., the section on *Property Observers* mentions *Overriding*, which a priori is not a related concept). This again would have required some manual curation.

Hierarchical links were modeled using 120 `skos:broader` relations. A dendrogram representation of the resulting hierarchy, generated using the *Ontospy*[14] library, is shown in Fig. 1.

Associative links were modeled using 218 `skos:related` relations, of which 55% were internal relations, the remainder being used for ontology alignment purposes (see Sect. 3.2). All concepts are covered by at least one associative or hierarchical link, effectively avoiding any orphan concepts. With respect to the

[13] https://protege.stanford.edu.
[14] https://lambdamusic.github.io/Ontospy/.

permalinks to excerpts in the reference book, the 139 concepts selected from the automatic concept extraction are directly related via the `skos:definition` property. The manually added 31 concepts are related to such an excerpt via a 1-hop link (either associative or hierarchical) to another concept. We described the metadata of the vocabulary using Dublin Core[15] and FOAF[16]. The final vocabulary comprises 1226 RDF triples.

3.2 Ontology Alignment

To connect the Swift vocabulary to the Linked Open Data cloud, we aligned the Swift vocabulary with concepts from DBpedia. Again, a manual alignment was feasible due to the relatively low number of concepts. 93 out of the 170 concepts could be directly related to matching DBpedia concepts. As already mentioned above, the `skos:related` relation was used for that purpose. The fact that only 55% of all concepts are directly mapped to DBpedia entities shows the specificity gap between our domain-specific ontology and the domain-general Wikipedia, which oftentimes does not have dedicated articles on certain programming language features. The remaining concepts are indirectly linked to DBpedia, over up to 2 hops via `skos:related` or 1 hop via `skos:broader`. Figure 2 shows a chord diagram of the semantic relations inside and beyond the Swift vocabulary. Concepts from the latter are represented in the orange arc, whereas mapped DBpedia entities are shown in the green arc. Hierarchical links inside the Swift vocabulary are represented through blue edges, associative links are shown through brown edges.

Aligning other programming language ontologies to DBpedia would establish an indirect relation to the Swift vocabulary, which ultimately could enable the retrieval of related learning material across different programming languages. This can be particularly useful, if a student has already seen a concept present in a language A in another language B, and wants to review or compare the resources on the concept from language B to better understand the concept in language A, fostering higher level thinking skills of students. In addition, collaboration among researchers could be improved this way, to avoid recreating an ontology on the same programming language, as shown in Sect. 2.

3.3 Available Resources

Table 1 lists all available resources related to the Swift vocabulary. It has been serialized in RDF/XML and Turtle format. The permanent URL (PURL) leads to the documentation, which has been generated using WIDOCO [10] and the aforementioned OntoSpy library. Through content negotiation, the RDF/Turtle representations can also be retrieved through the PURL, e.g.,

```
curl -sH "Accept: application/rdf+xml" \
     -L http://purl.org/lu/uni/alma/swift
```

[15] https://www.dublincore.org/specifications/dublin-core/dcmi-terms/.
[16] http://xmlns.com/foaf/spec/.

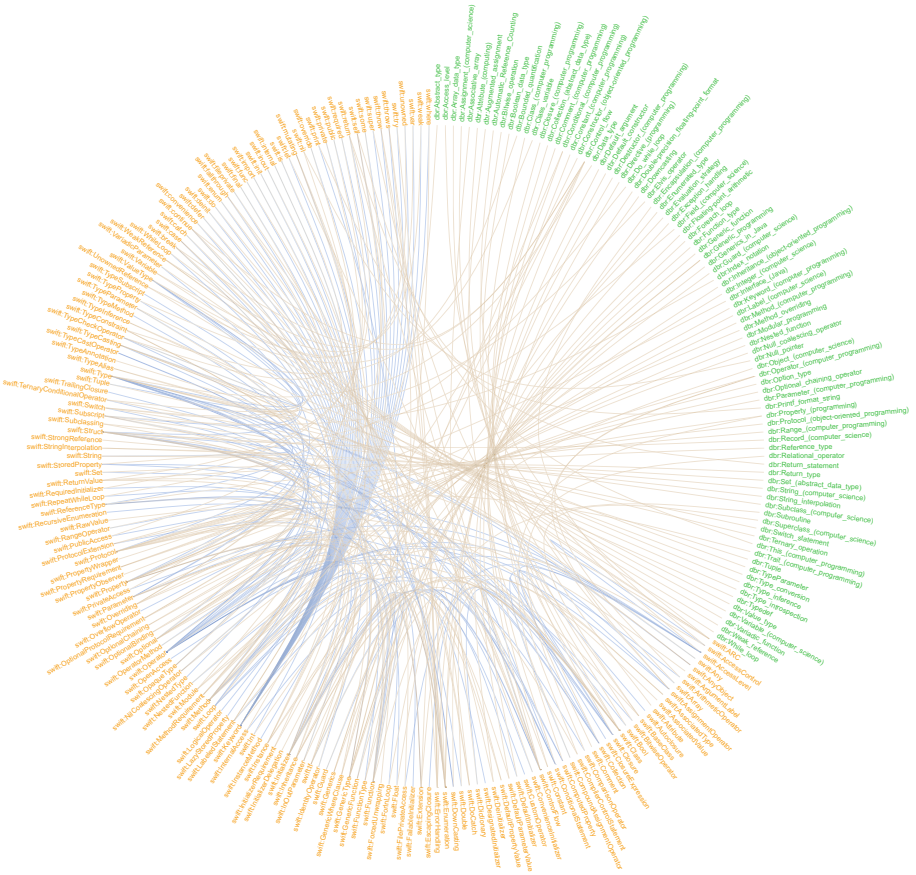

Fig. 2. Chord diagram showing semantic relations inside and beyond the Swift Vocabulary (Color figure online)

For the SPARQL endpoint, we use the *Virtuoso Open-Source Edition*[17]. It allows to retrieve, e.g., the resource from the Swift reference book on the "For-In Loop":

```
SELECT DISTINCT ?resource
WHERE {
    ?concept skos:prefLabel "For-In Loop" .
    ?concept skos:definition ?resource
}
```

Further learning resources could be indexed using an `INSERT` query, the necessary access rights provided.

[17] http://vos.openlinksw.com.

Table 1. Swift Vocabulary-related resources

Resource	URL
PURL	http://purl.org/lu/uni/alma/swift
Turtle file	https://alma.uni.lu/ontology/swift/5.2/swift.ttl
RDF file	https://alma.uni.lu/ontology/swift/5.2/swift.rdf
GitHub repository	https://github.com/cgrevisse/swift-vocabulary
SPARQL endpoint	https://alma.uni.lu/sparql

The vocabulary is released under the Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)[18] license. As a canonical citation, please use: "Grévisse, C. and Rothkugel, S. (2020). *Swift Vocabulary.* http://purl.org/lu/uni/alma/swift".

4 Applications

With the main focus on building a vocabulary suitable for annotating, retrieving and integrating learning material on the Swift programming language, this section will now present how this vocabulary is being used in different TEL tools.

4.1 Extension for Visual Studio Code

Visual Studio Code is a cross-platform source-code editor. Released in 2015, it is among the top 4 IDEs according to the *Top IDE index*[19], as of May 2020, and the most popular development environment according to the previously mentioned Stack Overflow development survey. We created the *ALMA 4 Code* extension[20], which enables the user to select a piece of code and be provided with related learning material. The extension currently supports Swift in its version 5.2. The learning resources are retrieved from the ALMA repository [12], which is hosted on the same server as our SPARQL endpoint. The extension focuses on providing resources to the select syntactical element(s), which can be useful to a student in a programming course to understand, e.g., code examples or exercise solutions.

Under the hood, the extension uses `swift-semantic`[21], a command-line utility we wrote that, for a given `.swift` file and selection range, returns the semantic concept from the Swift vocabulary of the top-most AST node. The underlying parser is given by the *SwiftSyntax*[22] library.

[18] https://creativecommons.org/licenses/by-sa/4.0/.
[19] https://pypl.github.io/IDE.html.
[20] https://marketplace.visualstudio.com/items?itemName=cgrevisse.alma4code.
[21] https://github.com/cgrevisse/swift-semantic.
[22] https://github.com/apple/swift-syntax.

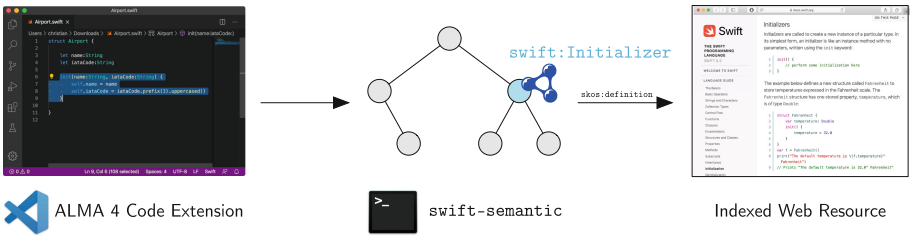

Fig. 3. Example workflow of the "ALMA 4 Code" extension for Visual Studio Code

An example workflow is shown in Fig. 3. The user has selected the initializer of a structure. Through a simple keyboard shortcut, the ALMA 4 Code extension is activated, which passes the source code as well as the selection parameters (start/end line/column) to the `swift-semantic` utility. The latter then builds the AST using the SwiftSyntax library and retrieves the top-most node inside of the selection. Through a mapping between AST node types and Swift vocabulary concepts, the corresponding concept is returned to the extension. Finally, the ALMA repository is searched for learning material indexed for this concept. In this example, the part on initializers from the Swift book is returned.

4.2 Annotation of Learning Material

The previously mentioned ALMA repository is not only used by the ALMA 4 Code extension. The ecosystem described in [12] also comprises Yactul, a gamified student response platform. The Yactul app can be used to replay quizzes from the class, keep track of the learning progress for an individual user and direct her to learning material related to a quiz activity. As shown in Fig. 4, if a player wants to know more about the concepts behind a question (e.g., when her answer was wrong), she can consult related resources, due to the fact that Yactul activities are tagged with semantic concepts. In this example, a question targeting the `mutating` keyword (required by methods that change the state of a Swift structure) is tagged with the concept `swift:mutating` from this vocabulary, such that the related excerpt from the Swift book can be retrieved and shown directly in the app.

Finally, we also use the concepts from our vocabulary as hashtags on our lab sheets (Fig. 5). Hashtags are a popular type of folksonomy used in many social networking sites. The LaTeX-generated lab sheets include these hashtags both for the learner (human agent) to understand the topic of an exercise, as well as for software agents in the form of machine-readable metadata of the PDF file. Clicking on a hashtag leads students to resources indexed in the ALMA repository, similar to the VS Code extension or the Yactul app. Our previous research [12] has shown that all 3 approaches of learning material integration, i.e., in an IDE, in the Yactul app and as hashtags on lab sheets are considered useful by first-year computer science students.

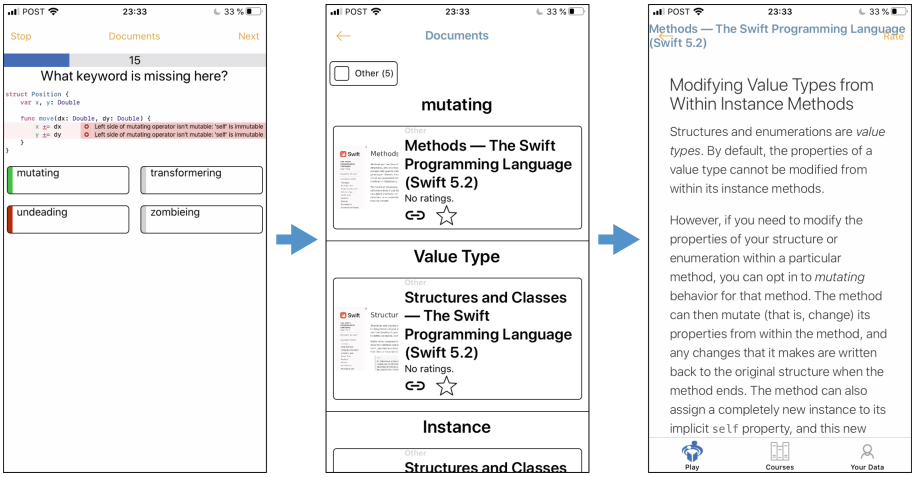

Fig. 4. Retrieval of learning material in the Yactul app

Exercise 31 – Hospital #Closure

The goal of this exercise is to write a Swift console application that models a patient management system for hospitals. On Moodle, you can find some boilerplate code, which defines a set of symptoms and hospital units, as well as permits the generation of random patients. You can also simply print a patient instance, a meaningful string representation has been provided.

Using closures, you will print patient lists according to certain criteria. Common operations like `filter`, `map`, `reduce`, `sorted` and `forEach` will be heavily used throughout this exercise. It is not permitted to use any explicit loops in this exercise.

Fig. 5. Use of a vocabulary concept in form of a hashtag on a lab sheet

During the summer semester 2020 (February - July), in which instruction was mostly delivered in a remote way due to the COVID-19 pandemic, the Swift vocabulary was first used in the context of a Swift course taught at the University of Luxembourg. At the time of writing, we have observed that learning resources from the Swift book were consulted 84 times through clicking on hashtags on the lab sheets. The most frequently visited resource was the excerpt on "Computed Properties"[23] (concept URI: `swift:ComputedProperty`), a feature unknown to the students from their prior knowledge on Java.

5 Discussion

Impact. The Swift vocabulary presented in this paper is, to the best of our knowledge, the first semantic knowledge representation about the Swift programming language. Similar to related work, which was mainly focusing on Java, this vocabulary was modeled using a glossary-based approach, relying on the Swift reference book. The benefits that related work has brought, mainly in the context of e-learning systems, can now be implemented for this relatively new, yet

[23] https://docs.swift.org/swift-book/LanguageGuide/Properties.html#ID259.

stable language, using this vocabulary. As previously mentioned, it has already been adopted in a Swift course for annotating and retrieving learning material, which is actively used by students.

Extensibility. Although the language has become rather stable by now, new language features are constantly integrated, and developers can submit proposals on a dedicated GitHub repository of Apple[24]. The architecture of SKOS, used in this vocabulary, follows the principle of making *minimal ontological commitment* [5], hence makes it easy to extend the vocabulary by integrating new language features.

Availability. As listed in Table 1, all resources are available through a permanent URL and on GitHub. The VS Code extension described in Sect. 4.1 is available on the Visual Studio Marketplace. The Swift Vocabulary has also been registered in the LOD Cloud[25].

Reproducibility. The code used in the ontology creation process described in Sect. 3 is available on the GitHub repository mentioned in Table 1. Apart from the reference serializations, it contains a Python script to generate the initial set of candidate concepts by extraction from the Swift book, along a roadmap for the manual curation.

Reusability. The only semantic commitment is given by the few formal requirements imposed by SKOS. No further domains or ranges have been defined as purely SKOS properties were used. Concepts can thus be easily reused in other ontologies, either directly or indirectly using the alignment with DBpedia.

Sustainability. As the Swift vocabulary is being actively used in the context of a Swift programming course, it is in the developing group's very own interest to keep it up-to-date with respect to further language features that would be introduced to students, either in a pure Swift programming course or an iOS development-related one. The vocabulary could also be pitched to receive its own sub-category in the "Related Projects" forum[26] to gain the recognition and support of the Swift open source community.

Maintenance. Should new language features be introduced or missing, the authors are always welcoming pull requests and issues on GitHub. We are also open for collaboration with other researchers in aligning their programming language ontologies with ours, either directly or indirectly through DBpedia.

[24] https://github.com/apple/swift-evolution.
[25] https://lod-cloud.net/dataset/Swift.
[26] https://forums.swift.org/c/related-projects/25.

Quality and Validation. The RDF serialization of the Swift vocabulary validated successfully according to the W3C RDF Validation Service[27]. We also used *qSKOS* [17] to check the quality of the SKOS vocabulary, with no *errors* found. The encountered warnings can be justified as follows: As previously mentioned, the 31 concepts manually added do not include a documentation property, but are linked to an excerpt of the Swift book through a 1-hop relation. The concepts `swift:Attribute` and `swift:Comment` are indeed not connected to other concepts in the Swift vocabulary, which is due to their true nature, but they are linked to matching DBpedia entities. Finally, reciprocal relations are not included, as `skos:related` is an `owl:SymmetricProperty`, whereas `skos:broader` and `skos:hasTopConcept` declare inverse properties (`owl:inverseOf`).

6 Conclusion and Future Work

In this paper, we presented an SKOS-based vocabulary on the Swift programming language. Previous work on other languages has shown the advantages of such domain ontologies, mainly in the context of TEL applications. Using a glossary-based modeling approach, we created a controlled set of concepts covering the majority of both syntactical and semantical elements for this increasingly popular language. Our vocabulary is aligned with DBpedia and registered in the LOD Cloud, making it 5-star open data. Furthermore, we have showcased applications of the vocabulary, both in an extension for the popular Visual Studio Code editor, as well as in form of annotations on learning material, such as quiz activities or hashtags on lab sheets. The latter, which is being actively used in our Swift course enables scaffolding support by integrating relevant resources from the Swift book. The Swift vocabulary can be beneficial to fellow programming teachers and Semantic Web researchers alike.

For future work, we plan on integrating a separate concept scheme on SwiftUI, a new framework for declarative user interface design for Apple's platforms. Furthermore, prerequisite relations could be introduced in the vocabulary, to determine learning paths. These relations could be deduced from the structure of the Swift book, and could ultimately be used to sequence other learning resources (e.g., videos) likewise. Finally, Swift code on GitHub could be indexed for concepts from the Swift vocabulary, to provide code examples on certain topics to students.

Acknowledgments. We would like to thank Rubén Manrique from the Universidad de los Andes (Bogotá, Colombia) for his valuable input on Virtuoso.

[27] https://www.w3.org/RDF/Validator/.

References

1. de Aguiar, C.Z., de Almeida Falbo, R., Souza, V.E.S.: OOC-O: a reference ontology on object-oriented code. In: Laender, A.H.F., Pernici, B., Lim, E.-P., de Oliveira, J.P.M. (eds.) ER 2019. LNCS, vol. 11788, pp. 13–27. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33223-5_3
2. Alnusair, A., Zhao, T.: Using ontology reasoning for reverse engineering design patterns. In: Ghosh, S. (ed.) MODELS 2009. LNCS, vol. 6002, pp. 344–358. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12261-3_32
3. Andrews, P., Zaihrayeu, I., Pane, J.: A classification of semantic annotation systems. Semant. Web **3**(3), 223–248 (2012). https://doi.org/10.3233/SW-2011-0056
4. Atzeni, M., Atzori, M.: CodeOntology: RDF-ization of source code. In: d'Amato, C., et al. (eds.) ISWC 2017. LNCS, vol. 10588, pp. 20–28. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68204-4_2
5. Baker, T., Bechhofer, S., Isaac, A., Miles, A., Schreiber, G., Summers, E.: Key choices in the design of Simple Knowledge Organization System (SKOS). J. Web Semant. **20**, 35–49 (2013). https://doi.org/10.1016/j.websem.2013.05.001
6. De Meester, B., Dimou, A., Verborgh, R., Mannens, E.: An ontology to semantically declare and describe functions. In: Sack, H., Rizzo, G., Steinmetz, N., Mladenić, D., Auer, S., Lange, C. (eds.) ESWC 2016. LNCS, vol. 9989, pp. 46–49. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47602-5_10
7. Dehors, S., Faron-Zucker, C.: QBLS: a semantic web based learning system. In: Proceedings of EdMedia: World Conference on Educational Media and Technology 2006, pp. 2795–2802. Association for the Advancement of Computing in Education (AACE) (2006)
8. Diatta, B., Basse, A., Ouya, S.: PasOnto: ontology for learning Pascal programming language. In: 2019 IEEE Global Engineering Education Conference (EDUCON), pp. 749–754 (2019). https://doi.org/10.1109/EDUCON.2019.8725092
9. Epure, C., Iftene, A.: Semantic analysis of source code in object oriented programming. A case study for C#. Romanian J. Human-Comput. Interact. **9**(2), 103–118 (2016)
10. Garijo, D.: WIDOCO: a wizard for documenting ontologies. In: d'Amato, C., et al. (eds.) ISWC 2017. LNCS, vol. 10588, pp. 94–102. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68204-4_9
11. Grévisse, C., Botev, J., Rothkugel, S.: An extensible and lightweight modular ontology for programming education. In: Solano, A., Ordoñez, H. (eds.) CCC 2017. CCIS, vol. 735, pp. 358–371. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66562-7_26
12. Grévisse, C., Rothkugel, S., Reuter, R.A.P.: Scaffolding support through integration of learning material. Smart Learn. Environ. **6**(1), 1–24 (2019). https://doi.org/10.1186/s40561-019-0107-0
13. Hosseini, R., Brusilovsky, P.: JavaParser: a fine-grain concept indexing tool for Java problems. In: First Workshop on AI-supported Education for Computer Science (AIEDCS) at the 16th Annual Conference on Artificial Intelligence in Education, pp. 60–63 (2013)
14. Ivanova, T.: Bilingual ontologies for teaching programming in Java. In: Proceedings of the International Conference on Information Technologies, pp. 182–194 (2014). https://doi.org/10.13140/2.1.2126.4967
15. Kernighan, B.W., Ritchie, D.M.: The C Programming Language. Prentice Hall, Englewood Cliffs (1988)

16. Kouneli, A., Solomou, G., Pierrakeas, C., Kameas, A.: Modeling the knowledge domain of the Java programming language as an ontology. In: Popescu, E., Li, Q., Klamma, R., Leung, H., Specht, M. (eds.) ICWL 2012. LNCS, vol. 7558, pp. 152–159. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33642-3_16
17. Mader, C., Haslhofer, B., Isaac, A.: Finding quality issues in SKOS vocabularies. In: Zaphiris, P., Buchanan, G., Rasmussen, E., Loizides, F. (eds.) TPDL 2012. LNCS, vol. 7489, pp. 222–233. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33290-6_25
18. Miranda, S., Orciuoli, F., Sampson, D.G.: A SKOS-based framework for subject ontologies to improve learning experiences. Comput. Hum. Behav. **61**, 609–621 (2016). https://doi.org/10.1016/j.chb.2016.03.066
19. Pierrakeas, C., Solomou, G., Kameas, A.: An ontology-based approach in learning programming languages. In: 2012 16th Panhellenic Conference on Informatics, pp. 393–398 (2012). https://doi.org/10.1109/PCi.2012.78
20. Sosnovsky, S., Gavrilova, T.: Development of educational ontology for C-programming. Int. J. Inf. Theor. Appl. **13**(4), 303–308 (2006)
21. Vesin, B., Ivanović, M., Klašnja-Milićević, A., Budimac, Z.: Protus 2.0: ontology-based semantic recommendation in programming tutoring system. Expert Syst. Appl. **39**(15), 12229–12246 (2012). https://doi.org/10.1016/j.eswa.2012.04.052
22. Wagelaar, D., Van Der Straeten, R.: Platform ontologies for the model-driven architecture. Eur. J. Inf. Syst. **16**(4), 362–373 (2007). https://doi.org/10.1057/palgrave.ejis.3000686