Chapter 10

Ontology Design Patterns for Linked Data Publishing

Adila Krisnadhi, Data Semantics Laboratory, Wright State University, Dayton, OH, USA; and Faculty of Computer Science, Universitas Indonesia

Nazifa Karima, Data Semantics Laboratory, Wright State University, Dayton, OH, USA

Pascal Hitzler, Data Semantics Laboratory, Wright State University, Dayton, OH, USA

Reihaneh Amini, Data Semantics Laboratory, Wright State University, Dayton, OH, USA

Víctor Rodríguez-Doncel, Ontology Engineering Group, Universidad Politécnica de Madrid, Spain

Krzysztof Janowicz, STKO Laboratory, University of California, Santa Barbara, CA, USA

10.1. Introduction

In Chapter 1 [17], we have given a worked example on how to develop a modular ontology using ontology design patterns. In the following, we will work through an example process how to use such an ontology for publishing Linked Data. Supplementary material, such as additional examples, OWL, and RDF files, are available at `http://dase.cs.wright.edu/content/pattern-driven-linked-data-publishing-primer`.

Linked Data – or Linked *Open* Data if the reference is to Linked Data that is openly available in the sense of liberal re-use licenses [25] – refers to the publishing of data on the Web following a few very straightforward principles [7], foremost among them the use of the Resource Description Framework (RDF).[1] Linked Data has seen very signficant uptake since its inception in 2006.[2] In 2014, over

[1] `https://www.w3.org/RDF/`
[2] `http://lod-cloud.net/`

1,000 linked datasets were counted [28], a more than threefold increase over 2011. Also, the LOD Laundromat [4, 24] website[3] currently reports over 650,000 RDF documents totaling over 38,600,000,000 triples.

The benefits of Linked Data are rather obvious and well publicized [12]: simply having large amounts of data on various subjects available in structured form using a standardized format like RDF makes it easier to find, ingest, reuse this data by third parties for all kinds of purposes. Some prominent examples include the performance improvements to the IBM Watson system due to ingesting the DBpedia linked dataset [10], and the use of Linked Data at the nytimes.com portal[4] and the bbc.com portal. The latter started with the deployment of the BBC Dynamic Semantic Publishing for the 2010 World Cup content [23], which was reused for the London 2012 Olympics content [3], and later on extended to the whole range of content that may be of interest to the BBC [2].

However, reuse of Linked Data has not taken off at large scale yet, and some of the reasons for this can be traced back to the often significant effort needed in finding, understanding and assessing such datasets, and in curating them to make them fit for the task at hand [26]. In our experience, some[5] of the significant cost factors arise out of the neglect of quality metadata aspects, such as the failure to provide an ontology underlying the data graph structure; graph structure (and underlying ontology) not being modeled according to best practices as they arise, e.g., out of ontology design patterns work; large, monolithic graphs and ontologies which are very hard to inspect and understand by humans; and so forth [15, 26].

Consequently, we advocate that linked data publishing should be done in accordance with a high-quality ontology to convey, explain, and enrich the bare linked data graph. In the sequel, we assume that such an ontology has already been developed based on ontology design pattern principles. This ontology provides us with definition of vocabulary terms that will be used to annotate the data and essentially act as the schema for the data. To make things more concrete, we shall describe the thought processes and an example workflow leading to the publishing of a linked dataset with vocabulary obtained from such an ontology. For the example workflow, we shall build on the ontology developed and described in Chapter 1 [17], make several adjustments to it and tie some loose ends to make it more suitable for linked data publishing. Afterwards, we will illustrate the steps one needs to take in setting up an appropriate infrastructure to publish linked data based on the ontology so that the data is published according to the Linked Data principles [7].

10.2. Vocabulary Preparation

Our starting point is Fig. 1.9 of Chapter 1 [17], which refers to several external patterns. We now need to make precise decisions how to resolve these external references, i.e., whether to indeed import some external model, or to use a stub, or use a simplified solution for the time being. In addition, we modify the axioms

[3] `http://lodlaundromat.org/` – retrieved 24 February 2016

[4] As reported by Evan Sandhaus in his keynote talk at the 9th International Semantic Web Conference (ISWC 2010) in Shanghai, China, November 2010.

[5] But certainly not all, see e.g. [30].

from Fig. 1.12 and 1.13 in accordance to the modeling decision we are about to make below. In making these changes, however, we generally remain truthful to the original model.

- We retain the use of stubs for Chess Tournament, Chess Opening, and Chess Game Result as indicated. That is, we shall have possibly blank nodes, which are simply typed as indicated. For the Chess Opening stub, we remove hasName property and consider the ECO code sufficient for our purpose.
- We use a similar stub for Agent and Place i.e. attaching a string as name using the hasName property.
- We omit axioms involving the startsAtTime and endsAtTime property of Agent Role.
- We remove Event because it was used for design, and there seems no added value in producing the additional triples for instantiating it at this stage.
- With Event removed, the inherited information regarding Place is attached to ChessGame directly, and additionally, in place of TemporalExtent, we also attach xsd:dateTime[6] to ChessGame. This is of course an oversimplification because several games may be on the same day, and the order may matter. However, in practice simplifications are sometimes needed to go forward; we acknowledge that this particular simplification may complicate the integration with more fine-grained data, and may preclude some uses that rely on a more fine-grained modeling. PGN files, which will be the data source we will currently only look at, provide the location and dates for chess games.
- Similarly, we attach xsd:dateTime through atTime and Place through atPlace to ChessTournament, which was originally also modeled as event.
- We remove the more general scoped domain and range restrictions for subEventOf property (with Event as both domain and range of this property). The axioms that express domain and range restrictions for this property where the domain and range pairs are ChessGame and ChessTournament, as well as HalfMove and ChessGame will be retained.
- We remove HalfMoveAnnotation for the time being, mainly because republishing of comments from PGN files may be difficult in terms of copyright; the move sequence itself is unproblematic, see [29].
- We remove originatesFrom for the time being, because we will look only at PGN files as sources for data at this stage.

The resulting graph is depicted in Fig. 10.1, while the modified set of axioms can be found in Fig. 10.2.

10.3. Views and Shortcuts

An expressive schema such as the one in Fig. 10.1 is extremely helpful to retain the benefits of high-quality ontology modeling for linked data publishing [8], including ease of understanding of the graph structure; ease of schema and thus graph

[6]The prefix xsd stands for `http://www.w3.org/2001/XMLSchema#`

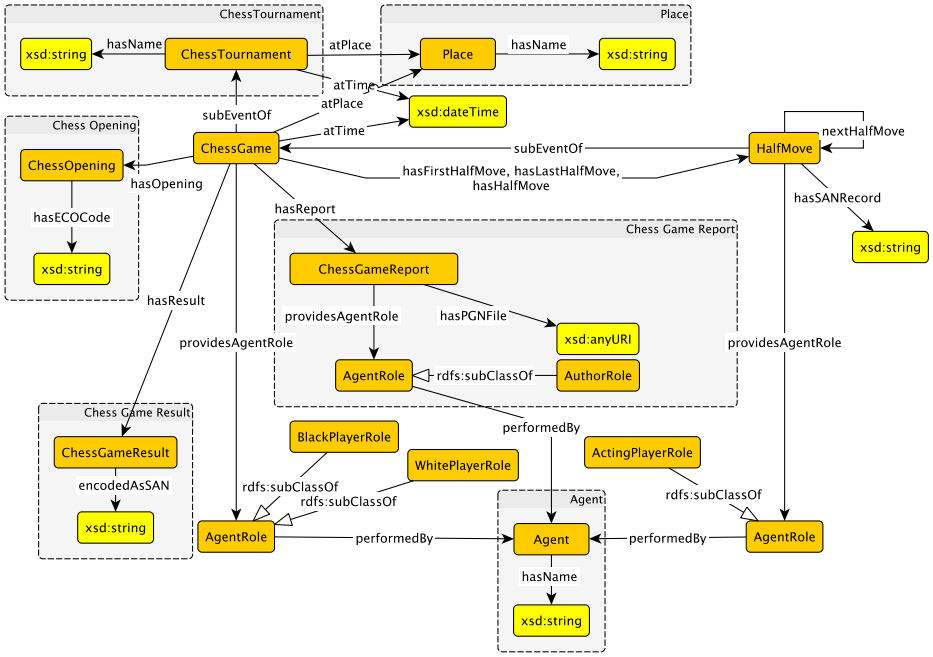

Figure 10.1. Chess Ontology modified from Fig. 1.9 of Chapter 1 [17].

extension without violating past modeling or requiring major modifications to past modeling; ease of reuse because the graph structure is easy to understand and modify; ease of integration with other data since the schema – and thus the RDF graph structure – lends itself more easily to such integration; ease of reuse of part of the RDF graph, since the schema – and thus the RDF graph – is already modularized.

However, the expressive schema also comes with a number of drawbacks: the graph structures are more complicated, and thus it takes more time and effort to cast data into it, they are more difficult to understand, and for those intending to re-use the data, they have to deal with a much more complicated schema than their specific use case requires. Furthermore, the resulting RDF graphs will usually be much larger than they would be if a much simpler schema were used.

Of course, the advantages of a complex schema are the disadvantages of a simple schema and vice versa. This raises the question about the best choice in any given situation, or the question about how to find a good compromise.

However, as we will argue and show below, we can also instead reap the best of both worlds, by moving to a *dual* schema, consisting of both a complex and one (or several) compatible simplified versions. In this case, users can choose the perspective which fits them, and even switch seamlessly between them if this is desired. We will also see below that the overhead resulting from such a dual schema is reasonable.

The idea is as follows: Once a complex schema like the one depicted in

$$\mathsf{AgentRole} \sqsubseteq (=1\ \mathsf{performedBy.Agent}) \sqcap \forall \mathsf{performedBy.Agent} \tag{10.1}$$

$$\exists \mathsf{performedBy.Agent} \sqsubseteq \mathsf{AgentRole} \tag{10.2}$$

$$\top \sqsubseteq \forall \mathsf{pAR.AgentRole} \tag{10.3}$$

$$\mathsf{ChessGame} \sqsubseteq \exists \mathsf{atPlace.Place} \sqcap \forall \mathsf{atPlace.Place} \tag{10.4}$$

$$\mathsf{ChessGame} \sqsubseteq \exists \mathsf{atTime.xsd{:}dateTime} \sqcap \forall \mathsf{atTime.xsd{:}dateTime} \tag{10.5}$$

$$\mathsf{ChessGame} \sqsubseteq \exists \mathsf{pAR.BlackPlayerRole} \sqcap \exists \mathsf{pAR.WhitePlayerRole} \tag{10.6}$$

$$\exists \mathsf{subEventOf.ChessTournament} \sqcup \exists \mathsf{hasOpening.ChessOpening} \sqsubseteq \mathsf{ChessGame} \tag{10.7}$$

$$\exists \mathsf{hasResult.ChessGameResult} \sqcup \exists \mathsf{hasReport.ChessGameReport} \sqsubseteq \mathsf{ChessGame} \tag{10.8}$$

$$\mathsf{ChessGame} \sqsubseteq \forall \mathsf{subEventOf.ChessTournament} \sqcap \forall \mathsf{hasOpening.ChessOpening} \tag{10.9}$$

$$\mathsf{ChessGame} \sqsubseteq \forall \mathsf{hasResult.ChessGameResult} \sqcap \forall \mathsf{hasReport.ChessGameReport} \tag{10.10}$$

$$\mathsf{BlackPlayerRole} \sqcup \mathsf{WhitePlayerRole} \sqsubseteq \mathsf{AgentRole} \sqcap (=1\ \mathsf{pAR}^{-}\mathsf{.ChessGame}) \tag{10.11}$$

$$\mathsf{ChessGame} \sqsubseteq (=1\ \mathsf{hasFirstHalfMove.HalfMove}) \sqcap (=1\ \mathsf{hasLastHalfMove.HalfMove}) \tag{10.12}$$

$$\mathsf{ChessGame} \sqsubseteq (=1\ \mathsf{hasLastHalfMove.HalfMove}) \tag{10.13}$$

$$\mathsf{hasHalfMove} \sqsubseteq \mathsf{subEventOf}^{-} \tag{10.14}$$

$$\mathsf{hasFirstHalfMove} \sqsubseteq \mathsf{hasHalfMove} \tag{10.15}$$

$$\mathsf{hasLastHalfMove} \sqsubseteq \mathsf{hasHalfMove} \tag{10.16}$$

$$\mathsf{HalfMove} \sqsubseteq \mathsf{Event} \sqcap \exists \mathsf{pAR.ActingPlayerRole} \sqcap (=1\ \mathsf{hasHalfMove}^{-}\mathsf{.ChessGame}) \tag{10.17}$$

$$\mathsf{ActingPlayerRole} \sqsubseteq \mathsf{AgentRole} \sqcap (=1\ \mathsf{pAR}^{-}\mathsf{.HalfMove}) \tag{10.18}$$

$$\mathsf{HalfMove} \sqsubseteq (\leq 1\ \mathsf{nextHalfMove.HalfMove}) \sqcap \neg \exists \mathsf{nextHalfMove.Self} \tag{10.19}$$

$$\exists \mathsf{subEventOf.ChessGame} \sqcup \exists \mathsf{nextHalfMove.HalfMove} \sqsubseteq \mathsf{HalfMove} \tag{10.20}$$

$$\exists \mathsf{hasSANRecord.xsd{:}string} \sqsubseteq \mathsf{HalfMove} \tag{10.21}$$

$$\mathsf{HalfMove} \sqsubseteq \forall \mathsf{subEventOf.ChessGame} \sqcap \forall \mathsf{nextHalfMove.HalfMove} \tag{10.22}$$

$$\mathsf{HalfMove} \sqsubseteq \forall \mathsf{hasSANRecord.xsd{:}string} \tag{10.23}$$

$$\mathsf{ChessTournament} \sqsubseteq \exists \mathsf{atPlace.Place} \sqcap \forall \mathsf{atPlace.Place} \tag{10.24}$$

$$\mathsf{ChessTournament} \sqsubseteq \exists \mathsf{atTime.xsd{:}dateTime} \sqcap \forall \mathsf{atTime.xsd{:}dateTime} \tag{10.25}$$

$$\mathsf{ChessTournament} \sqcup \mathsf{Place} \sqcup \mathsf{Agent} \sqsubseteq \forall \mathsf{hasName.xsd{:}string} \tag{10.26}$$

$$\exists \mathsf{hasECOCode.xsd{:}string} \sqsubseteq \mathsf{ChessOpening} \tag{10.27}$$

$$\mathsf{ChessOpening} \sqsubseteq \forall \mathsf{hasECOCode.xsd{:}string} \tag{10.28}$$

$$\exists \mathsf{encodedAsSAN.xsd{:}string} \sqsubseteq \mathsf{ChessGameResult} \tag{10.29}$$

$$\mathsf{ChessGameResult} \sqsubseteq \forall \mathsf{encodedAsSAN.xsd{:}string} \tag{10.30}$$

$$\mathsf{ChessGameReport} \sqsubseteq \exists \mathsf{pAR.AuthorRole} \tag{10.31}$$

$$\mathsf{AuthorRole} \sqsubseteq \mathsf{AgentRole} \sqcap (=1\ \mathsf{pAR}^{-}\mathsf{.ChessGameReport}) \tag{10.32}$$

$$\exists \mathsf{hasPGNFile.xsd{:}anyURI} \sqsubseteq \mathsf{ChessGameReport} \tag{10.33}$$

$$\mathsf{ChessGameReport} \sqsubseteq \forall \mathsf{hasPGNFile.xsd{:}anyURI} \tag{10.34}$$

$$\texttt{DisjointClasses}(\mathsf{AgentRole}, \mathsf{Agent}, \mathsf{ChessGame}, \mathsf{ChessTournament}, \mathsf{HalfMove}, \mathsf{Place}, \mathsf{ChessOpening}, \mathsf{ChessGameResult}, \mathsf{ChessGameReport}) \tag{10.35}$$

$$\texttt{DisjointClasses}(\mathsf{BlackPlayerRole}, \mathsf{WhitePlayerRole}, \mathsf{ActingPlayerRole}, \mathsf{AuthorRole}) \tag{10.36}$$

Figure 10.2. Chess Game axioms after simplifying Fig. 1.12 and 1.13 according to Section 10.2; pAR stands for providesAgentRole.

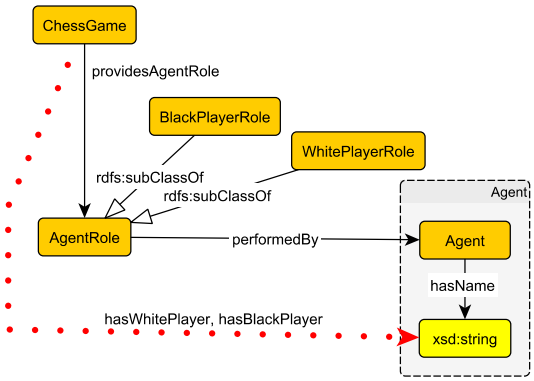

Figure 10.3. Example for a shortcut.

Fig. 10.1 has been developed, a simplified *view* can be established by way of so-called *shortcuts* through the graph [16]. Compilation of data from the complex schema to the simplified view is straightforward using rules, as we will see below.

We first give a small example before providing a complete simplified view for our chess ontology. Consider Fig. 10.3, which shows the part of the ontology connecting ChessGame to a player's name; the player could be the white or the black player. The dotted red arrow in this graph indicates two shortcuts, which we name hasWhitePlayer, for the WhitePlayerRole, and hasBlackPlayer for the BlackPlayerRole.

Given a populated ontology, we can then materialize the two shortcuts, by firing the following rules – pAR is used as abbreviation for providesAgentRole.

$$\begin{aligned}\mathsf{ChessGame}(x) \wedge \mathsf{pAR}(x,y) \wedge \mathsf{WhitePlayerRole}(y) \wedge \mathsf{performedBy}(y,z)\\ \wedge\, \mathsf{Agent}(z) \wedge \mathsf{hasName}(z,s) \rightarrow \mathsf{hasWhitePlayer}(x,s)\\ \mathsf{ChessGame}(x) \wedge \mathsf{pAR}(x,y) \wedge \mathsf{BlackPlayerRole}(y) \wedge \mathsf{performedBy}(y,z)\\ \wedge\, \mathsf{Agent}(z) \wedge \mathsf{hasName}(z,s) \rightarrow \mathsf{hasBlackPlayer}(x,s)\end{aligned}$$

These rules – like most rules in fact [18] – can also be expressed in description logics, and thus can often also be expressed in OWL DL [20]. We just give an example for the first rule and refer the interested reader to [18, 20] for further details on how to do this transformation. The main idea is that we associate each class name in the rule with a property that is not previously occurring in the ontology. In the case of the first rule above, we associate ChessGame with R_1, WhitePlayerRole with R_2, and Agent with R_3 where all R_1, R_2, and R_3 are freshly introduced. The rule then becomes the following set of axioms and the properties R_1, R_2, and R_3 are called the *rolifications* of ChessGame, WhitePlayerRole, and

Agent, respectively [18].

$$\begin{aligned} \mathsf{ChessGame} &\sqsubseteq \exists R_1.\mathrm{Self} \\ \mathsf{WhitePlayerRole} &\sqsubseteq \exists R_2.\mathrm{Self} \\ \mathsf{Agent} &\sqsubseteq \exists R_3.\mathrm{Self} \\ R_1 \circ \mathsf{pAR} \circ R_2 \circ \mathsf{performedBy} \circ R_3 \circ \mathsf{hasName} &\sqsubseteq \mathsf{hasWhitePlayer} \end{aligned}$$

In this particular case, though, the property chain in the last of these axioms cannot be converted into OWL DL, since hasName is a datatype property. This limitation of OWL was discussed in [19].

Mapping in the other direction, from the simplified view to the complex schema, is not as straightforward. First of all, even expressing the bare logical bones of this transformation in a knowledge representation language used in web ontologies is tricky, see the discussion in [19]. Furthermore, the transformation necessitates either generating blank nodes or, if the newly introduced nodes need to be named, minting new identifiers. In the aforementioned rules, the new nodes correspond to the variable y and z. Even more complicated is the fac that co-reference resolution may need to be performed on some of these new nodes, e.g., on the instances of Agent, since they may identify some entity occurring elsewhere in the dataset. That this direction is non-trivial indicates again that there is added value – more semantics – in the expressive schema.

The transformation can obviously be done using software specially written for this purpose, typically by making use of RDF APIs, some of which are listed at the end of Section 10.6.1.1. Alternatively, one could express the blank node generation and URI minting inline within a SPARQL query [11] with the help of a few SPARQL built-in functions and some naming convention. This way of expressing the transformation using SPARQL shall be briefly explained in Section 10.6.2

The complete set of shortcuts for our ontology are indicated by dotted arrows in Fig. 10.4. The corresponding rules are given in Fig. 10.5. The complete simplified view is depicted in Fig. 10.6. We are now ready to get down to a more concrete implementation work to publish linked data based on the Chess ontology pattern.

10.4. URI Minting

In Linked Data context, each term in linked datasets and ontologies are called *resources*. To conform with the Linked Data principles [7], we need to make all such resources Web-dereferenceable by assigning a *Uniform Resource Identifier* (URI) to each of them. Some resources may have been defined externally in another linked dataset, which is not under our control. For them, a URI is typically already provided and we just need to use it. For the rest, however, we need to *mint* appropriate URIs by following the design principles below [14].

1. Every resource, aside from literal values, should be assigned a HTPP URI as its identifier.[7]

[7] One can use the plain HTTP URI, i.e., with `http:` prefix, or HTTPS, i.e., with `https:`

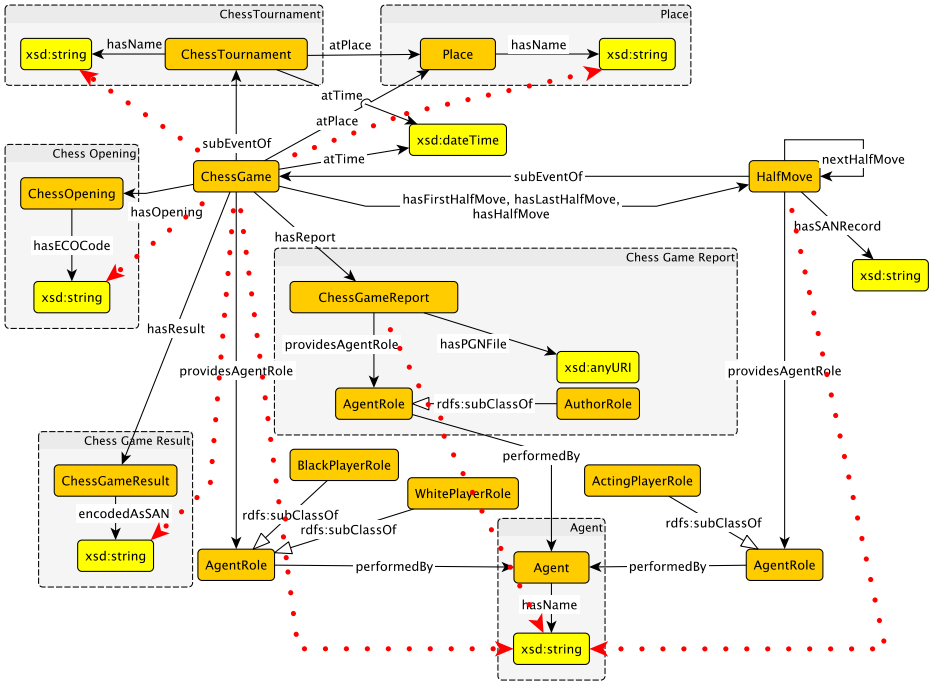

Figure 10.4. Shortcuts indicated by dotted arrows.

$$\begin{aligned}
&\mathsf{ChessGame}(x) \wedge \mathsf{pAR}(x,y) \wedge \mathsf{WhitePlayerRole}(y) \wedge \mathsf{performedBy}(y,z)\\
&\qquad \wedge \mathsf{Agent}(z) \wedge \mathsf{hasName}(z,s) \rightarrow \mathsf{hasWhitePlayer}(x,s)\\
&\mathsf{ChessGame}(x) \wedge \mathsf{pAR}(x,y) \wedge \mathsf{BlackPlayerRole}(y) \wedge \mathsf{performedBy}(y,z)\\
&\qquad \wedge \mathsf{Agent}(z) \wedge \mathsf{hasName}(z,s) \rightarrow \mathsf{hasBlackPlayer}(x,s)\\
&\mathsf{ChessGame}(x) \wedge \mathsf{hasResult}(x,y) \wedge \mathsf{ChessGameResult}(y) \wedge \mathsf{encodedAsSAN}(y,s)\\
&\qquad \rightarrow \mathsf{hasResultSAN}(x,s)\\
&\mathsf{ChessGame}(x) \wedge \mathsf{hasOpening}(x,y) \wedge \mathsf{ChessOpening}(y) \wedge \mathsf{hasECOCode}(y,s)\\
&\qquad \rightarrow \mathsf{hasOpeningECO}(x,s)\\
&\mathsf{ChessGame}(x) \wedge \mathsf{subEventOf}(x,y) \wedge \mathsf{ChessTournament}(y) \wedge \mathsf{hasName}(y,s)\\
&\qquad \rightarrow \mathsf{atChessTournament}(x,s)\\
&\mathsf{ChessGame}(x) \wedge \mathsf{subEventOf}(x,y) \wedge \mathsf{ChessTournament}(y) \wedge \mathsf{atPlace}(y,z)\\
&\qquad \wedge \mathsf{Place}(z) \wedge \mathsf{hasName}(z,s) \rightarrow \mathsf{atPlaceNamed}(x,s)\\
&\mathsf{ChessGameReport}(x) \wedge \mathsf{pAR}(x,y) \wedge \mathsf{AuthorRole}(y) \wedge \mathsf{performedBy}(y,z)\\
&\qquad \wedge \mathsf{Agent}(z) \wedge \mathsf{hasName}(z,s) \rightarrow \mathsf{hasAuthor}(x,s)\\
&\mathsf{HalfMove}(x) \wedge \mathsf{pAR}(x,y) \wedge \mathsf{ActingPlayerRole}(y) \wedge \mathsf{performedBy}(y,z)\\
&\qquad \wedge \mathsf{Agent}(z) \wedge \mathsf{hasName}(z,s) \rightarrow \mathsf{playedBy}(x,s)
\end{aligned}$$

Figure 10.5. Shortcut rules leading to our simplified view.

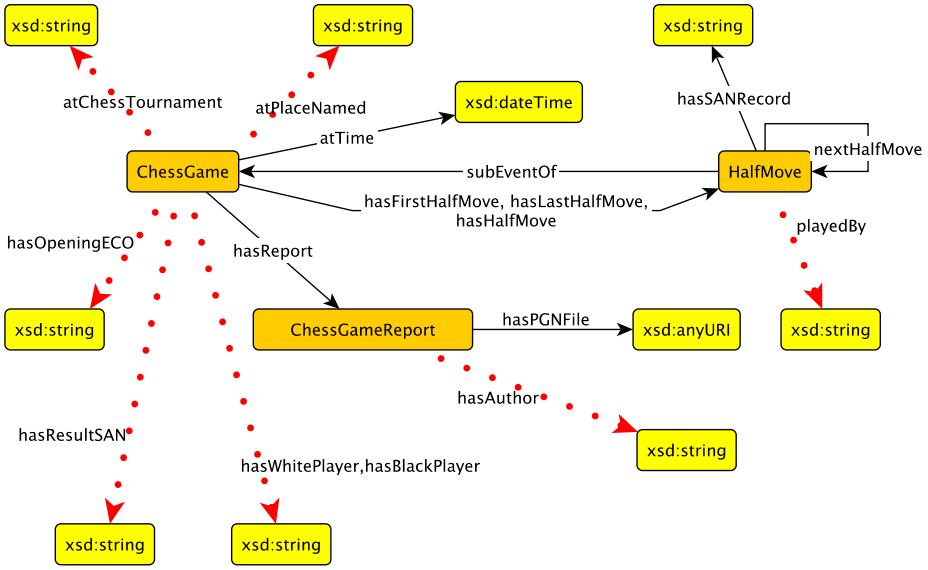

Figure 10.6. Simplified view of the ontology.

2. Data publishers should provide a machine-readable presentation for each URI they maintain in order to enable the URI to be looked up and dereferenced.

3. URIs should be persistent. In the literature, this usually means that they should not contain anything that will likely change such as session tokens or other state information. More generally, however, the persistence of URIs may go beyond that. That is, if a URI should live beyond the lifetime of the underlying infrastructure or the organization who maintains it. In short, once a URI is minted and published online, then it should ideally live forever.

4. URIs should be assumed to be opaque: agents and Web clients accessing a URI should not parse or read into the URI to infer anything about the referenced resource.

Note that URIs in Linked Data are used to identify not just Web documents, but also non-document resources, including real-world objects or even abstract entities. When designing a URI naming scheme, it is thus important to account for the distinction between a thing and the Web document about that thing.

prefix. The latter supposedly indicates that communication to the URI is done with encryption and authentication. Although there is a clear consensus of the need for a secure web for everyone, which motivated projects such as HTTPS Everywhere (`https://www.eff.org/Https-everywhere`) and Let's Encrypt (`https://letsencrypt.org/`), there is a debate as to whether a wholesale move to the HTTPS protocol is the best solution in the long run [5].

For example, consider the website of DaSeLab at Wright State University whose structure may look like this:

- `http://dase.cs.wright.edu/` – the homepage of DaSeLab;
- `http://dase.cs.wright.edu/people/adila-krisnadhi` – the homepage of Adila Krisnadhi in DaSeLab;
- `http://dase.cs.wright.edu/people/pascal-hitzler` – the homepage of Pascal Hitzler in DaSeLab.

Here, all three URIs reside in the same *URI namespace*, which is the DaSeLab namespace given by the URI: `http://dase.cs.wright.edu/`. Moreover, the latter two URIs above reside in a sub-namespace of the DaSeLab namespace given by `http://dase.cs.wright.edu/people/`. Namespaces here are simply a means to ensure the global uniqueness of identifiers. More precisely, within the `http://dase.cs.wright.edu/people/` namespace, the Web resource identified by `http://dase.cs.wright.edu/people/adila-krisnadhi` uniquely corresponds to the homepage of Adila Krisnadhi. Furthermore, on the Web, the URI `http://dase.cs.wright.edu/people/` uniquely correspond to a particular part of the DaSeLab namespace. As a result, one could expect that the URI `http://dase.cs.wright.edu/people/adila-krisnadhi` can be resolved only to a unique resource, which is the homepage of Adila Krisnadhi in DaSeLab, and not something else.

Now, we wish to use URIs also to identify DaSeLab and the two people: Adila Krisnadhi and Pascal Hitzler as *objects*. Since the URIs above have been used to identify the *homepages* of the DaSeLab, and the two people, one should not use them as URIs to identify the three entities as *objects*, i.e., different URIs need to be used. This is to avoid the confusion when using the URIs as part of the statements (concretely, RDF triples) in the linked dataset or ontologies: if we use, for example, `http://dase.cs.wright.edu/people/adila-krisnadhi` as the URI for the person Adila Krisnadhi, then the RDF triple

```
<http://dase.cs.wright.edu/people/adila-krisnadhi> foaf:givenName "Adila".
```

would actually be read as "the homepage of Adila Krisnadhi has a given name Adila".

10.4.1. Hash URIs and 303 URIs

There are two URI schemes commonly used by Semantic Web applications for identifying non-document resources [27]. They are *hash URIs* and *303 URIs*. Both schemes allow the server to serve both a machine-readable and a human-readable presentations of the same resource, depending on the client's request.

Hash URIs are URIs that contain a *fragment*, a special part of the URI preceded by a hash symbol ('`#`'). When a hash URI is retrieved from the server, HTTP protocol specified that the client application has to strip off the fragment part prior to sending the request to the server. In other words, such an URI, which includes the hash, cannot be retrieved directly, and thus does not necessarily identify a Web document. Therefore, we can use it to identify non-document

resources without raising confusion. For example, we could use the following URIs to identify DaSeLab (the organization), Adila Krisnadhi (the person), and Pascal Hitzler (the person), respectively:

- `http://dase.cs.wright.edu/about#daselab`
- `http://dase.cs.wright.edu/about#adila-krisnadhi`
- `http://dase.cs.wright.edu/about#pascal-hitzler`

When a client requests `http://dase.cs.wright.edu/about#adila-krisnadhi`, it strips off the fragment and only requests `http://dase.cs.wright.edu/about` from the server. Without content negotation, the server can respond by returning a machine-readable RDF document containing triples describing the three resources above. Meanwhile, with content negotation, it could send back a machine-readable RDF document or a human-readable HTML document, depending on the client's request. In this case, the client can indicate the preference in the `Accept` header by specifying, e.g., `application/rdf+xml` if the former is preferred, or `text/html` if the latter is preferred instead.

An alternative solution is to employ a special HTTP status code, `303 See Other`, hence the URIs are called 303 URIs. On the surface, there is no apparent distinction on 303 URIs except that they typically contain no fragment part. In this scheme, the URIs may look as follows[8] where we use the URI namespace `http://dase.cs.wright.edu/id/`:

- `http://dase.cs.wright.edu/id/daselab` – DaSeLab, the organization
- `http://dase.cs.wright.edu/id/adila-krisnadhi` – Adila Krisnadhi, the person
- `http://dase.cs.wright.edu/id/pascal-hitzler` – Pascal Hitzler, the person

The intuition is that according to the Web architecture, it is not appropriate to return a 200 status code[9] when the above URIs are requested by a client because the URIs actually possess no suitable presentation. Nonetheless, it is always more useful to provide more information about the requested resource, which is also in accordance with Linked Data principles. Here, the server should respond with a 303 status code, which indicates a redirection, and the response also includes `Location` header providing the location of the Web document containing information about the requested resource. Content negotiation can then be used to decide whether to serve a machine-readbale RDF document or a human-readable HTML.

For example, when `http://dase.cs.wright.edu/id/adila-krisnadhi` is requested by a client, the server returns a 303 code and redirects the request to the address given by the `Location` header. Here, we have two slightly different variants. In the first one, the server would redirect to a generic document, say `http://dase.cs.wright.edu/doc/adila-krisnadhi`, and then perform the content negotiation. If an RDF document is requested, it then returns an RDF document, say `http://dase.cs.wright.edu/doc/adila-krisnadhi.rdf`,

[8] Note that there is nothing special with the occurrence of `id` as part of the path in the URIs – one could use `data`, `foo`, or any other string to specify the path component of the URIs.

[9] OK status; meaning that the request has succeeded and the requested resource is returned.

while if an HTML document is requested, then it returns an HTML one, say `http://dase.cs.wright.edu/doc/adila-krisnadhi.html`. In the second variant, the server would perform content negotiation immediately. If the machine-readable RDF content is requested, then the server *redirects* to an RDF document, say `http://dase.cs.wright.edu/data/adila-krisnadhi`. Meanwhile, if HTML content is requested, then the server redirects to an HTML document, say `http://dase.cs.wright.edu/people/adila-krisnadhi`.

10.4.2. URI Naming Scheme

The next aspect that needs to be considered is the URI naming scheme. This aspect is, however, less clear than the URI scheme. Several organizations and communities have provided their own recommendations, conventions, and guidelines, which differ from each other. This is of course not surprising given that in the end, every URI is maintained by a particular party that is also responsible to provide an infrastructure that ensures the URI can be dereferenced.

For example, W3C uses a URI naming scheme of the form, among others, `http://www.w3.org/YYYY/MM/ssss` where YYYY and MM are decimal digits representing the year and month of URI allocation, and ssss is a short string [6].[10] The UK government uses `http://{domain}/id/{concept}/{reference}` as a naming scheme for URIs representing identifiers of entities in the UK government data [9], e.g., `http://education.data.gov.uk/id/school/78` is an identifier of some school in the UK. More examples of the different URI formats as well as their design rules and managements across several government agencies and standardization bodies can be found in a survey by Archer, et al. [1], while a general style guidelines that takes into account multilingual web can be consulted from a paper by Montiel-Ponsoda, et al. [22]. In such design rules and guidelines, one can also usually find recommendations such as preference of short URIs over longer ones, which case policy should be chosen (e.g., all lower case letters, CamelCase, etc.), how to deal with word separator if a URI is formed from several words, when using code value is appropriate, how versioning can be handled, etc. Before we proceed though, we need to choose appropriate namespaces for our identifiers.

If the data as well as ontologies being published are produced and maintained by a particular organization, one could use that organization's URI namespaces for the data and elements of the schema. This is usually a good choice assuming that the organization is not temporary in nature, and there is a clear policy that ensures the persistence of the URIs in case there is change in the underlying infrastructure. Unfortunately, this is in the end impossible to guarantee: organizations may cease to exist, and infrastructure and policies may change. Thus, relying on an organization's URI namespace may prove tricky in the long run.

To help extending the lifetime of URIs beyond the lifetime of the maintaining organizations, the Linked Data community has been using community-supported online services that provide permanent re-direction for Web applications. Such services facilitate data publishers to reserve a URI that is independent of the

[10] Quoted directly from the document: "ssss is a short string not causing confusion, alarm, or embarrassment."

actual address of the server that performs the dereferencing. When a user requests such a URI, the service will redirect the request to the actual location of the information on the Web. If this actual location changes, due to organizational, policy, or infrastructure changes, the URI maintainer can simply modify the redirection in the service to point to the new location. The URI itself is not changed, and hence, does not affect any user applications depending on the URI. Such services have been historicaly called Persistent Uniform Resource Locators (PURL) services,[11] and more recently, there is an ongoing community effort to provide a more unified form of such a service, called w3id,[12] through the W3C Permanent Identifier Community Group.[13]

For our example workflow here, we wish to distinguish two general categories of identifiers. The first category consists of ontology elements, i.e., classes, properties, and named individuals defined by the ontology. The second comprises non-literal resources in the data that are not vocabulary terms from the ontology. For both categories, we take the liberty of claiming `https://w3id.org/rdfchess/` as the top-level namespace.[14] For ontology elements, we assign a sub-namespace given by the ontology names. For non-literal resources that are not a vocabulary term, we reserve `https://w3id.org/rdfchess/id/` sub-namespace. Furthermore, following some of the guidelines from Montiel-Ponsoda, et al. [22], we decide on the following naming scheme.

- We use meaningful local names for ontology elements, and opaque names for data instances. For the latter, we use randomly generated strings as part of the identifiers.
- We employ CamelCase for class and property names from the ontology and lower case alphanumeric letters for named individuals from the ontology and non-literal in the RDF datasets. For the latter, underscore character ('_') is used as the token separator.
- We also employ lower case alphanumeric letters for naming the ontology where the dash character ('-') is used as the token separator if needed.
- We employ hash URIs for the ontology elements and 303 URIs for instances in the data.
- As much as possible, we provide human-readable labels for each resource in the ontology and linked datasets using `rdfs:label` property.
- We ignore versioning issues of the ontology elements to simplify the discussion.

The resulting URI patterns are provided below.

- The pattern `https://w3id.org/rdfchess/id/ssss` is used for non-literal resources in the linked datasets that are not a vocabulary term. Here, `ssss` is a random, unambiguous string. For example, we could use the

[11] A community site at `http://www.purlz.org/` listed several PURL services, including the oldest one called `purl.org`, which is hosted at OCLC and has been used for more than 15 years by the community.

[12] `https://w3id.org`; unlike most of other PURL services, this one operates in HTTPS-only mode.

[13] `https://www.w3.org/community/perma-id/`

[14] Of course, this namespace needs to be *actually* reserved, i.e., by actually performing all the steps specified in `https://w3id.org`

URI https://w3id.org/rdfchess/id/gam19e02 to represent an instance of the class ChessGame; the string gam19e02 is generated such that within the https://w3id.org/rdfchess/id/ namespace, it unambigously refers to this particular instance of ChessGame.

- The pattern https://w3id.org/rdfchess/ontology-name is used for the name of ontologies. For example, the URI of the complex version of our dual schema could be https://w3id.org/rdfchess/chessonto.
- The pattern https://w3id.org/rdfchess/ontology-name#ClassName is employed for class names defined in the ontology named ontology-name. For example, the ChessGame class in the ontology named above could have the URI: https://w3id.org/rdfchess/chessonto#ChessGame
- https://w3id.org/rdfchess/ontology-name#propertyName is used for property names defined in the ontology named ontology-name. For example, https://w3id.org/rdfchess/chessonto#hasHalfMove could be used for the hasHalfMove.
- https://w3id.org/rdfchess/ontology-name#named_individual is the URI pattern for named individuals defined in the ontology ontology-name. E.g., https://w3id.org/rdfchess/chessonto#grandmaster could be used to refer to the grandmaster title of chess players. Note that, although named individuals in the ontology are logically treated in the same way as instances of a class in the linked dataset that populates the ontology,[15] they use a different URI pattern. The reason is that named individuals are directly declared in the ontology, usually because they represent some controlled vocabulary in the domain; they are not viewed as data.

10.5. Publishing the Schema

After specifying the URI patterns, the next step is to prepare the OWL file(s) that contain all the axioms in the ontology. Since we are using dual schema, we essentially have two slightly different ontologies corresponding to Fig. 10.1 and 10.6, which can be merged into one according to Figure 10.4. So, our approach is to prepare the following OWL files. These OWL files can be created manually using any ontology editor, such as Protégé[16] or NeOn Toolkit.[17] Alternatively, one can also generate the OWL files programmatically using libraries such as OWL API.[18] For our example workflow, we shall use Protégé to create the OWL files. Please consult the RDF and OWL Primer in the appendix of this book [21] as well as the appropriate documentation of the ontology editor of choice for more information on how to write OWL axioms into the OWL files.

- chessonto.owl
  This OWL file captures the ontology according to Fig. 10.1, which forms the complex version of the dual schema. The corresponding axioms are given by Fig. 10.2. As ontology URI, we assign https://w3id.org/rdfchess/chessonto.

[15] In the description logic lingua, they are all ABox individuals.
[16] http://protege.stanford.edu/
[17] http://neon-toolkit.org/
[18] http://owlcs.github.io/owlapi/

The URIs for class names, property names, and named individuals in this OWL file are defined according to the patterns in the previous section.

- `chessonto-view.owl`
  This OWL file captures the ontology according to Fig. 10.6, expressing a simplified version of the dual schema. As ontology URI, we assign `https://w3id.org/rdfchess/chessonto-view`. Note that since this ontology defines the shortcuts involving classes and properties which are already in `chessonto.owl`, it does not introduce new URIs for those classes and properties, and instead, simply reuses the URIs already defined in `chessonto.owl`. For newly introduced properties such as hasWhitePlayer, hasBlackPlayer, etc., this ontology declares URIs for them according to the pattern in the previous section. Observe that all of the newly introduced properties in `chessonto-view.owl` are datatype properties. Hence, none of the rules in Fig. 10.5 can be expressed as OWL axioms. Instead, we can write them as DL-Safe SWRL rules [13], which are supported at least by OWL API and Protégé.[19] For example, the first rule in Fig. 10.5 can be expressed as the following SWRL rule where the prefix `chon:` refers to `<https://w3id.org/rdfchess/chessonto#>` and `chonv:` refers to `<https://w3id.org/rdfchess/chessonto-view#>`:

```
chon:ChessGame(?x) ^ chon:providesAgentRole(?x,?y)
         ^ chon:WhitePlayerRole(?y) ^ chon:performedBy(?y,?z)
         ^ chon:Agent(?z) -> chonv:hasWhitePlayer(?x,?z)
```

- `chessonto-full.owl`
  This OWL file captures the ontology according to Fig. 10.4. We assign `https://w3id.org/rdfchess/chessonto-full` as the ontology URI. This ontology does not define any class, property, or named individual. Instead, it simply imports the previous two OWL files.

The first OWL file is intended for users who wish to populate the Chess ontology before the shortcuts were created. The second is intended for users who prefer to populate the 'shortcut' version. Meanwhile, the third eases users who wish to access the whole dual schema.

After the OWL files are created, we need to publish them on the Web. This publishing step is not strictly needed for generating the linked datasets. Nevertheless, it is necessary if we wish to comply to the Linked Data principles. In particular, we need to ensure that the URIs of the ontologies as well as classes, properties, and individuals in the OWL files to be Web dereferenceable. Since we are using hash URIs, it suffices if the content negotiation module of the Web server[20] on which we host the OWL files is correctly configured so that whenever an ontology URI is requested, the correct OWL file is served, e.g., if `https://w3id.org/rdfchess/chessonto` is requested, then the server should serve `chessonto.owl` to the client. In this setting, whenever a class/property/individual name is requested, HTTP protocol would automatically strip off

[19] See `https://github.com/protegeproject/swrlapi/wiki/SWRLLanguageFAQ`

[20] For example, see `http://httpd.apache.org/docs/current/content-negotiation.html` for more information on how to configure content negotiation on the popular Apache web server.

```
[Event "WCh 2013"]
[Site "Chennai IND"]
[Date "2013.11.09"]
[Round "1"]
[White "Carlsen, Magnus"]
[Black "Anand, Viswanathan"]
[Result "1/2-1/2"]
[WhiteTitle "GM"]
[BlackTitle "GM"]
[WhiteElo "2870"]
[BlackElo "2775"]
[ECO "A07"]
[Opening "Reti"]
[Variation "King's Indian attack"]
[WhiteFideId "1503014"]
[BlackFideId "5000017"]
[EventDate "2013.11.09"]

1. Nf3 d5 2. g3 g6 3. Bg2 Bg7 4. d4 c6 5. O-O Nf6 6. b3 O-O 7. Bb2 Bf5 8. c4
Nbd7 9. Nc3 dxc4 10. bxc4 Nb6 11. c5 Nc4 12. Bc1 Nd5 13. Qb3 Na5 14. Qa3 Nc4 15.
Qb3 Na5 16. Qa3 Nc4 1/2-1/2
```

Figure 10.7. Example of PGN file for conversion to RDF triples. The file is obtained from `http://www.pgnmentor.com/events/WorldChamp2013.pgn` and we assume the author of the file is PGN Mentor

the fragment part that represents the class name and simply request the corresponding ontology to the server. If one wishes to have a nicer, human-readable presentation, one could set it up by manually creating an HTML page representing the ontology, or install a wrapper such as LODE[21] into the content negotiation.

10.6. Publishing the Linked Datasets on the Web

After the vocabulary/schema is ready, we can now populate it by generating the linked datasets. For our Chess example, we generate triples that correspond to the PGN files. Consider Fig. 10.7, which is another example of a PGN file (cf. Fig. 1.1 from Chapter 1). We shall convert data in this PGN file into a set of RDF triples. Since we two slightly different ontologies as part of the dual schema, we can proceed from either way. We shall describe how we can directly populate `chessonto.owl` (i.e., according to Fig. 10.1). Directly populating `chessonto-view.owl` (i.e., according to Fig. 10.6) is analogous. We shall also explain how we can actually populate `chessonto-view.owl` using the RDF triples that populates `chessonto.owl`, and vice versa.

[21] `http://www.essepuntato.it/lode`

10.6.1. Populating A Dual Schema: From the Complex Version to a Simplified Version

Recall that `chessonto.owl` represents the complex version of our dual schema, while `chessonto-view.owl` represents its simplified version. Our first approach to publish a linked dataset is by first populating `chessonto.owl` directly, and afterwards, populating `chessonto-view.owl` by making use the result of populating `chessonto.owl`. An alternative approach is in the opposite direction, which we shall explain in Section 10.6.2.

10.6.1.1. Populating `chessonto.owl` directly

Guided by the PGN specification[22], we can see that the content of the PGN file contains several pieces of information. We go through them one by one below and identify which ontology elements from Fig. 10.1 can be populated. We assume that the base prefix to be `https://w3id.org/rdfchess/id/`, while `xsd` refers to `http://www.w3.org/2001/XMLSchema#`.

- First of all, we create an instance of ChessGame, say `:gam19e02`.
- The name of the event in which the chess game was played is "WCh 2013". So we create an instance for ChessTournament, say `:tou23as5`, and assign for hasName property the literal value `"WCh 2013"^^xsd:string`. We connect `:gam19e02` to `:tou23as5` via subEventOf property.
- The site of the event is the city of Chennai, India, which is also the site of the game. We create an instance representing Chennai for the class Place, say `:pla537es8`, and assigns `"Chennai"^^xsd:string` as the value of hasName. We then attach it to the instance of `:gam19e02` and `:tou23as5` via atPlace property.
- The starting date of the chess game is November 10, 2013. Thus, we attach `"2013-11-09T00:00:00"^^xsd:dateTime`, a literal of type xsd:dateTime to `:gam19e02` via atTime property. The starting date of the event is also November 9, 2013, so we attach `"2013-11-09T00:00:00"^^xsd:dateTime` to `:tou23as5` via atTime property. Note of course that this is a simplification since the event obviously has an end date, though not specified in the PGN file.
- The round of the game in the context of the chess competition is the 1st round. We ignore this information since no classes or property corresponds to it.
- The name of the white player is "Carlsen, Magnus". Here, we create an instance of WhitePlayerRole, say `:rol88y76c`, and attach it to `:gamd19e02` via providesAgentRole property. We then create an instance of Agent, say `:ag422yt6`, attach it to `:rol88y76c` via performedBy property, and give `"Carlsen, Magnus"^^xsd:string` as the literal value for hasName property. We ignore the title of the white player, which is grandmaster ("GM").
- The name of the black player is "Anand, Viswanathan", so we use the literal value `"Anand, Viswanathan"^^xsd:string` for hasName.

[22] `http://www.thechessdrum.net/PGN_Reference.txt`

- The result of the game is "1/2-1/2" (i.e., a draw). We create an instance of ChessGameResult, say `:res23tu77h`, and then attach the literal value `"1/2-1/2"^^xsd:string` for encodedAsSAN property.
- The ELO rating of both the white and black players at the time of the game are 2870 and 2775, respectively. We ignore this information since it does not correspond to any classes or properties.
- The ECO code of the opening of the game is B18. So, we create an instance of ChessGameResult, say `:ope662vn2`, and then attach the literal value `"A07"^^xsd:string` for hasECOCode property.
- The file contain a numbered sequence of moves where each move consists of two half moves (except possibly the last one) with the white player's half move preceding the black player's half move. So, for each pair of half moves, we create an instance of HalfMove, attach an appropriate literal value representing the SAN encoding of the half move, and then attach that instance of HalfMove to `:gam19e02` using subEventOf and hasHalfMove (respecting the direction of the property as specified in Fig. 10.1). Instances representing two consecutive half moves should be connected via nextHalfMove property. The hasFirstHalfMove and hasLastHalfMove are used as appropriate to attach the first and last half moves to `:gam19e02`. Finally, for each instance of HalfMove we created, we attach a fresh instance of ActingPlayerRole via providesAgentRole property, and then we connect it to either the instance of Agent representing the white chess player (Magnus Carlsen) or the instance of Agent representing the black player (Viswanathan Anand). Both of these instances of Agent have been created earlier.
- We also create an instance of ChessGameReport, add an instance of AuthorRole and the corresponding instance of Agent representing PGN Mentor, and use `http://www.pgnmentor.com/events/WorldChamp1984.pgn` as the URI of the PGN file.

The above steps result in a set of overall 257 RDF triples given in Fig. 10.8, 10.9, 10.10, and 10.11. As the reader can see, the triples use properties and classes from `chessonto.owl`.

Naturally, writing them all by hand would be rather tedious. The mapping between tags in the PGN file and the vocabulary terms in the ontology also has to be set manually. Fortunately, there are already a number of programming libraries and APIs that can help us in generating and manipulating RDF triples, e.g., Apache Jena API (Java),[23] RDF4J (Java; formerly known as Sesame),[24] Redland (C),[25] rdflib (Python),[26] etc.

10.6.1.2. Storing RDF triples

The triples just generated can be stored as a file dump. The most recent version of RDF, i.e., RDF 1.1, specified several standard serialization formats:[27]

[23] `https://jena.apache.org/`
[24] `http://rdf4j.org/`
[25] `http://librdf.org/`
[26] `https://github.com/RDFLib/rdflib/`
[27] The older RDF 1.0 only has RDF/XML as the standard serialization format.

```
@prefix : <https://w3id.org/rdfchess/id/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix chon: <https://w3id.org/rdfchess/chessonto#> .

:gam19e02 rdf:type chon:ChessGame ;
          chon:subEventOf :tou23as5 ;
          chon:atPlace :pla537es8 ;
          chon:atTime "2013-11-09T00:00:00"^^xsd:dateTime ;
          chon:providesAgentRole :rol88y76c, :rol92z01m ;
          chon:hasResult :res23tu77h ;
          chon:hasOpening :ope662vn2 .

:tou23as5 rdf:type chon:ChessTournament ;
          chon:hasName "WCh 2013"^^xsd:string ;
          chon:atPlace :pla537es8 ;
          chon:atTime "2013-11-09T00:00:00"^^xsd:dateTime .

:pla537es8 rdf:type chon:Place ;
           chon:hasName "Chennai"^^xsd:string .

:rol88y76c rdf:type chon:WhitePlayerRole ; chon:performedBy :ag422yt6 .

:ag422yt6 rdf:type chon:Agent ;
          chon:hasName "Carlsen, Magnus"^^xsd:string .

:rol92z01m rdf:type chon:BlackPlayerRole ; chon:performedBy :ag79yy12 .

:ag79yy12 rdf:type chon:Agent ;
          chon:hasName "Anand, Viswanathan"^^xsd:string .

:res23tu77h rdf:type chon:ChessGameResult ;
            chon:encodedAsSAN "1/2-1/2"^^xsd:string .

:ope662vn2 rdf:type chon:ChessOpening ;
           chon:hasECOCode "A07"^^xsd:string .

:cgr448uy6 rdf:type chon:ChessGameReport ;
      chon:providesAgentRole :rol08jj2a ;
      chon:hasPGNFile <http://www.pgnmentor.com/events/WorldChamp2013.pgn> .

:rol08jj2a rdf:type chon:AuthorRole ;
           chon:performedBy :ag66bn89 .

:ag66bn89 rdf:type chon:Agent ;
          chon:hasName "PGN Mentor"^^xsd:string
```

Figure 10.8. RDF Triples (in Turtle syntax) populating the ontology in Fig. 10.1 — continued in Fig. 10.9

```
:gam19e02 chon:hasFirstHalfMove :hmgam19e021a ;
          chon:hasLastHalfMove :hmgam19e0216b ;
          chon:hasHalfMove
             :hmgam19e021a, :hmgam19e021b, :hmgam19e022a, :hmgam19e022b,
             :hmgam19e023a, :hmgam19e023b, :hmgam19e024a, :hmgam19e024b,
             :hmgam19e025a, :hmgam19e025b, :hmgam19e026a, :hmgam19e026b,
             :hmgam19e027a, :hmgam19e027b, :hmgam19e028a, :hmgam19e028b,
             :hmgam19e029a, :hmgam19e029b, :hmgam19e0210a, :hmgam19e0210b,
             :hmgam19e0211a, :hmgam19e0211b, :hmgam19e0212a, :hmgam19e0212b,
             :hmgam19e0213a, :hmgam19e0213b, :hmgam19e0214a, :hmgam19e0214b,
             :hmgam19e0215a, :hmgam19e0215b, :hmgam19e0216a, :hmgam19e0216b .

:hmgam19e021a rdf:type chon:HalfMove ;  chon:hasSANRecord "Nf3"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9021a ;  chon:nextHalfMove :hmgam19e021b .
:rolhg19e9021a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e021b rdf:type chon:HalfMove ;  chon:hasSANRecord "d5"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9021b ; chon:nextHalfMove :hmgam19e022a .
:rolhg19e9021a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e022a rdf:type chon:HalfMove ;  chon:hasSANRecord "g3"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9022a ;  chon:nextHalfMove :hmgam19e022b .
:rolhg19e9022a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e022b rdf:type chon:HalfMove ;  chon:hasSANRecord "g6"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9022b ; chon:nextHalfMove :hmgam19e023a .
:rolhg19e9022b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e023a rdf:type chon:HalfMove ;  chon:hasSANRecord "Bg2"^^xsd:string ;
   chon:providesAgentRole :rolhg19e9023a ;  chon:nextHalfMove :hmgam19e023b .
:rolhg19e9023a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e023b rdf:type chon:HalfMove ;  chon:hasSANRecord "Bg7"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9023b ; chon:nextHalfMove :hmgam19e024a .
:rolhg19e9023b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e024a rdf:type chon:HalfMove ;  chon:hasSANRecord "d4"^^xsd:string ;
   chon:providesAgentRole :rolhg19e9024a ;  chon:nextHalfMove :hmgam19e024b .
:rolhg19e9024a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e024b rdf:type chon:HalfMove ;  chon:hasSANRecord "c6"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9024b ; chon:nextHalfMove :hmgam19e025a .
:rolhg19e9024b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e025a rdf:type chon:HalfMove ;  chon:hasSANRecord "O-O"^^xsd:string ;
   chon:providesAgentRole :rolhg19e9025a ;  chon:nextHalfMove :hmgam19e025b .
:rolhg19e9025a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .
```

Figure 10.9. (continued from Fig. 10.8) RDF Triples (in Turtle syntax) populating the ontology in Fig. 10.1 – continued in Fig. 10.10

```
:hmgam19e025b rdf:type chon:HalfMove ;  chon:hasSANRecord "Nf6"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9025b ; chon:nextHalfMove :hmgam19e026a .
:rolhg19e9025b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e026a rdf:type chon:HalfMove ;  chon:hasSANRecord "b3"^^xsd:string ;
   chon:providesAgentRole :rolhg19e9026a ;  chon:nextHalfMove :hmgam19e026b .
:rolhg19e9026a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e026b rdf:type chon:HalfMove ;  chon:hasSANRecord "O-O"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9026b ; chon:nextHalfMove :hmgam19e027a .
:rolhg19e9026b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e027a rdf:type chon:HalfMove ;  chon:hasSANRecord "Bb2"^^xsd:string ;
   chon:providesAgentRole :rolhg19e9027a ;  chon:nextHalfMove :hmgam19e027b .
:rolhg19e9027a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e027b rdf:type chon:HalfMove ;  chon:hasSANRecord "Bf5"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9027b ; chon:nextHalfMove :hmgam19e028a .
:rolhg19e9027b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e028a rdf:type chon:HalfMove ;  chon:hasSANRecord "c4"^^xsd:string ;
   chon:providesAgentRole :rolhg19e9028a ;  chon:nextHalfMove :hmgam19e028b .
:rolhg19e9028a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e028b rdf:type chon:HalfMove ;  chon:hasSANRecord "Nbd7"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9028b ; chon:nextHalfMove :hmgam19e029a .
:rolhg19e9028b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e029a rdf:type chon:HalfMove ;  chon:hasSANRecord "Nc3"^^xsd:string ;
   chon:providesAgentRole :rolhg19e9029a ;  chon:nextHalfMove :hmgam19e029b .
:rolhg19e9029a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e029b rdf:type chon:HalfMove ;  chon:hasSANRecord "dxc4"^^xsd:string ;
    chon:providesAgentRole :rolhg19e9029b ; chon:nextHalfMove :hmgam19e0210a .
:rolhg19e9029b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e0210a rdf:type chon:HalfMove ;  chon:hasSANRecord "bxc4"^^xsd:string ;
   chon:providesAgentRole :rolhg19e90210a ;  chon:nextHalfMove :hmgam19e0210b .
:rolhg19e90210a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e0210b rdf:type chon:HalfMove ;  chon:hasSANRecord "Nb6"^^xsd:string ;
    chon:providesAgentRole :rolhg19e90210b ; chon:nextHalfMove :hmgam19e0211a .
:rolhg19e90210b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e0211a rdf:type chon:HalfMove ;  chon:hasSANRecord "c5"^^xsd:string ;
   chon:providesAgentRole :rolhg19e90211a ;  chon:nextHalfMove :hmgam19e0211b .
:rolhg19e90211a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .
```

Figure 10.10. (continued from Fig. 10.9) RDF Triples (in Turtle syntax) populating the ontology in Fig. 10.1 – continued to Fig. 10.11

```
:hmgam19e0211b rdf:type chon:HalfMove ;  chon:hasSANRecord "Nc4"^^xsd:string ;
    chon:providesAgentRole :rolhg19e90211b ; chon:nextHalfMove :hmgam19e0212a .
:rolhg19e90211b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e0212a rdf:type chon:HalfMove ;  chon:hasSANRecord "Bc1"^^xsd:string ;
   chon:providesAgentRole :rolhg19e90212a ;  chon:nextHalfMove :hmgam19e0212b .
:rolhg19e90212a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e0212b rdf:type chon:HalfMove ;  chon:hasSANRecord "Nd5"^^xsd:string ;
    chon:providesAgentRole :rolhg19e90212b ; chon:nextHalfMove :hmgam19e0213a .
:rolhg19e90212b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e0213a rdf:type chon:HalfMove ;  chon:hasSANRecord "Qb3"^^xsd:string ;
   chon:providesAgentRole :rolhg19e90213a ;  chon:nextHalfMove :hmgam19e0213b .
:rolhg19e90213a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e0213b rdf:type chon:HalfMove ;  chon:hasSANRecord "Na5"^^xsd:string ;
    chon:providesAgentRole :rolhg19e90213b ; chon:nextHalfMove :hmgam19e0214a .
:rolhg19e90213b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e0214a rdf:type chon:HalfMove ;  chon:hasSANRecord "Qa3"^^xsd:string ;
   chon:providesAgentRole :rolhg19e90214a ;  chon:nextHalfMove :hmgam19e0214b .
:rolhg19e90214a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e0214b rdf:type chon:HalfMove ;  chon:hasSANRecord "Nc4"^^xsd:string ;
    chon:providesAgentRole :rolhg19e90214b ; chon:nextHalfMove :hmgam19e0215a .
:rolhg19e90214b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e0215a rdf:type chon:HalfMove ;  chon:hasSANRecord "Qb3"^^xsd:string ;
   chon:providesAgentRole :rolhg19e90215a ;  chon:nextHalfMove :hmgam19e0215b .
:rolhg19e90215a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e0215b rdf:type chon:HalfMove ;  chon:hasSANRecord "Na5"^^xsd:string ;
    chon:providesAgentRole :rolhg19e90215b ; chon:nextHalfMove :hmgam19e0216a .
:rolhg19e90215b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .

:hmgam19e0216a rdf:type chon:HalfMove ;  chon:hasSANRecord "Qa3"^^xsd:string ;
   chon:providesAgentRole :rolhg19e90216a ;  chon:nextHalfMove :hmgam19e0216b .
:rolhg19e90216a rdf:type chon:ActingPlayerRole ; chon:performedBy :ag422yt6 .

:hmgam19e0216b rdf:type chon:HalfMove ;  chon:hasSANRecord "Nc4"^^xsd:string ;
    chon:providesAgentRole :rolhg19e90216b .
:rolhg19e90216b rdf:type chon:ActingPlayerRole ; chon:performedBy :ag79yy12 .
```

Figure 10.11. (continued from Fig. 10.10) RDF Triples (in Turtle syntax) populating the ontology in Fig. 10.1

RDF/XML, N-triples, Turtle (which we used in this chapter), RDFa, JSON-LD, TriG, and N-Quads. Most APIs for manipulating RDF would one to read and write those serializations formats. This would be sufficient if we just want to do some offline processing, do not need to pose custom queries to the data, and the number of triples is not too large that it is feasible to load them all in memory. In many other situations, we have to store the RDF triples in a triple store. One can choose any triple store product, both the open source as well as proprietary ones. Examples of triple stores as well as storage solutions that support storing RDF data include OpenLink Virtuoso,[28] TDB from Apache Jena,[29] RDF4J (formerly known as Sesame), Stardog,[30] Redland, GraphDB,[31] BrightStarDB,[32] Strabon,[33] Blazegraph[34], Dydra,[35], Mulgara,[36], CubicWeb,[37] AllegroGraph,[38], MarkLogic,[39] etc. These storage solutions support data retrieval using some query languages. Most of them support SPARQL and the others support other variants of graph querying language. Some additionally allows one to set up a SPARQL endpoint that is directly accessible from the Web typically via some kind of API, such as REST API.

10.6.1.3. Populating `chessonto-view.owl` using the populated `chessonto.owl`

Now that we have generated RDF triples to populate `chessonto.owl`, we can use them to populate `chessonto-view.owl`. The idea is to make use of the shortcuts we defined in Fig. 10.5. Note that `chessonto-view.owl` already contains a set of DL-safe SWRL rules expressing the shortcuts. Thus, one approach is to employ a SWRL or Datalog reasoner, e.g., Drools[40] or RDFox[41] to materialize the conclusions of each of those rules over the combination of the RDF triples and `chessonto-view.owl` as the underlying knowledge base, i.e., declaring all URIs in the RDF triples that are neither a class nor a property to be an OWL named individual. The materialization will result in a set of generated triples, which can then be added back to the original set of RDF triples.

Another approach is to either use any RDF API to search for the triples matching the body of the rules and then generate triples corresponding to the head of the rules. For example, the first rule in that figure is:

$$\mathsf{ChessGame}(x) \wedge \mathsf{pAR}(x, y) \wedge \mathsf{WhitePlayerRole}(y) \wedge \mathsf{performedBy}(y, z) \\ \wedge \mathsf{Agent}(z) \wedge \mathsf{hasName}(z, s) \rightarrow \mathsf{hasWhitePlayer}(x, s)$$

[28] `http://virtuoso.openlinksw.com/`
[29] `https://jena.apache.org/documentation/tdb/`
[30] `http://stardog.com/`
[31] `http://graphdb.ontotext.com/`
[32] `http://brightstardb.com/`
[33] `http://www.strabon.di.uoa.gr/`
[34] `https://www.blazegraph.com/`
[35] `http://dydra.com/`
[36] `http://www.mulgara.org/`
[37] `https://www.cubicweb.org/`
[38] `http://franz.com/agraph/allegrograph/`
[39] `http://www.marklogic.com/`
[40] `http://www.drools.org/`
[41] `http://www.cs.ox.ac.uk/isg/tools/RDFox/`

```
  ?x rdf:type chon:ChessGame ;
     chon:providesAgentRole ?y .
  ?y rdf:type chon:WhitePlayerRole ;
     chon:performedBy ?z .
  ?z rdf:type chon:Agent ;
     chon:hasName ?s .
```

Figure 10.12. Triple patterns for the body of the first rule from Fig. 10.5. The prefix `chon:` refers to `<https://w3id.org/rdfchess/chessonto#>`

```
CONSTRUCT {
  ?x chonv:hasWhitePlayer ?s
} WHERE {
  ?x rdf:type chon:ChessGame ;
     chon:providesAgentRole ?y .
  ?y rdf:type chon:WhitePlayerRole ;
     chon:performedBy ?z .
  ?z rdf:type chon:Agent ;
     chon:hasName ?s .
}
```

Figure 10.13. SPARQL CONSTRUCT query for the first rule from Fig. 10.5. The prefix `chon:` and `chonv:` refer to `<https://w3id.org/rdfchess/chessonto#>` and `<https://w3id.org/rdfchess/chessonto-view#>`, respectively.

The body of this rules can be generally read as a set of triple patterns. That is, it corresponds to the set of triple patterns given in Fig. 10.12.

One could also run SPARQL CONSTRUCT queries, which can be created for each rule, on the RDF triples, assuming that they are stored in a triple store. For example, the above rule can be expressed in the query given in Fig. 10.13. Running this query will yield a set of triples that can be added back to the original set of RDF triples, which now also populates `chessonto-view.owl`.

10.6.2. Alternative approach: Populating Dual Schema from a Simplified Version to the Complex Version

This approach may be appealing for some data providers because we start by populating `chessonto-view.owl`, which represents a simpler version of the dual schema. To populate `chessonto-view.owl` directly, the steps are essentially the same as the ones we took to directly populate `chessonto.owl` as described in Section 10.6.1.1. The only notable difference is that we do not need to generate a URI for WhitePlayerRole, providesAgentRole, Agent, etc. Be mindful that some of the classes and properties in `chessonto-view.owl` have the same URIs as those in `chessonto.owl`.

The more interesting part of this approach is the subsequent step where we can populate `chessonto.owl` based on the populated `chessonto-view.owl`. This

```
CONSTRUCT {
  ?x rdf:type chon:ChessGame ;
     chon:providesAgentRole
         [ rdf:type chon:WhitePlayerRole ;
           chon:performedBy
               [ rdf:type chon:Agent ;
                 chon:hasName ?s ]
         ]  .
} WHERE {
  ?x chonv:hasWhitePlayer ?s .
}
```

Figure 10.14. A SPARQL CONSTRUCT query expressing the reverse direction of the first rule from Fig. 10.5 where blank nodes are generated. The prefix `chon:` and `chonv:` refer to `<https://w3id.org/rdfchess/chessonto#>` and `<https://w3id.org/rdfchess/chessonto-view#>`, respectively.

time, however, we cannot use the SWRL rules in `chessonto-view.owl`, at least directly, because the process corresponds to the backward direction of the rules. That is, the triples we have correspond to the head of the rules and we have to generate new triples that correspond to the body of the rules. Therefore, SWRL reasoners cannot help us here. Fortunately, we can still approach it via RDF APIs and SPARQL CONSTRUCT queries. For example, based on the rule:

$$\mathsf{ChessGame}(x) \wedge \mathsf{pAR}(x,y) \wedge \mathsf{WhitePlayerRole}(y) \wedge \mathsf{performedBy}(y,z) \\ \wedge \mathsf{Agent}(z) \wedge \mathsf{hasName}(z,s) \rightarrow \mathsf{hasWhitePlayer}(x,s)$$

we can write a SPARQL CONSTRUCT query in Fig. 10.14 that intuitively runs in the opposite direction of the rule. Running this on the triples populates a part of `chessonto-view.owl`. Such a query can generally be created for each rule in Fig.. 10.5.

Observe that we generate blank nodes for WhitePlayerRole and Agent. If we wish to generate actual URIs, we need to hard-code parts of the URI namespaces for the instances of WhitePlayerRole and Agent and employ a number of SPARQL built-in functions. Generally, this also requires the help of some naming convention to ensure the unambiguity of the generated URIs within the same namespace. An example of the SPARQL query is given in Fig. 10.15.

After we generate the triples, we still have to do some post-processing, because according to `chessonto.owl` as also depicted in Fig. 10.1, the agent that acts as the white player (resp. black player) of a chess game should be the same as the agent that acts as the acting player of the half move of a white player (resp. black player) in the chess game.

For example, using the example of the chess game discussed in this chapter (as given by Fig. 10.7) and the SPARQL CONSTRUCT queries that generate blank nodes (the query in Fig. 10.14 and another similar query that is based on last rule in Fig. 10.5), we would have generated the triples in Fig. 10.16 where `:gm123` is the URI representing the instance of ChessGame that corresponds to

```
CONSTRUCT {
  ?x rdf:type chon:ChessGame ;
     chon:providesAgentRole ?roleuri .
  ?roleuri rdf:type chon:WhitePlayerRole ;
           chon:performedBy ?agenturi .
  ?agenturi rdf:type chon:Agent ;
            chon:hasName ?s .
} WHERE {
  ?x chonv:hasWhitePlayer ?s .
  BIND (STRUUID() AS ?roleid)
  BIND (STRUUID() AS ?agentid)
  BIND (URI(CONCAT("https://w3id.org/rdfchess/id/", "rol", ?roleid))
        AS ?roleuri)
  BIND (URI(CONCAT("https://w3id.org/rdfchess/id/", "ag", ?agentid))
        AS ?agenturi)
}
```

Figure 10.15. A SPARQL CONSTRUCT query expressing the reverse direction of the first rule from Fig. 10.5 that generates actual URIs, instead of blank nodes. The prefix `chon:` and `chonv:` refer to `<https://w3id.org/rdfchess/chessonto#>` and `<https://w3id.org/rdfchess/chessonto-view#>`, respectively. We use, as a part of naming convention, the string "`rol`" and "`ag`" as the initial part of the name of instances of AgentRole and Agent, respectively.

the discussed chess game and `:move1` is the URI representing the instance of HalfMove that corresponds to the first half move of the chess game, while `_:x`, `_:x1`, `_:y`, `_:y1` are all blank nodes. There, the blank nodes `_:x` and `_:y` must correspond to the same instance of Agent. Thus, we need to replace `_:x` with `_:y` or vice versa.

An analogous post-processing also needs to be done if we employ the SPARQL CONSTRUCT queries that generate actual URIs.

10.6.3. Redundancy in the Data and Linking with Other RDF Data

We now obtained a linked dataset that populate the dual schema. One problem that may still happen is redundancy inside the data, especially if the RDF triples are programmatically generated from the PGN files. This redundancy is mainly caused by the fact that two different PGN files may contain the same information that is expressed using different string values. One obvious example is the name of chess players. PGN Specification does not specify a fixed format for a chess player's name. One PGN file may contain "Carlsen, Magnus", while another may contain "Magnus Carlsen", and both refer to the same chess player Magnus Carlsen. If we can find two URIs that should correspond to the same entity, we can assert a link (i.e., as a triple) between these two entities using `owl:sameAs` predicate or weaker predicates such as the ones from the SKOS vocabulary.[42]

The above problem also extends to a more general problems of linking with

[42] `http://www.w3.org/2004/02/skos/core`

```
:gm123 rdf:type chon:ChessGame ;
       chon:providesAgentRole _:x1 .
_:x1 rdf:type chon:WhitePlayerRole ;
     chon:performedBy _:x .
_:x rdf:type chon:Agent ;
    chon:hasName "Carlsen, Magnus"^^xsd:string .
:gm123 chon:hasHalfMove :move1 .
:move1 chon:providesAgentRole _:y1 .
_:y1 rdf:type chon:ActingPlayerRole ;
     chon:performedBy _:y .
_:y rdf:type chon:Agent ;
    chon:hasName "Carlsen, Magnus"^^xsd:string .
```

Figure 10.16. Example RDF triples that need post-processing: `_:x` and `_:y` must correspond to the same instance of Agent.

other linked datasets in the Linked Open Data Cloud.[43] The latter is important to make our linked dataset truly in the five-star category [7]. As an example, we shall augment our dataset with links to some other linked dataset in the LOD Cloud. The simplest one to pick is DBpedia dataset, which is based on Wikipedia infoboxes. If we inspect our dataset, we can intuitively guess that a few pieces of information correspond to real world entities that may appear in Wikipedia. In particular, we may be able to find information about chess players, the chess tournaments, and location of the chess tournaments in DBpedia since those information are likely to be available in Wikipedia. With regards to the RDF triples we generated in this chapter, we find the following URIs from DBpedia datasets:

- `http://dbpedia.org/resource/Magnus_Carlsen` representing the chess player Magnus Carlsen;
- `http://dbpedia.org/resource/Viswanathan_Anand` representing the chess player Viswanathan Anand;
- `http://dbpedia.org/resource/World_Chess_Championship_2013` representing the World Chess Championship 2013;
- `http://dbpedia.org/resource/Chennai` representing the city of Chennai, India.

Hence, we can enrich our dataset by adding triples given in Fig. 10.17 to the ones we already obtained in Fig. 10.8, 10.9, 10.10, and 10.11.

One could easily see that finding links to other linked datasets as exemplified above can also be understood as detecting whether two URIs refer to the same entity. In our example above, we are quite fortunate since we are able to find the corresponding URIs in DBpedia manually. If we want to do this to every set of triples we generated from the PGN files, then we need to do it programmatically, and the general computational problem corresponds to the so-called *coreference resolution*, which is beyond the scope of this chapter. Nevertheless, this section

[43] `http://lod-cloud.net/`

```
:ag422yt6 owl:sameAs <http://dbpedia.org/resource/Magnus_Carlsen> .
:ag79yy12 owl:sameAs <http://dbpedia.org/resource/Viswanathan_Anand> .
:tou23as5 owl:sameAs <http://dbpedia.org/resource/World_Chess_Championship_2013>
:pla537es8 owl:sameAs <http://dbpedia.org/resource/Chennai>
```

Figure 10.17. Links to DBpedia data using `owl:sameAs` where the prefix `owl:` refers to `<http://www.w3.org/2002/07/owl#>`. If `owl:sameAs` is felt too strong, one can use other linking vocabulary such as the one from SKOS.

demonstrates the advantage of Linked Data for combining and integrating our chess dataset with other existing datasets.

10.6.4. Serving the Data on the Web

We can publish the triples we just generated as a downloadable file (RDF dumps) and putting it somewhere accessible by everyone on the Web. We assume that the ontologies as well as the vocabulary terms are all Web-dereferenceable. So, we just need to ensure that other URIs appearing in any of the RDF triple, which is of the form `https://w3id.org/rdfchess/id/ssss`, are Web-dereferenceable. This can be achieved by setting up an appropriate content negotiation so that whenever such URIs are requested, the RDF dump is returned. This is certainly a very crude way to serve the data, though arguably already satisfies the Linked Data principles.

A nicer way, albeit putting more burden on the data provider, is to set up a Linked Data infrastructure that host the RDF triples. The backend of such an infrastructure is typically a triple store that we already touched upon in one of the earlier section, although other types of database are also possible. The triple store is accessible from one or more endpoints that accept queries from users, usually expressed in SPARQL. On top of a SPARQL endpoint, one can then set up a front-end that, with appropriate Web server configuration, can handle content negotiation nicely. In addition, such a front-end allows one to either perform a follow-your-nose browsing of the data or formulate custom queries directly on the data. More fancy front-ends also provides visualization functionalities. One can certainly develop such a front-end in-house, but there are examples of front-end products that one can also use or adapt, e.g., Linked Media Framework,[44] D2RQ,[45] Pubby,[46] OpenLink Virtuoso's front-end,[47] PublishMyData,[48] Linked Data Fragments,[49] Exhibit,[50] etc.

[44] `https://bitbucket.org/srfgkmt/lmf/`
[45] `http://d2rq.org/`; intended to provide access to relational databases as virtual RDF graphs
[46] `http://wifo5-03.informatik.uni-mannheim.de/pubby/`
[47] `http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtDeployingLinkedDataGuide_UsingVirtuoso`
[48] `http://www.swirrl.com/`
[49] `http://linkeddatafragments.org/`
[50] `http://www.simile-widgets.org/exhibit3/`

10.7. Conclusion

This chapter presented a rather high-level description on the steps that one needs to take to publish Linked Data that is based on Ontology Design Patterns. Some technical pointers also have been provided. Obviously, we do not cover more specific technical details, which can normally be obtained from the documentation of the Linked Data infrastructure products that we refered to earlier. A rather prominent issue that we do not address concerns the possible redundancy of information particularly if we wish to generate one large linked dataset that cover not just one particular PGN file, but rather, a large collection of ones, e.g., the whole collection of PGN files from PGN Mentor. The way we generated RDF triples as discussed in this chapter assumes that for each PGN file, we generate a new URI for the chess tournament, the chess players (as instances of Agent), location of the tournament (as instance of Place), the author of the game description (as instance of Agent), and the chess opening that has its own ECO code. It is rather easy to see that those entities may occur across different PGN files, e.g., there may be more than one PGN files in which Magnus Carlsen appears as either the white or black player.

To address this redundancy, we can modify the procedure of generating RDF triples so that we do not look at one PGN file as an isolated data source, but look at one collection of PGN files as a whole. When generating the triples, we then need to reuse URIs for a particular entity (players, tournaments, places, etc.) if we encountered one that we have seen in the collection before. Alternatively, we can still consider each PGN file by itself, and after we finish processing all PGN files, we perform a post-processing step where we search for URIs that represent the exact same entity, and then add an `owl:sameAs` triple for each pair of such URIs. Both alternatives, however, boil down to coreference resolution problem, which generally may not be trivial, and thus, is beyond the scope of this chapter.

Acknowledgements. This work was partially supported by the National Science Foundation under award 1440202 *EarthCube Building Blocks: Collaborative Proposal: GeoLink – Leveraging Semantics and Linked Data for Data Sharing and Discovery in the Geosciences*, by the EC under the International Research Staff Exchange Scheme (IRSES) of the EU Marie Curie Actions, project *SemData – Semantic Data Management*, and by the Spanish Ministry of Economy and Competitiveness (project TIN2013-46238-C4-2-R).

Bibliography

[1] P. Archer, N. Loutas, S. Goedertier, and S. Kourtidis. Study on persistent URIs with identification of best practices and recommendations on the topic for the Member States and the European Commission, June 24, 2013. Available at `http://philarcher.org/diary/2013/uripersistence/`.

[2] O. Bartlett. Linked Data: Connecting together the BBC's online content. BBC Internet Blog, February 13, 2013. Available at `http://www.bbc.co.uk/blogs/internet/entries/af6b613e-6935-3165-93ca-9319e1887858`.

[3] O. Bartlett. Olympic Data services and the interactive video player. BBC Internet Blog, July 31, 2012. Available at http://www.bbc.co.uk/blogs/internet/entries/92c2ec84-6ec0-33f2-9dc7-0f915d28ff52.

[4] W. Beek, L. Rietveld, H. R. Bazoobandi, J. Wielemaker, and S. Schlobach. LOD laundromat: A uniform way of publishing other people's dirty data. In P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. A. Knoblock, D. Vrandecic, P. T. Groth, N. F. Noy, K. Janowicz, and C. A. Goble, editors, *The Semantic Web – ISWC 2014 – 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, volume 8796 of *Lecture Notes in Computer Science*, pages 213–228. Springer, 2014.

[5] T. Berners-Lee. Web security - "HTTPS Everywhere" harmful. Available at https://www.w3.org/DesignIssues/Security-NotTheS.html, 15 February 2012.

[6] T. Berners-Lee. *URIs for W3C Namespaces*. W3C Technical Report Publication Policies, 25 April 2006. Available at https://www.w3.org/2005/07/13-nsuri.

[7] T. Berners-Lee. Linked data. Available at https://www.w3.org/DesignIssues/LinkedData.html, 27 July 2006.

[8] E. Blomqvist, P. Hitzler, K. Janowicz, A. Krisnadhi, T. Narock, and M. Solanki. Considerations regarding ontology design patterns. *Semantic Web*, 7(1):1–7, 2015.

[9] P. Davidson. Designing URI sets for the UK Public Sector: A report from the Public Sector Information Domain of the CTO Council's cross-government enterprise architecture. Interim paper version 1.0, Chief Technology Officer Council, October 2009. Available at https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/60975/designing-URI-sets-uk-public-sector.pdf; Last accessed: May 12, 2016.

[10] D. A. Ferrucci, E. W. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. M. Prager, N. Schlaefer, and C. A. Welty. Building Watson: An overview of the DeepQA project. *AI Magazine*, 31(3):59–79, 2010.

[11] S. Harris and A. Seaborne, editors. *SPARQL 1.1 Query Language*. W3C Recommendation, 21 March 2013. Available at https://www.w3.org/TR/sparql11-query/.

[12] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.

[13] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C Member Submission, 21 May 2004. Available from http://www.w3.org/Submission/SWRL/.

[14] B. Hyland, G. Atemezing, and B. Villazón-Terrazas, editors. *Best Practices for Publishing Linked Data*. W3C Working Group Note, 09 January 2014. Available at https://www.w3.org/TR/ld-bp/.

[15] K. Janowicz, P. Hitzler, B. Adams, D. Kolas, and C. Vardeman. Five stars of linked data vocabulary use. *Semantic Web*, 5(3):173–176, 2014.

[16] A. Krisnadhi. *Ontology Pattern-Based Data Integration*. PhD thesis, Wright State University, 2015.

[17] A. Krisnadhi and P. Hitzler. Modeling with Ontology Design Patterns: Chess games as a worked example. In A. Gangemi, P. Hitzler, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.

[18] A. Krisnadhi, F. Maier, and P. Hitzler. OWL and Rules. In A. Polleres et al., editors, *Reasoning Web. Semantic Technologies for the Web of Data – 7th International Summer School 2011, Tutorial Lectures*, volume 6848 of *Lecture Notes in Computer Science*, pages 382–415. Springer, Heidelberg, 2011.

[19] A. A. Krisnadhi, P. Hitzler, and K. Janowicz. On the capabilities and limitations of OWL regarding typecasting and ontology design pattern views. In V. A. M. Tamma, M. Dragoni, R. Gonçalves, and A. Lawrynowicz, editors, *Ontology Engineering - 12th International Experiences and Directions Workshop on OWL, OWLED 2015, co-located with ISWC 2015, Bethlehem, PA, USA, October 9-10, 2015, Revised Selected Papers*, volume 9557 of *Lecture Notes in Computer Science*, pages 105–116. Springer, 2015.

[20] M. Krötzsch, F. Maier, A. A. Krisnadhi, and P. Hitzler. A better uncle for OWL: Nominal schemas for integrating rules and ontologies. In S. Sadagopan, K. Ramamritham, A. Kumar, M. Ravindra, E. Bertino, and R. Kumar, editors, *Proceedings of the 20th International World Wide Web Conference, WWW2011, Hyderabad, India, March/April 2011*, pages 645–654. ACM, New York, 2011.

[21] F. Maier. A primer on RDF and OWL. In A. Gangemi, P. Hitzler, K. Janowicz, A. Krisnadhi, and V. Presutti, editors, *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*, Studies on the Semantic Web. IOS Press, 2016. In this volume.

[22] E. Montiel-Ponsoda, D. Vila-Suero, B. Villazón-Terrazas, G. Dunsire, E. E. Rodriguez, and A. Gómez-Pérez. Style guidelines for naming and labeling ontologies in the multilingual web. In T. Baker, D. I. Hillmann, and A. Isaac, editors, *Proceedings of the 2011 International Conference on Dublin Core and Metadata Applications, DC 2011, The Hague, The Netherlands, September 21-23, 2011*, pages 105–115. Dublin Core Metadata Initiative, 2011.

[23] J. Rayfield. BBC World Cup 2010 dynamic semantic publishing. BBC Internet Blog, July 12, 2010. `http://www.bbc.co.uk/blogs/bbcinternet/2010/07/bbc_world_cup_2010_dynamic_sem.html`.

[24] L. Rietveld, W. Beek, R. Hoekstra, and S. Schlobach. Meta-data for a lot of LOD. *Semantic Web*, 2016. Conditionally accepted for publication.

[25] V. Rodríguez-Doncel, A. Gómez-Pérez, and N. Mihindukulasooriya. Rights declaration in linked data. In O. Hartig, J. Sequeda, A. Hogan, and T. Matsutsuka, editors, *Proceedings of the Fourth International Workshop on Consuming Linked Data, COLD 2013, Sydney, Australia, October 22, 2013*, volume 1034 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[26] V. Rodríguez-Doncel, A. A. Krisnadhi, P. Hitzler, M. Cheatham, N. Karima, and R. Amini. Pattern-based linked data publication: The linked chess

dataset case. In O. Hartig, J. Sequeda, and A. Hogan, editors, *Proceedings of the 6th International Workshop on Consuming Linked Data co-located with 14th International Semantic Web Conference (ISWC 2105), Bethlehem, Pennsylvania, US, October 12th, 2015.*, volume 1426 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.

[27] L. Sauermann and R. Cyganiak, editors. *Cool URIs for the Semantic Web*. W3C Interest Group Note, 03 December 2008. Available at `https://www.w3.org/TR/cooluris/`.

[28] M. Schmachtenberg, C. Bizer, and H. Paulheim. State of the LOD cloud 2014. http://linkeddatacatalog.dws.informatik.uni-mannheim.de/state/, 30 August 2014.

[29] E. Winter. Copyright on chess games. `http://www.chesshistory.com/winter/extra/copyright.html`, 1987.

[30] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, and S. Auer. Quality assessment for linked data: A survey. *Semantic Web*, 7(1):63–93, 2016.