Template Libraries for Industrial Asset Maintenance: A Methodology for Scalable and Maintainable Ontologies

Daniel P. Lupp[1], Melinda Hodkiewicz[2], and Martin G. Skjæveland[1]

[1] Department of Informatics, University of Oslo
[2] Faculty of Engineering and Mathematical Sciences, University of Western Australia

Abstract. Data in engineering industries is highly standardized and information-dense, and usually distributed across many different sources with incompatible formats and implicit semantics. As such, it provides an ideal use case for application of semantic technologies for data integration and access. However, adoption of semantic technologies is hampered due to the complex set of competencies required to construct a scalable and maintainable ontology. We present a methodology for aiding domain experts and ontology engineers in constructing and maintaining industry-viable ontologies using a template-based approach for ontology modeling and instantiation. Using the OTTR framework, the structure of the input formats and the semantics of the target domain are modeled and maintained in separate modularized template libraries. Data provided by domain experts is mapped to these templates, allowing end-users to extend and amend the model without the need to directly interact with semantic technology languages and tools. Our approach is demonstrated on a real-world use case from the asset maintenance domain which has applicability to a wide range of industries.

1 Introduction

Ontologies and knowledge graphs are becoming more commonly used within industry. However, the adoption of semantic technologies still faces challenges related to the scalability and maintainability of its artifacts and workflows. Developing an industry-viable ontology requires close collaboration between domain experts and ontology engineers to ensure accurate modeling, such that intended users gain ownership over the concepts and semantic relationships described in the model. The current state of the art of ontology engineering tools and methodologies require to a large extent that domain experts learn the intricacies of semantic technologies, tools, and languages. Additionally, ontology engineers need to become well-versed in the domain they are modeling. This places strong requirements on the competence and experience of the ontology engineering team which in turn represents a major hurdle to the industrial adoption of semantic technologies [25].

Ontology engineering is a labor-intensive effort that involves many tasks over a variety of disciplines: feasibility studies, domain analyses, conceptualization, implementation, maintenance, and use [22]. Various ontology engineering methodologies exist [14, 24, 19, 16], but few methodologies are supported by tools [11]. The methodology presented in this paper uses structured sets of OTTR templates [23] to represent end-users' and systems' domain conceptualizations in order to break these patterns

down into ontological axioms in an iterative and controlled manner. It provides a means by which these macro-like templates can be organized in libraries in order to improve maintainability of the model and reusability of the templates. As such, our methodology is focused primarily on addressing issues arising in the formalization, implementation, and maintenance tasks within the typical ontology engineering pipeline. As the templates match established domain conceptualization patterns, they are arguably more easily understood and instantiated by domain experts than working directly over the OWL meta model via ontology editors like Protégé [15]. Specifications of the OTTR framework and software for interpreting and instantiating OTTR templates is available under an open licence.

The intention of this tool-supported methodology is to clearly separate the different concerns of the domain expert and ontology expert involved in modeling: ontology engineers are tasked with providing relevant modeling patterns in the form of maintained template libraries, while domain experts participate in constructing the model by providing data in tabular form, which is used to populate the modeling patterns. The added abstraction layer introduced by OTTR templates also benefits the ontology expert by supporting the don't-repeat-yourself principle, ensuring uniform modeling and encapsulation of complexity [23]. The intended effect is to dramatically lower the barrier for domain experts' involvement in ontology engineering tasks as well as increase the efficiency and quality of ontology engineering in general.

In this paper, we present a methodology for the scalable construction and maintenance of ontologies via maintained template libraries. The main contribution is a manner of structuring template libraries and prototypical tool support in order to facilitate clearly defined tasks suitable for the actors involved in the ontology engineering process as well as maintainability of the resulting model. The presented methodology was applied to the life cycle management of engineering assets, where an ontology was built using real-world data from a global resources company.

In the following section, we introduce the domain of our use case and motivate the benefits that semantic technologies have in similarly standardized industries. In Sections 3 and 4 we give an introduction to the OTTR framework as well as the template library architecture which is central to the proposed methodology. The methodology is presented and discussed in Sections 5 and 6.

2 Use Case: Is My Maintenance Strategy Correct?

Work in the engineering sector is governed by international and domain-specific standards, issued by groups such as International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC). These documents describe minimal acceptable standards and procedures, and establish documentation requirements to ensure common standards for the consistency, quality, and safety of the process output, regardless of company or country of application. As a direct consequence, data captured as an output of applying a standard are fundamentally similar in the way they are structured, often in tables, due to the need to adhere to the language and data capture format prescribed by the standard.

All industrial physical assets should have a maintenance strategy. A strategy sets out when and how to maintain an asset to ensure it delivers the desired function in a cost-effective and safe manner. When you buy a car, the car manufacturer provides instructions in the manual for when to change the oil, inspect the fan belts, service the engine, and so on. Collectively the proposed activities and their timing form the car's maintenance strategy. Now imagine you are operating a manufacturing facility or large chemical plant. You have thousands of assets, each with individual strategies. A significant proportion of operational costs, in many cases millions of dollars per year, are spent on execution of maintenance according to these strategies. The motivation for this use case, from the General Manager of a large mineral processing plant, is *"can we automate and scale the process to assess if my maintenance strategies are correct?"* Currently this process involves a team of engineers examining maintenance work orders (MWO) to look for examples of failure events that are not consistent with the assumptions about asset performance that the asset's maintenance strategy was based on. This requires engineers to manually assess and integrate data stored in two different data sets. The process is described briefly below, but an important point for this paper is that the way the data is stored in each source is consistent, in terms of language used for column headings and structure of the tables, across companies and industries. Thus a solution for one company can be readily applied to other companies using the same templates.

One data source is a *Failure mode and effects analysis (FMEA)* table. FMEA is an engineering knowledge capture process to define the system, sub-system, and key components of an asset, document the different ways each can fail to fulfill their desired function, assign a failure mode, and describe the effects of the failure mode and possible causes. Maintenance strategies are then developed to address each important failure mode. The FMEA process is governed by engineering standards, specifically IEC 60812:2006 [6].

The second data source is reports, generated as CSV tables, from an organization's *Computerized Maintenance Management System (CMMS)*. A CMMS is a relational database for storing asset taxonomy and asset maintenance history. It is used for maintenance work management. The global CMMS market is dominated by one organization with a number of smaller competitors. Given the number of users of the CMMS of the large operator, the structure of their tables and the column headings form a defacto standard across many industries. Our example uses data from individual MWO's. An MWO is generated for every maintenance activity either 1) periodically by the CMMS based on a strategy (e.g., an MWO to replace the tires on the car every two years) or 2) by an asset operator or maintainers following a failure event or their identification of deteriorating asset health (e.g., noticing the pump is vibrating). The challenge for an engineer is to look at failure events in a list of hundreds, sometimes thousands of MWO's, and determine if this was a failure mode that was 'expected' according to the FMEA or not. Noting that the FMEA data is stored in a completely separate system and often 'owned' by a different group in the organization. This requires knowledge of the failure modes for each asset, and, in our use case, a review of tens of thousands of MWO's generated a year. The situation in industry currently is that this reconciliation work is not done and as a result there are two costly consequences. The first is that we continue to have to deal with failures that impact production and disrupt planned maintenance work because we do not have an appropriate maintenance strategy for a specific failure mode that was not

captured in the original FMEA work. The second is that we are doing work that is not effective. In other words, we are sending maintenance personnel to do work to address a failure mode we identified during the FMEA process but has not actually manifested in our specific situation. The maintenance work is therefore unnecessary. In both cases there are costs saved from a solution to the challenge question from the General Manager. Our approach is the construction of ontology templates for integrating information from MWO's with FMEA analyses and using reasoning. This is described in Section 5.

3 Pattern-Based Ontology Construction and Instantiation

Our methodology relies on using modularized, hierarchical template libraries within the Reasonable Ontology Templates (OTTR) framework [23] to create the ontology directly from specifications provided by domain experts in familiar formats, e.g., CSV and Excel, as well as from data exported directly from software systems. Using a new mapping feature in OTTR, data is translated into OTTR template instances which are finally expanded into RDF/OWL. This section gives an overview of the framework using examples from our use case and lays the groundwork for the presentation of the methodology in the following section.

The OTTR framework allows modeling patterns to be represented, instantiated, and translated in a precise manner using a macro-like [26] template mechanism. An OTTR template is *instantiated* by providing *arguments* that match the *parameters* of the template. The *signature* of a template is its name and its list of parameters. Templates are recursively defined: regular templates are defined using other templates, while special *base templates* are used to represent structures that cannot be represented in the OTTR language, typically basic structures of other formalisms. Currently, the OTTR framework specifies only one base template natively, the `ottr:Triple` template, which represents an RDF triple. Any correctly defined template instance may therefore be *expanded* to a set of RDF triples using a simple recursive macro expansion.

The OTTR framework is built to increase the efficiency of ontology development and maintenance, and to support and promote sound modeling practices such as don't-repeat-yourself (DRY), encapsulation of complexity, and separating the concerns of the groups involved in the ontology development process. To this end, the OTTR language include features such as parameter typing, list iterators, optional arguments and default values. Example 1 presents a template from our use case demonstrating the use of some of these features. The OTTR framework is formally defined by a set of specifications that describe its formal foundations, type system, serialization formats, and support for data consumption. For more information about OTTR, including formal specifications and interactive examples, visit the web page `ottr.xyz` and the references therein.

Example 1. Figure 1b) contains the `inter:FailureCodes` template. It specifies a single parameter `?codes` of type `NEList<owl:Class>`, which means the template only accepts a non-empty list of OWL classes as its argument. The template is defined using two other templates: `iso-tmp:AllDisjointFailureCodes` and `iso-tmp:FailureModeCode`. The latter is marked with a *list expander* `cross`, which here makes the template apply to each element in the list argument. The template illustrates of how the template mechanism helps hide complexity; the `iso-tmp:FailureModeCode` template is responsible for

```
ISO14224_FailureModeCode,ISO14224_FailureMode,ISO14224_Description
BRD,Breakdown,"Serious damage (Seizure, breakage)"
DOP,Delayed operation,Delayed response to commands
ELF,External leakage - fuel,External leakage of supplied fuel / gas
ELP,External leakage - process medium,"Oil, gas condensate, water"
```

a) Excerpt from CSV file `ISO14224_FMEA_codes_and_FM_description.csv`.

```
inter:FailureCodes[NEList<owl:Class> ?codes] :: {
    cross | iso-tmp:FailureModeCode(++?codes),
    iso-tmp:AllDisjointFailureCodes(?codes)
} .
macro:ObjectPartition[owl:Class ?partition, NEList<owl:Class> ?classes] :: {
    o-owl-ax:DisjointClasses(?classes),
    o-owl-ax:EquivObjectUnionOf(?partition, ?classes)
} .
```

b) The `inter:FailureCodes` and `macro:ObjectPartition` templates.

```
ex:FailureCode a :InstanceMap ;
  :source [ a :H2Source ] ;
  :template inter:FailureCodes ;
  :query """SELECT LISTAGG(CONCAT('fmea:',ISO14224_FailureModeCode),',')
              WITHIN GROUP (ORDER BY ISO14224_FailureModeCode)
    FROM CSVREAD('ISO14224_FMEA_codes_and_FM_description.csv');""" ;
  :argumentMaps([ :type (:NEList :IRI) ]) .
```

c) A bOTTR mapping which maps the CSV file in a) to the `inter:FailureCodes` template in b).

```
inter:FailureCodes((fmea:BRD, fmea:DOP, fmea:ELF, fmea:ELP)) .
```

d) The resulting instance from applying the mapping in c) to the excerpt in a).

```
fmea:ELF  rdfs:subClassOf  fmea:FailureCode .
fmea:ELP  rdfs:subClassOf  fmea:FailureCode .
fmea:DOP  rdfs:subClassOf  fmea:FailureCode .
fmea:BRD  rdfs:subClassOf  fmea:FailureCode .

fmea:FailureCode  a  owl:Class ;
  owl:equivalentClass  [ a  owl:Class ;
    owl:unionOf  ( fmea:BRD fmea:DOP fmea:ELF fmea:ELP ) ] .

[ a  owl:AllDisjointClasses ;
  owl:members  ( fmea:BRD fmea:DOP fmea:ELF fmea:ELP ) ] .
```

e) The resulting RDF output of expanding the instance in d).

Fig. 1: End-to-end example following the pipeline depicted in Figure 2. Figures 1a), 1c) and 1e) are sampled from the project published at the open git repository `gitlab.com/ottr/pub/2020-asset-maintenance`. Figure 1b) is taken from the template library at `tpl.ottr.xyz`.

modeling individual failure mode codes, while the `iso-tmp:AllDisjointFailureCodes` template models relationships between the complete set of failure mode codes. The template indirectly depends on the `macro:ObjectPartition` template, which represents a generic logical partitioning of classes, an axiom/macro that does not exist in OWL [26].

The ability to share and reuse templates for common modeling patterns and vocabularies is central to the OTTR framework. The template library at `tpl.ottr.xyz` contains all the templates used for our use case, including a "standardized" set of templates that model common patterns in OWL, RDFS, and RDF. The templates are published following many best practices for linked open data publication and is backed by an open git repository.[3] Our intention is that this library will grow to cover common patterns for central vocabularies, driven by community efforts and backed by a continuous delivery toolchain[4] to ensure quality and robustness of the library. Management guidelines for the library are in development.

Example 2. As an example of possible template reuse, consider the templates in Figure 1b). The `inter:FailureCodes` template may be used for capturing any list of failure mode codes from the standard (ISO 14224). The `macro:ObjectPartition` template may be used to express class partitions, a logical statement which can be useful for any OWL ontology.

A template represents a modeling pattern and is not meant to contain data cleaning and preparation instructions. This is by design so that these different tasks in the data pipeline are clearly separated, which again makes the templates easier to reuse. To support data extraction and preparation, the OTTR framework currently provides two options for translating data into the form specified by a template: either by adding tags to existing tabular data files (e.g., CSV or Excel spreadsheets), or via querying. The latter, called *bOTTR*,[5] specifies a mapping language where query result sets in tabular form are converted into template instances. bOTTR currently supports SPARQL queries and SQL queries over JDBC. Additionally, it is possible to use other transformation and mapping languages and write template instances directly in one of the available serialization formats, one of which is based on RDF. For our use case we have created a set of bOTTR mappings for each type of input source. The clear separation between bOTTR mappings that extract and normalize data, and OTTR templates for the modeling, allows these artifacts to be managed and maintained independently by different people with the necessary competence required for each task.

Example 3. Figure 1c) shows an example of a bOTTR mapping which instantiates the template from Example 1. bOTTR is specified as an RDF vocabulary. This mapping uses the H2 database engine[6] to load and query the CSV file `ISO14224_FMEA_codes_and_FM_description.csv` with the given SQL query. Each row in the query result set is converted into an instance of the `inter:FailureCodes` template. The `:argumentMaps` list specifies how the database string values are translated to RDF terms.

[3] `https://gitlab.com/ottr/templates`
[4] See, e.g., `https://en.wikipedia.org/wiki/Continuous_delivery`
[5] `https://spec.ottr.xyz/bOTTR/0.1/`
[6] `https://www.h2database.com`

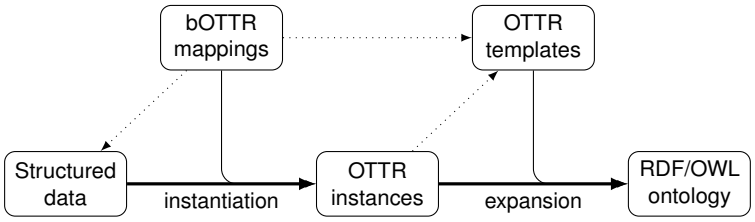

Fig. 2: OTTR framework data pipeline. The structure of the OTTR templates are discussed in further detail in Section 4.

The data pipeline for converting input data to ontologies using the OTTR framework is shown in Figure 2. Structured input data is translated to OTTR template instances as specified by bOTTR mappings. The bOTTR mappings contain references to the input sources and to OTTR templates. The OTTR template instances are then expanded to RDF triples (which represent an OWL ontology) using the template specifications. A complete end-to-end example of the pipeline shown in Figure 2 is given in Figure 1. The process is executed by the open source reference implementation for the OTTR framework, Lutra.[7]

4 Template Library Organization and Architecture

A *template library* is a set of templates that is published and curated. The templates within a library can be categorized according to the purpose they serve. A template of a certain type may only depend on templates of the same or lower type. Thus, these types form a layered structure, which can be seen in Figure 3. Each template type corresponds to a level of modeling granularity. The lowest type, *base*, consists of only one template, `ottr:Triple`, which simply corresponds to an RDF triple. *Utility* templates are used to improve template formulation, e.g., by grouping common triples together to avoid unnecessary repetition. Above these lie *logical* templates, which represent OWL axioms as well as convenient combinations of axioms. Base, utility, and logical templates are inherently domain-independent and as such reusable across various ontology engineering projects.

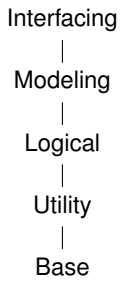

Fig. 3: Template types

Modeling templates represent modeling patterns in a specific domain. They are independent from specific input formats and represent the concepts and relationships occurring in the domain. For example, in our use case the template `iso-tmp:AllDisjoint FailureCodes` in Figure 1 is a modeling template which models the semantic relationship between all failure mode codes described in the ISO 14224 standard.

[7] `https://gitlab.com/ottr/lutra/lutra`

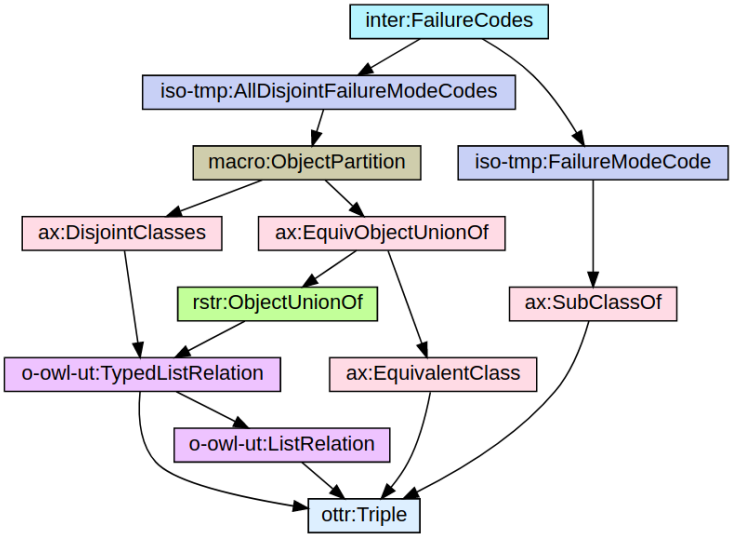

Fig. 4: The dependency graph for the `inter:FailureCodes` template. The nodes are colored according the namespace of the template. The graph is an adapted version of the one found at `http://tpl.ottr.xyz/p/asset-maintenance/interface/0.2/FailureCodes.html`

An *interface* template is a more specific type than the other template types, and is created for a specific input in mind. Their signatures correspond closely to that of the input data, and a key restriction is that they may depend only on modeling templates (and, explicitly, not logical, utility or base templates). For example, in the use case each interface template models the data in (possibly parts of) one CSV file.

This layered architecture helps in the use, development, and maintenance of template libraries. Each layer provides an "interface" for the layer(s) above to use, hence hiding the complexity of the layers below. The various layers typically fit different competencies and may be managed by people with different expertise. Base templates represent the core and are managed by the maintainers of the OTTR framework. Logical and utility templates require a firm grasp of semantic technologies. Modeling templates require domain knowledge and familiarity with the vocabularies used, while interface templates require a good understanding of the input data.

Example 4. Figure 4 depicts the dependency graph of the template discussed in Example 1. The graph illustrates how the template from our use case follows the template layering approach. The different layers can be seen from the namespaces of the template IRIs, where the prefix `inter` corresponds to interface templates and `iso-tmp` corresponds to modeling templates for a subdomain of our use case.

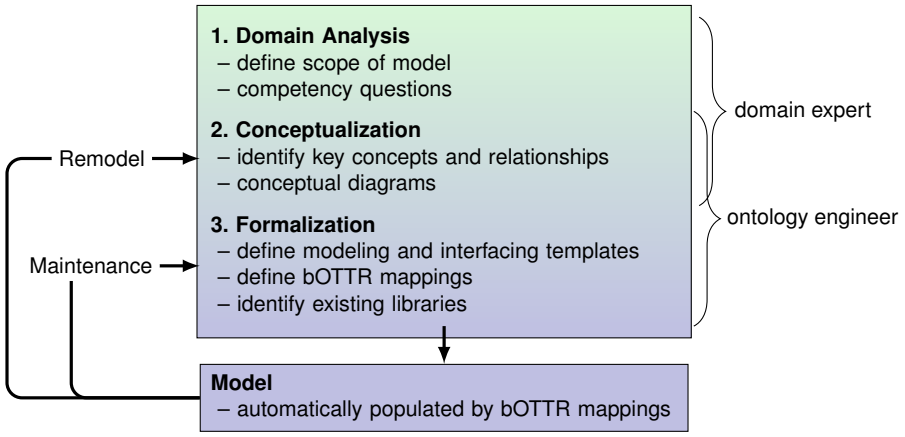

Fig. 5: An overview of the tasks in the methodology.

5 Methodology

In this section, we present in detail the methodology used to construct the ontology for our use case. The key component in the process is a well-constructed and curated template library as described in Section 4. In general, constructing an ontology involves many steps ranging from feasibility studies and technical tasks to evaluating its use (for a more detailed overview, see, e.g., [11]). The methodology described below strives to provide a guideline for the technical aspects of the ontology engineering process. In particular, it focuses on how to structure template libraries to bridge the gap between conceptual understanding, logical formulation, and existing tabular data. Other important tasks such as feasibility and use analyses should be performed, however they are outside of the scope of the presented methodology. An overview of the tasks within our methodology can be seen in Figure 5, and in the following sections we detail the steps involved for each of these tasks.

5.1 Domain Analysis

An important first step in constructing any ontology is defining the parts of the domain we wish to model. Thus, key questions to be answered are: (1) what key concepts and relationships should be modeled in the ontology; (2) what competency questions should the semantics of the model be able to answer; (3) is the domain we wish to model naturally divided into subdomains?

In our use case, the model should integrate information from distinct sources: existing international standards such as parts of ISO 14224 [7] for failure mode codes, FMEA tables based on IEC 60812 [6], asset breakdown hierarchies containing the relevant assets and their components, as well as maintenance work orders recorded in the CMMS. As such, this concrete use case has a natural subdivision in the desired model: *ISO 14224*, *FMEA information*, *asset breakdown structures*, and *maintenance work order information*. This separation should be reflected in the template library.

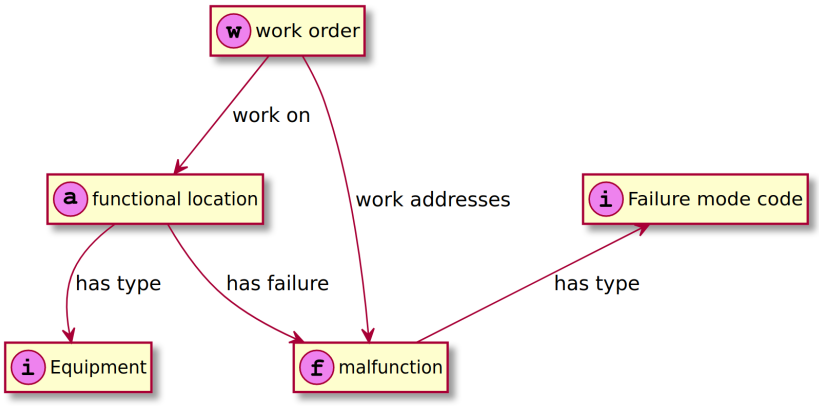

Fig. 6: A conceptual diagram representing some of the key terms in the FMEA model. The letter in the circle represents the subdomain the concept comes from: *w* for work orders, *i* for iso standard, *a* for asset, and *f* for FMEA.

The desired functionality of the ontology is to be able to detect malfunctions experienced during the life cycle of the asset, and recorded in the MWO's, that were not predicted in the FMEA. It is important to note that these sources provide different types of information in an ontological sense. Some of the sources, such as the international standards and parts of the FMEA tables will populate the TBox (i.e., need to be realized as axioms in the final ontology), whereas others serve as data to be integrated into the ABox. This will play a larger role in the Formalization task.

5.2 Conceptualization

In the Conceptualization task, the goal is to identify the important concepts and relationships the ontology should model. This stage does not yet involve any logical formalism; rather, ontology engineers and domain experts discuss the key relationships within the domain and represent them via conceptual modeling diagrams.

Many of the entities in our ontology map to the names in column headings in the CSV tables used. These are identical to entities also used in the literature, e.g., functional location, found for example in FMEA ontologies proposed by [13, 2, 21] and efforts to model failure events in MWO's by [3, 10, 20]. However we needed to create some additional entities not explicitly represented in the CSV files but which are necessary to support the desired semantics. An example would be the concept of a 'malfunction'.

Some of the key concepts from the use case, as identified by the domain expert and ontology engineer, are depicted in in Figure 6. Note that all concepts and relationships are strictly on a conceptual level and need not correspond to the final logical implementation in the ontology.

5.3 Formalization

The Formalization task consists of multiple steps: (1) encoding the key notions identified in the Conceptualization task as modeling templates, and (2) creating interfacing templates and bOTTR mappings that correspond to the provided input data schemas.

Modeling templates In this subtask, the conceptual diagrams from the previous task are formalized as modeling templates. The template name should represent a key notion or relationship within the domain that the domain expert understands. The template body provides the semantics of the modeling pattern and it may only depend on templates of a lower or equal type (see Section 4).

It is worth discussing the benefits of representing the relationships of the conceptual diagram as templates, as opposed to simply taking a box-is-class, arrow-is-property approach. Representing these as templates allows for more complex modeling without losing sight of the conceptual relationships being modeled. For instance, a domain expert need not be confronted with how concepts and classes are aligned with upper ontologies. The template signatures correspond to the terms occurring in the conceptual model. The template definitions, i.e., how they expand to an OWL ontology, is defined by the ontology expert in a dialogue with the domain expert. These modeling templates can be seen as an API for the model, allowing the ontology expert to construct interfacing templates corresponding to queries over the tabular data provided by the domain expert.

Interfacing templates and bOTTR mappings The final model is constructed by populating the modeling templates from the previous subtask using existing standards and structured information. In order for this to be possible, this structured information needs to be mapped into template instances. This can be achieved by using bOTTR mappings, as introduced in Section 3. As tabular data does not always correspond one-to-one with conceptual notions in a domain (a row in a database may, for instance, contain information about various different objects), domain experts can, with the help of ontology engineers, group modeling templates into *interfacing templates* which can then be populated using bOTTR mappings. As mentioned in Section 4, interfacing templates may only depend on modeling templates. This restriction might appear rather strict, however it ensures that the resulting ontology maintains modeling consistency by using only modeling patterns that the ontology engineer and domain expert have agreed upon. For example, a domain expert might discover during the formalization of interfacing templates that a specific modeling pattern is not captured by existing modeling templates. Rather than instantiating the logical and utility templates comprising the pattern directly, this pattern is a possible candidate for a new modeling template.

Example 5. Figure 1 provides an end-to-end example of this formalization process for a fragment of the use case. Figure 1b) shows an interfacing template `inter:FailureCodes` which depends on two modeling templates, `iso-tmp:FailureModeCode` and `iso-tmp:AllDisjointFailureModeCodes`. These two modeling templates represent the semantic modeling of the failure mode code concept from Figure 6, i.e., that failure mode codes are represented as classes and that the class `fmea:FailureCode` is the disjoint union of all failure mode codes. Figure 1c) shows the bOTTR map used to translate the source data into template instances.

5.4 Maintenance

One benefit to this approach is that, once the Formalization task is completed, the ontology can be constructed at scale: structured data is translated into interfacing templates, which via modeling templates expand into an ontology. This allows the domain expert to explore the model over actual data as opposed to smaller, prototype examples.

Of course, it is likely that at some point this model must be changed in some way. This is accomplished by maintaining the bOTTR mappings and interface templates as well as the modeling templates. Should the semantics of the final ontology not correspond to the domain experts' understanding, e.g., if the reasoning results in unexpected and undesirable inferences, then the offending modeling templates need to be revisited. This may result in isolated changes to certain modeling templates, or larger changes that also require amendments to lower layers of the template library. In the case of missing or wrong data translation, the interfacing templates and bOTTR mappings need to be altered.

Example 6. Due to the modular nature of a templates-based approach, any changes made to the templates get automatically transferred to the whole ontology. For instance, if the modeling of the concept *failure mode code* must be altered, this will be done in the modeling templates `iso-tmp:FailureModeCode` and `iso-tmp:AllDisjointFailureModeCodes` (see Figure 1b)). No changes need to be made to the interfacing templates or mappings, and the changes are automatically adopted by the model.

The input datasets, as well as versioned bOTTR mappings, batch scripts, and ontologies produced following this methodology are published at the open git repository `gitlab.com/ottr/pub/2020-asset-maintenance`.

6 Discussion

By following the methodology described above, we were able to rapidly construct a working ontology over real-world data that was capable of addressing the domain experts' needs. The use case data was supplied by global resources company and has been anonymized due to privacy issues. The use case had 87 functional locations (assets, subsystems, and components), 47 failure mode codes, and 2 years of MWO data. The ontology was able to identify five failures in flowmeters associated with seal systems on pressure vessel agitators, each failure causing process interruptions. It transpired that the failure mode had not been identified in the original FMEA work and hence there was no maintenance strategy (in this case a routine inspection) in place. Similar issues were identified at 36 other functional locations, some of them associated with safety critical elements.

There were some interesting challenges faced during the development process that are worth discussing. During the iterative development of the ontology, multiple types of modeling changes were made. These ranged from comparatively simple tasks (e.g., changing the naming scheme) to a complex reworking of the ontology's structure and semantics. As an example, in an early version the model made use of consistency checking to determine whether a malfunction was identified by the FMEA. This yielded the desired functionality, unexpected malfunctions were signaled as inconsistencies

and could be further examined using justification tools. However, this approach has limitations that could affect interoperability with other ontologies. During the operation of an asset, unexpected malfunctions might occur often and one will not always be able to remove this source of inconsistency. The benefits of semantic modeling are in this case limited, since an inconsistent ontology entails everything. This was remedied by altering the modeling of malfunctions, where malfunctions not predicted by the FMEA are inferred to be members of the `UnexpectedMalfunction` class. The resulting remodeling process, from logical formalization to implementation in the template library, was completed in a matter of a few hours. By nature of the architecture—the specification *is* the implementation—all changes are immediately deployed at scale, allowing the domain expert to directly see the effects of the changes.

A key motivator for the development of this methodology is to separate the concerns of the various parties involved in ontology engineering. The domain expert needs to be involved throughout the entire process but without requiring them to understand the technical and logical nuances of ontology engineering. Similarly, the ontology expert is adept at modeling logical relationships between entities, but cannot be expected to become an expert in the field they are modeling. This separation of concerns was evident during the development of our use case. The ontology engineer only needed to understand the relationship between concepts (e.g., how the relationship between equipment types and failure mode codes should be represented), as opposed to being concerned with concrete domain details (e.g., that the failure mode code INL stands for "internal leakage" and is only associated with specific equipment types). Additionally, in the remodeling process described above, all maintenance tasks were clearly defined and had their specific place in the hierarchy: all semantic modeling happens in the modeling templates, and linking these to the tabular data are the bOTTR and interfacing templates.

The methodology presented is designed specifically for domains where the majority of information and data is available in a structured format. Developing and maintaining a high-quality template library is a time-consuming task. However, we believe this pays off. The semantics of the model are encoded in the modeling templates. Thus, extending the model with further data (e.g., encoding the relationships of equipment types and failure codes not listed in ISO 14224) involves creating suitable interface templates as well as adding modeling templates for missing semantic relationships. Therefore, complexity of generating, prototyping and maintaining the ontology using the proposed methodology is not governed by the ontology's size but rather the regularity of its domain.

7 Related Work

Various ontology engineering methodologies have been proposed to streamline ontology development. High level process-oriented methodologies such as [14, 24] cover the complete ontology engineering process, whereas micro-level methodologies focus specifically on the ontology authoring task [16]. Furthermore, agile approaches like eXtreme Design (XD) [19] are specifically developed for ontology design patterns (ODP's) [4]. Of these, our methodology is perhaps most reminiscent of the eXtreme Design approach in that both are fundamentally based on using modeling patterns which are published

in repositories for reuse.[8] Yet an important difference is that Content ODP's, which are the pattern structure used in XD, are more generic or high-level in nature than OTTR templates. For example it is not possible to reliably instantiate a Content ODP in a functional manner, as the "reuse" of ODP's lacks a clear, formal definition and thus can be done in different ways and with different results [4]. For the OTTR framework, functional instantiation is a core design feature and allows OTTR templates to completely specify the translation from domain conceptualizations in a tabular format to ontologies.

A selection of different approaches that share similarities to ours is Generic Ontology Design Patterns (GOPD), ROBOT, Populus and Mapping Master. GOPD are formulated using GDOL [12], an extension of the Distributed Ontology, Modeling, and Specification Language (DOL) that supports a parameterization mechanism for ontologies. It is a meta-language for combining theories from a wide range of logics, including description logics and OWL, under one formalism while supporting features similar to OTTR framework like pattern definition, instantiation, and nesting.

ROBOT [8] is a tool for automating ontology workflows, including tasks such as managing ontology imports, apply ontology reasoning and running test suites, and, more relevant in this context, it supports instantiating modeling patterns in the form of Dead simple OWL design patterns [18].

Populous [9] presents users with a table based form, which can be exported as spreadsheets, where columns are constrained to take values from particular ontologies. Populated tables are mapped to patterns defined by the Ontology Pre-Processor Language (OPPL) [5] that in turn generate the ontology's content.

Mapping Master [17] is a Protégé plugin that uses a mapping language, M^2, to extend the Manchester OWL syntax for translating spreadsheet data into OWL ontologies.

These approaches differ from ours in that they are not based on compositional modeling patterns organized as hierarchical template libraries for ease of maintenance and reuse. Of these approaches only GDOL and OPPL are capable of expressing nested patterns. To the best of our knowledge, there exists no dedicated library of ontology templates published for sharing and reuse other than the OTTR template library.

8 Future Work

The methodology presented in this paper shows promise for streamlining ontology engineering projects within industries with large amounts of disjoint yet standardized data sources. The use case was built on real-world, anonymized data with the goal of demonstrating the feasibility of semantic technologies in such industries.

The modeling employed in the use case, in particular, serves as a proof-of-concept and is likely to be changed in the foreseeable future. In particular, alignment with an upper ontology such as Basic Formal Ontology (BFO) [1] is a priority to ensure interoperability with other ontologies. Throughout this process, the template library will be actively updated and maintained at `tpl.ottr.xyz` under the namespace `http://tpl.ottr.xyz/p/asset-maintenance/`.

The benefits of introducing our methodology based on use of OTTR template libraries comes with the added cost of maintaining these libraries. In order to facilitate easier

[8] A central repository for ODP's is `http://ontologydesignpatterns.org`.

maintenance, automated methods are necessary. The formal foundation of OTTR templates allows for formal relations and specification of anomalies in template libraries, and hence for automated procedures for detection and repair of these. Initial experiments on template library maintenance show promising results [23], but require more development to reach a mature state. Improving the tool support for the development and use of templates is also future work.

Our methodology contains concrete, formally checkable conditions, such as the dependency constraints placed on template types discussed in Section 4. Currently these conditions must be verified manually. It would be beneficial to develop tools designed to check that the methodology is used correctly, e.g., that bOTTR mappings only instantiate interfacing templates, that interfacing templates only depend on modeling templates, etc. Such functionality could be enhanced even more by allowing for public/private templates and submodules. This would give the ontology engineer more control over how the model is generated.

Finally, in addition to the maintenance of the template library, the bOTTR mappings must also be actively maintained. Currently, the translation from tabular data to IRIs is performed through functions built into the query language (see the use of `CONCAT` in Figure 1c) used to generate a valid IRI). These translations are interspersed in many of the bOTTR mappings, which is not ideal. A more robust, declarative approach to data preparation tasks, such as string manipulation and IRI generation, is crucial future work.

Acknowledgements We would like to thank Johan W. Klüwer for his constructive input on the use case modeling. We would also like to thank Leif Harald Karlsen for his assistance in structuring the template library and developing Lutra.

Contributions from the University of Western Australia for this work was supported in part by the BHP Fellowship for Engineering for Remote Operations.

References

[1] R. Arp, B. Smith, and A. D. Spear. *Building Ontologies with Basic Formal Ontology*. Massachusetts Institute of Technology, 2015.

[2] V. Ebrahimipour, K. Rezaie, and S. Shokravi. “An Ontology Approach to Support FMEA Studies”. In: *Expert Systems with Applications* 37.1 (2010).

[3] V. Ebrahimipour and S. Yacout. “Ontology-Based Schema to Support Maintenance Knowledge Representation with a Case Study of a Pneumatic Valve”. In: *IEEE Trans. Syst. Man Cybern. Syst.* 45.4 (2015).

[4] A. Gangemi and V. Presutti. “Ontology Design Patterns”. In: *Handbook on Ontologies*. Springer Berlin Heidelberg, 2009.

[5] L. Iannone, A. L. Rector, and R. Stevens. “Embedding Knowledge Patterns into OWL”. In: *The Semantic Web: Research and Applications, ESWC 2009*. Vol. 5554. LNCS. Springer, 2009.

[6] *IEC 60812: Analysis Techniques for System Reliability-Procedure for Failure Mode and Effects Analysis (FMEA)*. International Electrotechnical Commission, 2006.

[7] *ISO 14224:2016 Petroleum, Petrochemical and Natural Gas Industries — Collection and Exchange of Reliability and Maintenance Data for Equipment*. Standard. International Organization for Standardization, 2016.

[8] R. C. Jackson et al. “ROBOT: A Tool for Automating Ontology Workflows”. In: *BMC Bioinform.* 20.1 (2019).
[9] S. Jupp et al. “Populous: A Tool for Building OWL Ontologies from Templates”. In: *BMC Bioinform.* 13.S-1 (2012).
[10] M.-H. Karray, F. Ameri, M. Hodkiewicz, and T. Louge. “ROMAIN: Towards a BFO Compliant Reference Ontology for Industrial Maintenance”. In: *Appl. Ontology* 14.2 (2019).
[11] C. M. Keet. *An Introduction to Ontology Engineering*. College Publications, 2018.
[12] B. Krieg-Brückner and T. Mossakowski. “Generic Ontologies and Generic Ontology Design Patterns”. In: *Proceedings of the 8th Workshop on Ontology Design and Patterns (WOP 2017)*. Vol. 2043. CEUR. CEUR-WS.org, 2017.
[13] B. H. Lee. “Using FMEA Models and Ontologies to Build Diagnostic Models”. In: *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 15.4 (2001).
[14] M. Lopez, A. Gomez-Perez, J. Sierra, and A. Sierra. “Building a Chemical Ontology Using Methontology and the Ontology Design Environment”. In: *IEEE Intelligent Systems* 14.1 (1999).
[15] M. A. Musen. “The Protégé Project: A Look Back and a Look Forward”. In: *AI Matters* 1.4 (2015).
[16] N. F. Noy and D. L. McGuinness. *Ontology Development 101: A Guide to Creating Your First Ontology*. Technical Report KSL-01-05. Stanford Knowledge Systems Laboratory, 2001.
[17] M. J. O’Connor, C. Halaschek-Wiener, and M. A. Musen. “Mapping Master: A Flexible Approach for Mapping Spreadsheets to OWL”. In: *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference*. Vol. 6497. LNCS. Springer, 2010.
[18] D. Osumi-Sutherland, M. Courtot, J. P. Balhoff, and C. J. Mungall. “Dead Simple OWL Design Patterns”. In: *J. Biomed. Semant.* 8.1 (2017).
[19] V. Presutti, E. Daga, A. Gangemi, and E. Blomqvist. “eXtreme Design with Content Ontology Design Patterns”. In: *Proceedings of the Workshop on Ontology Patterns (WOP 2009)*. Vol. 516. CEUR. CEUR-WS.org, 2009.
[20] D. G. Rajpathak. “An Ontology Based Text Mining System for Knowledge Discovery from the Diagnosis Data in the Automotive Domain”. In: *Computers in Industry* 64.5 (2013).
[21] Z. Rehman and C. V. Kifor. “An Ontology to Support Semantic Management of FMEA Knowledge”. In: *Int. J. Comput. Commun. Control* 11.4 (2016).
[22] E. P. B. Simperl, M. Mochól, and T. Bürger. “Achieving Maturity: The State of Practice in Ontology Engineering in 2009”. In: *Int. J. Comput. Sci. Appl.* 7.1 (2010).
[23] M. G. Skjæveland, D. P. Lupp, L. H. Karlsen, and H. Forssell. “Practical Ontology Pattern Instantiation, Discovery, and Maintenance with Reasonable Ontology Templates”. In: *The Semantic Web - ISWC 2018*. Vol. 11136. LNCS. Springer, 2018.
[24] M. C. Suárez-Figueroa, A. Gómez-Pérez, and M. Fernández-López. “The NeOn Methodology for Ontology Engineering”. In: *Ontology Engineering in a Networked World*. Springer, 2012.
[25] T. Tudorache. “Ontology Engineering: Current State, Challenges, and Future Directions”. In: *Semantic Web* 11.1 (2020).
[26] D. Vrandečić. “Explicit Knowledge Engineering Patterns with Macros”. In: *Proceedings of the Ontology Patterns for the Semantic Web Workshop at ISWC 2005*. 2005.