Transformation of the Common Information Model to OWL

Andreas Textor[1,2], Jeanne Stynes[2], and Reinhold Kroeger[1]

[1] RheinMain University of Applied Sciences
Distributed Systems Lab
Kurt-Schumacher-Ring 18, D-65197 Wiesbaden, Germany
{andreas.textor,reinhold.kroeger}@hs-rm.de
[2] Cork Institute of Technology
Department of Computing
Rossa Avenue, Bishopstown, Cork, Ireland
jeanne.stynes@cit.ie

Abstract. Managing an IT environment requires the exchange of structured data between different agents. The Common Information Model (CIM) is a comprehensive open standard that specifies how managed elements in an IT environment are modelled as a set of common objects and relationships between them. It has however limited support for knowledge interoperability and aggregation, as well as reasoning. By converting the existing CIM model into a format that can be processed by semantic web tools, these limitations can be overcome. This paper describes how CIM can be converted into a Web Ontology Language (OWL) ontology including constructs for which no obvious direct conversion exists, such as CIM qualifiers.

1 Introduction

With the constantly growing size and complexity of IT environments, comprehensive management systems that can effectively and efficiently manage these environments, are essential. A number of commercial and free systems exist that cover various parts of the required feature set for such management tasks. Usually, the management system is not comprised by a single tool, but by a set of different tools. To provide a unified view on the environment and allow interoperability between multiple management tools and managed elements, several integrated network management models were developed. Notable examples are the OSI network management model (also known as CMIP, the name of its protocol) and the still widely used simple network management model (SNMP). A more recent approach to specify a comprehensive IT management model is the Common Information Model (CIM) which is described in more detail in section 3. This widely recognized Distributed Management Task Force (DMTF) standard allows consistent management of managed elements in an IT environment. CIM is actively used; for example, the storage standard SMI-S (Storage Management Initiative Specification) of the SNIA (Storage Networking Industry Association, [1]) is based on CIM.

F. Daniel and F.M. Facca (Eds.): ICWE 2010 Workshops, LNCS 6385, pp. 163–174, 2010.
© Springer-Verlag Berlin Heidelberg 2010

To establish interoperability mechanisms between the heterogenous integrated management models, mappings between different types of models can be defined. However, one question arises that can not be easily answered: "What happens when two different domains represent the same concept in a different way? A merely syntactic translation from the source model will not give the existing concept in the destination" [2]. This problem is known as *Semantic Matching* or *Ontology Matching* (see e.g. [3]). The problem can be approached by using ontologies to clearly define the semantics. The long term goal is to perform IT management based on a semantic foundation and a comprehensive domain model with the ability to model management rules in this model.

An IT management ontology can not only be used to clearly define the semantics of the managed elements in the managed system, but can also be used when describing associations of the managed system to adjacent domains. For example, if processes defined in an ontology for semantic business process management (see e.g. [4]) refer to physical or logical IT systems, such references can be directly and unambiguously expressed in the particular ontology. With the increasing development of semantic web techologies, including the specification of the Web Ontology Language (OWL) and numerous publications regarding merging and mapping of ontologies, querying, distributed reasoning etc., an ontology-based IT management approach can probably profit even more.

One aspect that is common to most recent approaches of applying semantic web technologies to the domain of IT management is the use of the Common Information Model as the domain model. To allow for semantic interoperability in IT management, a corresponding management ontology is required, and in [5], CIM has been proposed for this purpose. As [6] points out, CIM is usable for inferring properties about distributed systems, but is a semi-formal ontology with limited support for knowledge interoperability and aggregation, as well as reasoning. To create an IT management domain model that overcomes these shortcomings, a conversion of CIM to OWL is desirable. OWL can be used with reasoners, and can be augmented with rules formulated in the Semantic Web Rule Language (SWRL).

This paper presents how a conversion from CIM to OWL can be performed. Section 2 describes existing approaches to a conversion, section 3 gives an overview of the Common Information Model. Sections 5 and 6 describe the conversion of structural and additional CIM elements to OWL, respectively. Section 7 explains details of the implementation and examines the properties of the resulting ontology. The paper closes with a conclusion in section 8.

2 Related Work

The idea of applying semantic web technologies to the domain of IT management has been examined in several publications, e.g. [7,8]. Although CIM is often proposed for this purpose, the conversion of the native format in which CIM is specified into an ontology is not trivial. Several publications exist that attempt to create a conversion of CIM to OWL. In [6] the authors compare possible

conversions of CIM to RDFS (Resource Description Framework Schema) and to OWL. They find that RDFS is unsuitable to express CIM as it "does not provide constructs for expressing cardinality restrictions such as those used for describing association or aggregation relationships between CIM classes". Also, some CIM qualifiers can not be adequately expressed in RDFS. They go on to construct an OWL-based ontology for CIM by using a previously defined mapping of UML to DAML+OIL (which is the predecessor to OWL) and create a mapping of CIM to UML. The conversion of structural information (classes and properties) can be expressed, but most CIM concepts (e.g. CIM qualifiers) have no mappings to either UML constructs or OWL constructs, so the resulting ontology lacks a large part of the information that is provided in the original model.

In [9] the author also addresses the conversion of CIM to OWL, but several decisions in the conversion prevent a complete or near-complete conversion: Properties of CIM classes are mapped to OWL datatype properties, which prevents any property that has the type of another CIM class (object properties must be employed for this to be possible). The author addresses the issue of CIM properties that are explicitly scoped to the corresponding class, but in OWL have global scope, by using the OWL *allValuesFrom* restriction. This makes it impossible to specify properties for multiple classes that have the same name but different ranges. Apart from that, a lot of CIM constructs and qualifiers are not considered at all in the conversion, such as Aggregration/Composition, Value/ValueMap, Keys, etc.

A more complete conversion approach from CIM to OWL is described in [10]. The authors introduce a meta-ontology that is used to model the CIM constructs that have no direct OWL correspondence, e.g. default values or qualifiers that set a value read-only or write-only. However, they do not describe how the meta-ontology is constructed. The authors handle more of the CIM constructs than the previously described approaches, such as several qualifiers, but do not describe how more complex elements, such as CIM methods, can be converted. Also, they leave out several of the restrictions the CIM defines, e.g. data types of properties are lost and in their approach it is generally not possible to support constraints for properties, such as `MaxLen` for strings.

3 DMTF CIM Overview

This section briefly describes the basic properties of the Common Information Model. CIM defines how managed elements in an IT environment are represented as a set of objects and relationships between them. The model is intended to allow a consistent management of these managed elements, independent of their manufacturer or provider. Unlike SMI (Structure of Management Information, the model underlying the popular SNMP protocol), CIM is an object-oriented model. CIM consists of

– A basic information model called the *meta schema*. The meta schema is defined using the Unified Modeling Language (UML, [11]).

– A syntax for the description of management objects called the *Managed Object Format* (MOF).
– Two layers of generic management object classes called *Core Model* and *Common Model*.

Figure 1 shows the CIM meta schema definition in UML, as shown in the CIM specification [12]. The meta schema specifies most of the elements that are common in object-oriented modelling, namely

– *Classes*, *Properties* and *Methods*. The class hierarchy supports single inheritance (generalization) and overloading of methods. For methods, the CIM schema specifies only the prototypes of methods, not the implementation.
– *References* are a special kind of property that point to classes.
– *Qualifiers* are used to set additional characteristics of Named Elements, e.g. possible access rules for properties (`READ`, `WRITE`), marking a property as a key for instances (`Key`) or marking a class as one that can not be instantiated. Qualifiers can be compared to Java annotations; some qualifiers also have parameters.
– *Associations* are classes that are used to describe a relation between two classes. They usually contain two references.
– *Triggers* represent a state change (such as create, delete, update, or access) of a class instance, and update or access of a property.
– *Indications* are objects created as a result of a trigger. Instances of this class represent concrete events.
– *Schemas* group elements for administrative purposes (e.g. naming).

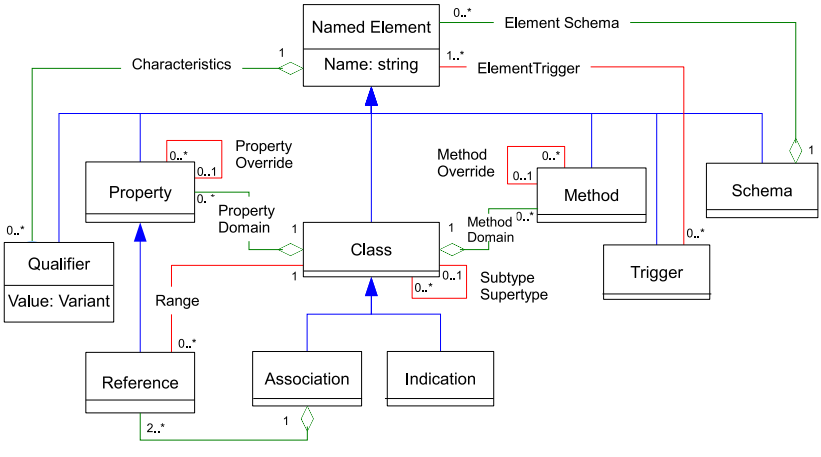

Fig. 1. CIM Meta Schema

Properties, references, parameters and methods (method return values) have a data type. Datatypes that are supported by CIM include {s,u}int{8,16,32,64} (e.g. uint8 or sint32), real{32,64}, string, boolean, datetime and strongly typed references (<classname> ref).

4 Translation Approach

For the construction of an OWL ontology, CIM schema elements have to be translated to OWL. In this case, the goal is an OWL 1 ontology. Some elements can be translated in a straightforward way, while other elements, especially qualifiers, can not be directly translated. To translate CIM constructs for which no direct translation exists, [10] proposes the use of a *CIM meta ontology*, although without describing it in detail. In our approach, we will create an ontology that consists of two parts: One part is the CIM meta ontology, which is statically modelled (i.e. manually) and which consists of super classes, properties and annotations that meta-model CIM constructs which can not be directly translated to OWL. The meta ontology has the namespace `meta`. The second part is the CIM schema ontology, which is modelled using OWL-, RDFS- and CIM meta constructs, and which represents the actual CIM model. It is automatically created by parsing and translating the MOF representation of the CIM schema, and has the namespace `cim`.

The translation of elements was performed incrementally. First of all, the basic CIM elements were translated to OWL constructs, such as classes. Possibilities for the translation of other elements were examined, and where possible, the corresponding OWL construct was used. In some cases a certain translation seems obvious, but isn't actually usable, e.g. using datatype properties to represent CIM properties where object properties are necessary. CIM concepts that have no corresponding OWL construct were modelled in the meta ontology, so that elements in the CIM schema part of the ontology can make use of these concepts. For the translation of the qualifiers, it was sometimes necessary to alter the structure of the model to be able to properly represent the qualifier; other qualifiers were modelled in the meta ontology.

When ontology elements such as classes and properties for corresponding CIM elements are created, the original name is retained. When "helper" elements are created (e.g. elements that do not exist in this form in the CIM schema), the identifiers of the original CIM class are suffixed with a double underscore "__", followed by the helper element name; further elements are concatenated with single underscores.

5 Translation of Structural Elements

The most basic element of the translation is a CIM class, which can be directly translated to `owl:Class`. Likewise, generalization (inheritance) can be expressed using the OWL subclass concept `rdfs:subClassOf`. Apart from generalization, CIM has another basic construct to express relationships between classes, the *Association*. An association is a special kind of a class describing a link between other classes. Associations are essentially normal classes with two typed reference properties, and are marked with the `Association` qualifier.

An *Aggregation* is a specialization of an association, where the two references are explicitly marked as parent and child (the class has both the `Association`

and `Aggregation` qualifiers, and the parent reference is also tagged with the `Associated` qualifier). The aggregation can be specialized even further using the `Composition` qualifier, which adds the semantics of a whole-part/compositional relationship to distinguish it from a collection or basic aggregation. None of the three constructs – association, aggregation and composition – can be directly translated into OWL. Using object properties for representing them is not possible, as association classes can inherit from other classes, but OWL object properties can not inherit from OWL classes. In order to model these constructs, corresponding classes and object properties are modelled in the CIM meta
ontology.

The class `CIM_Association` represents an association and is the domain of the object property `CIM_Association__Role`, from which concrete CIM association object properties can inherit. Each concrete association has two association roles (i.e. two instances of the object property). `CIM_Aggregation` is modelled as a subclass of `CIM_Association` and has two object properties which represent the parent and the child part of the association, respectively, and which are subproperties of `CIM_Association__Role`. The Composition is modelled analogously as a subclass of the aggregation class and the parent and child subproperties of the aggregation parent and child object properties.

The modelling of CIM properties proves to be not straightforward either. While the structural element of a property could be modelled by a simple object property, this approach is not applicable when taking into account that properties can have qualifiers that make assertions about the property. A simple example is the `MaxLen` qualifier that can be used with properties of type `string` to assert a maximum length for its value. Qualifiers that make assertions can, in some cases, be represented by OWL property restrictions, which constrain the range of a property in specific contexts in a variety of ways. One important case where this is not possible is the combination of `Values` and `ValueMap` qualifiers: The `ValueMap` qualifier defines the set of permissible values for a property. When it is used in combination with the `Values` qualifier, the location of the value in the `ValueMap` array determines the location of the corresponding entry in the `Values` array. Listing 1 shows an example property from the CIM class `CIM_Job`.

```
1       [Write, Description ( "This property indicates whether the times "
2       "represented in the RunStartInterval and UntilTime properties "
3       "represent local times or UTC times. Time values are synchronized "
4       "worldwide by using the enumeration value 2, \"UTC Time\"." ),
5          ValueMap { "1", "2" },
6          Values { "Local Time", "UTC Time" }]
7   uint16 LocalOrUtcTime;
```

Listing 1. ValueMap and Values qualifiers

To implement CIM properties in the OWL model, a structure as shown in figure 2 is used. In the figure, rectangular boxes represent OWL classes, rounded boxes with one connection represent datatype properties and rounded boxes with two connections represent object properties. Each CIM property is modelled as a combination of an OWL object property and an OWL class. This

class inherits from the meta ontology class `CIM_Value` that has two datatype properties, "`value`" and "`valueMap`". To model the semantics from the original `ValueMap` and `Values` qualifiers, the class is restricted using `owl:oneOf` to allow only instances from a list of `CIM_Value` instances (one for each `ValueMap`-`Values` combination).

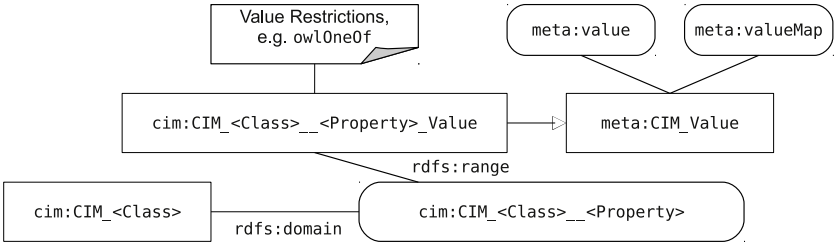

Fig. 2. Modelling of properties

For the example in listing 1, the following OWL elements are created: The class `CIM_Job__LocalOrUtcTime_Value` which is a subclass of `CIM_Value`, the object property `CIM_Job__LocalOrUtcTime` (with domain `CIM_Job` and range `CIM_Job_ _LocalOrUtcTime_Value`), the `CIM_Value` instance `CIM_Job__LocalOrUtcTime_ Value_Local_Time` with data property `value` set to "Local Time" and data property `valueMap` set to "1", likewise an instance for UTC Time, and finally, a subclass restriction on `CIM_Job` on the object property to limit all values to one of the two `CIM_Value` instances.

One important part that was left out so far, is the data type of the modelled property. To preserve the type information, the CIM primitive type is mapped to a corresponding XSD type, which is then added to the object property using the meta ontology annotation `type`. Another possibility would have been to create a subclass of `CIM_Value` for each primitive data type.

Each signed and unsigned number type is translated to its XSD equivalent, e.g. `uint8` becomes `xsd:unsignedByte`, `sint32` becomes `xsd:int`, `real32` becomes `xsd:float`, and so on. Translation is mostly straightforward, except for `char16`, which has to be translated into a string, and `datetime`, which has a corresponding XSD data type but which specifies a different lexical representation for the datetime string. While CIM uses the format "yyyymmddhhmmss.mmmmmmsutc" (where "mmmmmm" is the number of microseconds, and "utc" is the offset from UTC in minutes), XSD uses the format "yyyy-mm-ddThh:mm:ss" (where T is the date/time separator; a decimal point and additional digits to increase the precision of fractional seconds can be added after the seconds part). Datetime values have to be converted accordingly. Default values that can be specified for CIM properties unfortunately can not be expressed in OWL directly. To translate the default values into the ontology, the annotation `defaultValue` is defined in the meta ontology and added to the corresponding object property in the same way as the type annotation is added. Whenever an

instance of a class that has as property with a default value is to be created, this annotation has to be taken into account.

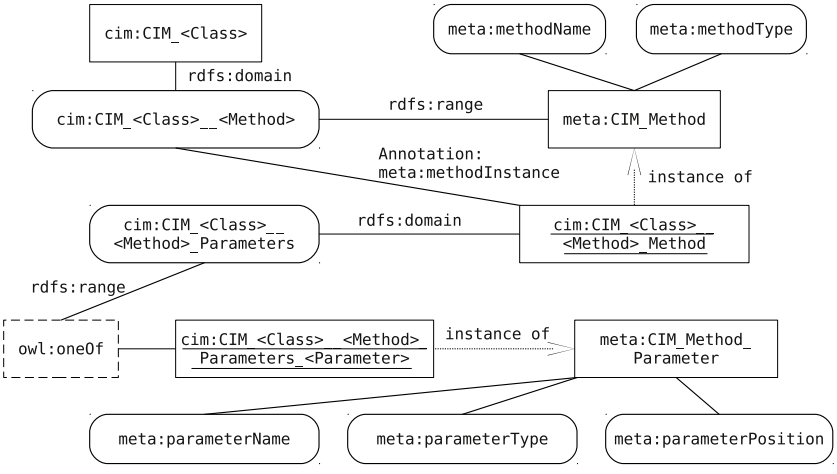

Fig. 3. Modelling of methods

The translation of CIM methods to OWL requires several OWL elements per method as well. Figure 3 shows how the translation is performed. A method basically has to provide the information about its name, type and its parameter list, where each parameter in turn has a name, a type and, in CIM, can have additional qualifiers. The meta ontology provides the class `CIM_Method`, that has two datatype properties, `methodName` and `methodType`. For each method of a CIM class, an OWL object property `CIM_<Class>__<Method>` is created, with the original CIM class as domain, and the `CIM_Method` class as range. An instance of the `CIM_Method` class is created and added as annotation to the object property. Generating both the object property and the instance is necessary, because in OWL, datatype properties can not be attached to object properties, but only to classes and instances. If the method overrides a method from a super class, a corresponding `rdfs:subPropertyOf` is added to the method object property.

Another object property is needed to associate the method instance with its parameters. Each method parameter is represented by an instance of the meta ontology class `CIM_Method_Parameter`, which in turn has datatype properties for its name, type and position. The position property is necessary, because OWL has no construct for ordered collections, and modelling a linked list would add unreasonable complexity. Note that although the OWL reference describes "Enumerated Datatypes" [13] that use `rdf:List` to define a list of values for a datatype property, a similar construct for OWL instances is not specified. The object property representing the parameter list has the method instance as domain and an `owl:oneOf` set of the method parameter instances as range.

6 Translation of Qualifiers

While section 5 described the translation of structural elements (classes, properties and methods), the second part to convert are the CIM qualifiers. Each qualifier can have a specific scope (e.g. can only be used in the context of classes, or in the context of properties and methods, etc.). A qualifier can also have parameters that have to be taken into account.

- *Values* and *ValueMap* are modelled in the meta ontology and were described in section 5.
- The *Override* qualifier can be used with properties and methods and indicates that the element in the derived class overrides the similar construct of the same name in the parent class in the inheritance tree. For the `Override` qualifier, an `rdfs:subPropertyOf` relationship is added to the object property that represents either the CIM property or the CIM method.
- The *Key* qualifier marks a property as the identifying property of the class. Keys are written once at object instantiation and are not modified thereafter. `Key` qualifiers on single properties can be translated by declaring the corresponding object property to be an instance of `owl:InverseFunctionalProperty`. If a property is declared to be inverse-functional, then the object of a property statement uniquely determines the subject (some individual). When more than one property in the source class is marked with the `Key` qualifier, they form a compound key. This poses the same problem as translating a SQL schema with composite primary keys to OWL (as proposed e.g. in [14]). One solution could be to create a synthetic class for the properties that comprise the key. This case is not handled by the converter.
- *Description* can be added to any element and provides a textual description of the element in human-readble format. It is converted into a `rdfs:comment`.
- The *Min* and *Max* qualifiers indicate the minimum and maximum cardinality of a reference property. They can be translated to `owl:minCardinality` and `owl:maxCardinality`.
- *MaxLen* is a qualifier that can only be attached to properties of type `string`, and specifies the maximum length of the string value. For this qualifier, an `owl:Restriction` is created for the class containing the property, as shown in the example in listing 2: using `owl:allValuesFrom` and `owl:withRestrictions`, the maximum length for the value can be specified as an XSD value restriction.
- *MaxValue* and *MinValue* specify the minimum and maximum values for `int` properties and can be translated analogously to the `MaxLen` qualifier, using `xsd:maxInclusive` and `xsd:minInclusive` datatype facets.
- The *Deprecated* qualifier marks an element as deprecated, and is converted usingg `owl:deprecatedClass` and `deprecatedProperty`, respectively.
- *Required* indicates that a non-NULL value is required for the property. It is translated by adding an `owl:minCardinality` of 1 to the object property.

– An *Alias* establishes an alternate name for a property or method and can be converted using the `owl:equivalentProperty` construct with the object property that represents the property or method.

```
<owl:Class rdf:about="#CIM_Account_CreationClassName_Value">
    <rdfs:subClassOf rdf:resource="&meta;CIM_Value"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="&meta;value"/>
        <owl:allValuesFrom>
          <rdf:Description>
             <rdf:type rdf:resource="&rdfs;Datatype"/>
             <owl:onDatatype rdf:resource="&xsd;string"/>
             <owl:withRestrictions rdf:parseType="Collection">
               <rdf:Description>
                  <xsd:maxLength
                       rdf:datatype="&xsd;integer">256</xsd:maxLength>
                  </rdf:Description>
             </owl:withRestrictions>
          </rdf:Description>
        </owl:allValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Listing 2. Conversion of MaxLen qualifier

– The *ModelCorrespondence* qualifier indicates a correspondence between two elements in the CIM schema. It is translated to the `rdfs:seeAlso` construct.
– *Read* and *Write* define that a property is readble or writeable. As no OWL construct exists to represent this feature, they are modelled as annotations in the meta ontology, that are attached to the particular object property.
– *Version* provides the version number of a schema object; it is converted to `owl:versionInfo`.
– The *Abstract* qualifier indicates that a class is abstract and serves only as a base for new classes. A translation of this concept to OWL does not make much sense, as OWL classes can not be compared to classes from object-orientation in this respect. Therefore, a marker annotation `abstract` is modelled in the meta ontology and attached to the OWL class. This does not preserve the orginal semantics, but can be accounted for by tools that create instances of CIM classes.
– The *Units* and *PUnit* qualifiers provide information about the unit in which a property or method is expressed. While the (now deprecated) `Units` qualifier specifies a human-readable format (e.g. "Tenths of Decibels"), the `PUnit` qualifier uses a machine-readable format (e.g. "decibels$*10^{-1}$"). The qualifiers are converted using the meta ontology annotations `units` and `punit`, respectively.
– *UMLPackagePath* specifies the a position within a UML package hierarchy for a CIM class. A class hierarchy other than the inheritance hierarchy is not modelled in the ontology, so the package path is also modelled as an annotation in the meta ontology.
– The *ClassConstraint*, *PropertyConstraint* and *MethodConstraint* qualifiers can be used to specify OCL constraints for the particular elements. The

Object Constraint Language (OCL) is a declarative language for describing rules that apply to UML models. These qualifiers are currently not handled by the converter. One possibility to handle these qualifiers is to convert the OCL expressions to SWRL, as proposed in [15], which can then be included in the ontology.

Some more CIM qualifiers exist, which are not discussed in detail here. These qualifiers mostly serve as flags, for example the `Experimental` qualifier, and can be converted by creating appropriate annotations in the meta ontology.

7 Implementation and Resulting Ontology

The converter was prototypically implemented in Scala [16], a programming language that integrates object-oriented and functional features and that compiles to Java Byte Code. Java libraries can be used in Scala programs, and OWLAPI 2.2.0 was used for the creation of the ontology. The conversion is performed in two steps: The first step is parsing the MOF representation of the CIM schema. The parser was implemented as a combinatory parser using parser combinators that are part of Scala's standard library; no external parser generator was necessary. The second step consists of traversing the resulting abstract syntax tree and mapping elements to OWL axioms, as described in sections 5 and 6.

CIM schema version 2.23.0 consists of 1379 MOF files, containing roughly the same amount of OWL classes, and is converted to a total amount of 69295 OWL axioms, which are serialized into a 12 MB OWL file. The conversion takes about 35 seconds on an Intel Core 2 Duo with 2 GHz and 2 GB RAM; this can be interesting when a newer version of the CIM schema is to be translated. Loading the ontology using OWLAPI takes approximately 10 seconds and uses 130 MB on the same computer.

Only constructs that are valid in OWL DL are used; the resulting ontology has the expressiveness of OWL DL, which allows for decidable reasoning. As `owl:InverseFunctionalProperty` for the conversion of the `Key` qualifier is only used with object properties, the ontology does not require OWL Full expressiveness (the inverse functional characteristic for datatype properties can only be specified in OWL Full). Pellet 2.0.1 affirms the consistency of the ontology and the reasoning complexity of $\mathcal{ALUHOIN}^{(\mathcal{D})}$ ($\mathcal{ALUE}$ is equivalent to $\mathcal{S}$, and $\mathcal{SHOIN}^{(\mathcal{D})}$ is the reasoning complexity of OWL DL).

8 Conclusion and Future Work

This paper presented a short introduction to the DMTF Common Information Model (CIM), and the motivation to convert CIM into the OWL format. It then explained in detail how CIM constructs such as classes, properties, methods and various qualifiers can be converted into OWL. The OWL ontology consists of two namespaces: `meta`, which contains manually modelled entities that represent CIM constructs for which no direct OWL translation exists, and `cim`, which

contains the actual OWL elements that model the CIM schema. The resulting ontology has the expressiveness of OWL DL. Future work includes the examination of reasoning performance in a real-world scenario using instance data and the possibility of a round-trip conversion (i.e. translating OWL back to CIM), as well as the application of the ontology in an IT management context using management rules formulated in SWRL.

References

1. Storage Networking Industry Association, http://www.snia.org/
2. De Vergara, J.E.L., Villagrá, V.A., Berrocal, J.: Semantic Management: advantages of using an ontology- based management information meta-model. In: Proceedings of the HP OpenView University Association Ninth Plenary Workshop (2002)
3. Shvaiko, P., Euzenat, J.: Ten Challenges for Ontology Matching. In: Proceedings of the 7th International Conference on Ontologies, DataBases, and Applications of Semantics, ODBASE (2008)
4. Hepp, M., Roman, D.: An Ontology Framework for Semantic Business Process Management. In: Proceedings of Wirtschaftsinformatik 2007, Karlsruhe (2007)
5. De Vergara, J.E.L., Villagrá, V.A., Asensio, J.I., Berrocal, J.: Ontologies: Giving Semantics to Network Management Models. IEEE Network 17 (May/June 2003)
6. Quirolgico, S., Assis, P., Westerinen, A., Baskey, M., Stokes, E.: Toward a Formal Common Information Model Ontology (2004)
7. De Vergara, J.E.L., Guerrero, A., Villagrá, V.A., Berrocal, J.: Ontology-Based Network Management: Study Cases and Lessons Learned. Journal of Network and Systems Management (2009)
8. Xu, H., Xiao, D.: A Common Ontology-based Intelligent Configuration Management Model for IP Network Devices. In: Proceedings of the First International Conference on Innovative Computing, Information and Control (ICICIC). IEEE Computer Society, Los Alamitos (2006)
9. Heimbigner, D.: DMTF - CIM to OWL: A Case Study in Ontology Conversion. In: Ontology in Action Workshop in conjunction with the 2004 Conference on Software Engineering and Knowledge Engineering, SEKE 2004 (2004)
10. Majewska, M., Kryza, B., Kitowski, J.: Translation of Common Information Model to Web Ontology Language. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007. LNCS, vol. 4487, pp. 414–417. Springer, Heidelberg (2007)
11. Object Management Group: Unified Modeling Language (UML), http://uml.org/
12. Distributed Management Task Force: Common Information Model (CIM), http://www.dmtf.org/standards/cim/
13. World Wide Web Consortium: OWL Web Ontology Language Reference, Enumerated Datatypes, http://www.w3.org/TR/owl-ref/#EnumeratedDatatype
14. Astrova, I.: Rules for Mapping SQL Relational Databases to OWL Ontologies. In: Metadata and Semantics, pp. 415–424. Springer, US (2009)
15. Milanović, M., Gašević, D., Giurca, A., Wagner, G.: On Interchanging Between OWL/SWRL and UML/OCL (2006)
16. The Scala Programming Language, http://www.scala-lang.org/