MODELING OF STANDARDS AND THE OPEN WORLD ASSUMPTION

Mihnea Muraru, Alexandru Agache, Matei Popovici, Ciprian Dobre
Computer Science Department
POLITEHNICA University of Bucharest, Romania
email: {mmihnea,alexandruag,pdmatei,cipsmm}@gmail.com

KEYWORDS
Standards, BACnet, Modeling, Automated verification, Ontology, OWL, Open World Assumption

ABSTRACT

To be able to connect with the emerging technologies, real world concepts must be translated into machine-processable formats. Standards represent such collections of widely-accepted concepts and conventions, applicable in certain areas of interest. We believe that adopting formal, machine-understandable representations of standards generates a series of advantages, that we describe in this paper. We use the BACnet standard as a concrete example, and we present a possible approach to modeling it in an ontology. We highlight various design decisions and pitfalls encountered within the design process.

INTRODUCTION

Standards represent means to describe widely-accepted conventions in certain areas of interest. In many situations, it is of utmost importance that specifications are free of any inconsistencies. Usually, this is achieved through a continuous revision process, which doesn't rule out the presence of errors completely, but rather makes it unlikely. Furthermore, checking the compliance of different products and services is also an important issue. This has relevant applications in the field of Quality Management.

An alternative approach to checking inconsistencies is the use of automatic reasoning mechanisms. However, such techniques can only be applied if the specifications are translated into rigorous and unambiguous representations. This can be done by adopting different formalisms for describing factual data and their corresponding relations. A possible result is an ontology, that is a collection of such formal specifications. Many ontology modeling methods, such as the Ontology Web Language (OWL) [3], define precise semantics, thus allowing automated reasoning.

Besides checking for inconsistencies, a resoner may also be used to infer new knowledge, starting from what is explicitly represented in the ontology. Regardless of its final purpose, the reasoning process will often face difficulties generated by incomplete information. In such scenarios, a certain perspective must be adopted. These aspects, regarding Open and Closed World Assumptions, are discussed in more detail in a following section.

Formal representations offer a more concise perspective over the matter at hand. Written standards expose information in a linear manner, thus making connections between concepts more difficult to identify. A formal specification of such a standard induces some degree of structuring, by individualizing specific concepts and the relations in which they are involved. Visual tools would be able to exploit this kind of representations, in order to facilitate navigation throughout the problem universe.

Using ontologies to aggregate information allows publication, exploration and interconnection with other resources on the web, according to the *Linked Data* [1] principle. Such an established vocabulary could constitute the common ground for the automation of business processes that operate with standard-related information.

The paper is structured as follows. *Related work* presents similar initiatives related to modeling standards in ontologies. We then turn our attention towards the BACnet standard, described briefly in the section bearing the same name. In the *BACnet Standard Ontology* section we propose an ontological representation of the standard. The last section comprises our conclusions and future work.

RELATED WORK

Traditionally, ontologies provide a conceptual model of a domain of discourse, ocasionally simplifying or reformulating elements from the domain. Rigorous descriptions such as the one proposed in the article, are, to our knowledge, inexistent. Ontological representations of devices such as [5] and [6] provide a high-level view, focused on functionality from the user standpoint as well as device behavior. There is no explicit adherence to any particular standard. [7] gives a uniform representation for devices complaint with BACnet, KNX, LonWorks and ZigBee communication standards. While this approach is rather low-level, and concerned with technical, device-related details, it focuses on creating a common model, compatible with any of the enumerated standards. The emphasis is less on expressing standard-

related constraints and more on flexibility and integration. While this approach is succesful in certain device-related applications, it is insufficient in describing the complex structure of standards such as BACnet.
An ontology-based approach for detecting specification inconsistencies is described in [8], and applied on software configuration management. In a similar manner to ours, reasoning is applied on OWL ontologies in order to identify invalid test configurations. A notable difference relies in the representation domain. Software configurations discussed in [8] have a rather straightforward behavior and structure, and limited number of restrictions. Such structures can be adjusted in certain regards, to fit to representational needs specific to OWL. In contrast, human-developed standards raise complex representation problems, and are more rigid. The issues raised by OWA are such an example. Also, the development of representations for standards is useful for detecting standard inconsistencies. This issue is not addressed in approaches such as [8].

THE BACNET STANDARD

BACnet (Building Automation and Control Network) [2] is an open standard for communications between building automation devices, that was developed by ASHRAE (American Society of Heating, Refrigerating and Air-Conditioning Engineering). Its aim is to allow interoperation of various devices developed by different manufacturers. In order to attain such a target, BACnet proposes a complex model for describing building systems, irrespective to implementation details specific to manufacturers, or to the physical communication medium. At the heart of this model lie three concepts: *object*, *property* and *service*.
Depending on its purpose, a BACnet device may be broken down into a series of BACnet *objects*, that represent the fundamental control and communication units. It is important to note that the objects contained in a device merely describe capabilities of interfacing with that device. They do not offer means of describing its internal structure.
Each BACnet object possesses a standard set of *properties*, that reflect its current state. They represent the only means through which other devices can interact with a specific object. Every property is assigned a *datatype* and a *conformance code*. The *datatype* determines the internal representation and interpretation of the value of a property. The *conformance code* dictates how the value of the property may be accessed and whether its presence is mandatory (read-only, writable, optional). The standard imposes certain constraints on groups of properties, such as the necessity that a *Schedule* object should possess at least one of the *Weekly_Schedule* or *Exception_Schedule* properties, which are optional.
BACnet *services* are mechanisms that allow operations such as: reading and writing property values, generating commands on other devices and signaling events.
The BACnet standard also addresses many other aspects regarding device communication, which are not presented here, due to the lack of relevance to the modeling issues discussed in this paper.

THE BACNET STANDARD ONTOLOGY

This section covers the main design considerations involved in the development of a BACnet ontology, written in OWL. We focus on a specific part of the BACnet standard (2004 version), namely the fundamental concepts and their relations, leaving out very technical details such as the description of low-level communication protocols. By fundamental we refer to the building blocks of the BACnet modeling paradigm: objects, properties and services.
We start by introducing the overall structure of the ontology and then we delve into more specific issues.

Overall structure

Classes
The fundamental concept in the BACnet ontology, designed to represent any BACnet-related notion, is *BACnetEntity*. Its direct subclasses are:

- *BACnetDevice*: represents the 6 BACnet device types, such as *BACnet Operator Workstation* or *BACnet Building Controller*;
- *BACnetObject*: represents the 25 object types in the BACnet standard, e.g., *BinaryOutputObject* or *ScheduleObject*;
- *BACnetProperty*: represents the various properties the objects may possess, such as *ObjectIdentifierProperty* and *OutOfServiceProperty*;
- *BACnetService*: represents the 5 categories of services described in the BACnet standard: *AlarmEventService*, *FileAccessService*, *ObjectAccessService*, *RemoteDeviceManagementService*, *VirtualTerminalService*;
- *BACnetDatatype*: represents the possible datatypes for object properties. *Simple datatypes*, like *BACnetObjectIdentifier*, are individuals of the subclass *BACnetSimpleDatatype*. *Complex datatypes*, such as arrays and lists, are represented as individuals of another subclass, *BACnetComplexDatatype*, that is further divided into *BACnetArrayDatatype* and *BACnetListDatatype*;
- *BACnetConformanceCode*: represents the possible conformance codes for the object properties. It contains only two individuals: $R(eadable)$ and $W(ritable)$.

Relations
We use relations to describe the connections between various concepts, according to the standard. From the OWL perspective, relations are divided into *object properties*[1] and *data properties*.
The object properties are:

- *hasProperty*[2]: associates an object with one of its properties. Its signature is: $BACnetObject \rightarrow BACnetProperty$;
- *hasDatatype*: associates a property with its datatype. Its signature is: $BACnetProperty \rightarrow BACnetDatatype$;
- *hasConformanceCode*: associates a property with its conformance code. Its signature is: $BACnetProperty \rightarrow BACnetConformanceCode$;
- *hasElementDatatype*: associates a complex type with the datatype of its elements. Its signature is: $BACnetComplexDatatype \rightarrow BACnetSimpleDatatype$[3].

The data properties are:

- *hasTheValue*: associates a property to its value, represented as a *string*. Its signature is: $BACnetProperty \rightarrow string$. It wasn't named *hasValue* so it wouldn't conflict with the OWL *hasValue* property restriction.
- *hasElementCount*: associates an array with its size, represented as an *integer*. Its signature is: $BACnetArrayDatatype \rightarrow integer$.

Naming conventions

Classes
The classes in the ontology have *suffixes* that correspond to their BACnet significance, e.g., *BinaryOutputObject*, *ObjectIdentifierProperty* and *BACnetArrayDatatype*.

Individuals
Individuals representing complex datatypes are named using the following conventions:

- *array* datatypes, that are instances of the *BACnetArrayDatatype* concept, appear as *NxElementType* (read *N times ElementType*). If the array size is fixed, *N* is replaced by the actual number of elements. If the array size is variable, the letter *N* is used directly. For example, the individual *3xBACnetTimeStampDatatype* stands for the type *BACnetARRAY*[3] *of BACnetTimeStamp*;
- *list* datatypes, that are instances of the *BACnetListDatatype* concept, appear as *ListofElementType*. For example, the individual *ListOfBACnetCalendarEntryDatatype* stands for the type *List of BACnetCalendarEntry*.

Property attributes

Any property of a given object is characterized by a datatype and a conformance code. They are not intrinsically associated to a property and must be defined for *every* object.
For example, the BACnet specification of the *Calendar* object includes the *ObjectIdentifierProperty*, having the *BACnetObjectIdentifier* datatype and the *R* conformance code. This is represented as a *superclass* of the *CalendarObject* concept, as shown in Fig. 1. Note the use of the Manchester syntax [4].

Open vs. Closed World Assumptions

According to the Open World Assumption[4] (OWA), assertions are generally used to infer new knowledge rather than constrain existing knowledge. For example, the restriction (*hasProperty exactly* 1 *ActiveTextProperty*) produces the following behavior:

- if a particular object isn't explicitly asserted to possess an *Active_Text* property, the reasoner will conclude that in fact it possesses one that we don't know about, and no inconsistency is signalled;
- if a particular object is asserted to possess 2 *Active_Text* properties, the reasoner will infer that the two properties must represent the same entity. However, all the property instances can be declared different, in which case the reasoner will signal an inconsistency.

As one can notice, both scenarios are counterintuitive. Although in the second case, the constraint can be effectively enforced, in the first case it is *impossible* to impose the explicit presence of at least one property.
According to our current knowledge, reasoner extensions exist, that use the Closed World Assumption[5] (CWA).

```
hasProperty exactly 1
  (ObjectIdentifierProperty
    and (hasDatatype value
          BACnetObjectIdentifierDatatype)
    and (hasConformanceCode value R))
```

Figure 1: Superclass of *CalendarObject*, denoting the attributes of its *Object_Identifier* BACnet property

[1] Although the expression *object properties* is perfectly valid in the BACnet context, we use it here to refer to OWL properties, in the ontological sense.
[2] Disambiguation is needed: *hasProperty* is a property in OWL sense, that links a BACnet object to a BACnet property.
[3] We have assumed that values having complex types can only be made out of values having simple types.
[4] If something isn't explicitly stated to be true, it doesn't mean that it is false.
[5] If something isn't explicitly stated to be true, then it is false.

We might find good use of such a reasoner, in order to enforce constraints in a more intuitive way. This is similar to the role of database schemas. For instance, defining a type for a table column is a means for restricting the possible values of that field, and not a mechanism for inferring types.

Disjointness

Because of the OWA, used by OWL, we must explicitly state as many facts as possible. Thus, all the concepts on the same hierarchical level are made *disjoint*. For example, the top-level concepts describing devices, objects, properties, are disjoint. Furthermore, within each concept, its direct subclasses are also disjoint, and so on.

Similarly, all the individuals that are direct or indirect instances of top-level concepts must be declared *different*.

Closure axioms

As stated in the *Property attributes* section, the definition of a BACnet object concept comprises qualified[6] cardinality restrictions for each associated property. The *ObjectIdentifierProperty*, mentioned in Fig. 1, was such an example. However, due to the OWA, stating that an object must possess certain properties does not enforce the absence of other properties, that are not explicitly mentioned. As an example, the *Active_Text* property could be added to a *Calendar* instance, although it is not part of the BACnet standard specification. In order to implement this constraint, a *closure axiom*, such as the one in Fig. 2, must be added as a superclass of the object in consideration.

This means that whatever properties our object possesses, they must belong to one of the concepts separated by the *or* operator.

[6] A *qualified* cardinality restriction constrains the range class of the property involved.

```
hasProperty only (
  (DateListProperty
    and (hasDatatype value
      ListOfBACnetCalendarEntryDatatype)
    and (hasConformanceCode value R))
  or
  (DescriptionProperty
    and (hasDatatype value
          CharacterStringDatatype)
    and (hasConformanceCode exactly 1
          BACnetConformanceCode))
  or ...)
```

Figure 2: Closure axiom for the *Calendar* object

Covering axioms

Similarly to the mechanisms described in the previous section, we also define *covering axioms* for the concepts in the ontology. Their purpose is to state that a class possesses only the explicitly defined subclasses and that no other unknown subclasses exist. The subclasses are said to *cover* the superclass.

We apply this principle to every concept in the ontology, by stating that each class is *equivalent* to the union of its subclasses. An example regarding the *BACnetEntity* concept is given in Fig. 3.

A particular situation regards those classes that contain only individuals. In this case, instead of being covered by its subclasses, each class is defined to be *equivalent* to an enumerated class containing the individuals. For example, the *BACnetConformanceCode* concept is equivalent to the class $\{R, W\}$, where R and W are individuals presented in the *Conformance codes* section.

Functional properties

Certain relations are inherently functional. This means that an individual is associated along these relations with *at most one* other individual. For example, a property associated with an object cannot have more than one datatype or conformance code.

If this characteristic was absent then, due to the OWA, one would be able to assign multiple datatypes and conformance codes to the same property within the same object.

The *Any* datatype

Several objects do not assign a specific datatype to some of their properties. This is encoded within the BACnet standard using the *Any* datatype. An example is the *Present_Value* property of the *Schedule* object, as shown in Fig. 4. The freedom in type choice is specified using the constraint (*hasDatatype exactly* 1 *BACnetDatatype*). This is different from properties having a fixed datatype, that are defined using the OWL *hasValue* property restriction, as in (*hasDatatype value CharacterStringDatatype*).

```
BACnetConformanceCode
or BACnetDatatype
or BACnetDevice
or BACnetObject
or BACnetProperty
or BACnetService
```

Figure 3: Covering axiom for the *BACnetEntity* concept

```
hasProperty exactly 1
  (PresentValueProperty
    and (hasDatatype exactly 1
          BACnetDatatype)
    and (hasConformanceCode value R))
```

Figure 4: Specifying the *Any* datatype of the *Present_Value* BACnet property

Conformance codes

In the BACnet standard, there are three conformance codes associated with properties: *R*(*eadable*), *W*(*ritable*) and *O*(*ptional*). The first two imply that the property is mandatory, while the last one allows its absence.
We have decided to explicitly model only the *R* and *W* conformance codes. Instead of explicitly stating that an object possesses a property having the *O* conformance code, we simply choose to allow at most one property of that kind. This will be reflected in the superclasses of an object. For example, the restriction in Fig. 1 will change from *exactly* to *max*, as depicted in Fig. 5. Also, notice that the conformance code isn't explicitly specified anymore.

Property constraints

The standard imposes certain constraints on groups of properties. Some examples are given below:

1. At least one of these properties is required.
2. If one of the optional properties *Inactive_Text* or *Active_Text* is present, then both of these properties shall be present.
3. If *Present_Value* is commandable, then it is required to be writable. This property is required to be writable when *Out_Of_Service* is TRUE.

Constraint #1 is applicable for the *Schedule* object and its optional *Weekly_Schedule* and *Exception_Schedule* properties. It is encoded as a superclass of the *ScheduleObject* concept, as shown in Fig. 6.
Constraint #2 is modeled as a superclass of the *BACnetObject* concept, as shown in Fig. 7. Unlike constraint

```
hasProperty max 1
  (DescriptionProperty
    and (hasDatatype value
          CharacterStringDatatype)
    and (hasConformanceCode exactly 1
          BACnetConformanceCode))
```

Figure 5: Specifying the *optional* conformance code of the *Description* BACnet property

```
(hasProperty exactly 1
    ExceptionScheduleProperty) or
(hasProperty exactly 1
    WeeklyScheduleProperty)
```

Figure 6: Specifying the mandatory presence of at least one of the *Weekly_Schedule* and *Exception_Schedule* properties for the *Schedule* object

```
((hasProperty exactly 0
    ActiveTextProperty) and
 (hasProperty exactly 0
    InactiveTextProperty))
or
((hasProperty exactly 1
    ActiveTextProperty) and
 (hasProperty exactly 1
    InactiveTextProperty))
```

Figure 7: Specifying the requirement of having the *Active_Text* and *Inactive_Text* properties both present or absent

#1, that was specified as a superclass of a particular object concept (*ScheduleObject*), constraint #2 was modeled as a superclass of the general *BACnetObject*, as it applies to several BACnet objects. The objects that do not possess these two properties are also members of the same superclass, because their closure axioms forbid the presence of both properties.
Constraint #3 proves different, as it involves dynamic conditions, that may change during the functioning of a device, e.g., the value of the *Out_Of_Service* property. This cannot be explicitly modeled in the ontology, using an approach similar to the one in the first two examples. Thus, we introduce a special conformance code, that will implicitly indicate this constraint, and assign it to the *Present_Value* property.

CONCLUSIONS

Although the OWA has its own advantages in knowledge representation, it proves inappropriate when modeling a standard, as they are closed in nature. As stated in the dedicated section, reasoners based on CWA may be used, but the available options are quite limited.
Using the proposed ontology, device specifications may be written and automatically validated against the standard. However, regarding the correctness of the ontology itself, specific tests have been used. We plan on developing an unit testing methodology that will allow automated verification of the ontology. The testing should verify that correct specifications are accepted (*positive tests*) and that invalid ones are rejected (*negative tests*).
The ontology was developed as part of research work for FCINT project (*Framework for service composition*

based on ontologies for the aggregation of knowledge and information for intelligent buildings).

ACKNOWLEDGMENT

The research presented in this paper is supported by national project "TRANSYS Models and Techniques for Traffic Optimizing in Urban Environments", Project "CNCSIS-PD" ID: 238.

REFERENCES

[1] Linked Data. http://linkeddata.org/.

[2] BACnet Standard, 2004. http://www.bacnet.org/.

[3] OWL 2 Web Ontology Language, October 2009. http://www.w3.org/TR/owl2-overview/.

[4] OWL 2 Web Ontology Language Manchester Syntax, October 2009. http://www.w3.org/TR/owl2-manchester-syntax/.

[5] Dario Bonino and Fulvio Corno. DogOnt — Ontology Modeling for Intelligent Domotic Environments. pages 790–803. 2008.

[6] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. *International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 258–267, August 2004.

[7] Christian Reinisch, Wolfgang Granzer, Fritz Praus, and Wolfgang Kastner. Integration of heterogeneous building automation systems using ontologies.

[8] Hamid Haidarian Shahri, James A. Hendler, and Adam A. Porter. Software configuration management using ontologies.