SEMANTIC QUERY ON MATERIALS DATA BASED ON MAPPING MATML TO AN OWL ONTOLOGY

Xiaoming Zhang[1*, 2], *Changjun Hu*[1], *and Huayu Li*[1]

[1]*School of Information Engineering School, University of Science and Technology Beijing, Beijing, 100083, China*
**Email:* zxm1975@gmail.com
[2]*School of Information Science and Engineering, Hebei University of Science and Technology, Shijiazhuang, 050054, China*

ABSTRACT

MatML plays an important role in materials data applications while structure-aware query techniques (e.g., XPath and XQuery) are used to search the content of MatML. However, both XPath and XQuery cannot efficiently retrieve sets of MatML on a conceptual level. In this paper, we propose an approach to transform MatML-based materials data into an OWL ontology. As such, materials data can then be explored in a more semantic way. The proposed method formally defines a set of rules to extract the corresponding OWL ontology (named MatOWL) from a given MatML schema. The instance transformation from MatML to MatOWL is implemented with the help of an intermediate object model. The algorithm for instance transformation is also given. Further, MatOWL can be mapped to other ontologies with logic rules to provide more semantic context for domain experts, and more materials knowledge can be obtained by reasoning on the OWL ontology. An experimental prototype demonstrates the effectiveness of our proposed approach.

Keywords: MatML, OWL Ontology, Semantic query, Materials data, Domain data engineering

1 INTRODUCTION

In recent years, industrial communities and research institutions have accumulated immeasurable amounts of materials data (Westbrook, 2003), which are sure to facilitate materials science research. However, the complexity of materials data makes it hard for users to integrate and share these scientific data (Iwata, Shichijo, & Ashino, 2001). Materials data uses, such as access, acquisition, and interoperability, have gradually become the most challenging problems facing materials informatics (Hunt, 2006). As stated at the materials informatics workshop held at University of Queensland in 2006 (U Queensland, 2006), informatics is still in its infancy in the materials domain, and researchers of materials informatics should keep up with other scientific communities such as biology, astronomy, and social sciences. In order to exchange, integrate, and share materials data, a unified materials data model is needed. MatML (MatML Schema, 2004) and NMC-MatDB (NMC-MatDB Schema, 2007) are materials data models built for this purpose. A CODATA Task Group for Exchangeable Materials Data Representation (Task Group, 2006) was established at 2006 to focus on topics related to interoperability of heterogeneous data resources. Swindells (2002) described the representation of engineering properties, which includes material properties, by the use of ISO 10303 standards (ISO, 1994). ISO 10303-45 Edition 2 (ISO, 2008) defines an information model to enable the representation of materials and other engineering properties of a product. This standard is extended by the development of ISO 10303-235, which focuses on the representation of the measurement process of a property and will shortly be published. The terminologies of properties and measurement processes are specified in ISO 13584 (ISO, 2001) as a series of dictionaries.

MatML is an extensible markup language developed especially to facilitate the exchange of materials information. It can uniformly represent materials property data to resolve syntactic and structural heterogeneity. As MatML is simple, flexible, and understandable, it offers many benefits (Sturrock, Begley, & Kaufman, 2001) to materials scientists and engineers. Research and applications about MatML have emerged in recent years. MatML is used in NSDL MatDL (Bartolo, Lowe, Sadoway, Powell, & Glotzer, 2005), and a web GUI has been developed to compare the properties of more than 80 types of materials. Begley and Howard-Reed (2005) have discussed the application of MatML to containment emission data, and a mapping approach from RDBMS to MatML is presented to transform these emission data into an exchangeable format. Bartolo & Lowe (2003) mapped MatML tags to Dublin Core elements to reuse the detailed information provided by MatML, so as to equip Dublin Core metadata with the capability of describing materials domain features. Varde, Begley, and Fahrenholz-Mann (2006) have presented some success stories for MatML in data mining applications, such as failure analysis and decision support systems.

As MatML uses a tree-based structure to represent its content, queries (XPath or XQuery) on MatML are tightly coupled with the tree structure of the XML documents. However, materials scientists are more likely to write a semantic query by using domain terms that they are familiar with, instead of exploring the complex document structure. Therefore, MatML can not meet this requirement elegantly. MatML currently has not defined the names of materials properties and experiment data items (Ashino & Fujita, 2006). As such, it lacks the capability to describe the high-level abstraction of concept semantics, and machine-processable ontologies are required to provide the semantic mappings between related terms (Cheung, Drennan, & Hunter, 2008). Furthermore, from the perspective of data integration, the semantics of XML data should be specified explicitly to resolve semantic heterogeneity. Ashino and Oka (2007) have shown that MatML is not adequate for data exchanges between heterogeneous materials databases and have proposed a framework by using an ontology to define the structure of domain concepts. Hunter, Little, and Schroeter (2008) have developed an ontology for integration of disparate materials databases. These excellent research achievements pay more attention to the semantic integration of databases. Differing from the existing work, we believe that both integration and utilization of MatML data are becoming more and more important. However, this topic has not yet been well studied.

In this paper, we propose an approach to use the achievements of the semantic web (Berners-Lee, Hendler, & Lassila, 2001) by transforming MatML to an OWL (Bechhofer, Harmelen, Hendler, Horrocks, McGuinness, Patel-Schneider, et al., 2004) ontology, so that materials scientists can perform semantic queries on materials data derived from the MatML documents. The approach formally defines a set of rules to extract the ontology (called MatOWL) from a given MatML schema and utilizes an intermediate object model to help instance transformation from MatML to MatOWL. The proposed MatOWL can be easily extended by mapping it to other ontologies with logic rules. As such, more semantic context for domain experts can be provided for semantic-related retrieval tasks.

The remainder of this paper is organized as follows. The related work is discussed in Section 2. Section 3 gives the problem description. Section 4 introduces the rules used for extracting MatOWL. Section 5 describes the process and algorithm for the instance transformation. Section 6 discusses how to enhance MatOWL and compose SPARQL query on the translated materials data. Section 7 shows experimental results and demonstrates the prototype. Section 8 concludes the paper.

2 RELATED WORK

In the recent years, much research has concentrated on how to build mapping from XML to an ontology efficiently. From the view point of real-life applications, some researchers focus on metadata generation for web pages, while others focus on data integration. From the view point of implementation techniques, some approaches aim at extracting an ontology from XML documents or XML schema (Ferdinand, Zirpins, & Trastour, 2004; Bohring & Auer, 2005), while others are dedicated to mapping the XML documents to an existing ontology (Reif, Jazayeri, & Gall, 2004; Rodrigues, Rosa, & Cardoso, 2005; Kobeissy, Genet, & Zeghlache, 2007).

WEESA (Reif, et al., 2004), JXML2OWL (Rodrigues, et al., 2005) and XMLTOWL (Kobeissy, et al., 2007) generate mappings between an XML schema and a specific OWL ontology. WEESA and XMLTOWL use XML to express mappings, whereas JXML2OWL adopts XSLT. Once the mappings are defined, these approaches can transform XML documents to the corresponding OWL instances automatically. The advantage of this kind of solution is that an existing ontology can be utilized to provide a semantic interpretation for the XML data.

The XML2OWL (Bohring & Auer, 2005) framework also supports instance transformation, and it can extract an OWL ontology from a XML schema or even from XML documents. The extraction and transformation rules are implemented by XSLT. On the other hand, Ferdinand, et al. (2004) proposed a method to convert XML to RDF as well as to transform a XML schema to OWL, though the two transforming processes are independent of each other.

From the perspective of data integration, Lehti & Fankhauser (2004) use an OWL ontology as the global view in the data integration system and then construct mappings between the XML schema of a data source and the OWL ontology. It does not dump all instances from the data source, whereas the original query on the OWL ontology has to be translated into the corresponding XQuery. Different from the simple value correspondence between XML and ontology, An, Borgida, & Mylopoulos (2005) propose an approach to help end-users construct complex mapping formulas between XML schemas and OWL ontologies. The mappings are expressed as a subset of First Order Logic.

Compared with the approaches mentioned above, our method has the following features. First, we do not map a XML document to a particular ontology. Alternatively, we focus on how to extract the OWL ontology from the XML document and then how to associate it with other OWL ontologies. Second, we are concerned with a particular type of XML document (i.e., MatML) in the materials domain. Hence, we must use some specific rules to extract information from MatML. Third, we use logic rules to express the relationships between the extracted ontology and other domain ontologies, and therefore the extracted ontology can be further used in a user-preferred manner.

3 PROBLEM DESCRIPTION

As more and more materials resources emerge in the MatML format, it is significant and necessary to provide a convenient interface for users to access these scientific data. The motivation of our work is to bridge the semantic gap between materials experts and structure-aware queries on MatML, so that users can exploit the content of MatML in a more semantic way. Structure-aware query here means that the query should more or less be aware of the structural information of the source data. Thanks to the rapid development of semantic web technology, we therefore can use technologies such as ontology and logic rules to help to solve this problem. Before we give the overview of our approach, some related definitions are given below.

The MatML version 3.1 schema contains more than 50 complex and simple types. Elements and attributes of MatML are declared as these types. Accordingly, the MatML schema can be formally defined as follows.

Definition 1. The MatML schema is a 2-tuple $MatML\_Schema = (CT, ST)$, where $CT = \{ct \mid ct \text{ is xsd:complexType}\}$ and $ST = \{st \mid st \text{ is xsd:simpleType}\}$. Further, $ST = ST_{NR} \bigcup ST_{R}$ where simple types with enumeration restriction belong to ST_R and the other simple types belong to ST_{NR}; $CT = (E, ATT)$ where $E = \{e \mid e \text{ is a xsd:Element}\}$ and $ATT = \{att \mid att \text{ is a xsd:Attribute}\}$. For each $ct \in CT$, *ct.Elements* and *ct.Attributes* are used to get *E* and *ATT* of *CT* respectively. For each $st \in ST$, *getBaseType(st)* is defined to get the base type of *st.* For each $x \in E \bigcup ATT$, *x.name* and *x.type* are used to get the name and type of *x* respectively. If *x.type* is *xsd:IDREF*, *getRefType(x)* is used to get the type referenced by *x.* The set of XML Schema types (e.g., xsd:string) are represented by *XSDType*.

Applications in the materials science domain need more semantic support, such as that offered by an ontology. An ontology is a formal, explicit specification of a shared conceptualization (Gruber, 1993; Studer, Benjamins, & Fensel, 1998). An ontology can be used to capture some shared domain knowledge (e.g., the key concepts of a domain and important relationships between them), and it can also serve as the basis for logic reasoning on the information content in a specific domain. Based on the language description of an OWL (Bechhofer, et al., 2004), the ontology extracted from MatML can be defined as follows.

Definition 2. *MatOWL* is an OWL ontology extracted from the MatML schema, which is a 5-tuple $MatOWL=(C,OP,DP,I,A^{o})$, where *C* is a finite set of concepts; *OP* is a finite set of object properties; *DP* is a finite set of data type properties; *I* is a set of instances; and A^{o} is a set of axioms. Given $\sum=C,OP,DP,I,A^{o}$ is a logic axiom set over $\sum$. For each $x \in \sum$, we use *x.id* to get a resource identifier of *x.* For $c, x \in C$, *c.hasSuperclass(x)* is defined to express that class *c* is a subclass of class *x.* For $p \in OP \bigcup DP$ and $x, y \in C \bigcup XSDType$, *p.hasDomain(x)* is defined to represent that the domain of property *p* contains class *x,* while *p.hasRange(y)* is defined to represent that the range of property *p* contains class *y.*

Definition 3. *f(x)* is a name mapping function from the MatML schema into the MatOWL, where *x* can be a name of a type, element, or attribute defined in the MatML schema, and the value of *f(x)* is the name of a class or property defined in MatOWL.

Based on these definitions, we propose an approach for mapping MatML to MatOWL so that users can perform semantic queries on materials data. The overview of our approach is illustrated in Figure 1. After four steps, users can semantically access MatML-based materials data by SPARQL queries.

Step 1: Extraction of MatOWL. This step analyzes the structure of the MatML schema and extracts the concepts and properties of a MatOWL ontology from the MatML schema by a set of heuristic rules. Through this step, the TBox of MatOWL is constructed.

Step 2: Instance transformation. Instance data are transformed from MatML documents to MatOWL. Consequently, MatOWL is populated with materials property data, and the ABox of MatOWL is constructed.

Step 3: Enhancement of MatOWL. This step adds more semantics to MatOWL. MatOWL is associated with concepts from other existing domain ontology by logic rules.

Step 4: Query building. A SPARQL query is built based on the domain terms that come from MatOWL or the associated domain ontology.

In the following sections, each of the four steps is described in more detail.

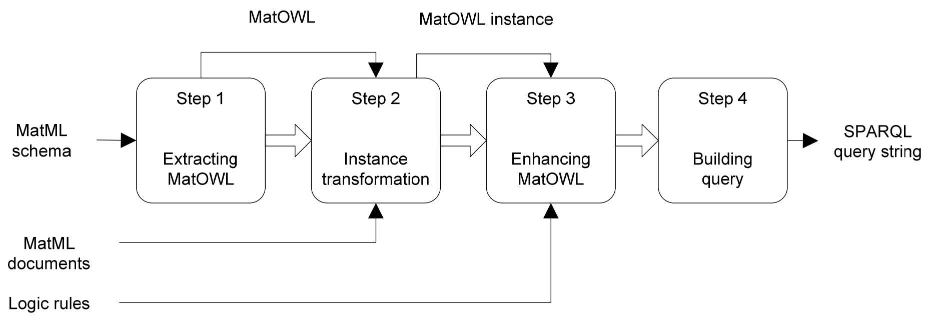


Figure 1. Overview of the approach

4 EXTRACTING MATOWL

Because MatML is a data model for materials property data, the MatML schema can provide the basic vocabulary and structure for the description of this domain. Therefore, extracting an ontology from the MatML schema is not only a good idea but also a feasible solution. Hunter, et al. (2008) have designed an ontology based on MatML for database integration. Inspired by this idea, we deeply analyze the complex types and simple types defined in the MatML schema and propose a set of heuristic rules for extracting both concepts and properties from the MatML schema to build MatOWL. The extracted ontology involves the concepts and properties about materials property data. The main rules for extracting MatOWL from the MatML schema are given as follows.

Rule 1. Rule for class generation.

$$\forall t \in (CT \cup ST_R) \rightarrow (\exists c \in C) \wedge (c.id = f(t.name)) \quad (1)$$

This rule implies that each *t*, which is a complex type or simple type with enumeration restriction, is transformed to a corresponding class in MatOWL. For example, complex type *PropertyData* and simple type *ChemicalElementSymbol* in MatML are transformed to owl:Class.

Rule 2. Rule for object property generation

$$\forall ct \in CT \wedge \forall x \in (ct.Elements \cup ct.Attributes) \wedge x.type \in (CT \cup ST_R)$$
$$\rightarrow (\exists op \in OP) \wedge (op.id = f(x.name)) \wedge (op.hasDomain(f(ct.name))) \wedge (op.hasRange(f(x.type.name))) \quad (2)$$

$$\forall ct \in CT \wedge \forall att \in ct.Attributes \wedge (att.type = xsd : IDREF)$$
$$\to (\exists op \in OP) \wedge (op.id = f(att.name)) \wedge (op.hasDomain(f(ct.name))) \wedge (op.hasRange(getRefType(att.type))) \quad (3)$$

Formula (2) implies that for each complex type *ct*, if the type of its element (or attribute) *x* is a complex type or simple type with enumeration restriction, *x* is extracted as an object property *op*. Meanwhile, class *f(ct.name)* is added to the domain of *op*, and class *f(x.type.name)* is added to the range of *op*. For example, the element *Name* in the complex type *PropertyDetails* is extracted as an object property *hasName*. On the other hand, formula (3) implies that for each complex type *ct*, if the type of its attribute *x* is *xsd:IDREF*, *x* is extracted as an object property. As the attribute *Property* in the complex type *Propertydata* has type of *xsd:IDREF*, it is transformed to an object property.

Rule 3. Rule for data type property generation.

$$\forall ct \in CT \wedge \forall x \in (ct.Elements \bigcup ct.Attributes) \wedge x.type \in ST_{NR}$$
$$\to (\exists dp \in DP) \wedge (dp.id = f(x.name)) \wedge (dp.hasDomain(f(ct.name))) \wedge (dp.hasRange(getBaseType(x.type))) \quad (4)$$
$$\forall ct \in CT \wedge \forall x \in (ct.Elements \bigcup ct.Attributes) \wedge x.type \in XSDType$$
$$\to (\exists dp \in DP) \wedge (dp.id = f(x.name)) \wedge (dp.hasDomain(f(ct.name))) \wedge (dp.hasRange(x.type)) \quad (5)$$

Formula (4) implies that for each complex type *ct*, if the type of its element (or attribute) *x* is a simple type without enumeration restriction, *x* is extracted as a data type property *dp*. At the same time, class *f(ct.name)* is added to the domain of *dp*, and class *f(getBaseType(x.type))* is added to the range of *dp*. For example, the element *Notes* in the complex type *PropertyData* is extracted as a data type property *notes* whose range is *xsd:string*. As shown in Formula (5), if *x.type* is a kind of *xsd* type, the range of *dp* is just *x.type*.

Rule 4. Rule for cardinality generation.

$$\forall ct \in CT \wedge \forall x \in (ct.Elements \bigcup ct.Attributes) \wedge \neg Exist(x.minOccurs) \wedge \neg Exist(x.maxOccurs)$$
$$\to (\exists c \in C) \wedge (c.id = f(ct.name)) \wedge (\exists p \in OP \bigcup DP) \wedge (p.id = f(x.name)) \wedge p^c.Cardinality = 1) \quad (6)$$
$$\forall ct \in CT \wedge \forall x \in (ct.Elements \bigcup ct.Attributes) \wedge x.minOccurs = 0 \wedge \neg Exist(x.maxOccurs)$$
$$\to (\exists c \in C) \wedge (c.id = f(ct.name)) \wedge (\exists p \in OP \bigcup DP) \wedge (p.id = f(x.name)) \wedge (p^c.MinCardinality = 1) \wedge (p^c.MaxCardinality = 1) \quad (7)$$
$$\forall ct \in CT \wedge \forall x \in (ct.Elements \bigcup ct.Attributes) \wedge \neg Exist(x.minOccurs) \wedge (x.maxOccurs = 'unbounded')$$
$$\to (\exists c \in C) \wedge (c.id = f(ct.name)) \wedge (\exists p \in OP \bigcup DP) \wedge (p.id = f(x.name)) \wedge p^c.MinCardinality = 1) \quad (8)$$
$$\forall ct \in CT \wedge \forall x \in (ct.Elements \bigcup ct.Attributes) \wedge (x.minOccurs = m) \wedge (x.maxOccurs = 'unbounded')$$
$$\to (\exists c \in C) \wedge (c.id = f(ct.name)) \wedge (\exists p \in OP \bigcup DP) \wedge (p.id = f(x.name)) \wedge p^c.MinCardinality = m) \quad (9)$$

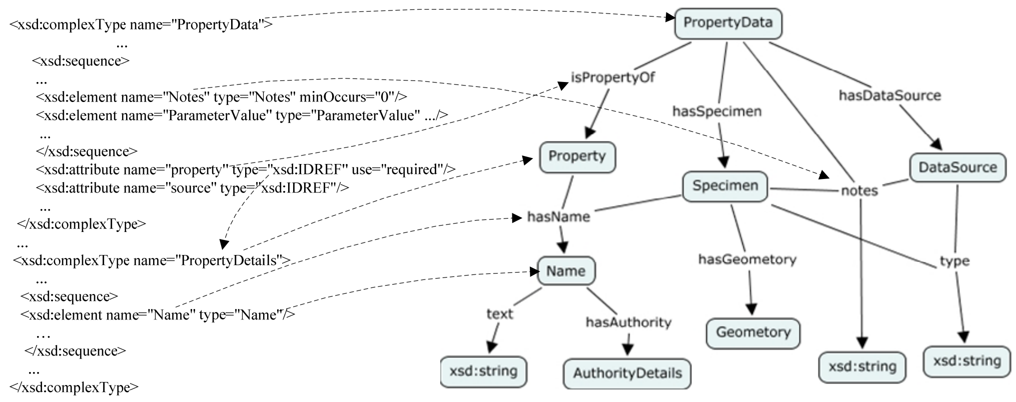
This set of rules is used to generate cardinality restrictions for the generated property. *Exist(x.att)* is used to test whether a *xsd:Element* (or *xsd:Attribute*) *x* has the attribute *att*. $p^c.Cardinality$. ($p^c.MaxCardinality$, $p^c.MinCardinality$) represents a *Cardinality* (*MaxCardinality*, *MinCardinality*) restriction on property *p* for class *c*. In the MatML schema, if there is no *miniOccurs* (*maxOccurs*) declared in an element or attribute, it implies *miniOccurs*=1 (*maxOccurs*=1). Therefore, if neither *miniOccurs* nor *maxOccurs* exists, it indicates "exactly one." The expression *maxOccurs='unbounded'* means there is no upper bound for occurring times, so the corresponding property has no *MaxCardinality* restriction.

Rule 5. Rule for hierarchy relationship generation.

$$\exists ct \in CT \wedge (ct.name = 'Metadata') \wedge \forall e \in ct.Elements$$
$$\to (\exists c \in C) \wedge (c.id = f(e.type.name)) \wedge (c.hasSuperclass(f('Metadata'))) \quad (10)$$

The complex type *Metadata* in the MatML schema has eight sub-Elements (i.e., *AuthorityDetails*, *DataSourceDetails*, *MeasurementTechniqueDetails*, *ParameterDetails*, *PropertyDetails*, *SourceDetails*, *SpecimenDetails*, *TestConditionDetails*), each of which has its own declared types. The corresponding classes to these types are defined as subclasses of *f('Metadata')*.

An example is given in Figure 2, which illustrates how the above extraction rules are used to accomplish the MatML schema transformation. The left part of Figure 2 is a snippet of the MatML schema, and the corresponding MatOWL is on the right. The complex type *PropertyData* is extracted as the class *PropertyData*. The attribute *property*, whose type is *xsd:IDREF*, is transformed to the object property *isPropertyOf*, and the referenced type *PropertyDetails* is translated into the class *Property*, which becomes the range of the property *isPropertyOf*. The element *Name* is converted into the object property *hasName* whose domain contains the class *Property*, and the range is the class *Name*. The element *Notes* is mapped to the data type property *notes* whose range is *xsd:string* and the domain contains the classes *PropertyData*, *Specimen*, *DataSource*, etc.



```
<xsd:complexType name="PropertyData">
            ...
   <xsd:sequence>
    ...
    <xsd:element name="Notes" type="Notes" minOccurs="0"/>
    <xsd:element name="ParameterValue" type="ParameterValue" .../>
     ...
    </xsd:sequence>
    <xsd:attribute name="property" type="xsd:IDREF" use="required"/>
    <xsd:attribute name="source" type="xsd:IDREF"/>
     ...
 </xsd:complexType>
 ...
<xsd:complexType name="PropertyDetails">
   ...
  <xsd:sequence>
  <xsd:element name="Name" type="Name"/>
    …
   </xsd:sequence>
    ...
</xsd:complexType>
```

Figure 2. Example for extracting MatOWL

5 INSTANCE TRANSFORMATION

After extracting MatOWL from the MatML schema, instance data of MatML can be populated into MatOWL. As XML has a very different structure from OWL, it is a nontrivial task to get data from MatML and change them to OWL instances. However, MatML can be naturally described by a set of associated objects because the structure of MatML can be considered as an element tree. Hence, we design an object model (named MatOO) to represent the hierarchy of MatML. The object model has a similar structure to MatML, and it has the similar style (classes and relationships) to the OWL model. Thus, the object model is built as the intermediate model for instance transformation. The process of instance transformation is shown in Figure 3.

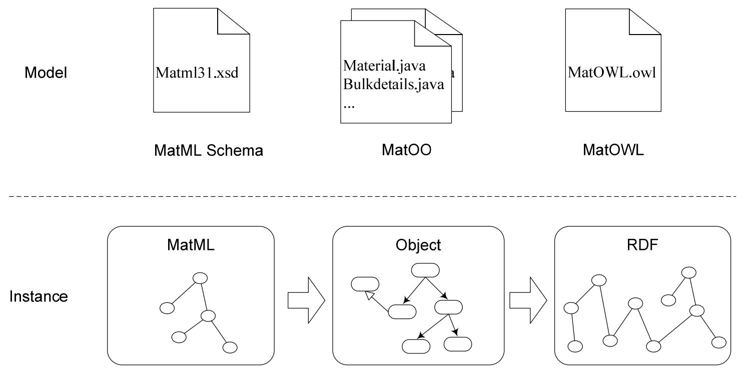


Figure 3. Process of instance transformation

5.1 MatOO model

MatOO is a model that is used to facilitate the process of instance transformation. Actually, MatOO is an object-tree with the same hierarchy as the element-tree of MatML. The elements are defined as Java classes, and the children of an individual element are defined as its member variables. If a child element can occur multiple times, then the corresponding member variable is defined as an object array; if the type of the child is *xsd:IDREF*, it is defined as the correcponding referenced type. Figure 4 shows a part of the MatOO model. For example, *PropertyData*'s child *specimen* is the type of *xsd:IDREF*, so it is defined as the class *SpecimenDetails*; because *PropertyData*'s child *ParameterValues* may occur multiple times, it is defined as an array. The names of the Java classes and member variables come from MatML schema, and both getter and setter methods are also defined in these classes. We also define a mapping table to describe the name correspondence between MatOO and MatOWL. Currently, we just use a XML document to describe the name mappings by which the name of class and property from MatOO can be mapped to the identifier of MatOWL class and property.

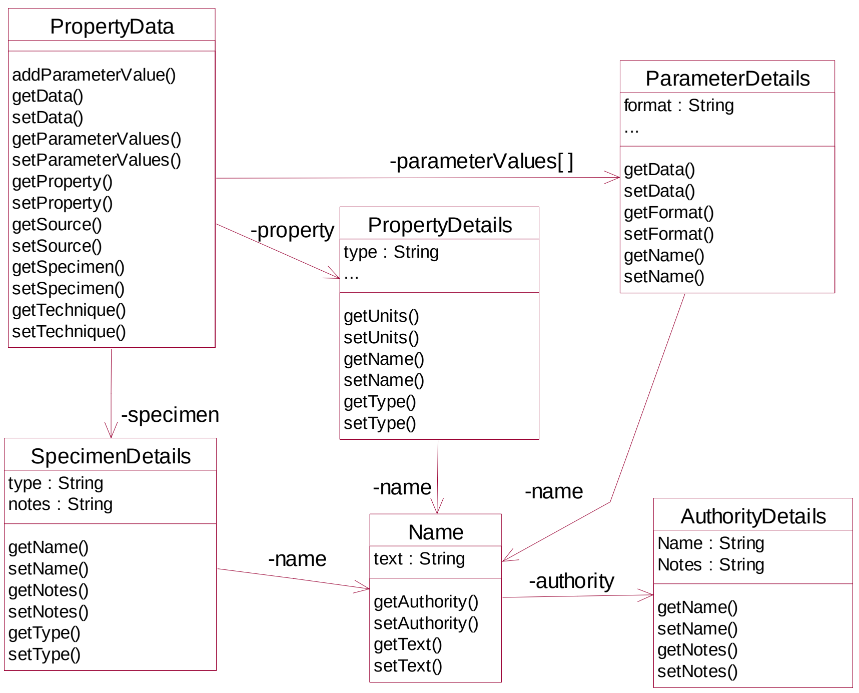


Figure 4. MatOO Model (partial)

5.2 Algorithm for instance transformation

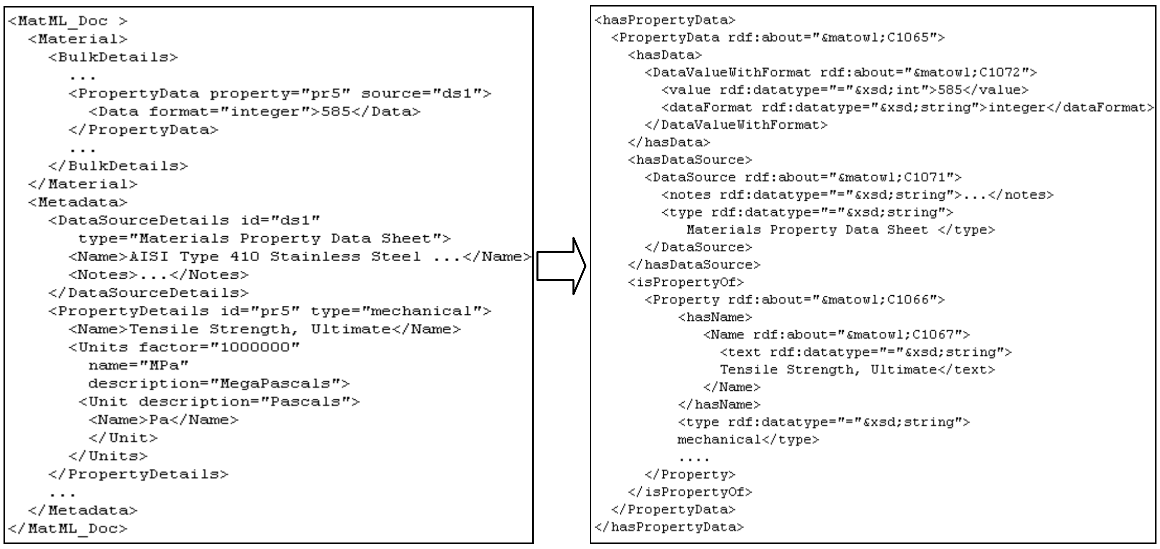
Due to the similar structure of MatOO and MatML, it is easy to parse a MatML document and populate the instance data into the objects of MatOO. How to dump data from MatOO to MatOWL is the key step of our approach. The algorithm for the instance transformation from MatOO to MatOWL is given in Table 1 as pseudo code. The input of the algorithm is an ontology (i.e., MatOWL) model *OM*, a root class *C* of MatOO, and an initial instance *I*. The output of the algorithm is an ontology model *OM'* with populated OWL instances.

Table 1. An algorithm for instance transformation from MatOO to MatOWL

```
Algorithm. transform (OntModel OM, Object C, Individual I)
Input: the MatOWL ontology model OM,
       a root class C of MatOO model,
       an initial individual I
Output: the ontology model OM' with populated instances.
1.    $F^C$=getAllFieldsFromClass(C);
2.    for each $f_i^C$ in $F^C$
3.        pstr=getOntPropNameFromMappingTable( $f_i^C$ );
4.        if pstr!=null
5.            p=OM.getOntProperty(pstr);
6.            for each $fO_{i,k}^C$ in $f_i^C$
7.                if p.isDatatypeProperty()
8.                    I.addProperty(p, $fO_{i,k}^C$ );
9.                else if p.isObjectProperty()
10.                   $RI_p$ =CreateRangeIndividual(p);
11.                   I.addProperty(p, $RI_p$);
12.                   transform(OM, $fO_{i,k}^C$ , $RI_p$);
13.       else    transform(OM, $f_i^C$ , I);
```

The algorithm first gets all fields (members) of the root class *C*, with F^C being a set of fields of class *C* (line 1). For each field $f_i^C \in F^C$, the algorithm then tries to transform its value to the MatOWL instance (line 2). In the next step, the property name *pstr* in MatOWL corresponding to f_i^C is retrieved from the mapping table between MatOO and MatOWL (line 3). If the property name is not null, the algorithm gets the ontology property object *p* from the ontology model *OM* (line 4-5). Because f_i^C may be an array, for each object $fO_{i,k}^C$ in f_i^C array, the algorithm then tries to transform $fO_{i,k}^C$ to the MatOWL instance (line 6). Obviously, if f_i^C is not an array, it can be considered as an array with only one array element. Then we will judge whether *p* is a data type property or an object property. If *p* is a data type property, the content of $fO_{i,k}^C$ will be assigned to the property *p* of the instance *I* (line 7-8); otherwise, if *p* is an object property, we first get the range of *p* (named RI_p), and link *I* and RI_p by property *p*; then the algorithm makes a recursive call using $fO_{i,k}^C$ as the root class and RI_p as the initial instance respectively (line 9-12). Finally, if *pstr* is null, it means that no corresponding property name is found in the mapping table. The algorithm recursively executes lookup from the next level in the object-tree in order to check if the corresponding property name can be found (line 13). When this recursive algorithm stops, it has traversed all MatOO objects whose instance data have been populated into the ontology model *OM*, which now becomes the result ontology model *OM'*.

Figure 5 shows an example of the instance transformation from MatML to MatOWL. The left part is a snippet code of an input MatML document, while the right is the partial MatOWL instances generated by the algorithm.



```
<MatML_Doc >
  <Material>
    <BulkDetails>
      ...
      <PropertyData property="pr5" source="ds1">
        <Data format="integer">585</Data>
      </PropertyData>
      ...
    </BulkDetails>
  </Material>
  <Metadata>
    <DataSourceDetails id="ds1"
       type="Materials Property Data Sheet">
      <Name>AISI Type 410 Stainless Steel ...</Name>
      <Notes>...</Notes>
    </DataSourceDetails>
    <PropertyDetails id="pr5" type="mechanical">
      <Name>Tensile Strength, Ultimate</Name>
      <Units factor="1000000"
        name="MPa"
        description="MegaPascals">
       <Unit description="Pascals">
        <Name>Pa</Name>
        </Unit>
      </Units>
    </PropertyDetails>
    ...
  </Metadata>
</MatML_Doc>
```

```
<hasPropertyData>
  <PropertyData rdf:about="&matowl;C1065">
    <hasData>
      <DataValueWithFormat rdf:about="&matowl;C1072">
        <value rdf:datatype="="&xsd;int">585</value>
        <dataFormat rdf:datatype="&xsd;string">integer</dataFormat>
      </DataValueWithFormat>
    </hasData>
    <hasDataSource>
      <DataSource rdf:about="&matowl;C1071">
        <notes rdf:datatype="="&xsd;string">...</notes>
        <type rdf:datatype="="&xsd;string">
           Materials Property Data Sheet </type>
      </DataSource>
    </hasDataSource>
    <isPropertyOf>
      <Property rdf:about="&matowl;C1066">
          <hasName>
             <Name rdf:about="&matowl;C1067">
               <text rdf:datatype="="&xsd;string">
               Tensile Strength, Ultimate</text>
             </Name>
          </hasName>
          <type rdf:datatype="="&xsd;string">
          mechanical</type>
          ....
      </Property>
    </isPropertyOf>
  </PropertyData>
</hasPropertyData>
```

Figure 5. An example for instance transformation

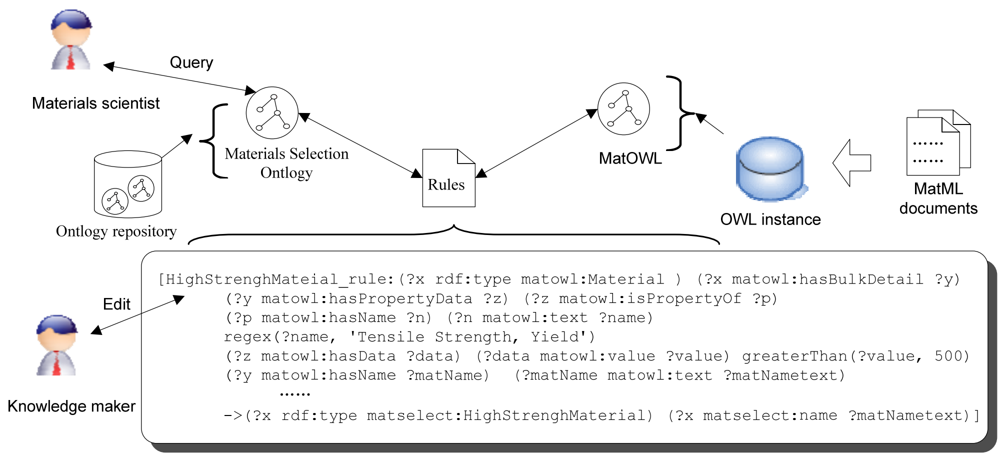
6 MATOWL EXTENSION AND SEMANTIC QUERY

After getting the OWL instances of MatOWL, we can use the concepts and properties from MatOWL to retrieve these instances with semantic queries. However, two issues should be taken into consideration in the semantic query. First, although OWL is more expressive than XML, all contents of MatOWL come from the MatML schema. Actually, MatOWL is an OWL version of MatML. Therefore, it is hard to say that MatOWL can provide more semantics than the MatML schema. Second, MatOWL is not convenient enough for users to write a query. On the contrary, users are more willing to write a query by using domain terms that they are familiar with, i.e., one user-oriented query view is more convenient for materials scientists. If we would like to get more interesting query results from MatOWL with an OWL inference engine, additional concepts, axioms, and rules should be added to MatOWL. Therefore, it is necessary to extend MatOWL.

In our opinion, there are two ways to make MatOWL more useful in the semantic world: (1) to add domain concepts, property, and axioms to MatOWL directly; (2) to map MatOWL to other existing ontologies in materials science. Then the DL axiom or logic rules could be built to associate MatOWL with the new contents in both ways.

After enhancing MatOWL by new domain concepts with the logic rules, writing semantic queries on the materials data becomes fairly intuitive to materials scientists. When users want to retrieve materials data from MatOWL, they can utilize not only concepts and properties from MatOWL but also domain concepts and properties from other domain ontologies. In this way, we can implement a user-oriented and semantic-based query on the MatOWL instances which are derived from MatML.

For example, a user wants to perform a semantic query for materials selection using domain terms such as high-strength materials or corrosion-resistant materials. We should build a (or use an existing) materials selection ontology that defines the related domain concepts and properties. Then we can map the materials selection ontology to MatOWL by adding the logic rules between them. In other words, we use knowledge from MatOWL to explain the concepts of materials selection ontology (shown in Figure 6). Once the mapping rules are created, we can compose a query using terms from the materials selection ontology, which is just a virtual view for users. The rule engine can help to answer query automatically using the instances from MatOWL.



```
[HighStrenghMateial_rule:(?x rdf:type matowl:Material ) (?x matowl:hasBulkDetail ?y)
      (?y matowl:hasPropertyData ?z) (?z matowl:isPropertyOf ?p)
      (?p matowl:hasName ?n) (?n matowl:text ?name)
      regex(?name, 'Tensile Strength, Yield')
      (?z matowl:hasData ?data) (?data matowl:value ?value) greaterThan(?value, 500)
      (?y matowl:hasName ?matName)  (?matName matowl:text ?matNametext)
          ……
      ->(?x rdf:type matselect:HighStrenghMaterial) (?x matselect:name ?matNametext)]
```

Figure 6. Mapping between ontologies using logic rules

When users select materials, assuming they intend to find high-strength materials, they can just use the term *HighStrengthMaterial* to make a query. The SPARQL query is presented as follows:

```
SELECT ?mat   ?name
WHERE { ?mat   rdf:type   HighStrengthMaterial.
          ?mat   matselect:name   ?name
        }
```

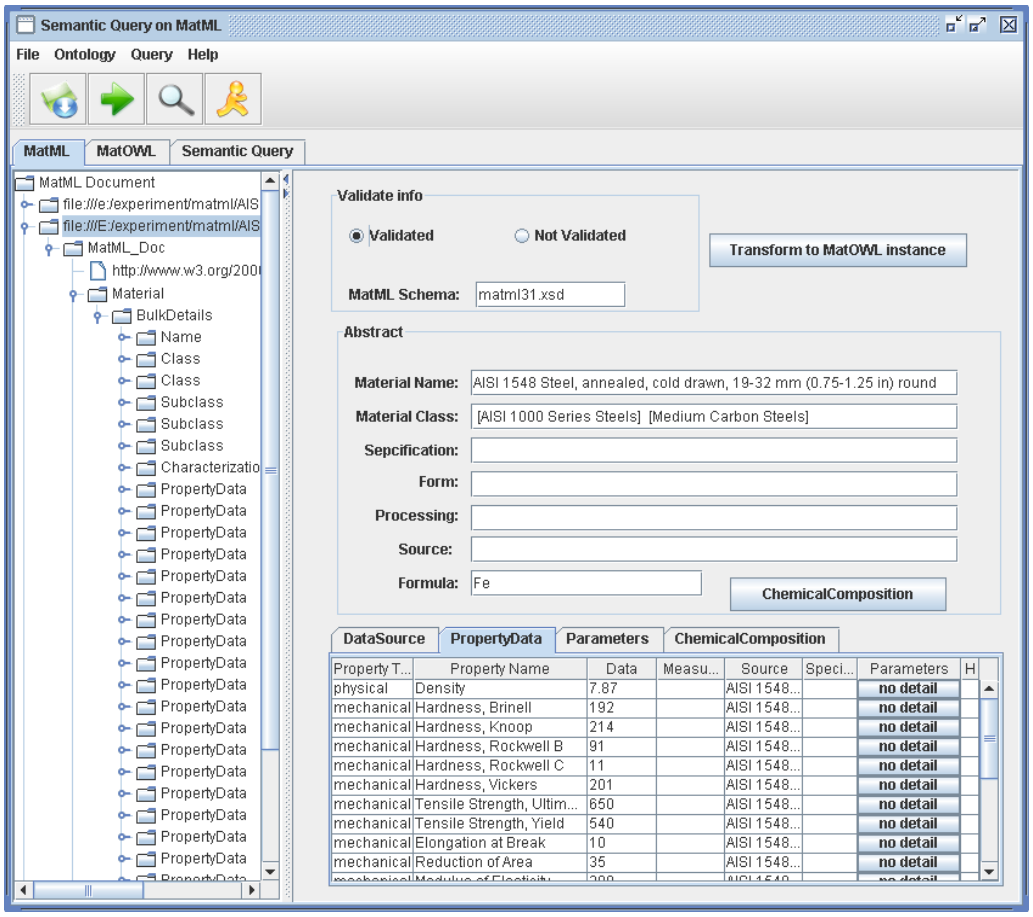
The rule engine can automatically classify the related instances from MatOWL to *HighStrengthMaterial*. The materials scientists need to know only the domain terms they are familiar with instead of the concepts from MatOWL.

7 EXPERIMENTAL PROTOTYPE

An experimental prototype has been implemented to evaluate our proposed method. In particular, the prototype has the following four functionalities: (1) it can be used to browse materials property data in MatML documents; (2) one or more MatML documents can be transformed into corresponding MatOWL instances; (3) we can use this prototype to execute a SPARQL query to retrieve the desired resources from the translated MatOWL; and (4) the prototype utilizes the OWL reasoner and rule engine to return the inferred query results.

The graphic user interface (GUI) of the prototype is developed by Java Swing, and DOM4j (DOM4j, 2005) is used to parse MatML. The Jena API (Jena, 2007) is used to manipulate the extracted OWL, and logic rules are defined based on the Jena rules. We use the MatML schema version 3.1 as on the MatML website, with the MatML documents from MatDL Materials Selector (MatDL) as our experimental data set.

As shown in Figure 7, we can import one or more MatML documents into our system. Thanks to the proposed MatOO model, we can easily browse both structure and materials data from an imported MatML document in detail. When a user selects a MatML document from the tree on the left, he/she can browse detailed information such as property data, chemical composition, and parameter values.



```
MatML Document
  file:///e:/experiment/matml/AIS
  file:///E:/experiment/matml/AIS
    MatML_Doc
      http://www.w3.org/200
      Material
        BulkDetails
          Name
          Class
          Class
          Subclass
          Subclass
          Subclass
          Characterizatio
          PropertyData
          PropertyData
          PropertyData
          PropertyData
          PropertyData
          PropertyData
          PropertyData
          PropertyData
          PropertyData
          PropertyData
          PropertyData
          PropertyData
          PropertyData
          PropertyData
          PropertyData
          PropertyData
          PropertyData
          PropertyData
```

Validate info

[✓ Validated] Not Validated

MatML Schema: matml31.xsd

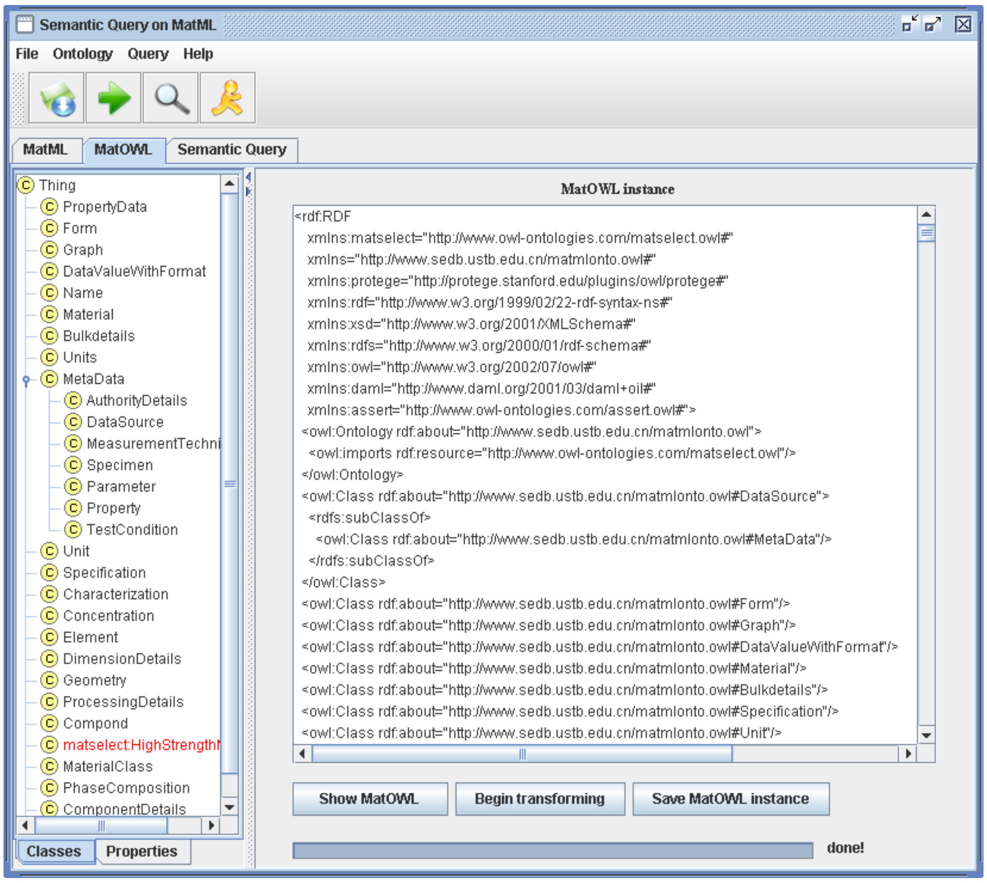
Abstract

Material Name:	AISI 1548 Steel, annealed, cold drawn, 19-32 mm (0.75-1.25 in) round
Material Class:	[AISI 1000 Series Steels] [Medium Carbon Steels]
Sepcification:	
Form:	
Processing:	
Source:	
Formula:	Fe

Property T...	Property Name	Data	Measu...	Source	Speci...	Parameters	H
physical	Density	7.87		AISI 1548...		no detail	
mechanical	Hardness, Brinell	192		AISI 1548...		no detail	
mechanical	Hardness, Knoop	214		AISI 1548...		no detail	
mechanical	Hardness, Rockwell B	91		AISI 1548...		no detail	
mechanical	Hardness, Rockwell C	11		AISI 1548...		no detail	
mechanical	Hardness, Vickers	201		AISI 1548...		no detail	
mechanical	Tensile Strength, Ultim...	650		AISI 1548...		no detail	
mechanical	Tensile Strength, Yield	540		AISI 1548...		no detail	
mechanical	Elongation at Break	10		AISI 1548...		no detail	
mechanical	Reduction of Area	35		AISI 1548...		no detail	

Figure 7. Loading and browsing MatML

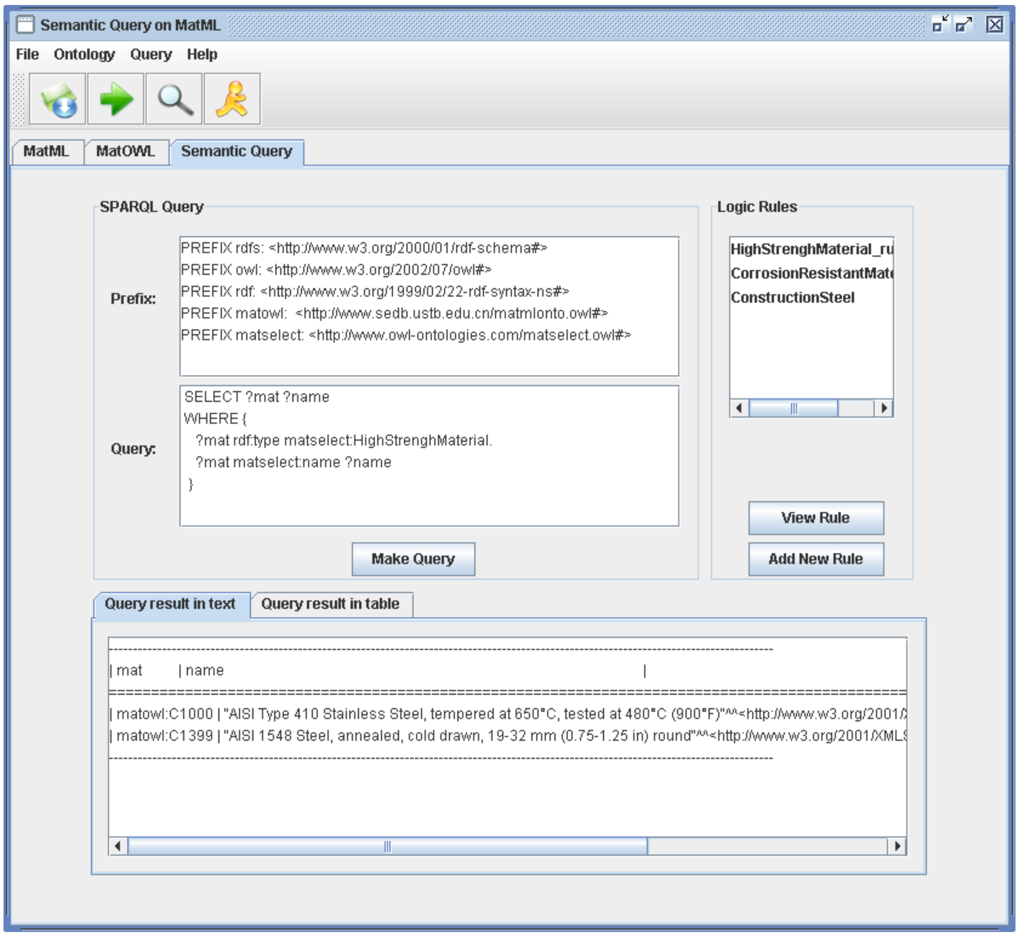
As shown in Figure 8, the button *show MatOWL* displays the translated MatOWL on the left as a concept tree. For example, the concept *matselect:HighStrengthMaterial* is an imported class from a certain ontology, and it has been mapped to a set of MatOWL concepts and properties by a logic rule. All MatML documents that have already been imported in the system can be transformed into the corresponding MatOWL instances one at a time by clicking the button *begin transforming*. The resulting instances are shown in the text area on the right side after the transformation.



```
<rdf:RDF
  xmlns:matselect="http://www.owl-ontologies.com/matselect.owl#"
  xmlns="http://www.sedb.ustb.edu.cn/matmlonto.owl#"
  xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:assert="http://www.owl-ontologies.com/assert.owl#">
 <owl:Ontology rdf:about="http://www.sedb.ustb.edu.cn/matmlonto.owl">
  <owl:imports rdf:resource="http://www.owl-ontologies.com/matselect.owl"/>
 </owl:Ontology>
 <owl:Class rdf:about="http://www.sedb.ustb.edu.cn/matmlonto.owl#DataSource">
  <rdfs:subClassOf>
   <owl:Class rdf:about="http://www.sedb.ustb.edu.cn/matmlonto.owl#MetaData"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:about="http://www.sedb.ustb.edu.cn/matmlonto.owl#Form"/>
 <owl:Class rdf:about="http://www.sedb.ustb.edu.cn/matmlonto.owl#Graph"/>
 <owl:Class rdf:about="http://www.sedb.ustb.edu.cn/matmlonto.owl#DataValueWithFormat"/>
 <owl:Class rdf:about="http://www.sedb.ustb.edu.cn/matmlonto.owl#Material"/>
 <owl:Class rdf:about="http://www.sedb.ustb.edu.cn/matmlonto.owl#Bulkdetails"/>
 <owl:Class rdf:about="http://www.sedb.ustb.edu.cn/matmlonto.owl#Specification"/>
 <owl:Class rdf:about="http://www.sedb.ustb.edu.cn/matmlonto.owl#Unit"/>
```

Figure 8. MatOWL instance transformation

The SPARQL query generator is still in development and will provide a GUI to help users build a SPARQL query based on an ontology. Currently, users can use a SPARQL query form to input SPARQL queries as shown in Figure 9. The right rule list shows the logic rules that have been defined in advance. With user-defined rules, the system can automatically perform ontology reasoning with the Jena rule engine and return the inferred answers to the user.



```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX matowl:  <http://www.sedb.ustb.edu.cn/matmlonto.owl#>
PREFIX matselect: <http://www.owl-ontologies.com/matselect.owl#>
```

```
SELECT ?mat ?name
WHERE {
  ?mat rdf:type matselect:HighStrenghMaterial.
  ?mat matselect:name ?name
 }
```

```
---------------------------------------------------------------------------------------------------------
| mat        | name                                                                                    |
=========================================================================================================
| matowl:C1000 | "AISI Type 410 Stainless Steel, tempered at 650°C, tested at 480°C (900°F)"^^<http://www.w3.org/2001/
| matowl:C1399 | "AISI 1548 Steel, annealed, cold drawn, 19-32 mm (0.75-1.25 in) round"^^<http://www.w3.org/2001/XML
---------------------------------------------------------------------------------------------------------
```

Figure 9. Query interface

By translating the MatML into our MatOWL, we can conclude the following advantages: (1) both XQuery and XPath on the MatML documents are essentially queries on the structural tree, whereas the SPARQL queries on the MatOWL instances are queries on a semantic graph. Obviously, the SPARQL query allows users to explore the content of a MatML document more semantically. (2) Once a common conceptual model (i.e., MatOWL) is built, the integration of different MatML data sets will become more convenient. (3) Combined with the existing domain ontology by logic rules, the MatOWL instances can be accessed in a more semantic way. (4) The existing inference mechanism can be easily integrated into the system to find more useful knowledge from the original MatML data. The detailed comparison between MatML and MatOWL is shown in Table 2.

Table 2. MatML vs. MatOWL

Comparison Item	**MatML**	**MatOWL**
Structure definition	XML Schema	Ontology
Data structure	tree	graph
Query Language	XPath/XQuery	SPARQL
Query construction	structure aware	semantic aware
Relationship	parent/son/brother	subclass /object property
Reasoning ability	no	yes

8 CONCLUSION

In this paper, we have presented an approach to support semantic query on MatML-based materials data by transforming the MatML into the corresponding OWL ontology. A set of rules are formally defined for extracting MatOWL, and an intermediate object model is built to facilitate the instance transformation. The proposed method allows users to conveniently exploit the original MatML from the view of different domain ontologies by rule-based mapping between them. The experimental prototype has shown the feasibility and effectiveness of our proposed approach. The major limitation of the current prototype tool is the lack of GUI support for both logic rules construction and query generation. In the future, we plan to support SWRL (Horrocks, Patel-Schneider, Boley, Tabet, Grosof, & Dean, 2004) rules, implement a logic rule editor to simplify the process of writing logic rules, and develop a query generator based on domain terms, which will make it much easier for materials scientists to build a semantic query.

9 ACKNOWLEDGEMENTS

This work is partially supported by the Key Technologies Research and Development Program of China under Grant No. 2006BAK11B03, the R&D Infrastructure and Facility Development Program under Grant No. 2005DKA32800, and the Hi-Tech Research and Development Program (863) of China under Grant No. 2006AA01Z105.

10 REFERENCES

An, Y., Borgida, A., & Mylopoulos, J. (2005) Constructing Complex Semantic Mappings between XML Data and Ontologies. *Proc. 4th International Semantic Web Conference* (pp. 6-20). Galway, Ireland.

Ashino, T. & Fujita, M. (2006) Definition of a Web Ontology for Design-Oriented Material Selection. *Data Science Journal 5(Jun)*, pp 52-63.

Ashino T. & Oka N. (2007) Development of Information Platform for Data Exchange between Heterogeneous Material Data Resources. *Proc. Ensuring the Long-Term Preservation and Value Adding to Scientific and Technical Data.* Munich, Germany.

Bartolo, L.M., Lowe, C.S., Sadoway, D., Powell, A., & Glotzer, S. (2005) NSDL MatDL: Exploring Digital Library Roles. *D-Lib Magazine 11(3).*

Bartolo, L.M. & Lowe, C.S. (2003) A Preliminary Investigation of Metadata Description Mechanisms for Materials Science. Proc. 2003 Dublin Core Conference. Seattle, Washington.

Bechhofer, S., Harmelen, F.v., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., et al. (2004) OWL Web Ontology Language Reference, W3C Recommendation. Retrieved Mar 10, 2008 from the World Wide Web: http://www.w3.org/TR/owl-ref/

Begley, E. & Howard-Reed, C. (2005) Application of MatML to Contaminant Emissions Data. *ASTM Standardization News*, pp 52-59.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001) The Semantic Web. *Scientific American Magazine 284(5*), pp

34-43.

Bohring, H. & Auer, S. (2005) Mapping XML to OWL Ontologies. *Proc.* 13. *Leipziger Informatik-Tage* (pp. 147-156). Leipzig, Germany.

Cheung, K., Drennan, J., & Hunter, J. (2008) Towards an Ontology for Data-Driven Discovery of New Materials. *Proc. Semantic Scientific Knowledge Integration AAAI/SSS Workshop*, Stanford, USA.

DOM4j Version 1.6.1 (2005). Retrieved Jan 5, 2008 from the World Wide Web: http://www.dom4j.org/

Ferdinand, M., Zirpins, C., & Trastour, D. (2004) Lifting XML Schema to OWL. *Proc. International Conference on Web Engineering* (pp. 354-358). Munich, Germany.

Gruber, T.R. (1993) A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition 5(2),* pp 199-220.

Hunt, W. (2006) Materials Informatics: Growing from the Bio World. *JOM Journal of the Minerals, Metals and Materials Society 58(7)*, pp 88-88.

Hunter, J., Little, S., & Schroeter, R. (2008) The Application of Semantic Web Technologies to Multimedia Data Fusion within eScience. In Kompatsiaris, Y. & Hobson, P., (Eds.*), Semantic Multimedia and Ontologies*, London: Springer.

Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., & Dean, M. (2004) SWRL: A Semantic Web Rule Language, Combining OWL and RuleML. Retrieved Mar 10, 2008 from the World Wide Web: http://www.w3.org/Submission/SWRL/

ISO (1994) ISO 10303–1:1994, Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 1: Overview and Fundamental Principles. Geneva: International Organization for Standardization.

ISO (2001) ISO 13584–1:2001, Industrial Automation Systems and Integration - Parts library - Part 1: Overview and Fundamental Principles. Geneva: International Organization for Standardization.

ISO (2008) ISO 10303–45:2008 Edition 2, Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 45: Integrated generic resource: Material and other engineering properties. Geneva: International Organization for Standardization.

Iwata, S., Shichijo, N., & Ashino, T. (2001) Modular Simulation Technique for Virtual Experiment of Complex Phenomena in Materials. *Materials & Design 22(1*), pp 77-79.

Jena Version 2.5.2 (2007). Retrieved Jan 5, 2008 from the World Wide Web: http://jena.sourceforge.net/

Kobeissy, N., Genet, M.G., & Zeghlache, D. (2007) Mapping XML to OWL for Seamless Information Retrieval in Context-Aware Environments. *Proc. IEEE International Conference on Pervasive Services* (pp. 349-354). Istanbul, Turkey.

Lehti, P. & Fankhauser, P. (2004) XML Data Integration with OWL: Experiences and Challenges. *Proc. 2004*

International Symposium on Applications and the Internet (pp. 160-167). Tokyo, Japan.

MatDL: MatML and Material Grapher. Retrieved Mar 10, 2008 from the World Wide Web: http://orbis.kent.edu/matdl/matml/select.php

MatML Schema (2004) MatML Schema Version 3.1. Retrieved Mar 10, 2008 from the World Wide Web: http://www.matml.org/downloads/matml31.xsd

NMC-MatDB Schema (2007) NMC-MatDB Schema Version 3.08. Retrieved Mar 10, 2008 from the World Wide Web: http://www.nims.go.jp/vamas_twa10/AMM_DB/XML-Schema.LZH.

Prud'hommeaux, E. & Seaborne, A. (Eds.) (2008) SPARQL Query Language for RDF, W3C Recommendation Retrieved Mar 10, 2008 from the World Wide Web: http://www.w3.org/TR/rdf-sparql-query/

Reif, G., Jazayeri, M. & Gall, H. (2004) Towards Semantic Web Engineering: WEESA-Mapping Xml Schema to Ontologies. *Proc. WWW2004 Workshop on Application Design, Development and Implementation Issues in the Semantic Web*, New York, USA.

Rodrigues, T., Rosa, P., & Cardoso, J. (2005) Mapping Xml to Existing OWL Ontologies. *Proc*. IADIS International Conference WWW/Internet 2006 (pp. 72-77). Murcia, Spain.

Studer, R., Benjamins, V.R., & Fensel, D. (1998) Knowledge Engineering: Principles and Methods. *Data & Knowledge Engineering 25(1-2)*, pp 161-197.

Sturrock, C.P., Begley, E.F., & Kaufman, J.G. (2001) MatML –Materials Markup Language Workshop Report, *Technical Report NISTIR 6785*, National Institute of Standards and Technology, Gaithersburg, MD.

Swindells N. (2002) Communicating Materials Information: Product Data Technology for Materials. *International Materials Reviews*, February 47(1), pp 31-46.

Task Group (2006) CODATA Task Group for Exchangeable Materials Data Representation. Retrieved Mar 10, 2008 from the World Wide Web: http://www.codata.org/taskgroups/TGmatlsdata/

U Queensland (2006) Materials Informatics Workshop. Retrieved Mar 10, 2008 from the World Wide Web: http://www.itee.uq.edu.au/~eresearch/workshops/materials_informatics_06/

Varde, A.S., Begley, E.F., & Fahrenholz-Mann, S. (2006) MatML: XML for Information Exchange with Materials Property Data. *Proc. ACM KDD's 4th international workshop on Data mining standards, services and platforms* (pp. 47-54). Philadelphia, Pennsylvania.

Westbrook, J.H. (2003) Materials Data on the Internet. *Data Science Journal 2(Nov)*, pp 198-212.