EXTENSIBLE ONTOLOGICAL MODELING FRAMEFORK FOR SUBJECT MEDIATION

L. A. Kalinichenko, N. A. Skvortsov
Institute for Problems of Informatics RAS
Moscow, Vavilova, 44/2
{leonidk,scvora}@synth.ipi.ac.ru

Abstract

An approach for extensible ontological model construction in a mediation environment intended for heterogeneous information sources integration in various subject domains is presented. A mediator ontological language (MOL) may depend on a subject domain and is to be defined at the mediator consolidation phase. On the other hand, for different information sources different ontological models (languages) can be used to define their own ontologies. Reversible mapping of the source ontological models into MOL is needed for information sources registration at the mediator. An approach for such reversible mapping is demonstrated for a class of the Web information sources. It is assumed that such sources apply the DAML+OIL ontological model. A subset of the hybrid object-oriented and semi-structured canonical mediator data model is used for the core of MOL. Construction of a reversible mapping of DAML+OIL into an extension of the core of MOL is presented in the paper. Such mapping is a necessary pre-requisite for contextualizing and registration of information sources at the mediator. The mapping shows how extensible MOL can be constructed. The approach proposed is oriented on digital libraries where retrieval is focused on information content, rather than on information entities.

1 Introduction

This paper has been written in context of a project[1] investigating *subject mediation* approach supporting information integration in a particular subject domain. Web information integration systems, digital libraries providing content interoperability, digital repositories of knowledge in certain domains (like: Digital Earth, Digital Sky, Digital Bio, Digital Law, Digital Art, Digital Music) are few examples of respective areas. In digital libraries created in such areas retrieval is focused on information content, rather than on information entities. For example, in digital libraries for education semantic conceptual modeling is required going beyond conventional textbooks and courses as information entities. Another example is an interoperability of metadata registries where crossing of boundaries between different information contexts is required.

For such areas, according to the approach, the application domain model is to be defined by the experts in the field independently of potentially relevant

[1] This research has been partially supported by the grant of the Russian Foundations for Basic Research N 01-07-90084

information sources. This model may include specifications of data structures, terminologies (thesauri), concepts (ontologies), methods applicable to data, processes (workflows) characteristic for the domain. These definitions constitute specification of a *subject mediator*. After subject mediator had been specified, information providers can disseminate their information for integration in the subject domain independently of each other and at any time. To disseminate they should *register* their information sources at the subject mediator. Users may not know anything about the registration process and about the sources that have been registered. Users should know only subject domain definitions that contain concepts, structures, methods approved by the subject domain community. Thus various information sources belonging to different providers can be registered at a mediator.

The subject mediation approach is applicable to various subject domains in science, cultural heritage, mass media, e-commerce, etc. This technology is contrasted with the widely used general purpose Web search engines characterized by very low precision of search due to uncontrollable use of terms for indexing and search. This is unavoidable payment for simplicity of sites “registration” at the Web.

Local as View (LAV) mediator architecture [5] is assumed as a basis for the subject mediation. According to LAV, schemas exported by sources are taken as materialized views over virtual classes of the mediated schema. Queries are expressed in terms of the mediated schema. The LAV architecture is designed to cope with a dynamic, possibly incomplete set of sources. Sources may change their exported schemas, become unavailable from time to time. LAV is potentially scalable with respect to a number of sources involved.

Two separate phases of the subject mediator functioning are distinguished: *consolidation* phase and *operational* phase. Consolidation phase is intended for the subject model definition. During this phase the mediator's schema metainformation is formed. During operational phase the burden of the sources registration process is imposed on the information providers. They formulate sources' specifications (schemas, concept definitions, vocabularies) in terms of the subject mediator's metainformation. In process of registration the local metainformation sublayer of the mediator is formed expressing source schemas in the mediator's canonical model as views above the mediator schema. Managing source registration concurrently by providers is the way to reach the mediator's scalability.

In [3] methods and tools required to support information source registration process at the mediator are presented. For the LAV mediation strategy, [3] considers information source registration as the process of compositional information systems development [2]. Source metainformation definitions are treated as specifications of requirements and classes of the mediator level with the related metainformation – as specifications of pre-existing components. To get

source classes definitions as views above the mediator level, facilities of specific method and tool [3] are applied.

Specifications of information sources and mediator specifications must be associated with *ontological contexts* defining concepts of the respective subject areas. For the uniformity reasons, the ontological concepts are described by means of the mediator's canonical model as type specifications. Ontological concepts are also described with their verbal definitions similar to definitions of words in an explanatory dictionary. Verbal definitions are required for establishing preliminary semantic relationships between concepts. Several kinds of semantic relationships (positive, hyponym/hypernym) can be discovered between ontological concepts as fuzzy ones.

To contextualize information sources at the mediator during their registration, it is required to map local ontological contexts of the sources into the mediator's ontology. Fuzzy relationships between concepts of different contexts are established by calculating correlation coefficients between concepts on the basis of their verbal definitions. The correlation coefficients are calculated using the vector-space approach [10]. This technique of the ontological composition (*loose ontological integration*) provides for establishing fuzzy correlation between related concepts [13]. Another technique (*tight ontological integration*) consists in composition of ontological modules using complete specifications of concepts as type definitions.

In [7] the GIMP protocol (GIMP – Generalized Intermediator Protocol) supporting registration of an information source at a mediator is considered. A subset of OAI protocol [11] is used during registration to exchange metainformation uniformly represented in the canonical model of the intermediator framework. According to GIMP, a canonical schema is organized in modules, each of them containing all subject mediator specifications of one of the following kinds: structural specifications (definitions of subject mediator types and classes), ontological specifications, thesaurus specifications, classifier specifications, etc. For each kind of metainformation format, an XML-oriented definition and a respective namespace are provided. In terms of OAI, a subject mediator can disseminate metainformation in multiple formats mentioned. After obtaining the required specifications from the mediator, the provider contextualizes its source metainformation in the mediator's context, maps local source structural definitions into the canonical model and constructs representation of local classes in terms of the mediator's classes. The results of the registration are formed as a local source schema expressed in terms of the canonical model with the appropriate terminological, structural and ontological links to the respective components of the mediator schema. Mediator takes registration information from the provider to complete the registration.

One of the main difficulties of the tight ontological integration in frame of GIMP consists in reconciling of different ontological models in the mediator

and information sources. This problem is discussed in this paper in details regarding contextualization of the Web information sources in the subject mediator as an example. Basic model considered now for the Semantic Web is a Web ontology language which can formally describe the semantics of classes and properties used in Web documents. DAML+OIL has been submitted to W3C as a candidate for the Web ontology language [1,4]. Ontologies are intended to improve existing Web-based applications and may enable new uses of the Web. DAML+OIL builds on earlier W3C standards such as RDF and RDF Schema, and extends these languages with richer modeling primitives. In the mediator considered here the SYNTHESIS language [6] is used for the canonical modeling. This is a hybrid object-oriented and semi-structured data model with a logic capability. To make registration of a Web source at the mediator possible, a *reversible mapping* of the Web ontology language into the mediator ontological model is required. Under this condition, applying the GIMP protocol, transformation of the mediator ontological specifications into DAML+OIL becomes possible and after contextualization made in DAML+OIL model, the result is transformed to the mediator representation and is returned back to the mediator.

Since the registration is done by providers in their native environments, DAML+OIL capabilities can be applied using methods of proving structural subsumption and satisfiability of concepts after mapping of the mediator ADT concept specifications into DAML+OIL. Proof of a subsumption of DAML+OIL classes implies subtyping relationship of the respective ADTs after the reverse mapping of DAML+OIL into the canonical mediator model.

The paper presents in the subsequent sections a reversible mapping of DAML+OIL into the mediator ontological model the core of which is defined as the SYNTHESIS language subset. An extension of this core having equal to DAML+OIL expressive power is introduced showing mapping of DAML+OIL into such extension. Reverse mapping of the resulted MOL into DAML+OIL is presented in the next section showing that the mapping commutes. A correspondence between DAML+OIL, MOL constructs and Description Logic is summarized in the table given in Appendix.

The knowledge of DAML+OIL [1,4] by readers is assumed.

2 Mediator's Ontological Framework

The mediator's ontological framework introduced here is based on the following considerations. For different information sources different ontological models (languages) can be used to define their ontologies. At the consolidation phase a concrete mediator ontological model is to be chosen (constructed) for the definition of the ontology of a specific subject domain. Different models may be required for different domains. Here we assume that the core of the mediator ontological language is defined as a subset of the mediator canonical model. Various approaches for construction of subject domain-oriented ontolog-

ical models can be designed. For instance, such model can be developed based on a prediction of a set *M* of ontological models that are used for the sources potentially relevant to the mediator subject domain. In this case the mediator model can be constructed as a union of extensions of the MOL core equivalent to each of the model in *M*.

During operational phase, to make tight ontological integration possible, each source ontological model should be mapped into the MOL. It is well known that each such mapping should be formed as an extension of the MOL core having equal capabilities with the source model [9].

Due to the above, the following ontological language framework for the mediator is assumed. To construct various ontological models, a core of MOL is defined as a subset of the mediator canonical model. The main constituent to be defined in the model is a concept. A concept is an entity of knowledge representation that reflects characteristics of all similar objects of real world that could exist in a given subject domain. In terms of the canonical model of a mediator a concept is natural to define as an abstract data type (ADT). ADT in the mediator canonical model (SYNTHESIS [6]) is an object-oriented notion applied for modeling of any kind of real-world entities. As ADT, a concept may be characterized by subtyping relationship forming concept hierarchy. Attributes, relationships, invariants can be imposed on instances of ADT.

Besides ADT, an ontological concept specification can contain its verbal description expressing the concept meaning in a subject domain. Verbal description is a natural-language definition of a concept, which may be used during registration for preliminary interrelating of mediator concepts with concepts from different source ontologies. For this purpose descriptor lists for concepts are taken from their verbal definitions to apply weighted vector-space concept similarity (hierarchy) evaluation method [10]. Descriptor list of a concept consists of lexical units characterizing given concept.

Using verbal description, between two concepts one of several kinds of semantic relationships can be established (such as positive and hypernym relationships). These relationships are treated as fuzzy ones. Using properties of the respective relations (e.g., transitivity) it is possible to infer concept relationships that are not represented directly. During the process of source registration at the mediator, loose integration of the source ontology with that of the mediator is applied [13] to establish verbal-based fuzzy relationships between concepts.

Tight ontology integration based on concept definitions as ADT is used during registration process for more precise reconciliation of a source ontological context with that of the subject mediator. Semantic relationships deduced from the verbal definitions of the concepts provide an intuition to look for their more sound interpretation. For instance, a positive verbal concept relationship assumes that equivalence of respective concept ADT specifications is expected.

Hyponym/hypernym relationship assumes that subtyping of respective concepts is expected.

The MOL core provides only hierarchical relationship between the concepts. All other relationships and constraints are provided in core extensions. Each extension together with the core is equal in its capabilities to a specific (source) ontological model. More details on the MOL core and constructing the extensions are provided in the following sections.

3 Mediator Ontological Language: the Core

3.1 General Features of the SYNTHESIS Language

For the canonical model of a mediator the SYNTHESIS language has been chosen [6]. This language uses a hybrid semi-structured/object data model [8]. The canonical model considered provides support of wide range of data - from untyped data on one end of the range to strictly typed data on another. Self-descriptive, semi-structured data are represented as *frames* that are used as symbolic models of some entities or concepts. The language uses frames to describe any entity, including the entities of the language itself, such as types, classes, functions, assertions. A frame at any moment of its life cycle can be declared belonging to an admissible class (class is a collection of typed objects). At that moment the frame becomes an *object*.

Typed data should conform to *abstract data types* (ADT) prescribing behaviour of their instances by means of the type's operations. ADT describes an interface of a type whose signature defines names and types of its operations. State-based attributes of ADT are defined with a shorthand: `<attribute name>: <type>`. Type invariants (constraints expressed as closed logic formulae) can be included into the type definition. Besides ADT, the language contains also a comprehensive collection of built-in datatypes. Subtyping relation over a collection of ADT forms a lattice with `Taval` as a root and `Tnone` as a bottom of the lattice.

Types in the language are objects themselves. The multilevel type system of the language is organized as follows. On the level of types the type objects are located providing for definition of concrete and generic types. On the second level (the level of "types of types") the metatype objects are located that include as their instances the types of the first level. On the third level the metatypes objects are located that include the metatypes of the second level as their instances, and so forth. Thus the multilevel type system sets a classification relationship on types that is orthogonal to the subtype relationship.

Metatypes behave like (meta)classes that in their turn are organized as follows. A class specification combines information about two kinds of objects: about a class as an object itself and about objects - instances of the class. Generally metaclasses provide for introducing of generic concepts and of common attributes (or of their categories) for similar classes, for introducing of common

consistency constraints and deductive rules for such classes and their attributes. Metaclasses provide for proper grouping of application domain information and for proper differentiation of various application domains.

In the language the attribute specifications of objects may be treated in their turn also as types of association objects establishing a correspondence between a set of objects in an association domain and a set of objects in an association range. Thus a specification of a type attribute may be considered as a specification of an association type.

3.2 The Core of the Mediator Ontology Language

A subset of the mediator's canonical model (the Core of the Mediator Ontology Language (MOL)) sufficient for constructing extensions equivalent to various ontological models is characterized in more details. Only small subset of the SYNTHESIS ADT specification is included with state-based attributes and invariants. ADT specifications are syntactically represented by frames, their attributes – by slots of the frames. Additional information related to attributes can be included into metaslots. Syntactically frames are included into figure brackets `{` and `}`, slots are represented as pairs `<slot name>:<slot value>` (a frame in its turn can be used as a slot value), slots in a frame are separated by semi-colons. Metaslots (that are represented by frames) are written immediately after the slots to which they are related.

Invariants are expressed using formulae of the SYNTHESIS object calculus. To specify formulae, a variant of a typed (multisorted) first order predicate logic language is used. Every predicate, constant and variable in formulae is typed. Predicates in formulae correspond to types or their Boolean combinations (expressed with `&` (intersection), `|` (union), `^` (complement)). Variables and constants are used as terms. Each term has a well-defined type. Explicitly typed variables are denoted as `<identifier>/<type>`.

Throughout the paper we use an example of ontology presented in [1]. This ontology uses classes: `Animal, Male, Female, Man, Woman, Person`, properties: `hasParent, hasFather, hasMother, hasSpouse, hasOccupation`, class elements `DisjointWith, DisjointUnionOf`, and properties restrictions. In the first part of the paper we show how an extension of the MOL core can be constructed with the same meaning as DAML+OIL. Later we show how the constructed model can be mapped back to DAML+OIL getting specifications similar to those that can be found in [1]. Please, note that all examples presented in the paper are pieces of one and the same ontology expressed in two different languages.

```
{Animal;
(1)
 in: type, daml_oil;
 hasParent:  Animal;
```

```
    metaslot
       in: HasParent
    end
 hasFather:  Male;
    metaslot
       in: HasFather
    end
 hasMother:  Female;
    metaslot
       in: HasMother;
    end
 hasMom:  Female;
    metaslot
       in: HasMother
    end
 age: integer;
    metaslot
       in: Age
    end
 sameAs: {in: invariant, samePropertyAs,
    {{ all a/Animal ( hasMother(a) = HasMom(a) }}}
};
{Male;
(2)
 in: type, daml_oil;
 supertype: Animal
};
{Female;
(3)
 in: type, daml_oil;
 supertype: Animal;
 disjoint: {in: invariant, disjointWith,
    {{Male(a/Animal) & Female(a/Animal) = {} }}}
};
```

(1) defines a type `Animal`. `type, daml_oil, invariant, sameProperyAs` are names of built-in metatypes. Attributes of `Animal` belong to association metatypes defined in the sequel. `sameAs` is a type invariant expressing semantics of `sameProperyAs` of DAML+OIL. Formulae in invariants are put into double figure brackets. (2), (3) are subtypes of `Animal` that correspond to the respective class elements in DAML+OIL example.

Treating of an object attribute as an association motivates an introduction of association metatypes establishing properties of such association types. It is said that an object attribute (as an association) belongs to a particular attribute category that is explicitly introduced by an association metatype. Association metatypes (or categories) are built-in or user defined.

User-defined association metatype is structured for our limited purpose as follows. Association metatype names (for direct and inverse associations) are used for category names of type attributes. An association type is defined in an `instance_section` of an association metatype. In an association type speci-

fication two pre-defined attributes `domain` and `range` can be given. Otherwise domain and (or) range of an association metatype are not restricted.

An association type (as a type of binary relation) is set by an attribute `association_type`. If an association *R* is defined on a domain C_1 and a range C_2 then the bounds (pairs of positive integers) define the following. The first bound gives for any object c_1 of C_1 an admissible range (minimal and maximal value) of a number of different objects c_2 of C_2 such that $\{<c_1,c_2>\}$ belongs to *R*. The second bound for any c_2 of C_2 gives a minimal and maximal value of a number of objects c_1 of C_1 such that $\{<c_2,c_1>\}$ belongs to an association inverse to *R*. `inf` is a constant denoting an arbitrary positive integer. Subtyping relation can be established on association metatypes.

```
{HasParent;
(4)
 in: metatype, association, daml_oil;
 inverse: HasChild;
 instance_section:
 {domain: Animal;
  range: Animal}
};
{HasFather;
(5)
 in: metatype, association, daml_oil;
 superclass: HasParent;
 instance_section:
 {domain: Animal;
  range: Male}
};
{HasFatherCard;
(6)
 in: metatype, association, daml_oil;
 superclass: HasFather;
 instance_section:
 {domain: Animal;
  range:Male;
  association_type: {{1,1},{0,inf}};
     metaslot
        in: onProperty, cardinality, UniqueProperty,
     end}
};
{HasMother;
(7)
 in: metatype, association, daml_oil;
 superclass: HasParent;
 instance_section:
 {domain: Animal;
  range: Female}
};
```

(4) – (7) are association metatypes defining features of type attributes (treated as properties in DAML+OIL). Domain, range and association type can

be defined. Slot inverse defines inverse association. (6) is a specialization of HasFather adding association type. This type is used to express partially a role of onProperty cardinality restriction in DAML+OIL.

4 MOL core extension equivalent to DAML+OIL model

We consider mapping of the object world of DAML+OIL [1,4] into MOL. Roughly the object world of DAML+OIL is mapped into MOL as follows: classes and restrictions are mapped into types, properties are mapped into type attributes (that may belong to certain association metatypes), individuals are mapped into frames. Before providing further details, new built-in SYNTHESIS metatypes (induced by DAML+OIL) are introduced:

- daml_oil: types and associations of ontology become instances of this metatype;
- restriction: classifies types belonging to it as modeling restrictions of DAML+OIL;
- disjointWith, disjointUnionOf, onProperty, toClass, hasValue, hasClass, cardinality, maxCardinality, minCardinality, cardinalityQ, maxCardinalityQ, minCardinalityQ, sameTypeAs, equivalentTo, samePropertyAs: metatypes to which invariants in type definitions can be associated. These invariants interpret respective DAML+OIL elements.

Other extensions of the MOL core follow.

4.1 Class Elements Interpretation

Classes Thing, Nothing of DAML+OIL are mapped into Taval, Tnone types of MOL respectively. subClassOf <class-expression list> is mapped into supertype <type name list>.

A type with name T in the type name list results of mapping of a class expression CE (this mapping is denoted for short CE>T; such double denotation will help explaining mapping of DAML+OIL constructs into MOL). For different cases of class expression (class name, enumeration type, property restriction, Boolean combination of class expressions) the mapping is denoted as:

- a type name T resulting from a mapping of a class name C (denoted for short as C>T);
- a name of the enumeration type - mapping of the DAML+OIL enumeration (denoted as E>EM);
- a name of type T - mapping of the DAML+OIL property-restriction R (denoted as R>T);
- a name T of a type defined by invariant showing that the set of admissible values of this type is equal to a resulting set of a formula F. F is a result of

mapping of Boolean combination Bexp of class expressions (denoted as `Bexp>T:F`).

A type corresponding to a class defined by a Boolean combination of class expressions look as follows:

```
{TallMan;
(8)
 in: type, daml_oil;
 inters: {in: invariant, Boolean_combination,
    {{TallMan(a/Taval) = (Man(a/Taval) & Tall-
Thing(a/Taval))}}}
};
```

`disjointWith` element asserting that C is disjoint with the class-expression CE in the element (i.e. C must have no instances in common with it) is mapped into the invariant (example (3)):

```
I: {in: invariant, disjointWith, {{(C>T1  & CE>T2  = {}) }}}
```

`disjointUnionOf` element asserts that C has the same instances as the disjoint union of the class-expressions element (all of the classes defined by the class-expressions of a `disjointUnionOf` element must be pairwise disjoint). For two class expressions CE1 and CE2 the resulting invariant looks as:

```
I: {in:predicate, invariant, disjointUnionOf,
   {{(C>T = (CE1>T1 | CE2>T2)) & (CE1>T1 & CE2>T2 = {}) }}}
```

Type `Person` (example 11) contains such an invariant. Types `Man, Woman, Person` correspond to the respective class elements in DAML+OIL ontology example [1].

```
{Man;
(9)
 in: type, daml_oil;
 supertype: Person, Male
};
{Woman;
(10)
 in: type, daml_oil;
 supertype: Person, Female
};
{Person;
(11)
 in: type, daml_oil;
 supertype: R_Person;
 hasSpouse: Person;
    metaslot
       in: HasSpouse;
    {comment; if hasSpouse would be defined as
     in:HasSpouseMarried then this would be a definition
     of the married person concept}
    end
```

```
 hasOccupation: FullTimeOccupation;
    metaslot
       in: HasOccupation
    end
 dunion: {in: invariant, disjointUnionOf
    {{(Person(p/Person) = (Man(p/Person) | Woman(p/Person))) &
      (Man(p/Person) & Woman(p/Person) = {}) }}}
};
```

In this type definition we assume:

```
{HasSpouse;
(12)
 in: metatype, association, daml_oil;
 instance_section:
 {domain: Person;
  range: Person;
  association_type: {{0,1},{0,1}}
     metaslot
        in: onProperty, maxCardinality
     end}
};
{HasSpouseMarried;
(13)
 in: metatype, association, daml_oil;
 superclass: HasSpouse;
 instance_section:
 {domain: Person;
  range: Person;
  association_type: {{1,1},{1,1}}
     metaslot
        in:onProperty, Cardinality
     end}
};
{HasOccupation;
(14)
 in: metatype, association, daml_oil;
 instance_section:
 {domain: Person;
  range: FullTimeOccupation;
  association_type: {{0,1},{0,inf}}
     metaslot
        in: onProperty, Cardinality0
     end}
};
```

`sameClassAs` element asserts that C is equivalent to the class-expression in the element. The resulting invariant (for `equivalentTo` element the invariant is similar) looks as:

```
I: {in: invariant, sameClassAs, {{(C>T1 = CE>T2)}}}
```

`HumanBeing` concept is the same as `Person`:

```
{HumanBeing;
(15)
 in: type, daml_oil;
 supertype: Person;
 sameTypeAs: {in: invariant, sameClassAs
    {{all p/Person (HumanBeing(p) = Person(p)) }}}
};
```

4.2 Property Restrictions Interpretation

A property restriction is a special kind of class expression in DAML+OIL. It implicitly defines an anonymous class, namely the class of all objects that satisfy the restriction. A restriction is mapped into a subtype of a property domain type.

`daml:toClass` element defines the class of all objects for whom the values of property P all belong to the class expression. In other words, it defines the class of object x for which it holds that if the pair (x,y) is an instance of P, then y is an instance of the class-expression CE>T or datatype. The following invariant in a subtype ST of a property domain type is included:

```
I: {in: invariant, onProperty, toClass,
   {{all x (ST(x) & (P(x) ⊆ CE>T))}}}
```

Type `R_Person` is a subtype of `Animal`. Parents of `R_Person` instances should be `Persons`.

```
{R_Person;
(16)
 in: type, daml_oil, restriction;
 supertype: Animal;
 hp: {in: invariant, onProperty, toClass,
    {{all ap (R_Person(ap) &  hasParent(ap) <= Person) }}};
 hasFather: Male;
    metaslot
       in: HasFatherCard
    end
};
```

`daml:hasValue` element defines the class of all objects for whom the property P has at least one value equal to the named object or datatype value (and perhaps other values as well). In other words, if we call the instance y, then it defines the class of objects x for which (x,y) is an instance of P. The following invariant in a subtype ST of a property domain type is included:

```
I: {in: invariant, onProperty, hasValue,
   {{all x (ST(x) & y∈P(x)) }}}
```

daml:hasClass element defines the class of all objects for which at least one value of the property P is a member of the class expression or datatype. In other words, it defines the class of objects x for which there is at least one instance y of the class-expression CE or datatype such that (x,y) is an instance of P. The following invariant in a subtype ST of a property domain type is included:

```
I: {in: invariant, onProperty, hasClass,
   {{all x (ST(x) & ex y (P(x,y) & CE>T(y))) }}}
```

4.3 Cardinality Constraints Interpretation

daml:cardinality element. This defines the class of all objects that have exactly N distinct values for the property P, i.e. x is an instance of the defined class if and only if there are N distinct values y such that (x,y) is an instance of P. For property P an association metatype is defined with an association type specified as:

```
association_type: {{N,N},{?,?}}
```

HasFatherCard (example 6) and HasSpouseMarried (example 13) are of this association type.

daml:maxCardinality element. This defines the class of all objects that have at most N distinct values for the property P. For property P an association metatype is defined with an association type specified as:

```
association_type: {{0,N},{?,?}}
```

HasSpouse (example 12) is of this association type.

daml:minCardinality element. This defines the class of all objects that have at least N distinct values for the property P. For property P an association metatype is defined with an association type specified as:

```
association_type: {{N,inf},{?,?}}
```

daml:cardinalityQ element defines the class C of all objects that have exactly N distinct values for the property P that are instances of the class expression CE or datatype (and possibly other values not belonging to the class expression or datatype). In other words: x is an instance of the defined class (x satisfies the restriction) if and only if there are exactly N distinct values y such that (x,y) is an instance of P and y is an instance of the class expression or datatype. For property P an association metatype is defined with an instance type defined as:

```
{domain: C>T1;
 range: CE>T2;
 association_type: {{N,N},{?,?}};
    metaslot
       in: onProperty, cardinalityQ
    end }
```

This is the same as:

```
I: {in: invariant, onProperty, cardinalityQ,
   {{all x(C>T1(x) & count (P(x) & CE>T2)  = N) }}}
```

`daml:maxCardinalityQ` element. This defines the class C of all objects that have at most N distinct values for the property P that are instances of the class expression CE or datatype (and possibly other values not belonging to the class expression or datatype). For property P an association metatype is defined with an instance type defined as (example 14):

```
{domain: C>T1 ;
 range: CE>T2;
 association_type: {{0,N},{?,?}};
    metaslot
       in: onProperty, CardinalityQ
    end}
```

`daml:minCardinalityQ` element. This defines the class C of all objects that have at least N distinct values for the property P that are instances of the class expression CE or datatype (and possibly other values not belonging to the class expression or datatype). For property P an association metatype is defined with an instance type defined as:

```
{domain: C>T1 ;
 range: CE>T2;
 association_type: {{N,inf},{?,?}};
    metaslot
       in: onProperty, CardinalityQ
    end}
```

We have considered mostly the object world of DAML+OIL so far. Datatype world is treated similarly. XML Schema datatypes are mapped into SYNTHESIS built-in or abstract types. Datatype values get adequate representation in SYNTHESIS. For instance, age attribute of Animal (example 1) is of datatype `integer`. A property restriction based on such type is expressed as a type:

```
{Adult;
(17)
```

```
 in: type, daml_oil, restriction;
 supertype: Person;
 adultage : {in: invariant, onProperty, hasClass,
    {{all p (Adult(p) & age(p) > 17) }}}
};
```

5 Reverse Mapping of the MOL obtained into DAML+OIL

The intention of this section is to show that the extended MOL specifications contain enough information to map them into DAML+OIL definitions preserving original meaning. General rules for such reverse mapping are the following:

- MOL type that does not belong to restriction metatype is mapped into DAML+OIL class (subclass). Type invariants belonging to `disjointWith, disjointUnionOf, sameClassAs` metatypes are mapped into the respective elements of the resulting class definition.
- MOL type attributes together with the semantics introduced by the related association metatypes are mapped into DAML+OIL properties (subproperties).
- MOL type that belongs to restriction metatype is mapped into DAML+OIL restriction. In such cases invariants of MOL type belonging to metatypes `toClass, hasValue, hasClass` are mapped into the respective elements of the resulting restriction.
- MOL type attributes P of a type C>T defined in an association metatype having association types related to `onProperty` metatype are mapped into `cardinality` or `cardinalityQ` property restriction of a class C.

Due to the fact that element definitions in DAML+OIL have a form of various clichés in RDF, to generate a DAML+OIL definition for the cases 1 – 4 above it is required to extract from the MOL specifications the required parameters and to insert them into the appropriate positions of the respective DAML+OIL cliché. This process is illustrated with the following examples:

Case 1. For `Female` type (example 3) the following class definition will be generated:

```
<daml:Class rdf:ID="Female">
   <rdfs:subClassOf rdf:resource="#Animal"/>
   <daml:disjointWith rdf:resource="#Male"/>
</daml:Class>
```

Case 2. For `hasFather` attribute of `Animal` type (example 1) the following property definition will be generated:

```
<daml:ObjectProperty rdf:ID="hasFather">
   <rdfs:subPropertyOf rdf:resource="#hasParent"/>
   <rdfs:range rdf:resource="#Male"/>
</daml:ObjectProperty>
```

Case 3. For `R_Person` type (example 16) the following restrictions will be generated (`toClass` and `cardinality` restrictions, the latter is related to the Case 4):

```
<daml:Class rdf:about="#Person">
   <rdfs:subClassOf>
      <daml:Restriction>
         <daml:onProperty rdf:resource="#hasParent"/>
         <daml:toClass rdf:resource="#Person"/>
      </daml:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
      <daml:Restriction daml:cardinality="1">
         <daml:onProperty rdf:resource="#hasFather"/>
      </daml:Restriction>
   </rdfs:subClassOf>
</daml:Class>
```

Case 4. For `hasOccupation` attribute of `Person` type (example 11) specifying `cardinalityQ` restriction the following definition will be generated:

```
<daml:Class rdf:about="#Person">
   <rdfs:subClassOf>
      <daml:Restriction daml:maxCardinalityQ="1">
         <daml:onProperty rdf:resource="#hasOccupation"/>
         <daml:hasClassQ rdf:resource="#FullTimeOccupation"/>
      </daml:Restriction>
   </rdfs:subClassOf>
</daml:Class>
```

Difficulties arise with reversible mapping of MOL constants to XML Schema datatype instances. DAML+OIL requires introduction of constants and types of the following syntax (a type like that is to be produced for example 17):

```
rdf:resource=http://www.w3.org/TR/2001/NOTE-daml+oil-walkthru-
20011218/daml+oil-ex-dt#over17
```

Such definitions cannot be mapped reversibly without providing of a specific normalized procedure of forming such specifications.

6 Related work

Ontolingua [12] provides maintaining of ontologies with multiple representation languages to port from system to system. In contrast, the proposed approach is focused on compositions of ontologies with multiple representation languages in a mediator context.

[9] presents an approach for development of *commutative* data model mappings in process of design of heterogeneous database integration systems. The method introduced provides for constructing canonical data model kernel extensions equivalent to various source data models. The method is based on the data model axiomatic extension principle. The canonical data model in the integrating system should be extensible while new source data models are integrated. Such extension is implemented axiomatically - by adding to the data definition language of a set of axioms determining (in terms of the canonical model) of logical dependencies of the source data model. The result of the extension is proved to be equivalent to the source data model.

Similarly, reversible ontological model mapping technique proposed here applies the axiomatic extension principle to the ontologies (Section 3).

Reversible data model mappings methods are considered in [14] focusing on the need for formal interpretation in which it becomes possible to transform databases between different data models.

7 Conclusion

The paper discusses ontological modeling framework issues in the mediation environment. For different information sources different ontological models (languages) can be used to define their ontologies. Different ontological models can be required for appropriate modeling of different subject domains. How the mediator ontological language (MOL) should be positioned w.r.t. such modeling variety is analyzed in the paper.

It is assumed that the core of MOL is defined as a subset of the mediator canonical model. Based on such core, one and the same procedure is required to design MOL for a specific domain as well as to map a source ontological model into MOL. This procedure consists in development of the extension of the MOL core equivalent to a given ontological model.

An approach for such reversible mapping construction is demonstrated for a class of the Web information sources. It is assumed that such sources apply the DAML+OIL ontological model. A subset of the hybrid object-oriented and semi-structured canonical mediator data model is used for the core of MOL. Construction of a reversible mapping of DAML+OIL into an extension of the core of MOL is presented in the paper. Such mapping is a necessary prerequisite for contextualizing and registration of information sources at the mediator. The mapping shows how a problem of reconciling of non-compatible ontology modeling languages of the mediator and of heterogeneous information sources can be solved.

The approach proposed is applicable to heterogeneous ontological models integration in various subject domains. Such integration may be required in digital libraries where retrieval is based on the information content, rather than

on information entities, or where an interoperability of metadata registries is required to cross the boundaries between different information contexts, etc.

[1] *Annotated DAML+OIL Ontology Markup.* http://www.w3.org/TR/daml+oil-walkthru/

[2] Briukhov, D.O., Kalinichenko, L.A. Component-based information systems development tool supporting the SYNTHESIS design method. In *Proceedings of the East European Symposium on Advances in Databases and Information Systems (ADBIS'98)*, Springer, LNCS No. 1475, 1998.

[3] Briukhov, D.O., Kalinichenko, L.A., Skvortsov, N.A. Information sources registration at a subject mediator as compositional development. In *Proceedings of the Fifth East European Symposium on Advances in Databases and Information Systems (ADBIS'01)*, Springer-Verlag, 2001, pp. 70-83.

[4] *DAML+OIL (March 2001) reference description.* http://www.w3.org/TR/daml+oil-reference

[5] Friedman, M., Levy, A. and Millstein, T. Navigational Plans for Data Integration, In *Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, Orlando, Florida, 1999.

[6] Kalinichenko, L. A. SYNTHESIS: the language for description, design and programming of the heterogeneous interoperable information resource environment. *Institute for Problems of informatics, Russian Academy of Sciences, Moscow*, 1995.

[7] Kalinichenko, L.A., Briukhov D.O., Tyurin I.N., Skvortsov N.A. Intermediator framework protocol for information sources registration at heterogeneous mediators In *Proceedings of the DELOS Workshop on Interoperability in Digital Libraries*, September 8-9, 2001, GMD-IPSI, Darmstadt, Germany

[8] Kalinichenko, L. A. Integration of Heterogeneous Semistructured Data Models in the Canonical One. In *Proceedings of the First Russian Conference on Digital Libraries*, St. Petersburg, 1999.

[9] Kalinichenko L.A. Methods and tools for equivalent data model mapping construction In *Proceedings of the International Conference on Extending Database Technology (EDBT'90)*, Venice, 1990

[10] G. Salton, C. Backley. Term-Weighting Approaches in Automatic Text Retrieval. *Readings in Information Retrieval under edition of K. S. Jones and P. Willett, Morgan Kaufmann Publishers Inc.*, 1997

[11] The Open Archives Initiative Protocol for Metadata Harvesting Protocol Version 1.0 of 2001-01-21,DocumentVersion2001-04-24, http://www.openarchives.org/OAI/openarchivesprotocol.htm

[12] Ontolingua. http://ontolingua.stanford.edu

[13] N. A. Skvortsov, L. A. Kalinichenko. An Approach to Ontological Modeling and Establishing Intercontext Correlation in the Semistructured Envi-

ronment. In *Proceedings of the Second Russian Conference on Digital Libraries*, Protvino, Sep 26-28, 2000

[14] De Troyer O.M.F. On data schema transformation. *Ph. D. Thesis*, Brabant Univ., 1993

Appendix

A correspondence between DAML+OIL and MOL constructs is summarized in the following table. To show DAML+OIL element semantics, equivalent Description Logic (DL) encoding is represented.

DAML+OIL Element (and DL expression)	**DAML+OIL RDF Encoding**	**MOL specification**
Nothing ($\bot$)	<daml:Nothing/>	Tnone
Thing ($\top$)	<daml:Thing/>	Taval
class (C)	<daml:Class rdf:ID="C"/>	{C; in: type }
individual (I)	<daml:Thing rdf:ID="I"/>	{I; in: frame }
property (P)	<daml:ObjectProperty rdf:ID="P"/>	{P; in: association, metatype, daml_oil; }
domain	<rdfs:domain rdf:resource = "#C"/>	domain: C;
range	<rdfs:range rdf:resource = "#C"/>	range: C;
intersectionOf ($C_1 \sqcap C_2$)	<daml:intersectionOf rdf:parseType = "daml:collection"> <daml:Class rdf:about="#C1"/> <daml:Class rdf:about="#C2"/> </daml:intersectionOf>	I: {in:predicate, invariant, intersectionOf {{C3(a/Taval) = C1(a/Taval) & C2(a/Taval)}} }
unionOf ($C_1 \sqcup C_2$)	<daml:unionOf rdf:parseType= "daml:collection"> <daml:Class rdf:about="#C1"/> <daml:Class rdf:about="#C2"/> </daml:intersectionOf>	I: {in:predicate, invariant, unionOf {{C3(p/C3) = C1(p/C3) | C2(p/C3)}} }
complementOf ($\neg C$)	<daml:complementOf rdf:resource="#C"/>	I: {in:predicate, invariant, ComplementOf {{C1(p/C1)=^C(p/C)}} }
oneOf ($\{x_1 \ldots x_n\}$)	<daml:oneOf parseType = "daml:collection"> <daml:Thing rdf:about = "#x1"/> <daml:Thing rdf:about = "#xn"/> </oneOf>	{enum; enum_list: x1,… xn}
toClass ($\forall R.C$)	<daml:Restriction> <daml:onProperty rdf:resource="#R"/> <daml:toClass rdf:resource="#C"/> </daml:Restriction>	I: {in: invariant, onProperty, toClass, {{all x (C1(x) & (R(x,y) $\subseteq$ C(y)))}}}
hasClass ($\exists R.C$)	<daml:Restriction> <daml:onProperty rdf:resource="#R"/> <daml:hasClass rdf:resource="#C"/> </daml:Restriction>	I: {in: invariant, onProperty, hasClass, {{all x (C1(x) & ex y (P(x,y) & C(y)))}}}
hasValue ($\exists R.\{x\}$)	<daml:Restriction> <daml:onProperty rdf:resource="#R"/> <daml:hasValue rdf:resource="#x"/> </daml:Restriction>	I: {in: invariant, onProperty, hasValue, {{all y (C1(y) & x$\in$R(y))}} }
minCardinalityQ ($\geq_n R.C$)	<daml:Restriction daml:minCardinalityQ="n"> <daml:onProperty rdf:resource="#R"/> <daml:hasClassQ rdf:resource="#C"/> </daml:Restriction>	{domain: C1; range: C; association_type: {{n,inf},{?,?}}; metaslot in: onProperty, minCardinalityQ end}
minCardinality ($\geq_n R$)	<daml:Restriction> <daml:onProperty rdf:resource="#R"/> <daml:minCardinality>n </daml:minCardinality> </daml:Restriction>	association_type: {{n,inf},{?,?}}

maxCardinalityQ ($\leq_n R.C$)	<daml:Restriction daml:maxCardinalityQ="n"> <daml:onProperty rdf:resource="#R"/> <daml:hasClassQ rdf:resource="#C"/> </daml:Restriction>	{domain: C1; range: C; association_type: {{0,n},{?,?}}; metaslot in: onProperty, maxCardinalityQ end }
maxCardinality ($\leq_n R$)	<daml:Restriction> <daml:onProperty rdf:resource="#R"/> <daml:minCardinality>n </daml:minCardinality> </daml:Restriction>	association_type: {{0,n},{?,?}}
cardinalityQ ($=_n R.C$)	<daml:Restriction daml:CardinalityQ="n"> <daml:onProperty rdf:resource="#R"/> <daml:hasClassQ rdf:resource="#C"/> </daml:Restriction>	{domain: C1; range: C; association_type: {{n,n},{0,inf}} metaslot in: onProperty, CardinalityQ end}
cardinality ($=_n R$)	<daml:Restriction daml:cardinality="n"> <daml:onProperty rdf:resource="#R"/> </daml:Restriction>	association_type: {{n,n},{?,?}}
subClassOf ($C_1 \sqsubseteq C_2$)	<rdfs:subClassOf rdf:resource="#C2"/>	{C1; in: type; supertype: C2}
sameClassAs ($C_1 \equiv C_2$)	<daml:sameClassAs rdf:resource="#C2"/>	I: {in: invariant, sameClassAs, {{(C1(x/C1) = C2(x)}}}
subPropertyOf ($P_1 \sqsubseteq P_2$)	<rdfs:subPropertyOf rdf:resource="#P2"/>	{P1; in: association, metatype; superclass: P2 }
samePropertyAs ($P_1 \equiv P_2$)	<daml:samePropertyAs rdf:resource="#P2"/>	I: {in: invariant, samePropertyAs, {{(P1(x) = P2(x))}}}
disjointWith ($C_1 \sqsubseteq \neg C_2$)	<daml:disjointWith rdf:resource="#C2"/>	disjoint: {in:predicate, invariant, disjointWith, {{C1(a/C3) & C2(a/C3) = {} }} }
inverseOf ($P_1 \equiv P_2^-$)	<daml:inverseOf rdf:resource="#P2"/>	inverse: P2;
transitiveProperty ($P^+ \sqsubseteq P$)	<daml:TransitiveProperty rdf:ID="P"/>	metaslot in: onProperty, transitiveProperty end
uniqueProperty ($\top \sqsubseteq \leq_1 P$)	<daml:UniqueProperty rdf:ID="P">	{association_type: {{1,1},{0,inf}}; metaslot in: onProperty, cardinality, UniqueProperty end}}
unambiguousProperty ($\top \sqsubseteq \leq_1 P^-$)	<daml: UnambiguousProperty rdf:ID="P">	metaslot in: onProperty, unambigousProperty end
disjointUnionOf ($C_1 \sqcup C_2$, $\bot \equiv C_1 \sqcap C_2$)	<daml:disjointUnionOf rdf:parseType="daml:collection"> <daml:Class rdf:about="#C1"/> <daml:Class rdf:about="#C2"/> </daml:disjointUnionOf>	dunion: {in:predicate, invariant, disjointUnio- nOf, {{(C3(x/C3) = (C1(x/C3)|C2(x/C3))) & (C1(x/C3)&C2(x/C3) = {}) }}}