Formalization of the CRM: Initial Thoughts

Carlo Meghini

Istituto di Scienza e Tecnologie della Informazione
Consiglio Nazionale delle Ricerche – Pisa

CRM SIG Meeting
Iraklio, October 1st, 2014

Outline

- Overture: why, how, what
- Andante Maestoso: a first-order theory of the CRM
- Adagio: post-reflections
- Allegretto: an OWL ontology of the CRM
- Allegro con spirito: conclusions

Overture: Why doing it?

Communication to other researchers

- Understanding
- Computational Research
- Extension
- Experimentation

Methodological Foundations

- Soundness
- Solidness

Inference

- Querying

How to do it?

Choose your favourite theoretical tool:

- Mathematical Logic
- Computational Logic
- Set Theory
- Category Theory
- *you name it*

My choice: mathematical logic, because I know it (a bit)

But the are also advantages:

- discourse formalization (syntax and semantics)
- argument formalization (proof)
- many results to use (*e.g.*, description logics)

What did I do?

1. First-order logic translation of the specs
   - ...so that I understand what we are talking about
2. OWL expression of the resulting theory
   - ...so that we can use some tools:
     - the Protégé ontology editor
     - the Virtuoso triple store
     - and maybe more

Some things remain to be done.

Andante Maestoso: First-order logic translation

Translation of class specifications

CRM Specification	*Translation into first-order logic*
Class E*n c-name*	Unary Predicate *En*
A Superclass of B	$B(x) \supset A(x)$
A Subclass of B	$A(x) \supset B(x)$
A Disjoint from B	$A(x) \supset \neg B(x)$

So we have the following two axioms in $\mathcal{CRM}$:

$$E2(x) \supset \neg E77(x)$$
$$E18(x) \supset \neg E28(x)$$

Since disjointness axioms propagate down the IsA hierarchy, these two axioms sanction the disjointness of many classes.

Translation of property specifications

CRM Specification	*Translation into first-order logic*
Property P*n* *p-name*	Binary Predicate *Pn*
P has Domain C	$P(x, y) \supset C(x)$
P has Range D	$P(x, y) \supset D(y)$
P is a Superproperty of Q	$Q(x, y) \supset P(x, y)$
P is a Subproperty of Q	$P(x, y) \supset Q(x, y)$

Meta-Properties

CRM Specification	*Translation into first-order logic*
Property P*n* *p-name*	Binary Predicate *Pn*
P has Meta-Property P.*n*: C	$P.n(x, y, z) \supset [P(x, y) \wedge C(z)]$
P is Symmetric	$P(x, y) \supset P(y, x)$
P has Asymmetric Meta-Prop. P.*n*: C	$P.n(x, y, z) \supset [P(x, y) \wedge \neg P.n(y, x, z) \wedge C(z)]$

A meta-property is a property whose domain is a property.

Meta-properties are modelled as 3-place predicate symbols:

- the first two places are given to the terms in the domain property,
- the last place is used for the type.

The corresponding axiom includes the assertion of the domain property in the consequent, thus making it possible to omit it whenever a typing statment is present.

Shortcuts

CRM Specification	*Translation into first-order logic*
Property P*n* *p-name*	Binary Predicate *Pn*
Weak Shortcut $P_1 \ldots P_n$	$[P_1(x, z_1) \wedge P_2(z_1, z_2) \wedge \ldots \wedge P_n(z_n, y)] \supset P(x, y)$
Strong Shortcut $P_1 \ldots P_n$	$P(x, y) \equiv \exists z_1 \ldots z_n [P_1(x, z_1) \wedge \ldots \wedge P_n(z_n, y)]$

Shortcuts come in two sorts:

- *weak* shortcuts, interpreted as implications:

$$[P24(x, y) \wedge (P23(y, z) \vee P22(y, z))] \supset P51(x, z)$$

- *strong* shortcut, interpreted as abbreviations:

$$P2(x, y) \equiv \exists z_1 [P41(x, z_1) \wedge P42(z_1, y)]$$

Notice that weak shortcuts do not require existential variables, due to the semantics of conditionals.

Property quantification

The definition of quantifiers is given in terms of two features:

- *total property* and
- *functional property*

that can be applied to a property or to its inverse. Therefore a property or its inverse fall exactly into one of the following cases:

1. total and not functional, *i.e.*, defined on every element of its domain and can take up more than one value;
2. functional and not total, *i.e.*, at most one value is provided for any element of its domain;
3. the property is neither total, *i.e.*, some domain elements can miss it, nor functional, *i.e.*, can take up more than one value for any element of its domain;
4. both total and functional, *i.e.*all domain element must have one value for it, and no more than one.

Translation:

- P is functional: $[P(x,y) \wedge P(x,y')] \supset (y = y')$
- P is total (on domain A): $A(x) \supset \exists y P(x,y)$
- the inverse of P is functional: $[P(x,y) \wedge P(x',y)] \supset (x = x')$
- the inverse of P is total (having range A): $A(x) \supset \exists y P(y,x)$

The complete translation of each quantifier can be obtained by conjoining the translation of the corresponding features. For instance:

- *many to many* (0,n:0,n): P and its inverse are neither total nor functional: no axiom
- *one to one* (1,1:1,1): P and its inverse are total and functional:

$$A(x) \supset \exists y P(x,y)$$
$$[P(x,y) \wedge P(x,y')] \supset (y = y')$$
$$B(x) \supset \exists y P(y,x)$$
$$[P(x,y) \wedge P(x',y)] \supset (x = x')$$

Adagio

By applying the rules above to the specification of classes and properties, we obtain a set of axioms that make up the $\mathcal{CRM}$ first-order theory.

Some pleasant consequences (based on standard logical notions):

- we can talk of the $\mathcal{CRM}$ language, as the set of predicate symbols that occur in the axioms
- we can talk of a $\mathcal{CRM}$ knowledge base (KB) as a set of sentences of the $\mathcal{CRM}$ language plus the axioms
- we can talk about a model of a KB, as any interpretation of the language that satisfies all the axioms and the sentences in the KB
- we can talk about the *consistency* of a KB
- we can talk about *reasoning* in CRM because we have an inference relation $KB \models \alpha$

We can define formally the *interaction* with a KB, thereby separating *once for all* the theory from its implementation:

- $\textsc{Tell}$(KB,s), where s is a sentence of our language
- we have a query language: the set of open formulas of the language
  - *e.g.*, $E55(x) \wedge \exists y[P27(x,y) \vee \neg P72(x,y)]$
- $\textsc{Ask}$(KB,α), wehre α is a query
- we can use the inference relation to define the answer to a query

We can *validate* the CRM:

- we can *prove* that the $\mathcal{CRM}$ axioms are consistent (hopefully :-))
- as well as any other property we think it's there

We can *defend* the CRM:

- we can challenge the CRM's detractors to prove what they say

We can formally test whether:

- a language is equivalent to, or less/more powerful than the CRM
- an implementation is sound and complete with respect to the $\mathcal{CRM}$

Nothing particularly surprising, but a firm ground to start building.

Allegretto: The OWL ontology CRM

Encoding the $\mathcal{CRM}$ in OWL, as a step towards the implementation of the CRM.

Not the only way of implementing the CRM, but it allows us to use *at no cost*:

- implementations of OWL to manage the creation and the evolution of KBs (*e.g.*, TELL)
- the SPARQL query language to extract information from instances of the CRM (*e.g.*, ASK)
- the Protégé ontology editor for fast prototyping of extensions.

it presents at least two advantages:

The CRM ontology is given in the functional notation.

Translation of class specifications

CRM Specification	*OWL Specification*
Class E*n c-name*	`Declaration(Class(crm:En))`
A Superclass of B	`SubClassOf(B A)`
A Subclass of B	`SubClassOf(A B)`

OWL comes with built-in classes whose intended meaning overlaps with the intended meaning of some CRM classes. So we have some *class mapping axioms.*

```
SubClassOf(crm:E1 owl:Thing)
SubClassOf(crm:E41 rdf:Literal)
SubClassOf(crm:E59 rdf:Literal)
SubClassOf(owl:real crm:E60)
```

What about using XML Schema datatypes (*e.g.*, `xsd:date`) for some CRM classes (eg, E50 Date)?

Class disjointness axioms:

- `DisjointClasses(crm:E2 crm:E77)`
- `DisjointClasses(crm:E18 crm:E28)`

Translation of property specifications

OWL distinguishes between two kinds of properties:

- *object properties,* connecting two individuals, and
- *data properties,* connecting an individual and a literal.

Therefore it must be determined, for each CRM property, whether it is encoded as an OWL object or data property.

Since E41 Appellation and E59 Primitive Value are sub-classes of `rdf:Literals`, all properties having either `crm:E41`, or `crm:E59`, or a subclass of theirs, are translated as OWL *data* properties, and all remaining properties are translated as OWL *object* properties.

Is this correct?

The easy part:

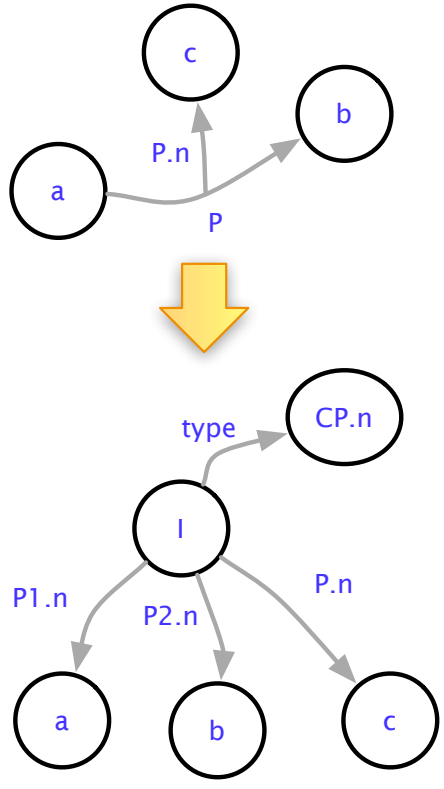
CRM Specification	*Translation into OWL*
Property P*n p-name*	`Declaration(DataProperty(crm:Pn))` or `Declaration(ObjectProperty(crm:Pn))`
P has Domain C	`ObjectPropertyDomain(crm:Pn crm:C)`
P has Range D	`ObjectPropertyRange(crm:Pn crm:D)`
P is a Superproperty of Q	`SubObjectPropertyOf(crm:Q crm:Pn)`
P is a Subproperty of Q	`SubObjectPropertyOf(crm:Pn crm:Q)`
P is Symmetric	`SymmetricObjectProperty(crm:Pn)` *
P is Transitive	`TransitiveObjectProperty(crm:Pn)`

Meta-properties

A meta-property $P.n$ of a property P associates an instance (a, b) of P with a value c in the meta-property range.

It can be modeled by a class `CP.n` each instance of which stands for an instance of P and is connected to the individual c via meta-property `P.n`.

```
Declaration(Class(crm:CP.n))
Declaration(ObjectProperty(crm:P1.n))
Declaration(ObjectProperty(crm:P2.n))
Declaration(ObjectProperty(crm:P.n))
ClassAssertion(crm:CP.n I)
ObjectPropertyAssertion(crm:P1.n I a)
ObjectPropertyAssertion(crm:P2.n I b)
ObjectPropertyAssertion(crm:P.n I c)
ObjectPropertyAssertion(crm:P a b)
```



If the meta-property is asymmetric, then add:

```
NegativeObjectPropertyAssertion(crm:P.n c I)
```

The assumption here is that the same anonymous individual `I` is *always* used to reify the instance (a, b) of P.

Bad news: all the above assertions cannot be obtained as implicit knowledge via axioms, thus their insertion in the ontology has to be performed procedurally.

Property quantification

The specification of the CRM describe properties, or their inverses, as functional or total (neither or both).

```
FunctionalObjectProperty(crm:P) or FunctionalDataProperty(crm:P)
SubClassOf(crm:A ObjectSomeValuesFrom(crm:P owl:Thing))
InverseFunctionalObjectProperty(crm:P)
SubClassOf(crm:A ObjectSomeValuesFrom(ObjectInverseOf(crm:P) owl:Thing))
```

Shortcuts

P Weak Shortcut $P_1 \ldots P_n$ is captured by:

```
SubObjectPropertyOf(ObjectPropertyChain(crm:P1 ...  crm:Pn) crm:Pn)
```

Strong shortcuts are equivalence statements consisting of an *if* and an *only-if* part. For example:

P2 has type: P41 classified, P42 assigned

The *only-if* part of the equivalence is a weak shortcut:

```
SubObjectPropertyOf(ObjectPropertyChain(crm:P41 crm:P42) crm:P2)
```

For each instance (a, b) of the property expressed via the assertion:

```
ObjectPropertyAssertion(crm:a P2 crm:b)
```

the *if* part of the equivalence implies the following assertions:

```
ObjectPropertyAssertion(crm:a P41 _:a1)
ObjectPropertyAssertion(_:a1 P42 crm:b)
```

where `_:a1` is an anonymous individual.

Bad news: While these assertions can be expressed on an individual basis, there is no way of obtaining them as implicit knowledge via some axioms.

Allegro con Spirito: Conclusions

`CRM` is an OWL ontology including the axioms that capture the semantics of the CRM vocabulary:

- *class axioms*
- *class mapping axioms*
- *class disjointness axioms*
- *property axioms*

The full expression of the CRM is not possible in OWL, since neither strong shortcuts nor meta-properties are directly expressible.

The problem is not severe for meta-properties because meta-property instances never result as implicit knowledge, they always result from manual insertion.

The problem is much more severe for shortcuts, because *in principle* a shortcut property instance may result as implicit knowledge, therefore the insertion of the corresponding assertions is considerably more difficult.

Next steps

But this is only half of the story.

The other half is *translating* and *computing answers* to queries.

Would this work:

$$\text{OWL} \xrightarrow{\text{encode}} \text{RDF} \xrightarrow{\text{query}} \text{SPARQL}$$

?

Provided it does, queries stated against `CRM` will have smaller answers than the equivalent queries stated against $\mathcal{CRM}$.

But these would be canonical queries. What about *user-friendly* queries?

Narratives?