



A formal, scalable approach to semantic interoperability

Jim Davies^{a,b,*}, James Welch^b, David Milward^a, Steve Harris^b

^a Oxford University, Department of Computer Science, Oxford OX1 3QD, UK

^b Big Data Institute, Oxford OX3 7LF, UK



ARTICLE INFO

Article history:

Received 16 July 2019

Received in revised form 1 February 2020

Accepted 11 February 2020

Available online 14 February 2020

Keywords:

Semantic interoperability

Refinement

Formal methods

ABSTRACT

Scientific progress is increasingly dependent upon the acquisition, processing, and analysis of large volumes of data. The validity of results and the safety of applications rely upon an adequate understanding of the real-world semantics of this data: its intended interpretation, and the context in which it is acquired and processed. This presents a challenge: interpretations vary, context is infinite, and either may change over time.

This paper addresses that challenge. It introduces a language for the description of real-world semantics that allows for multiple, evolving interpretations, together with a high degree of automation in the capture and creation of contextual metadata. The language itself has a mathematical semantics, and supports a notion of semantic interoperability closely related to existing, formal notions of refinement.

The language represents a scalable approach in three respects: it is compositional, in terms of composing real-world semantics piece by piece; it allows for multiple perspectives, allowing the parallel development of different interpretations; and it supports automatic transformations to and from implementation languages. The practical application of the approach is illustrated with examples from large-scale medical research.

© 2020 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Semantic interoperability is a key consideration in information systems design. Whenever data produced in one system is used in another, it is important that this usage is consistent with the meaning, or semantics, of the data. If this is not the case, then—as far as this data, and this usage is concerned—the two systems are not semantically interoperable, and may fail to work effectively together.

A simple but dramatic example is afforded by the case of the Mars Climate Orbiter [17]. Here, the system responsible for trajectory calculations delivered numeric ‘thruster performance data’ in units of pound-seconds (lbf-s). The navigation software, receiving the data, expected numeric data in units of Newton-seconds. This inconsistency in data semantics led to the destruction of the \$125m spacecraft.

Other questions of semantic interoperability require more than a careful consideration of units. A survey of water-quality data collected in the United States [22] found that the majority of the available data lacked information regarding the methodology employed, the chemical forms considered, and even the locations in which the samples were taken.

* Corresponding author at: Oxford University Department of Computer Science, Oxford OX1 3QD UK.

E-mail addresses: Jim.Davies@cs.ox.ac.uk (J. Davies), James.Welch@bdi.ox.ac.uk (J. Welch), dmmilward@gmail.com (D. Milward), Steve.Harris@bdi.ox.ac.uk (S. Harris).

<https://doi.org/10.1016/j.scico.2020.102426>

0167-6423/© 2020 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

The cost of collection of this ambiguous data—14.5 m of the 25 m records surveyed—is estimated at \$12b. It cannot be combined or analysed without additional effort to obtain the missing information, and hence ensure that this usage is consistent with the original meaning. In some cases, the information needed will be too difficult to obtain, and the data will remain unused.

As our society becomes increasing reliant upon ‘data-driven’ approaches to the delivery of services in business, in government, and in healthcare, the importance of achieving semantic interoperability, at scale, should be clear enough. This raises two questions: how do we achieve semantic interoperability at scale? and how can we be sure that we have done so?

This paper sets out to answer these questions, presenting an approach to interoperability that is scalable and formal. Scalability is achieved through automation and composition: the information we need is represented as computable metadata; this metadata may be automatically processed; it may also be composed, extended, and re-used. Formality comes from a sound mathematical basis: a foundation for guarantees of correctness and consistency.

At the heart of this approach is a metadata language: a language for describing data. The descriptions of data will include structured, computable information about the context in which the data is collected, processed, or used, through the representation of the software artefacts involved: forms, schemas, databases, workflows, or programs.

The descriptions will include also textual explanations: instructions to the person collecting the data, relevant information about the context, and accounts of how the data is to be interpreted. These explanations need to be managed in a consistent fashion, combined and presented in the right context. Furthermore, where these explanations are used to support arguments of interoperability or consistency, we can capture the resulting assertions in computable form.

The paper begins with a review of the mathematical notation required. This is followed by an introduction to the language and a presentation of its formal semantics, in Section 3 and Section 4, respectively. The formal basis of our approach is set out in Section 5. In Section 6, we explain the tool support that has been developed, and how these tools have been employed in a particular application domain—that of health research informatics. The paper ends with a discussion of related work.

2. Notation

The mathematical notation employed in this paper is the formal, set-theoretic notation \mathbb{Z} [25]. This notation has been applied in the design and development of large software systems [7,11,12]. It is particularly suitable for the definition of formal, denotational semantics: it allows more concise descriptions, through the naming and re-use of patterns of declaration and constraint, and it is supported by a range of open-source tools.

In the \mathbb{Z} notation, we write $[S, T]$ to introduce S and T as basic sets or given types of symbols. We write $\mathbb{P}S$ to denote the set of all subsets of a set S , and we write $S \leftrightarrow T$ and $S \rightarrow T$ to denote the set of all binary relations between S and T , and the set of all partial functions from S to T , respectively.

If r is a function or relation, we write: $\text{dom } r$ to denote the domain of r , as the set of all elements a in s for which there is at least one element b in t such that the pair (a, b) appears in r ; $\text{ran } r$ to denote the range of r , as the corresponding subset of T ; and r^\sim to denote the inverse of r .

In addition, if A is a subset of S and B is a subset of T , then we write: $r \ll A \gg$ to denote the relational image of A under r , being the set of all elements t of B for which there is at least one element a in S such that the pair (a, b) appears in r ; and $r \triangleright B$ to denote the subset of r in which the second component of each pair is an element of B .

Sets may be written in extension using curly braces— $\{$ and $\}$ —or as set comprehensions in the format $\{ \text{decl} \mid \text{cons} \bullet \text{term} \}$, where decl introduces a set of variables ranging over sets identified in decl or already in scope, satisfying the logical constraint cons , to produce a set composed of the possible values of expression term .

For example, if $S = \{1, 2, 3\}$ and $T = \{3, 4\}$, then

$$\{x : S; y : T \mid x < y \bullet (x, y)\}$$

produces the set $\{(1, 3), (1, 4), (2, 3), (2, 4), (3, 4)\}$ containing all pairs of numbers such that: the first component comes from the set $\{1, 2, 3\}$; the second component comes from the set $\{3, 4\}$; and the value of the first component is strictly less than that of the second.

Furthermore, if we define relation r such that

$$r = \{x : S; y : T \mid x < y \bullet (x, y)\}$$

then $r \ll \{2, 3\} \gg = r \ll \{2\} \gg = \{3, 4\}$ and $r \triangleright \{3\} = \{(1, 3), (2, 3)\}$. To simplify the presentation of sets of pairs, we write $a \mapsto b$ to denote the pair (a, b) .

Sequences may be written in extension using angle brackets— \langle and \rangle . If s and t are sequences, then we write: heads to denote the first element of s ; lasts to denote the last element of s ; s_n to denote the n th element of s , for any number n between 1 and the length of s ; and $s \frown t$ to denote the concatenation of the two sequences.

The declaration $a : S$ introduces a variable a as an element of the set S . A declaration may appear as part of a universal (\forall) or existential (\exists) quantification. For example, we may assert that every element a of S satisfies the property $a < 4$ by writing $\forall a : S \bullet a < 4$. A global constant c , drawn from set S , with property p , may be declared as

$$\frac{c : S}{p}$$

A free type U , akin to a type union, containing images of sets S and T under injective (1 to 1) functions f and g , may be declared as

$$U ::= f\langle\langle S \rangle\rangle \mid g\langle\langle T \rangle\rangle$$

A schema, as a pattern of declaration and constraint, may be named in 'vertical' form using the following syntax:

$$\frac{\text{Name} \quad \text{declaration}}{\text{constraint}}$$

or in 'horizontal' form as

$$\text{Name} \hat{=} [\text{declaration} \mid \text{constraint}]$$

Once a schema name has been introduced, it may be used wherever a declaration, constraint, or set-valued expression is expected.

We will include a schema name SA within the declaration part of another schema SB to introduce the variables declared within SA , under the constraint of SA . We will use this approach to build up a complex declaration, bringing new functions or properties into scope step-by-step, one schema at a time.

We may write SC to denote the set of all objects of 'schema type SC '. Each of these objects—often called 'bindings'—has a component or property for each of the variables introduced in the declaration part of the schema. For each object, the combination of values for these components satisfies the constraint part of the schema. If o is an object of schema type SC , and variable c is introduced in the declaration part of that schema, then we may write $o.c$ to represent the value of component c in object o .

3. A data modelling language

In this section, we will describe a language of data models and semantic relationships. The data models—corresponding to software artefacts or data standards—allow us to express the structural and contextual relationships between data points, based upon an existing or intended implementation, whether this is a database, a data schema, or a more abstract data standard.

Our notion of model should be familiar to anyone who has worked with object-oriented models or languages: our models are composed of classes; each class has a set of properties or attributes; attributes may correspond to local variables, or they may be references to other classes. Our models may introduce also enumerated sets of possible values, and make use of shared terminologies and data types.

Our notion of semantic relationships reflects our intention to use linked collections of data models to record or determine that data collected in one context can be safely re-used in another context. For this to be the case, the defining properties of data in the first context must be enough to guarantee the properties of data in the second context.

In our language, we may introduce a link between definitions to indicate that one definition *refines* another: it says everything that the other definition says, and possibly more. So that we may make this assertion from the perspective of either definition, we will introduce also the opposite notion: one definition *abstracts* another if and only if that other definition refines it.

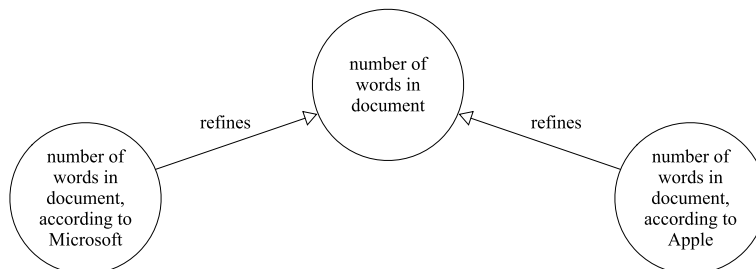


Fig. 1. Two refinements of a definition.

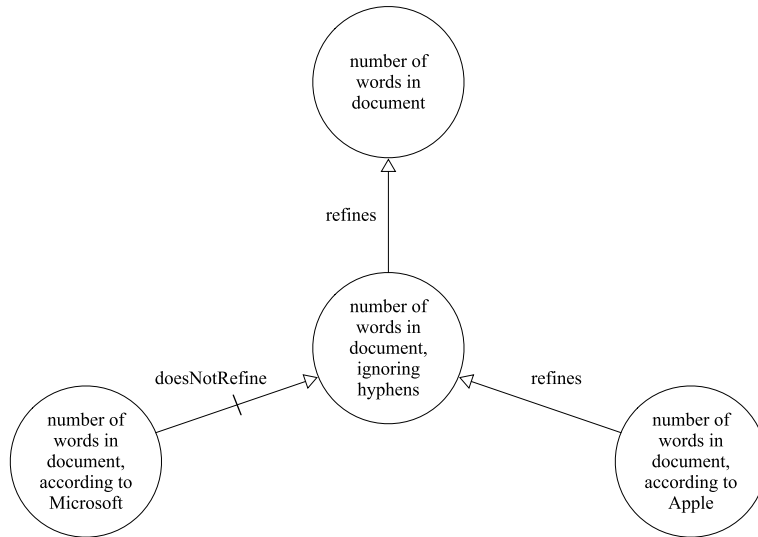


Fig. 2. A definition that is not a refinement.

As an example, consider the three definitions given in Fig. 1. Here, we are assuming that the name of the attribute—e.g. ‘word count’—and the datatype—e.g. positive integer—is the same in each case, and we are focusing simply upon the explanatory text. The two refinement arrows represent assertions that the definitions to the left and right are both refinements of the definition at the centre.

Such assertions cannot be derived automatically from the explanatory text. The explanations may be subjective, and will certainly give only a partial account of the context. We will need to provide some mechanism for managing the addition of assertions to a model; we will need to provide also some mechanism for determining which models, and which assertions, are to be ‘believed’, or considered valid, in a particular application.

As our models may represent incomplete information about the artefacts they represent, and the semantic relationships between them, we cannot infer that no refinement exists simply because there is no *refines* or *abstracts* link present. To record the assertion or conclusion that no refinement exists, we have two other forms of link: *does not refine* links and *does not abstract* links.

As a further example, consider the three attribute definitions given in Fig. 2. Here, we have a more specific definition at the centre of diagram: ‘number of words in a document ignoring hyphens’. We have also an assertion that this refines the definition with the explanatory text ‘number of words in a document’. If we accept this assertion, then any data collected against the new definition can be used in any situation in which the original definition was accepted.

We have also an assertion that the definition ‘number of words in a document according to Microsoft’ is not a refinement of this new, more specific definition. The ‘word count’ feature in Microsoft’s Word application treats hyphenated phrases as single words: it does not ignore hyphens. In contrast, the same feature in Apple’s Pages application does ignore hyphens, treating them as if they were spaces. For example, Word will count the phrase ‘strongly-connected’ as one word, whereas Pages will count it as two words.

3.1. Attributes and links

We will describe each individual data point as an *attribute*, corresponding to an individual field, variable, column, or property in a software artefact, or to an individual item in a data standard. Each attribute definition will include a name, a textual explanation, and a reference to an enumeration or data type, representing the set of values that the attribute may take.

To facilitate the capture of contextual information from, and the generation of, software artefacts we will include also multiplicity and logical constraint information. The multiplicity of an attribute is described in the usual way, as a pair of numbers or symbols, representing the lower and upper bounds upon the number of values associated with a single instance.

Attribute definitions may be linked to indicate that one refines, abstracts, does not refine, or does not abstract the other. A link indicating refinement or abstraction may make reference to a data transformation, indicating how the names and values of attributes in one context may be related to those of corresponding attributes in the other, linked context.

The type of an attribute may be a primitive type, corresponding to a string or numeric type in an implementation. In our language the description of a numeric type may include also an indication of the units employed, such as kilograms or metres per second. Alternatively, the type may be an enumeration, a terminology, or a reference to a class.

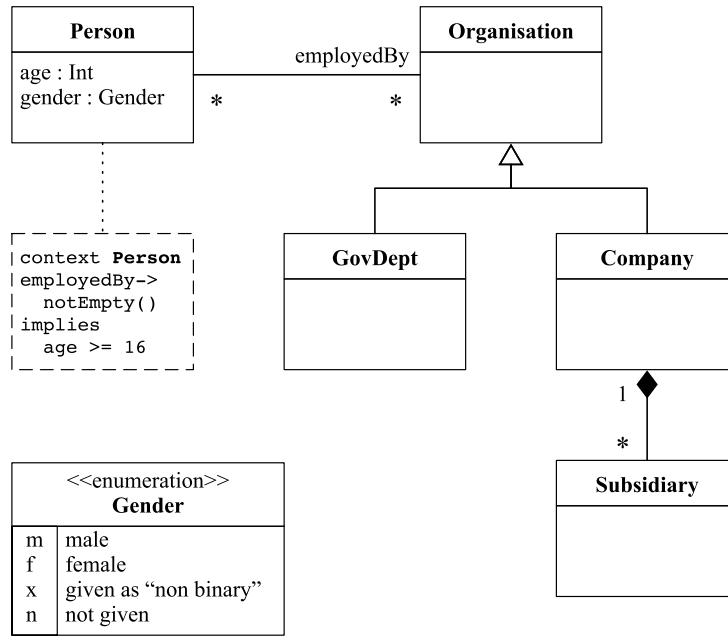


Fig. 3. A class diagram for the model 'Employment'.

3.2. Classes, models, and paths

We will use the notion of a *class* to describe a collection of attributes that are defined, populated, or accessed together. A class definition will include the definitions of its attributes, together with a logical constraint, a textual explanation, and a set of links of its own. As in the case of attributes, links between classes may indicate that the definition of one refines or abstracts the definition of another.

Our notions of extension and containment between classes correspond to the familiar notions of inheritance and composition used in object-oriented design [21]. As our principal concern is the description of the data, we may remain agnostic with regard to behavioural aspects of inheritance, as detailed in [15].

A fully-qualified class name may be used also as the type of an attribute. In this case, the attribute represents a navigable association between two classes. As in object-oriented design, the constraint and the textual explanation of one class may make reference to the attributes of another, associated class. For this reason, it is advisable to manage the definitions of associated classes, and the constraints they impose, together.

We will use the notion of a *model* to describe a collection of class definitions. We will use models as our 'units of context and versioning': attribute and class definitions will be developed, released, and updated only as components of models.

3.3. Enumerations and terminologies

An *enumeration* contains a list of enumerated items or terms that may appear as values for attributes. It may contain also explanatory text, providing additional information regarding the interpretation of one or more of the enumerated items it contains. Each individual item may contain further textual explanation, together with a set of links to other items. We will manage enumerations as components of models, just as we manage classes and attributes.

A *terminology* is also a set of linked items or terms. The difference is that we expect a terminology to be used across many models, and to contain other items that cannot be used as values, along with other kinds of links. Furthermore, a terminology will typically be maintained as an artefact in its own right, and not as a part of the design of a software artefact or data standard.

In either case, the links that concern us are those indicating that the definition of one term, or one enumerated item, is broader or narrower than the definition of another. We will allow the definition of the same four kinds of links, indicating refinement, abstraction, and their negations.

3.4. Example

The class diagram shown in Fig. 3 presented using the class diagram notation of the Unified Modeling Language (UML) [18]. The model contains five classes: **Person**, **Organisation**, **GovDept**, **Company**, and **Subsidiary**. The class **Person** is shown with two attributes: 'age' and 'gender'. The type of 'gender' is given by an enumeration included in the same model; there are four possible values for 'gender', each of which comes with a textual explanation.

```

Catalogue ::= catalogue
  models Model* transformations Transformation*
  terminologies Terminology* mappings Mapping*
  types Type* conversions Conversion*

Model ::= model
  name Name text Text constraint Constraint links Link*
  classes Class* enumerations Enumeration*

Class ::= class
  name Name extends Path* contains Path*
  text Text constraint Constraint links Link* attributes Attribute*

Attribute ::= attribute
  name Name text Text constraint Constraint links Link*
  multiplicity Multiplicity type Path

Enumeration ::= enumeration
  name Name text Text enums Enum*

Enum ::= enum
  name Name text Text links Link*

Link ::= refines Path[ using Path ] | doesNotRefine Path |
  abstracts Path[ using Path ] | doesNotAbstract Path

Type ::= type
  name Name text Text constraint Constraint
  primitive Primitive units Units

Path ::= Name*

```

Fig. 4. The Catalogue Language (partial).

There is an association between **Person** and **Organisation**: this may be seen as a reference-valued attribute of **Person**, called 'employedBy'. This association is many-to-many, as indicated by the asterisks at either end. There is a constraint, written in the Object Constraint Language (OCL) designed for use with UML. This constraint is defined in the context of the class **Person**, and insists that whenever the reference-valued attribute 'employedBy' is non-empty, the value of the attribute 'age' must be greater than or equal to 16.

There are two subclasses of **Organisation**, as indicated by the lines leading to the hollow arrowhead: this is the notation for inheritance in the UML. An instance of one of the subclasses, **Company**, can 'contain', or take sole responsibility for, a number of instances of **Subsidiary**. This is indicated by a line with a solid diamond: this is the notation for composition in the UML. If this association were annotated with 'role names' at either end, then these would be treated as reference-valued attributes within the classes, in the same way as 'employedBy' within **Person**.

3.5. Catalogues

In our language, a *catalogue* is a collection of models and related entities: transformations, terminologies, mappings, types, and conversions. Each transformation will describe a relation between classifiers: attributes, classes, or complete data models; each mapping will describe a relation between terms in enumerations or terminologies; each conversion will represent a function between data types. These related entities will find application across multiple models, and will be managed and versioned separately.

A grammar for part of the language is shown in Fig. 4. The existing implementation includes several other features, to support: the transfer and use of models and other entities across catalogue instances; the organisation of models into collections; the organisation of users into communities; model versioning and publication status; representations of workflows and dataflows; links to software artefacts and instances of datasets.

4. A formal semantics

We will give a formal, denotational semantics to the language in terms of first-order logic, basic set theory, and binary relations. This semantics will serve three purposes: to demonstrate that the language admits a consistent, compositional interpretation; to support the definition of additional constraints upon usage, not expressed in the simple grammar above; and to provide a basis for a simple theory of refinement and interoperability.

4.1. Catalogue items

We will write *Name* to denote the set of all item names, *Text* to denote the set of all possible textual explanations, and *Constraint* to denote the set of all possible logical constraints.

[*Name*, *Text*, *Constraint*]

To support our description of numeric data types, we will introduce also the concepts of units and sign:

[*Units*, *Sign*]

A path, or fully-qualified name, is a non-empty sequence of names:

$Path == seq_1 Name$

Each link is associated with a path, leading to the classifier or term that is the subject of the assertion. In the case of a positive assertion regarding a classifier there may be an additional path, the first component of the pair, leading to the transformation needed to achieve the refinement or the abstraction.

$Link ::= \begin{array}{l} \text{refines} \langle Path \rangle \\ | \text{doesNotRefine} \langle Path \rangle \\ | \text{doesNotAbstract} \langle Path \rangle \\ | \text{abstracts} \langle Path \rangle \\ | \text{refinesUsing} \langle Path \times Path \rangle \\ | \text{abstractsUsing} \langle Path \times Path \rangle \end{array}$

Each item in a catalogue will have a name, a path, and a textual explanation; the name of the item will be the last name appearing in the path.

<i>Item</i> <i>name</i> : <i>Name</i> <i>path</i> : <i>Path</i> <i>text</i> : <i>Text</i>
<i>name</i> = last path

If the item in the catalogue is a classifier, it will have two further properties: a logical constraint and a set of links.

<i>Cltem</i> <i>Item</i> <i>constraint</i> : <i>Constraint</i> <i>links</i> : $\mathbb{P} Link$
--

If the classifier is a data attribute, it will have another property: a path leading to a type definition.

<i>Attribute</i> <i>Cltem</i> <i>type</i> : <i>Path</i>

If the classifier is a data class, it will have three other properties: a set of paths leading to classes that the current class extends or inherits from; a set of paths leading to classes that the current class notionally contains; and a set of attributes.

<i>Class</i> <i>Cltem</i> <i>extends, contains</i> : $\mathbb{P} Path$ <i>attributes</i> : $\mathbb{P} Attribute$
$\forall a : attributes \bullet a.path = path \hat{\ } \langle a.name \rangle$

An attribute can belong to only one class: the attribute definitions are contained within the class definition. For each attribute, the attribute path is obtained by extending the class path with the name of the attribute.

An enumerated item has a set of links,

<i>Enum</i>
<i>Item</i>
$links : \mathbb{P} Link$

and an enumeration contains a set of enumerated items:

<i>Enumeration</i>
<i>Item</i>
$enums : \mathbb{P} Enum$
$\forall e : enums \bullet e.path = path \frown \langle e.name \rangle$

A model is itself a classifier, bringing additional constraint information and its own set of links. Models are ‘top level’ items within a catalogue: a model path contains only the name of the model in question. A model contains classes and enumerations: in each case, the path to the contained item consists of the name of the model followed by the name of the item.

<i>Model</i>
<i>CItem</i>
$classes : \mathbb{P} Class$
$enumerations : \mathbb{P} Enumeration$
$path = \langle name \rangle$
$\forall c : classes \bullet c.path = \langle name, c.name \rangle$
$\forall e : enumerations \bullet e.path = \langle name, e.name \rangle$
$\forall c : classes \bullet c.contains \cup c.extends \subseteq \{d : classes \bullet d.path\}$

The last constraint expresses the requirement that composition and inheritance relationships can exist only between classes in the same model.

Transformations are also top-level items. The additional property of a transformation is a function on sets of attribute-value pairs.

<i>Transformation</i>
<i>Item</i>
$transform : (Path \rightarrow Path) \rightarrow (Path \rightarrow Path)$
$path = \langle name \rangle$

Each attribute or value is identified by a path. Each set of attribute-value pairs will be functional, in the sense that each attribute will have at most one value.

A term has the same properties as an enumerated item:

<i>Term</i>
<i>Item</i>
$links : \mathbb{P} Link$

A terminology is similar to an enumeration, except that it is a top-level item rather than a component of a model.

A mapping is a top-level item relating values—enumerated items or terms—to values, a type is a top-level item with a range of values from some primitive or native datatype and—for numeric types—a description of the units employed, and a type conversion is a top-level item with a function acting on primitive datatypes and—for numeric types—source and target units. For the purposes of this paper, it is enough to know that these items are maintained within the catalogue, alongside the data models, and that they can be referenced using fully-qualified names or paths.

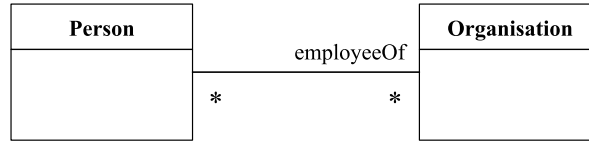


Fig. 5. The model 'HMRC'.

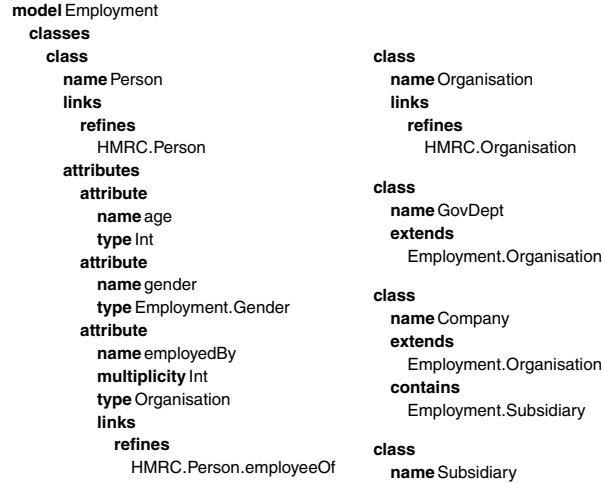


Fig. 6. Employment model (classes).

A catalogue is simply a collection of these top-level items:

```

Catalogue
models :  $\mathbb{P}$  Model
transformations :  $\mathbb{P}$  Transformation
terminologies :  $\mathbb{P}$  Terminology
mappings :  $\mathbb{P}$  Mapping
types :  $\mathbb{P}$  Type
conversions :  $\mathbb{P}$  Conversion
  
```

4.2. Example

Suppose that our catalogue contains also a model 'HMRC', shown in Fig. 5. This model has two classes, **Person** and **Organisation**, and an association between them that corresponds to a reference-valued attribute of **Person** named 'employeeOf'. The model might contain also constraints, setting out the conditions under which Her Majesty's Revenue and Customs (HMRC) would consider a person to be an employee of an organisation.

To re-use this information, we may add semantic links to the relevant components of our 'Employment' model. In the language of Fig. 4, this would produce the model shown in Fig. 6: the links properties of these components indicate that their interpretations should be seen as refinements of the interpretations of the corresponding components of the 'HMRC' model.

Fig. 7 can be represented as an instance of the *Model* schema type. The $\langle \dots \rangle$ notation is used to denote a binding of a *Z* schema type, and the \rightsquigarrow symbol is used to indicate the value taken by each of the schema attributes. In this example, only the values of 'Gender' have any explanatory text.

4.3. Catalogue properties

To simplify the presentation of our analysis, without compromising our ability to type-check or reason about the specification, we will introduce the notion of a generic catalogue item:

```

CatalogueItem  $\hat{=}$ 
  Model  $\vee$  Class  $\vee$  Attribute  $\vee$  Enumeration  $\vee$  Enum  $\vee$  Type  $\vee$ 
  Transformation  $\vee$  Term  $\vee$  Terminology  $\vee$  Mapping  $\vee$  Conversion
  
```

```

⟦ name ∼ Employment, path ∼ ⟨Employment⟩, text ∼ " ", links ∼ {} ,
classes ∼ {⟦ name ∼ Person, path ∼ ⟨Employment, Person⟩, text ∼ " ",
links ∼ {refines ⟨HMRC, Person⟩}, extends ∼ {}, contains ∼ {},
constraint ∼ employedBy → notEmpty() implies age ≥ 16
attributes ∼
  {⟦ name ∼ age, text ∼ " ", links ∼ {},
    path ∼ ⟨Employment, Person, age⟩,
    multiplicity ∼ {1}, type ∼ Age⟧,
  ⟦ name ∼ gender, text ∼ " ", links ∼ {},
    path ∼ ⟨Employment, Person, gender⟩,
    multiplicity ∼ {1}, type ∼ ⟨Employment, Gender⟩⟧,
  ⟦ name ∼ employedBy, text ∼ " ",
    links ∼ {refines ⟨HMRC, Person, employeeOf⟩},
    path ∼ ⟨Employment, Person, employedBy⟩,
    multiplicity ∼ N, type ∼ ⟨Employment, Organisation⟩⟧} ,
⟦ name ∼ Organisation, path ∼ ⟨Employment, Organisation⟩,
text ∼ " ", links ∼ {refines ⟨HMRC, Organisation⟩},
extends ∼ {}, contains ∼ {}, attributes ∼ {}⟧,
⟦ name ∼ GovDept, path ∼ ⟨Employment, GovDept⟩,
links ∼ {}, extends ∼ {⟦ Employment, Organisation⟧},
contains ∼ {}, attributes ∼ {}⟧,
⟦ name ∼ Company, path ∼ ⟨Employment, Company⟩,
links ∼ {}, extends ∼ {⟦ Employment, Organisation⟧},
contains ∼ {⟦ Employment, Subsidiary⟧}, attributes ∼ {}⟧,
⟦ name ∼ Subsidiary, path ∼ ⟨Employment, Subsidiary⟩,
links ∼ {}, extends ∼ {}, contains ∼ {}, attributes ∼ {}⟧
enumerations ∼ {⟦ name ∼ Gender, path ∼ ⟨Employment, Gender⟩,
  enums ∼ {⟦ name ∼ m, path ∼ ⟨Employment, Gender, m⟩,
    text ∼ "male", links ∼ {}⟧,
  ⟦ name ∼ f, path ∼ ⟨Employment, Gender, f⟩,
    text ∼ "female", links ∼ {}⟧,
  ⟦ name ∼ x, path ∼ ⟨Employment, Gender, x⟩,
    text ∼ "given as non binary", links ∼ {}⟧,
  ⟦ name ∼ n, path ∼ ⟨Employment, Gender, n⟩,
    text ∼ "not given", links ∼ {}⟧} } }

```

Fig. 7. Employment model: abstract representation.

together with a lookup function mapping paths to items, and use these to shadow the contents of the catalogue:

CatalogueItems

Catalogue

item : Path → CatalogueItem

We will introduce a specialisation of this lookup function for each different kind of item, and insist that the properties of a shadow, generic item have the same values as those of the corresponding, actual item:

CatalogueObjects

CatalogueItems

model : Path → Model

class : Path → Class

attribute : Path → Attribute

...

model = {*m* : models • *m.path* ↦ *m*}

...

∀ *p* : dom *model* • (*item p*).constraint = (*model p*).constraint

...

We will introduce also the set of all currently-valid paths for a catalogue, as the union of the domains of these lookup functions:

<i>CataloguePaths</i>
...
$paths : \mathbb{P} Path$
$paths = \bigcup \{ \text{dom } model, \text{dom } class, \text{dom } attribute, \text{dom } enumeration, \text{dom } enum, \text{dom } transformation, \text{dom } term, \text{dom } terminology, \text{dom } mapping, \text{dom } type, \text{dom } conversion \}$

The definition of one item may re-use and include the definition of another by way of a *refines* link or—in the case of a class—in terms of extension.

<i>CatalogueUses</i>
...
$uses : Path \leftrightarrow Path$
$uses = \{ p1, p2 : paths \mid \exists t : paths \bullet \text{refines } p2 \in (item\ p1).links \vee \text{refinesUsing } (t, p2) \in (item\ p1).links \vee p2 \in (class\ p1).extends \}^*$

The relation *uses* is defined as the reflexive transitive closure of the relation corresponding to refinement or extension links.

One catalogue item may contain another in terms of its path or—in the case of a class—through a class composition association:

<i>CatalogueContains</i>
...
$contains : Path \rightarrow Path$
$contains = (_ \leq _) \triangleright paths \cup \{ p, q : \text{dom } class \mid q \in (class\ p).contains \}$

where \leq denotes sequence prefix and \triangleright denotes relational range restriction.

We can use the reflexive transitive closure of the union of *uses* and *contains*[~], the inverse of this relation, to identify the context in which any item in the catalogue is defined:

<i>CatalogueContext</i>
...
$context : Path \rightarrow \mathbb{P} Path$
$\forall p : paths \bullet context\ p = (uses \cup contains^{\sim})^* \langle \{ p \} \rangle$

Again, we do this in terms of paths: the *context* of a path to an item is the set of paths to items providing context for its definition.

4.4. Values and interpretations

The fundamental purpose of the catalogue is to provide definitive descriptions for a set of classifiers. It does this by associating each classifier with a set of possible values, subject to a logical constraint, and a textual explanation.

A value may be: an element of a primitive type, such as numbers or strings; a term, or an item in an enumeration; or an instance of a class. We will refer to the last of these as an ‘object value’.

```
Value ::= primitiveValue⟨⟨Type⟩⟩
        | termValue⟨⟨Term⟩⟩
        | enumValue⟨⟨Enum⟩⟩
        | objectValue⟨⟨Object⟩⟩
```

The logical constraint defining the set of possible values for a classifier, as a subset of this type, will be formed as the conjunction of all of the logical constraints in the context of its definition.

We may identify the constraint information associated with each classifier:

CatalogueConstraint

```
...
constraint : Path → Constraint
constraint =
  {p : dom model ∩ dom class ∩ dom attribute •
   p ↦ (item p).constraint}
```

and define the set of possible values as follows:

CatalogueValues

```
...
values : Path → ℙ Value
∀ p : dom attribute • (∃ t : Path | t = (attribute p).type •
  t ∈ dom enumeration ⇒
    values p = enumValue( (enumeration t).enums ) ∩
    extent (∧ (constraint ∥ context p ))) ∧
  ...
  t ∈ dom class ⇒
    values p = values t ∩ extent (∧ (constraint ∥ context p )))
∀ p : dom class • values p =
  objectValue( {n : Name; v : Value | ∃ a : (class p).attributes •
    n = a.name ∧ v ∈ values a.path} ) ∩
  extent (∧ (constraint ∥ context p )))
...
```

Here we show only the values for classes, and for enumeration- and class-valued attributes; the values for models, and for term- and primitive-valued attributes, are defined in an entirely similar fashion.

We may identify the textual explanation directly associated with each item, together with its ‘complete’ text, being the combination of explanations for the items in context:

CatalogueText

```
...
text, completeText : Path → Text
text = {p : paths • p ↦ (item p).text}
completeText = {p : paths • p ↦ ⊔ (text ∥ context p )}
```

Here, we have written \sqcup to denote the combination of a set of textual explanations. The path associated with a particular value is given by

CatalogueValuePath

```
...
valuePath : Value → Path
∀ v : Value •
  v ∈ ran primitiveValue ⇒
    valuePath v = (primitiveValue~ v).path ∧
  v ∈ ran termValue ⇒
    valuePath v = (termValue~ v).path
...
```

The definitive description of a classifier, its definition in the context of a catalogue, is a function from possible values to textual explanations.

<i>CatalogueDefinition</i>
...
<i>definition</i> : <i>Path</i> \rightarrow (<i>Value</i> \rightarrow <i>Text</i>)
$\forall p : \text{dom } \textit{model} \cap \text{dom } \textit{class} \cap \text{dom } \textit{attribute} \bullet$
<i>definition</i> $p = \{v : \text{values } p \bullet$
$v \mapsto \text{completeText } p \uplus \text{completeText } (\text{valuePath } v)\}$

Here, we write \uplus to denote the combination of two textual explanations. The logical constraint that defines *values* p depends upon the context of the classifier; the associated text depends also upon the context of the value.

4.5. Example

The definition of the association *employedBy* within the model *Employment* has the following context within the catalogue:

```
context (Employment, Person, employedBy)
= (uses  $\cup$  contains~)*( $\{ \langle \text{Employment}, \text{Person}, \text{employedBy} \rangle \}$ )
=  $\{ \langle \text{Employment} \rangle, \langle \text{Employment}, \text{Person} \rangle, \langle \text{HMRC}, \text{Person}, \text{employeeOf} \rangle, \langle \text{HMRC} \rangle, \langle \text{HMRC}, \text{Person} \rangle \}$ 
```

The reflexive transitive closure of the union of the usage and containment relations—the latter reversed here—maps this path to a set of paths within the *Employment* and *HMRC* models. This set consists of the paths of items whose definitions have been re-used, together with the paths of any other items that contain them.

This is the context in which the definition of *employedBy* is made, according to the catalogue. We can obtain all of the relevant text and all of the relevant constraint information as the relational image of this set under the functions *text* and *constraint*, respectively. The latter will give us the set of values that may be associated with *employedBy*: in this case, a set of object values satisfying the constraints of *Employment.Organisation* and *HMRC.Organisation*.

4.6. Refinement and belief

We may now define a formal notion of semantic refinement between classifiers in the catalogue, again identified by their paths. A classifier p_1 is a semantic refinement of another classifier p_2 if the definition of p_1 is consistent with that of p_2 . In the case where no transformation, mapping, or conversion is applied, this means that:

- every value that p_1 can take is also a possible value of p_2
- for every value of p_1 , the explanation provided in the context of p_1 is consistent with, or expands upon, the explanation of the same value provided in the context of p_2

We will write $p_1 \sqsupseteq p_2$ to indicate that the item with path p_1 is a semantic refinement of the item with path p_2 , defined as follows:

<i>CatalogueRefinement</i>
...
$\sqsupseteq : \text{Path} \leftrightarrow \text{Path}$
$\forall p_1, p_2 : \text{Path} \bullet p_1 \sqsupseteq p_2 \Leftrightarrow$
$\text{dom}(\text{definition } p_2) \subseteq \text{dom}(\text{definition } p_1) \wedge$
$\forall v : \text{dom}(\text{definition } p_1) \bullet$
$\text{definition } p_1 v \supset \text{definition } p_2 v$

We write ' \supset ' to denote logical implication between textual explanations. In the absence of any contradictions between the additional explanation t_3 and either of the explanations t_1 and t_2 , it should be the case that

$$(t_1 \uplus t_3) \supset (t_2 \uplus t_3) \Leftrightarrow (t_1 \supset t_2)$$

In our definition of \sqsupseteq , this corresponds to the assumption that the explanation of each of the possible values does not conflict with the explanation of either classifier. Where this is the case, the defining constraint can be simplified:

$$\forall p_1, p_2 : \text{Path} \bullet p_1 \sqsupseteq p_2 \Leftrightarrow \text{values } p_2 \subseteq \text{values } p_1 \wedge \text{completeText } p_1 \supset \text{completeText } p_2$$

Where the refinement relationship between two classifiers relies upon the application of a data transformation, this transformation is applied to *values* $p2$ and *completeText* $p2$ before the above comparisons are made.

In applications of a metadata catalogue, refinement between classifiers will normally be manually asserted rather than derived from a formal analysis of the constraints and explanations involved. A user will add a link to the catalogue to indicate that, in their opinion, one classifier is a refinement of another. The same is true of refinement between enumerated items or terms.

If we assume that our notions of logical implication for constraints and textual explanations are both reflexive and transitive, then the same will be true of our notion of refinement. Once a link has been added to the catalogue, the assertion that it represents can be used to reason not only about the two items concerned, but also about the semantic relationships between other items.

Whether we are able to establish that one item refines another may depend upon which definitions, and hence which assertions, we choose to believe. The extent of our belief is represented by a set of paths: the *belief* set. This set should be closed under any combination of *uses*, *contains*, and *contains*[~].

CatalogueBelief

```
...
belief :  $\mathbb{P}$  Path
(uses  $\cup$  contains~  $\cup$  contains)(belief)  $\subseteq$  belief
```

If we choose to believe the definition of an item, then we must believe the definitions of other items that this definition refers to, the definitions that form part of its context, and the definitions of any items it contains.

We will write '*p1* refines *p2*' to indicate that the item with path *p1* has a link asserting that it refines the item with path *p2*. Similarly, we will write '*p1* abstracts *p2*', '*p1* doesNotRefine *p2*', and '*p1* doesNotAbstract *p2*' to indicate that the same item has a link asserting that it abstracts, does not refine, or does not abstract the item with path *p2*, respectively.

CatalogueStatements

```
...
_ refines _, _ doesNotRefine _ : Path  $\leftrightarrow$  Path
_ abstracts _, _ doesNotAbstract _ : Path  $\leftrightarrow$  Path
_ extends _, _ contains _ : Path  $\leftrightarrow$  Path

(_ refines _) = { p1, p2 : paths |  $\exists t : \text{dom transformation} \bullet$ 
  refines p2  $\in$  (item p1).links  $\vee$ 
  refinesUsing (t, p2)  $\in$  (item p1).links }

(_ abstracts _) = { p1, p2 : paths |  $\exists t : \text{dom transformation} \bullet$ 
  abstracts p2  $\in$  (item p1).links  $\vee$ 
  abstractsUsing (t, p2)  $\in$  (item p1).links }

(_ doesNotRefine _) = { p1, p2 : paths |
  doesNotRefine p2  $\in$  (item p1).links }

(_ doesNotAbstract _) = { p1, p2 : paths |
  doesNotAbstract p1  $\in$  (item p2).links }

(_ extends _) = { p1, p2 : paths |
  p2  $\in$  (item p1).extends }

(_ contains _) = { p1, p2 : paths |
  p2  $\in$  (item p1).contains }
```

The definition of a class *p1* may include the statement that it extends the definition of another class *p2*. Where this is the case, we will write '*p1* extends *p2*'. Similarly, the definition of a class *p1* may include the statement that this class contains another class *p2*, in the sense of composition in implementation—any instances of *p2* should be treated as components of an instance of *p1*. Where this is the case, we will write '*p1* contains *p2*'.

Where a refinement or abstraction makes use of a data transformation, then the path *t* to the transformation is included as a parameter in the assertion.

The fact that we are able to give a denotational semantics to our catalogue language in this fashion, through the structurally-recursive definition of a function *definition*, confirms that our language is compositional with respect to this notion of language semantics. Furthermore, the schemas that define the language, presented in Section 4.1, are easily seen to be satisfied by an empty catalogue, containing no models at all.

To confirm that our *approach* is compositional, in the sense that we can add more information about an existing item, or add information about other items—while guaranteeing that existing information is preserved, and the whole of the

information presented about each item is precisely the sum of its various parts—we need to consider how the information obtained from a catalogue depends upon which parts of its content we choose to believe.

5. Refinement and interpretation

The formal semantics given above provides a basis for reasoning about refinement between items described in the catalogue language, and hence establishing semantic interoperability between information systems and software artefacts. In this section, we explore what may be inferred, in terms of refinements, from the current state of a catalogue, under a particular belief set. We present a set of inference rules, and show how the catalogue language supports multiple perspectives.

5.1. Refinement and belief

To explore what may be inferred, under a particular belief set, we will introduce a further refinement relation \rightarrow . For any paths $p1$ and $p2$, and for any belief set B , we will write

$$B \vdash p1 \rightarrow p2$$

to indicate that, within the current catalogue, $p1$ is a refinement of $p2$, if the definitions with paths in B are to be believed or accepted.

Formally, we define

$\begin{array}{l} \text{Refinement} \\ \text{CatalogueStatements} \\ \hline _ \rightarrow _ : \text{Path} \leftrightarrow \text{Path} \\ \hline _ \rightarrow _ = \\ \quad ((\sqsupseteq \cup \text{refines} \cup \text{abstracts}) \cap (\text{belief} \times \text{belief}))^* \end{array}$

The relation \rightarrow is formed as the reflexive transitive closure of the intersection of $\text{belief} \times \text{belief}$ with the union of three relations: ' \sqsupseteq ', 'refines', and the inverse of 'abstracts'. Two paths are related under this closure if and only if there is a sequence of steps from one to the other, using only pairs from those three relations for which both components of the pair lie in the belief set.

Our turnstile entailment relation \vdash is then defined as the relationship between values of the set belief and the relation \rightarrow given by the extent of the schema *Refinement*: that is, for any set of paths B :

$$B \vdash p1 \rightarrow p2 \Leftrightarrow B = \text{belief} \wedge p1 \rightarrow p2$$

If we need to reason about more than one catalogue, we may parameterise this relation to identify the catalogue in which the entailment holds: for example, we may write \vdash_C to indicate that it holds for catalogue C .

Our first inference rule follows immediately from the definition of \rightarrow . If the formal semantics of their definitions shows that $p1$ is a refinement of $p2$, and our belief set B indicates that we believe both definitions, then we may conclude that $p1$ is a refinement of $p2$:

Rule 1 (Refinement by definition) For paths $p1$ and $p2$ and belief set B :

$$\frac{p1 \sqsupseteq p2 \wedge p1 \in B \wedge p2 \in B}{B \vdash p1 \rightarrow p2}$$

Similarly, if the definition of an item includes the assertion that it refines another item, then it is enough to believe the definition in question to conclude that the refinement holds:

Rule 2 (Refinement by assertion) For paths $p1$ and $p2$ and belief set B :

$$\frac{p1 \text{ refines } p2 \wedge p1 \in B}{B \vdash p1 \rightarrow p2}$$

It is not necessary to require also that $p2 \in B$: if $p1 \in B$ and $p1$ refers to $p2$, then $p2 \in B$, as belief sets are closed under definition re-use.

If the definition of an item includes the assertion that it abstracts another item, and we are to believe the definition of that other item, then we may conclude that refinement holds in the other direction:

Rule 3 (Abstraction by assertion) For paths $p1$ and $p2$ and belief set B :

$$\frac{p1 \text{ abstracts } p2 \wedge p1 \in B \wedge p2 \in B}{B \vdash p2 \rightarrow p1}$$

The fact that $p2$ needs to be included in the belief set for the conclusion to hold, reflects the fact that although belief sets are closed under 'refines', they do not need to be closed under 'abstracts'.

5.2. Refinement and inheritance

As described above, the catalogue language does not include any explicit representation of behaviour or functionality. Even where a corresponds to an abstraction of a program or a workflow, the classes of the model will have no explicit behavioural properties: no methods, functions, or operations will be described. Our notion of inheritance is therefore based entirely upon the textual explanations and constraints present in class definitions.

The semantics given to inheritance takes into account the fact that some of the constraint information and textual explanation pertaining to a particular class may not appear in the properties of the class. Instead, it may appear in the definition of any item that forms part of the context for the class in question. If $p1$ and $p2$ are paths leading to classes in a model, then

$$\begin{aligned} p1 \text{ extends } p2 & \Rightarrow p2 \in (\text{item } p1).\text{extends} & [\text{CatalogueStatements}] \\ & \Rightarrow p2 \in (\text{class } p1).\text{extends} & [\text{CatalogueItems}] \\ & \Rightarrow p1 \mapsto p2 \in \text{uses} & [\text{CatalogueUses}] \\ & \Rightarrow p2 \in \text{context } p1 & [\text{CatalogueContext}] \\ & \Rightarrow \text{values } p1 \subseteq \text{values } p2 \wedge & [\text{CatalogueValues}] \\ & \quad \text{completeText } p1 \supset \text{completeText } p2 & [\text{CatalogueText}] \\ & \Rightarrow \text{dom}(\text{definition } p2) \subseteq \text{dom}(\text{definition } p1) \wedge \\ & \quad \forall v : \text{dom}(\text{definition } p1) \bullet \\ & \quad \quad \text{definition } p1 \ v \supset \text{definition } p2 \ v & [\text{CatalogueDefinition}] \\ & \Rightarrow p1 \sqsupseteq p2 & [\text{CatalogueRefinement}] \end{aligned}$$

The implication for the step with labels *CatalogueValues* and *CatalogueText* relies upon three pieces of information: *context* is defined in terms of a reflexive transitive closure; \wedge is monotonic with respect to logical implication on formal constraints; \sqsupseteq is monotonic with respect to logical implication on textual explanations. The last of these is an assumed property or axiom of the operator in question.

This is enough to establish that:

Rule 4 (Extension) For paths $p1$ and $p2$ and belief set B :

$$\frac{p1 \text{ extends } p2 \wedge p1 \in B}{B \vdash p1 \rightarrow p2}$$

The orientation of the arrow symbol denoting our refinement relation matches that of the arrow denoting inheritance or subclassing in UML class diagrams.

5.3. Closure of belief sets

The formal definition of our catalogue language includes the requirement, in the constraint of schema *CatalogueBelief*, that any belief set should be closed under the re-use of definitions, whether this is achieved through refinement or through extension. From this, we may derive the following rules:

Rule 5 (Closure under re-use) For paths $p1$ and $p2$ and belief set B ,

$$\frac{p1 \in B \wedge p1 \text{ uses } p2}{p2 \in B}$$

where 'uses' is either of 'refines' or 'extends'.

If path $p1$ is a prefix of path $p2$, then the item that $p2$ points to is a subcomponent of the item that $p1$ points to. It would be inconsistent to believe in the whole of a definition, and yet not believe in some part of it. The same closure property given above allows us to derive the following rule:

Rule 6 (Downward closure) For paths $p1$ and $p2$ and belief set B ,

$$\frac{p1 \in B \wedge p1 \preceq p2}{p2 \in B}$$

Another requirement is that belief should be closed in the opposite direction. This reflects our view that classes, attributes, and enumerations should not be defined and managed independently, but only as part of coherent data models, and that models should be our ‘units of context and versioning’, as explained in Section 1.

Rule 7 (Upward closure) For $p1$ and $p2$ and belief set B ,

$$\frac{p1 \in B \wedge p2 \preceq p1}{p2 \in B}$$

Any of the items that contain the item with path $p1$ may include textual explanations or logical constraints that refer directly to item $p1$.

In combination, these three rules confirm that the context for a given definition includes the definition of every item in the same model, and every item in every model that these definitions refer to, directly or indirectly, with refinement or extension links.

We will find it useful to introduce also the concept of a ‘belief closure’ of a given set of paths. If P is a set of paths, we write \bar{P} to denote its belief closure, defined by

$\begin{array}{l} \text{BeliefClosure} \\ \dots \\ \neg : \mathbb{P}Path \rightarrow \mathbb{P}Path \\ \forall P : \mathbb{P}Path \bullet \\ \quad \bar{P} = ((_ \preceq _) \cup (_ \preceq _)^\sim \cup (_ \text{refines } _))^*(P) \end{array}$

The belief closure of P is the smallest set of paths that we are required to believe in, if we wish to believe in P . It is the set of all paths that can be reached from paths in P by following refinement assertions and/or selecting items in the same model, repeatedly.

5.4. Compositionality

Where a belief set entails several requirements, we may choose to present a single entailment in which those requirements appear as a common-separated list, rather than write out a conjunction of entailments: for example,

$$B \vdash r1, r2, \dots \Leftrightarrow B \vdash r1 \wedge B \vdash r2 \wedge \dots$$

We will use this convention in our next rule, which is again a consequence of the definitions of \rightarrow and \vdash :

Rule 8 (Combination) For paths $p1$, $p2$, $p3$, and $p4$, and belief sets B and C ,

$$\frac{B \vdash p1 \rightarrow p2 \wedge C \vdash p3 \rightarrow p4}{B \cup C \vdash p1 \rightarrow p2, p3 \rightarrow p4}$$

As \rightarrow is defined as a reflexive, transitive closure, we have also the following rule:

Rule 9 (Transitivity) For paths $p1$, $p2$, and $p3$, and belief set B ,

$$\frac{B \vdash p1 \rightarrow p2, p2 \rightarrow p3}{B \vdash p1 \rightarrow p3}$$

Together, these rules tell us that we can add paths—and hence new content—to a catalogue without detracting from the information that it already contains, and that by adding new content that includes refinement or abstraction links, we can

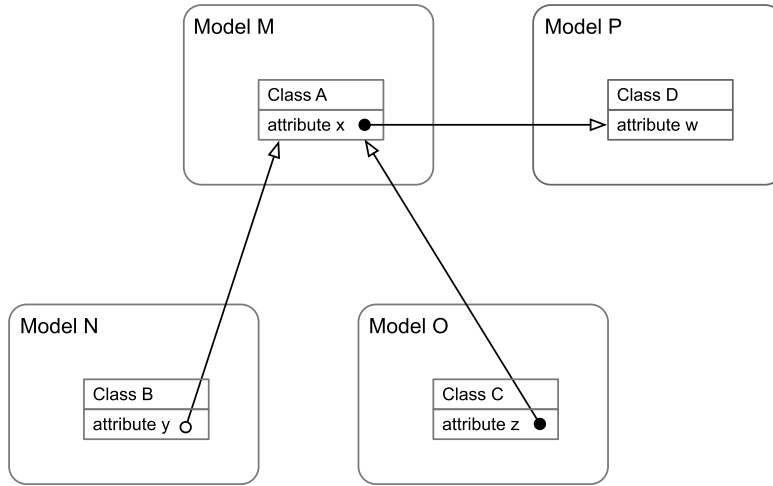


Fig. 8. Picturing refinement.

provide more information about any item that is already in the catalogue. We may express this as a further, derived rule: one that is an immediate consequence of Rules 8 and 9.

Rule 10 (Composition) For paths $p1$, $p2$, and $p3$, and belief sets B and C ,

$$\frac{B \vdash p1 \rightarrow p2 \wedge C \vdash p2 \rightarrow p3}{B \cup C \vdash p1 \rightarrow p3}$$

Not only is our language compositional, but so too—in this sense—is our approach to the inference of refinement and semantic interoperability.

5.5. Perspectives

Although every ‘outgoing’ semantic link forms an inherent part of the definition of the item in question, the same is not true of all ‘incoming’ semantic links—links that refer to this item, but are properties of some other item. If the definition of that other item sits in another model, and that model is not referred to, then we can choose whether or not to believe it.

To illustrate this and other situations, we will use decorated forms of our refinement symbol \rightarrow to represent the various forms of direct semantic link that may be introduced. We will add a solid disc to indicate an outgoing semantic link, corresponding the inclusion of a *refines* link in the *links* property of the source item. In text, the resulting symbol could equally serve as a synonym for ‘refines’: for any paths $p1$ and $p2$, we have that

$$p1 \bullet \rightarrow p2 \Leftrightarrow p1 \text{ refines } p2 \quad (\Leftrightarrow \text{refines } p2 \in (\text{item } p1).links)$$

We will add a hollow disc to indicate an incoming link: one in which the assertion is a property of the target item. In text, the resulting symbol could equally serve as a synonym for *the inverse of ‘abstracts’*: for any paths $p1$ and $p2$,

$$p1 \circ \rightarrow p2 \Leftrightarrow p2 \text{ abstracts } p1 \quad (\Leftrightarrow \text{abstracts } p1 \in (\text{item } p2).links)$$

Fig. 8 illustrates a scenario in which the definition of attribute M.A.x includes an abstraction link to attribute N.B.y and a refinement link to attribute P.D.w—indicated by the arrow on the right with a hollow disc, and the horizontal arrow on the left with a solid disc—and the definition of another attribute, O.C.z, includes a refinement link to M.A.x. For economy, we will refer to these attributes below using the attribute part of the name: that is, we will write x for M.A.x.

The definition of attribute x re-uses that of attribute w , in terms of the outgoing refinement link. If we wish to make use of the definition of x , we must believe not only model M, in which x is defined, but also model P, in which w is defined. A catalogue with these contents would tell us that

$$M \cup P \vdash x \rightarrow w$$

The other models provide no additional context for the definition of x : we can use the definition of x whether or not we believe model N or model O.

In the case of attribute y , the refinement link is incoming: the statement that the definition of y refines that of x is made in model M. There are no outgoing links from N to O, corresponding to definition usage or class extension, and so we are

free to believe N without also believing M . Simply believing N would give us merely the definition of y , with no additional refinement information, whereas believing also in M and P would give us

$$N \cup M \cup P \vdash y \rightarrow x, x \rightarrow w$$

and hence also that $y \rightarrow w$.

In the case of attribute z , the refinement link to x is outgoing, as is the onward refinement link to w . We may use the definition of x without needing to believe model O , just as we have above:

$$M \cup P \vdash x \rightarrow w$$

If we are content to believe in model O , however, we may infer also that

$$O \cup M \cup P \vdash z \rightarrow x, x \rightarrow w$$

and hence also that $z \rightarrow w$.

In this scenario, we might imagine that models N and O corresponds to artefacts used for data collection, such as forms, schemas, or databases, that model M corresponds to a data standard, and that model P corresponds to some proposed usage or application of data. If we believe all four of the models, then the catalogue tells us that:

- data that meets the requirement of x within standard M will be fit for purpose w in P ;
- data collected by N , against attribute y , meets the requirements of x within M ;
- data collected by O , against attribute z , also meets the requirements of x within M ;

and hence that data collected by either N or O , against y or z , respectively, can be combined and/or used for purpose w within P .

5.6. Negation

The catalogue language also has support for negative assertions: statements that refinement, or abstraction, does not hold. We may define a negative counterpart \nrightarrow to our refinement relation \rightarrow , and write

$$B \vdash p1 \nrightarrow p2$$

to indicate that the current catalogue tells us that, given belief set B , the item with $p1$ is not a refinement of the item with path $p2$. Formally,

$\begin{array}{l} \text{Negation} \\ \hline \text{CatalogueStatements} \\ \neg \nrightarrow : \text{Path} \leftrightarrow \text{Path} \\ \hline \neg \nrightarrow = \\ ((\text{belief} \times \text{belief}) \setminus \supseteq) \cup \\ \text{doesNotRefine} \cup \text{doesNotAbstract} \cap (\text{belief} \times \text{belief}) \end{array}$

Note that this relation is defined simply as the union of three relations, constrained to apply only to paths appearing in *belief*: unlike the definition given in *Refinement*, it does not make use of reflexive transitive closure $*$.

Using this definition, in the context of the properties set out in the schema *CatalogueStatements*, we may derive a set of rules broadly similar to those defined above for \rightarrow . As the form of the rules is now familiar, we may save space by omitting the information that $p1$, $p2$, $p3$, and $p4$ denote paths and that B and C denote belief sets.

Rule 11 (Negation definition)

$$\frac{\neg (p1 \supseteq p2) \wedge p1 \in B \wedge p2 \in B}{B \vdash p1 \nrightarrow p2}$$

Rule 12 (Negation by assertion)

$$\frac{p1 \text{ doesNotRefine } p2 \wedge p1 \in B \wedge p2 \in B}{B \vdash p1 \nrightarrow p2}$$

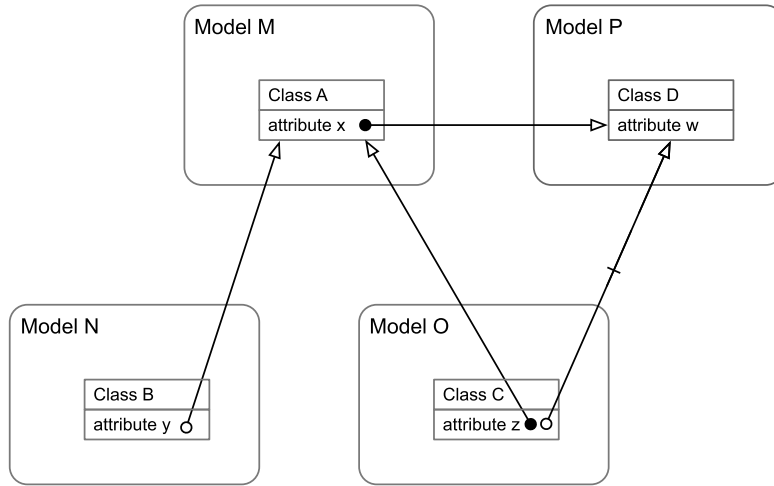


Fig. 9. Conflicting assertions.

Rule 13 (Negation by inverse assertion)

$$\frac{p1 \text{ doesNotAbstract } p2 \wedge p1 \in B \wedge p2 \in B}{B \vdash p2 \nrightarrow p1}$$

As we might expect, this relation does not correspond to class extension; neither is it transitive. It does, however, satisfy the following pair of rules:

Rule 14 (Combination of negation)

$$\frac{B \vdash p1 \nrightarrow p2 \wedge C \vdash p3 \nrightarrow p4}{B \cup C \vdash p1 \nrightarrow p2, p3 \nrightarrow p4}$$

Rule 15 (Combination of refinement and negation)

$$\frac{B \vdash p1 \rightarrow p2 \wedge C \vdash p3 \nrightarrow p4}{B \cup C \vdash p1 \rightarrow p2, p3 \nrightarrow p4}$$

The second rule admits an immediate corollary, showing that it is perfectly feasible to add conflicting information about existing items by adding new paths to the belief set.

Rule 16 (Introduction of conflict)

$$\frac{B \vdash p1 \rightarrow p2 \wedge C \vdash p1 \nrightarrow p2}{B \cup C \vdash p1 \rightarrow p2, p1 \nrightarrow p2}$$

As an illustration of how this might arise in practice, consider the scenario shown in Fig. 9. In this scenario, in addition to the links shown in Fig. 8, there is a negative abstraction link from P.D.w to O.C.z. We will decorate the ‘does not refine’ arrow with a hollow disc $\circ \nrightarrow$ and a solid disc $\bullet \nrightarrow$ to represent incoming and outgoing assertions, respectively.

If we believe O, then we must believe also M and P, as belief sets are closed under outgoing refinement assertions. This leads us to the following entailment

$$O \cup M \cup P \vdash z \rightarrow w, z \nrightarrow w$$

Here, the catalogue is presenting us with conflicting information on refinement between attributes z and w.

This does not prevent us from obtaining consistent information from the catalogue regarding the relationships between other attributes. The negative abstraction link is an incoming assertion, owned by model P; it does not come into play unless we believe also model O. If we believe the other three models, then we may reliably establish that data from y can

be re-used in w :

$$\text{NUMUP} \vdash y \rightarrow w$$

5.7. Consistency

While it is important that our catalogue should support differences of opinion as to whether or not one item refines another, it is important also that it should be able to provide a consistent account of items it contains: both in terms of logical consistency of the account itself, on any given occasion; and in terms of consistency of the account over time.

It would be unreasonable for us to expect consistency during the development of a model. Once this development is complete, however, we might reasonably insist that each model is consistent, in terms of the items and links it introduces, and that the definition of each item in the model should remain constant from that point onwards—that is, consistent over time.

We can achieve this by maintaining a *status* value for each model

$$\text{Status} ::= \text{draft} \mid \text{finalised}$$

and imposing the following four constraints:

1. once the status of a model has changed from *draft* to *finalised*, it remains *finalised* from that point onwards;
2. new items may be added to a model, and the properties of existing items—including outgoing refinement links—may be modified, only if the status of that model is *draft*;
3. outgoing refinement links may be created only if the target item lies within a *finalised* model;
4. only models that are ‘self consistent’ may be finalised.

We say that a model M is self consistent if the belief closure \overline{M} yields no logical contradictions—arising from structural or constraint properties—and no conflicting refinement assertions: that is, pairs of assertions of the form $p1 \rightarrow p2$ and $p \dashv p2$.

An inductive argument, using the properties of the catalogue language, can be used to establish that these constraints are enough to guarantee that:

- for any finalised model M , and the definition of any item within M will remain constant—even if other models, and other links, are added to the catalogue;
- for any belief set B containing only paths to items in finalised models, the catalogue will yield only consistent information regarding refinement.

With these constraints in mind, we should consider again the scenario of Fig. 9. Model P must be finalised before model M , as there is an outgoing refinement link from x to w . Model N can be finalised without reference to any of the other models, provided that the constraint properties of its items contain no logical inconsistencies.

Model O , however, is not self-consistent. The outgoing assertion from z to x places M and hence P in the belief closure:

$$\overline{O} = O \cup M \cup P$$

and the outgoing semantic assertion conflicts with the negative assertion made in P :

$$\overline{O} \vdash z \rightarrow w, \quad z \dashv w$$

Model O cannot then be finalised without violating Constraint 4 above. The imposition of these constraints ensures that such a scenario cannot occur with all four models finalised.

6. Application

In this section, we explain the tools that have been developed to support our use of the language, and hence our approach to semantic interoperability, and discuss their application in the domain of health research informatics.

6.1. Implementation

To support the use of the language, we have developed a ‘metadata catalogue toolkit’, consisting of a metadata datastore or catalogue—matching the schema *Catalogue*—and a collection of software utilities for supporting the development of new models within the catalogue, for automatically creating models of existing software artefacts, and for creating software artefacts from existing models.

The screenshot displays the Metadata Catalogue web interface. On the left, a tree view shows a hierarchy of models under 'Models'. The selected model is 'HES: A&E attendance', which is expanded to show 'Clinical diagnoses'. The right-hand side shows the details for the selected 'Data Element', 'A&E diagnosis: 3 character'. This section includes an 'Aliases' tab, a 'Description' field, a 'Data Type' dropdown set to 'DIAG3_NN (Enumeration)', a table of key-value pairs for the enumeration, a 'Parent Hierarchy' link, a 'Multiplicity' of '0..1', and a 'Classifications' section. Below this, there are tabs for 'Properties', 'Comments', 'Links', 'Summary', and 'Attachments'. The 'Properties' tab is active, showing a table with columns 'Namespace', 'Key', and 'Value'.

Namespace	Key	Value
ox.softeng.metadataloaders.hdf	Technical Field Name	DIAG3_NN
ox.softeng.metadataloaders.hdf	Sensitive	N/A

Fig. 10. A screenshot of the Metadata Catalogue web interface.

The catalogue itself is implemented as a web-based application. A Representational State Transfer (REST) Application Programming Interface (API) allows language-independent communication with other tools and services. A plug-in architecture allows for the integration of additional functionality: in particular, for utilities supporting new types of software artefact, or new data standards.

The present implementation represents the third generation of the technology. In the first generation [5,8], we used the eXist [14] database as a simple store of XML representations of metadata items as individual attributes with arbitrary XML document types. This version of the catalogue did not make use of or enforce the constraints of a data modelling language, and the approach to semantic linkage was through classification, rather than individual assertions of refinement. It was applied successfully in two large projects, but proved difficult to maintain.

The second generation of the technology [6] was developed for the National Institute for Health Research (NIHR) Health Informatics Collaborative: a national programme aimed at facilitating the re-use of routinely-collected hospital data for medical research. This version of the catalogue allowed arbitrary links, and included some of the features of the data modelling language presented here, but without a consistent, formal interpretation.

The design included our principle that models should be the units of context and versioning. However, links were not contained within attributes, classes, or even models, and hence not subject to model finalisation, making it impossible to ensure consistency. Furthermore, it was possible to 'import' arbitrary items across models, without any guarantee that the definition of these items was independent of context. Despite these challenges, the second generation catalogue was taken up by contractors and successfully applied to the delivery of the UK 100,000 Genomes Project [16].

The third generation of the technology was developed from scratch, based on the experience of the previous implementations, and—crucially—developed in parallel with the formal semantics presented above. The result was a catalogue in which it was possible to guarantee consistency of definitions, and to achieve a greater degree of scalability through automation and federation.

Fig. 10 shows the catalogue web application interface. Models, terminologies, transformations, mappings types and conversions are organised and managed according to a folder structure shown on the left-hand side of the screen. The selected model, "HES: A&E attendance" (for 'Hospital Episode Statistics: Accident and Emergency Attendance') is shown expanded to reveal the contained classes. The selected class, "Clinical diagnoses" is displayed in bold.

The right-hand side of the screen shows a selected attribute or 'data element': "A&E diagnosis: 3 character". A textual description is given, along with details of the element's enumerated data type. The hierarchy of the element within its containing class provides links to the containing model and class respectively. The multiplicity of the data element—in this case '0..1'—is shown below.

The catalogue toolkit supports all of the language features described in this paper, and includes import and export functions for a range of languages and systems: MS SQLServer, Oracle, and other databases; XML schema and JSON formats; Excel spreadsheets; UML models; and OWL for ontologies. It supports also a range of healthcare data and metadata standards, including the CDISC ODM (Operational Data Model) [13].

Role-based access control is implemented through administrative and organisational groups, and a notion of model ownership. A folder-based filing system for models, linked to authority, enables collections of models to be managed together. As with any collaborative information system, every edit is logged and maybe subsequently audited; social-media-style commenting allows more informal interactions.

Although the access control built into the application means that a single instance can support multiple organisations, some organisations prefer to manage their own instances, to provide an additional level of security around their metadata. Support for catalogue federation allows these organisations to share definitions with others, model by model, when they are ready. The semantics presented here, including the consistency constraints, has been applied in every application of the third-generation technology.

As an indication of scale, one instance of the catalogue currently holds more than 600 models, with more than 55,000 classes and 370,000 attributes, and 15 major terminologies, containing more than 1.4 m terms. There are currently 14 instances of the catalogue in use, across more than 20 organisations.

6.2. Cardiovascular research

One instance of the catalogue is being used to support a NIHR Health Informatics Collaborative project in cardiovascular medicine. Clinical researchers from five different university hospitals used the tool to define a minimum dataset to address a range of research questions [1]. This dataset was based in part upon an existing national audit—Myocardial Ischaemia National Audit Project (MINAP)—for cardiovascular disease.

The dataset went through a number of iterations, each managed, versioned and distributed through the catalogue. Textual descriptions of each data attribute were added, together with refinement links pointing at the MINAP data specification or the National Health Service (NHS) data dictionary.

An XML schema was automatically generated from the data model, complete with textual annotations. The XML schema export plugin converted primitive data types into their XML equivalents, attributes into simple XML elements, and classes to XML elements of complex type. The schema was presented to data managers at each contributing site, who were tasked with collecting data and uploading it in the prescribed format.

The catalogue was used to generate a relational database implementation for the lead site to store the submitted data. The export plugin produced a series of simple Data Definition Language (DDL) statements, converting data classes into tables and mandatory or optional elements into database columns. The treatment of inheritance, associations and containment employed standard object-relational bridge techniques. Code for decomposing XML submissions into appropriate database 'select' statements was also generated.

6.3. Cancer reporting

Another instance of the catalogue is supporting the re-use of routinely-collected data in cancer research: in particular, the re-use of detailed, statutory reporting data on cancer service delivery.

The Cancer and Outcomes Services Dataset (COSD) is a large dataset containing attributes pertaining to key parts of the patient pathway, specialised for each of the main cancer sites (for example lung, or skin). The specification for this dataset is defined within an XML schema, itself spread over around 380 files, and textual descriptions in an associated Excel spreadsheet. The specification has a supplementary model specifically for pathology data. Both specifications have been through four major versions.

Other datasets, including the Systemic Anti-Cancer Therapy (SACT) dataset and the RadioTherapy Data Set (RTDS) are reported in comma-separated-value (csv) format, or in Microsoft Access format according to a specification provided in a Microsoft Word document. The various monthly reports of these datasets are not considered together at the providing hospitals, neither are they reviewed and re-used for local service improvement.

The catalogue was used to create models of the different versions of the various submission formats, and then to define transformations from these models to a single, unified cancer model—produced separately and imported from a UML modelling tool. Relational database schemas were generated to hold the data as submitted, together with additional staging schemas to facilitate the transformations.

A database implementation of the unified model was also generated, and SQL transformations were used to move data from the staging schemas into this implementation. The transformations were partly generated from the semantic links, but were extended manually with additional procedures; these procedures were then stored in the catalogue for reproducibility and re-use.

The resulting database served as an integrated cancer record, combining information from the different reporting sets to produce a more complete picture of referral, diagnosis, treatment, and outcomes.

7. Discussion

In this paper, we have presented an approach to semantic interoperability, with a formal, mathematical foundation, and argued that it is scalable. Our arguments for scalability are based upon support for compositional development of definitions, the exploitation of structural information in supporting automation, and the ability to support multiple perspectives or

purposes—allowing us to consider two definitions semantically equivalent for one purpose, but semantically distinct for another.

Our approach has the additional advantage of affording a uniform treatment of semantic refinement and interoperability. The fact that two classifiers are connected by a refinement arrow tells us that the properties of the first are enough to guarantee the properties of the second. What this means in practice then depends upon the purpose of the models that contain these classifiers:

- if both models represent artefacts used for data collection, then data collected against the first classifier has all of the properties of data collected against the second;
- if the first represents an artefact used for data collection, and the second represents an intended usage or application, then data collected against the first classifier is suitable for use in the role described by the second;
- if the first represents an artefact used for data collection, and the second represents a data standard, then data collected against the first classifier conforms with the data standard;
- if both models represent data standards, then the arrow tells us that the first standard imposes all of the requirements of the second.

An advantage of working at the level of models, rather than concrete artefacts, is that we are able to define a single, unifying notion of refinement—one arrow captures all of these situations—and hence achieve a uniform approach to semantic interoperability.

The importance of semantic interoperability has been widely recognised, and many different approaches have been developed. Most of these involve agreement upon a single data model, or a single ontology, for a particular domain or ‘community of interest’. These approaches have met with some success, where the meaning of the data that is to be shared is largely unaffected by context, and the nature of the data to be shared remains relatively constant.

The difficulty in achieving a single, unified view was a concern for those working upon the ‘Semantic Web’, where a single ontology was often the ultimate aim. In [20], for example, the authors consider information or views in terms of: *perspectives*, which are typically complementary and stem from an expert’s incomplete view of the world; and *opinions*, which are likely to be collectively inconsistent and reflect an individual’s convictions. A methodology is presented for integrating consistent views into a single ontology.

Similarly in [23], ontologies are registered against a user; the collective facts may be inconsistent or conflicting, but ultimately “... the correct knowledge is taken to be equivalent to the consensual knowledge, and thus the knowledge engineer’s task becomes one of finding consensus within the divergent knowledge of the experts”.

Furthermore, a particular challenge in the application of the semantic web language OWL is the semantics of the *owl:imports* construct. This can be used to combine multiple ontologies, losing the context of each, and potentially introducing conflicts. In [3], the authors extend OWL with \sim -connections in order to achieve modularity when combining ontologies. However, in these approaches, no reasoning is possible in the presence of contradictions: a single knowledge base must remain entirely consistent.

The ISO/IEC 11179/3 standard for metadata registration [19] avoided the trap of aiming to establish a single, agreed model for data, aiming instead to establish a single, agreed model for metadata. This standard was the original inspiration for our approach, and the catalogue language was developed in an attempt to address three particular, decisive shortcomings:

- The standard has no notion of the composition of data items: each attribute is defined separately. This means that every attribute needs to be described in full, and that there is no re-use of textual explanation or logical constraints across the items of a class, or the classes in a model. Furthermore, it means that there is no basis for inferring context from the structural relationships in software artefacts, nor is there any basis for generating those artefacts.
- The standard has no support for links between attributes. Instead, attributes are grouped according to ‘data element concepts’, requiring that all of the definitions involved are considered to be equivalent. This requires the same, impossible degree of coordination as that required for agreement upon a single data model, except that now we are proceeding one attribute at a time.
- The standard supports only a single ontology for the organisation of ‘data element concepts’, meaning that all users of a particular registry must agree upon the same set of relationships between these concepts.

Nevertheless, a number of ISO/IEC 11179/3-compatible metadata repositories or catalogues have been implemented. [24] argues that such tools “should result in fewer errors and less effort required for designing clinical studies”.

In [9], the authors explain how they augmented the ISO/IEC standard with CDISC ODM data models. The resulting catalogue contained around 800,000 terms with semantic annotations, derived from around 5,800 models. They concluded that “Uniform manual semantic annotation of data models is feasible in principle, but requires a large-scale collaborative effort due to the semantic richness of patient data.”

There have been other formally-based approaches platform-independent modelling and analysis. The Heterogeneous Tool Set (HETS) [4] defines a language for the analysis of models and transformations, based upon the modelling language UML and the associated transformation language QVT. The toolset is focussed on the automatic construction of code from models, and supports the verification of some simple structural and non-structural constraints.

HeteroGenius [10] represents a similar approach, although focussed instead upon the analysis of software specifications and the management of proofs. In [2], the authors present an abstract notation suitable for reasoning about refinements in a variety of different formalisms. This category-theoretic approach can be used for building language-independent models, but with a focus upon behavioural properties.

To understand how our notion of semantic refinement corresponds to existing, formal notions of refinement, we may consider each classifier as an abstract data type where we have no information regarding the operations. Our notion of refinement is then consistent with the notion of data refinement for abstract data types, as described in [25]. Where models correspond to artefacts that include operation specifications, we could extend our language to support their description within the catalogue.

Our work thus far has focussed upon establishing semantic interoperability based upon data abstractions of software artefacts. We have identified an approach that can be applied at scale, having support for both automation and compositional, potentially-federated development. We have developed a formal semantics for the language that underpins this approach, and used this semantics in the development of a metadata catalogue that supports multiple, potentially-conflicting perspectives and yet can be guaranteed to provide consistent information.

Our opportunities to apply our work, and much of the funding for the development of the catalogue toolkit, have come from the domain of health research. However, we believe that the approach set out here is equally applicable to data management challenges in other domains: the need for new technologies to support reliable interpretation and re-use of data at scale exists also in areas such as electronic government, supply chain management, and financial regulation.

Acknowledgements

This research has been supported by the National Institute for Health Research (NIHR) Oxford Biomedical Research Centre (BRC). We are grateful to Charles Crichton, Kinga Várnai, and Kerrie Woods for helpful discussions regarding theories of semantic interoperability, to Oliver Freeman, Peter Gammon, Soheil Saifipour, Anshita Shrivastava, Andrew Tsui, and Ayshe Yalmaz for their work on the catalogue toolkit implementation, and to our colleagues in the NIHR Health Informatics Collaborative for helping us to evaluate and develop our technology. A complete implementation of the toolkit is available at metadata-catalogue.org.

References

- [1] A. Kaura, et al., Association of troponin level and age with mortality in 250 000 patients: cohort study across five UK acute care centres, *BMJ Clin. Res.* 367 (November 2019).
- [2] P. Castro, A. Nazareno, Algebraic foundations for specification refinements, in: L. Ribeiro, T. Lecomte (Eds.), *Formal Methods: Foundations and Applications - 19th Brazilian Symposium, SBMF 2016, Natal, Brazil, November 23–25, 2016, Proceedings*, in: *Lecture Notes in Computer Science*, 2016, pp. 112–128.
- [3] B. Cuenca Grau, B. Parsia, E. Sirin, Working with multiple ontologies on the semantic web, in: *International Semantic Web Conference*, in: *Lecture Notes in Computer Science*, vol. 3298, Springer, 2004, pp. 620–634.
- [4] C. Daniel, M. Till, S. Nora, Model-driven engineering in the heterogeneous tool set, in: C. Braga, N. Martí-Oliet (Eds.), *Formal Methods: Foundations and Applications - 17th Brazilian Symposium, SBMF 2014, Maceió, AL, Brazil, September 29–October 1, 2014. Proceedings*, in: *Lecture Notes in Computer Science*, vol. 8941, Springer, 2014, pp. 64–79.
- [5] J. Davies, J. Gibbons, Semantic frameworks: meanings in the architecture, in: *Distributed Computing and Internet Technology*, in: *LNCS*, vol. 5966, 2010, pp. 40–54.
- [6] J. Davies, J. Gibbons, A. Milward, D. Milward, S. Shah, M. Solanki, J. Welch, Domain-specific modelling for clinical research, in: *SPLASH Workshop on Domain-Specific Modelling*, October 2015.
- [7] J. Davies, J. Gibbons, J. Welch, E. Crichton, Model-driven engineering of information systems: 10 years and 1000 versions, *Sci. Comput. Program.* 89B (September 2014) 88–104.
- [8] J. Davies, S. Harris, C. Crichton, A. Shukla, J. Gibbons, Metadata standards for semantic interoperability in electronic government, in: *International Conference on Theory and Practice of Electronic Governance*, Cairo, December 2008.
- [9] M. Dugas, A. Meidt, P. Neuhaus, M. Storck, J. Varghese, ODMedit: uniform semantic annotation for data integration in medicine based on a public metadata repository, *BMC Med. Res. Methodol.* 16 (1) (2016) 65.
- [10] M. Giménez, M. Moscato, C.G. Lopez Pombo, M. Frias, HeteroGenius: a framework for hybrid analysis of heterogeneous software specifications, *arXiv preprint*, arXiv:1401.0974, 2014.
- [11] A. Gomes, M. Oliveira, Formal development of a cardiac pacemaker: from specification to code, in: *Brazilian Symposium on Formal Methods*, Springer, 2010, pp. 210–225.
- [12] M. Hinchey, J. Bowen, *Industrial-Strength Formal Methods in Practice*, Springer Science & Business Media, 2012.
- [13] W. Kuchinke, J. Aerts, S. Semler, C. Ohmann, CDISC standard-based electronic archiving of clinical trials, *Methods Inf. Med.* 48 (05) (2009) 408–413.
- [14] W. Meier, eXist: an open source native XML database, in: A. Chaudhri, M. Jeckle, E. Rahm, R. Unland (Eds.), *Web, Web-Services, and Database Systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 169–183.
- [15] B. Meyer, *Object-Oriented Software Construction*, Prentice Hall, Upper Saddle River, NJ, USA, 2000.
- [16] D. Milward, Model driven data management in healthcare, in: *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development - Vol. 1: MODELSWARD, INSTICC, SciTePress*, 2019, pp. 107–118.
- [17] NASA, Mars Climate Orbiter Mishap Investigation Board Phase I Report, Technical report, NASA, November 1999.
- [18] OMG, UML 2.0 superstructure specification, <http://www.omg.org/cgi-bin/doc?ptc/05-07-04>, 2005.
- [19] International Standards Organisation, International standard for metadata registries, <http://metadata-standards.org/11179/>.
- [20] M. Ribière, R. Dieng-Kuntz, A viewpoint model for cooperative building of an ontology, in: *International Conference on Conceptual Structures*, Springer, 2002, pp. 220–234.
- [21] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W.E. Lorensen, *Object-Oriented Modeling and Design*, vol. 199, Prentice Hall, 1991.
- [22] L. Sprague, G. Oelsner, D. Argue, Challenges with secondary use of multi-source water-quality data in the United States, *Water Res.* 110 (2017) 252–261.

- [23] J. Tennison, N. Shadbolt, APECKS: a tool to support living ontologies, in: *Proceedings of the 11th Knowledge Acquisition Workshop, KAW'98*, 1998, pp. 18–23.
- [24] H. Ulrich, A.-K. Kock, P. Duhm-Harbeck, J. Habermann, J. Ingenerf, Metadata repository for improved data sharing and reuse based on HL7 FHIR, in: *MIE*, 2016, pp. 162–166.
- [25] J. Woodcock, J. Davies, Using Z, www.usingz.com, 1996.