Conceptual Model Interoperability: a Metamodel-driven Approach

Pablo Rubén Fillottrani[1,2] and C. Maria Keet[3]

[1] Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur, Bahía Blanca, Argentina, prf@cs.uns.edu.ar
[2]Comisión de Investigaciones Científicas, Provincia de Buenos Aires, Argentina
[3]Department of Computer Science, University of Cape Town, South Africa, mkeet@cs.uct.ac.za

Abstract. Linking, integrating, or converting conceptual data models represented in different modelling languages is a common aspect in the design and maintenance of complex information systems. While such languages seem similar, they are known to be distinct and no unifying framework exists that respects all of their language features in either model transformations or inter-model assertions to relate them. We aim to address this issue using an approach where the rules are enhanced with a logic-based metamodel. We present the main approach and some essential metamodel-driven rules for the static, structural, components of ER, EER, UML v2.4.1, ORM, and ORM2. The transformations for model elements and patterns are used with the metamodel to verify correctness of inter-model assertions across models in different languages.

1 Introduction

The volume and the need to share existing data sources becomes increasingly important, like in enterprise information integration [11], company mergers and acquisitions [4], scientific collaborations in several fields [1,20,18], e-government initiatives [14,19,21], and in general the broader adoption of the Semantic Web. Interoperability at the level of conceptual models is a key in this goal, in order to maximize the extent to which data can be exchanged while preserving its original meaning. This involves linking, converting, and integrating conceptual models represented in different modelling languages; e.g., when a database back-end is designed with EER, the application layer that uses the database is specified in UML, and the business rules were extracted from the experts using ORM.

Results have been obtained to address this issue. Besides one-off unidirectional algorithms to transform a language, e.g., from ORM to UML [5], several multi-language approaches exist, ranging from linking each model to a graph [7] or description logic language [15] to transformations mediated by a dictionary of common terms [3]. However, these solutions are only partial, for they, among others, omit several constructs (e.g., weak entity types, roles) or modify the language (e.g., by removing datatypes from UML), and therewith have imprecise

'equivalence' mappings or the algorithms are not available. Overall, there is very limited interoperability of conceptual data models in praxis.

To address these issues we have developed an approach that uses a formalised metamodel with a set of modular rules to mediate the linking and transformation of elements in the conceptual models represented in different languages, which simplifies the verification of inter-model assertions and model conversion. The previously developed metamodel [16,17] with all static structural entities (including constraints) of the main conceptual modelling languages—UML v2.4.1, EER, and ORM2—has been formalised and has a table with mappings between terms used in the different languages (e.g., UML association and EER relationship). This is used with a newly specified set of rules from/to the metamodel and within-metamodel conversions to convert model elements and check the validity of inter-model assertions. A major advantage of using a formalised metamodel is that it also can induce a series of transformations/link checks, thanks to the constraints specified in the metamodel. For instance, relating a relationship induces the checking of its roles, of the object types that participate in it, and their identifiers, due to the chain of mandatory participations in the metamodel.

We discuss related works in Section 2 and introduce the metamodel-driven approach in Section 3. Section 4 contains a selection of the rules for the main elements, which is elaborated in Section 5 concerning mapping validations in a broader context. We discuss and conclude in Section 6.

2 Related works

Several papers mainly have proposed transformations from one conceptual modelling language to another, without considering the case of validating inter-model assertions. Nevertheless, it is useful to consider also their approaches, for transformations could be used to check inter-model assertions (out of scope are transformations other than between conceptual models, such as QVT for MOF and ATL for the Eclipse platform).

Venable and Grundy's work [22,10,23] uses a metamodel in the CoCoA graphical language that covers a part of ER and a part of NIAM (a precursor to ORM), and it was implemented in MView and Ponamu. Their metamodel omits, mainly, value types, nested entity types, and composite attributes, and NIAM is forced to have attributes as in ER in the 'integrated' metamodel. Their "dynamic" ad hoc mappings are thus limited, and they have not been made public.

Boyd and McBrien [7] use the Hypergraph Data Model to relate ER, relational, UML, and ORM schemas, and include transformation rules between them through mapping each schema into the graph. The advantage is that it provides a simple irreducible form for schemas that can be used to prove schema equivalence, but it does not consider inter-model assertions and the specification omits roles, aggregation, weak entity types, several constraints, and it removes the data type specification from UML to match EER's partial attribute.

Atzeni et al [2,3] devised a comprehensive approach with extensible automatic schema translations, which has been implemented. Unlike [22], they have

a term dictionary that aids with the transformation, and the translations are produced in Datalog. However, it does not include ORM, the dictionary has only 9 constructs, it lacks metamodel relations and constraints between them, and the system considers model transformations only.

Bowers and Delcambre's framework [6] is a flat representation of schema and data. Its representational language ULD covers only ordinal, set and union class constructs, and cardinality constraints, and it operates at the implementation-level, providing examples for the relational model, XML, RDF, and RDF Schema only. The transformations are handled by Datalog.

Two principal works to relate ORM to UML or ER are [12,5]. Halpin provides diagram element to diagram element mappings and approximations [12] and some conversions from ORM to ER are implemented with undisclosed algorithms. Bollen does provide comprehensive rules for transforming ORM's object types, nested object types, fact types and some constraints into UML diagrams [5]. Those rules are then combined in a sequence of algorithms to transform a ORM conceptual schema into a UML class diagram. While the rules are sound, some of the algorithms exhibit steps and iterations which are not clearly defined leading to ambiguous results.

Fill and Burzynski [9] outline three metamodel-mediated approaches to integrate conceptual models and ontologies, being integration on the level of meta models, by using references in the meta model of the existing conceptual models, and a hybrid of the two, but no details. The tool it is said to be implemented in, ADONIS, now focuses entirely on BPMN.

A different strand of research on unification that does not use a metamodel, is to use one logic formalism for several conceptual modelling languages, notably a Description Logic language [8,13,15]. Different logics are used, however, and they do not cover all features of the language due to the complexity trade-offs made. For instance, in [8] identifiers are absent, and the $\mathcal{DLR}_{ifd}$ used in [15] does not consider the ORM's relationship constraints or UML's aggregation. Also, approximate transformations are not represented.

3 The Metamodel-driven Approaches

The focus is interoperability and integration of conceptual data models represented in different languages, but to be able to assert a link between two entities in different models and *evaluate automatically* whether it is a valid assertion and what it does entail, one has to know what type of entities they are, whether they are the same, and if not, whether one can be transformed into the other for that particular selection. That is, we first need an approach for transforming a model (or a selection thereof) in one language into another. This is depicted in Fig. 1 and illustrated with some sample data. There are three input items at the top, the algorithms on the right, and the two output items at the bottom. In this paper we focus on the rules and algorithms, but they avail of the formalised metamodel and term mapping table to function well. For instance, it needs to

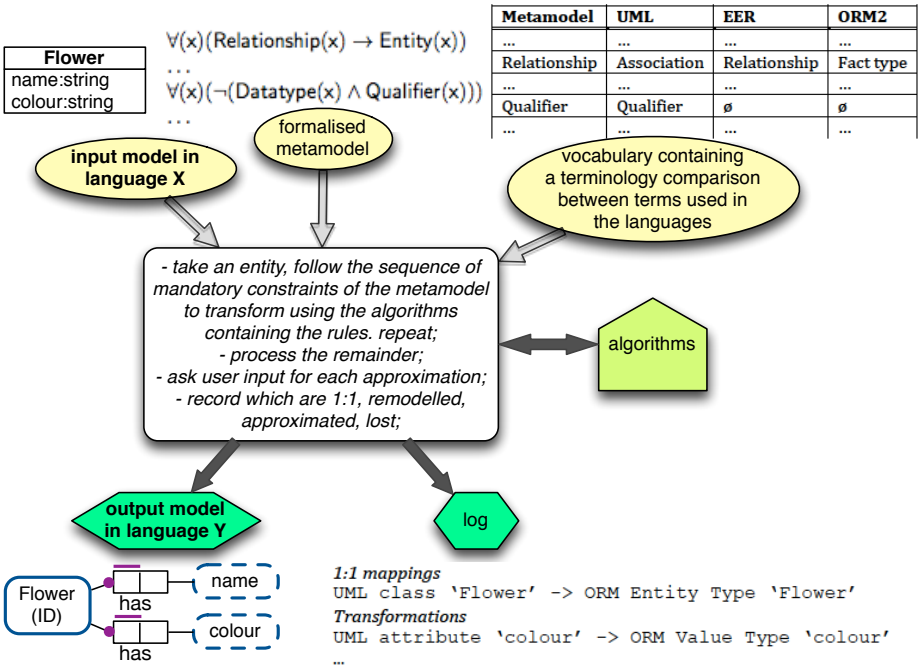

Fig. 1. Approach for transforming a model in one language into another, with some sample data.

recognise that a UML class in the diagram can be mapped 1:1 to a ORM entity type, and transform a UML attribute to an ORM value type.

This knowledge is then used for the inter-model assertions, whose approach is illustrated in Fig. 2. It uses both the formalised metamodel and the algorithms; a structured version of this approach is included as Algorithm 1 in the Appendix. In addition, compared to transformation, it can be run in both directions from one fragment to the other, where one direction is chosen arbitrarily. The next two subsections provide some detail on the formalised metamodel and transformation lists, before we proceed to the rules in Section 4.

3.1 Formalised metamodel and term mappings

The metamodel, described in [16,17] and represented as a set of UML v.2.4 diagrams with annotations, is a consistent conceptual model about the entities and constraints in the selected modelling languages, covering almost all their native features. It aims at representing in a unified way whatever is present in the languages, and several notions from Ontology (philosophy) and ontologies (Artificial intelligence) were used in its development so as to increase understanding of the language features, to reconcile or unify perceived differences, and to improve

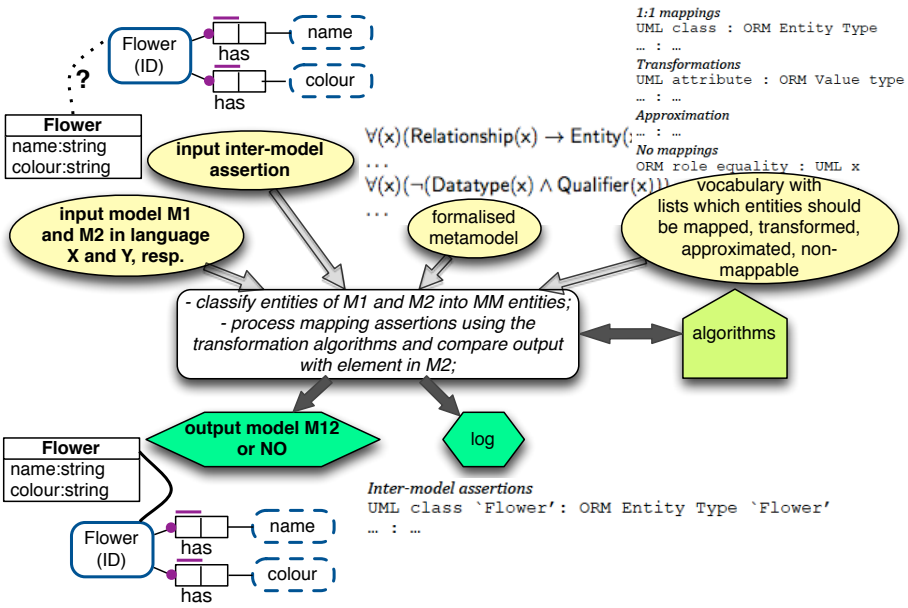

Fig. 2. Approach for adding inter-model assertions, with some sample data. The "algorithms" box is essentially the same as the algorithms in Fig. 1.

the quality of the metamodel; e.g., on attributes and the positionalist nature of relationships [17].

It is formalized in two versions, both available at `http://www.meteck.org/SAAR.html`. The first is a set of function-free first order logic set of formula with equality. Fig. 3 shows a fragment relating relationship, role, and object type with identification constraints. The second is a subset of the first that is representable and approximated in OWL 2 for easy computational use, represented in $\mathcal{SHIQ}(D)$, with some 100 classes, 70 object properties (partially due to encodings of ternaries), and 663 axioms.

The metamodel is complemented with a vocabulary in the form of a list of terminology comparison and conventions of the entities in UML Class Diagrams, EER, and ORM, and their corresponding names in the metamodel (see [17]).

3.2 Categorisation of rules

As the model features are more or less similar across the languages, we have divided them into four groups: 1:1 mappings, transformations, approximations, and those for which there are no alternatives. The four lists largely follow from the metamodel of the static, structural components and constraints [17], although in some cases there is a conceptual equivalence, but not exactly in the representation; e.g., UML and EER both have attributes, but EER does not record the datatype, so they are not 1:1 from an algorithmic viewpoint.

$$\forall(x,y)(\texttt{Contains}(x,y) \rightarrow \texttt{Relationship}(x) \wedge \texttt{Role}(y))$$
$$\forall(x)\exists^{\leq 2}y(\texttt{Contains}(x,y))$$
$$\forall(x)(\texttt{Role}(x) \rightarrow \exists(y)(\texttt{Contains}(y,x)))$$
$$\forall(x,y,z)(\texttt{Contains}(x,y) \wedge \texttt{Contains}(z,y) \rightarrow (x=z))$$
$$\forall(x,y,z)(\texttt{RolePlaying}(x,y,z) \rightarrow \texttt{Role}(x) \wedge \texttt{CardinalityConstraint}(y) \wedge \texttt{EntityType}(z))$$
$$\forall(x)(\texttt{Role}(x) \rightarrow \exists(y,z)(\texttt{RolePlaying}(x,y,z)))$$
$$\forall(x,y,z,v,w)(\texttt{RolePlaying}(x,y,z) \wedge \texttt{RolePlaying}(x,v,w) \rightarrow (y=v) \wedge (z=w))$$
$$\forall(x,y,z,v,w)(\texttt{RolePlaying}(x,y,z) \wedge \texttt{RolePlaying}(v,y,w) \rightarrow (x=v) \wedge (z=w))$$
$$\forall(x)(\texttt{CardinalityConstraint}(x) \rightarrow \exists(y)(\texttt{MinimumCardinality}(x,y) \wedge \texttt{Integer}(y)))$$
$$\forall(x)(\texttt{CardinalityConstraint}(x) \rightarrow \exists(y)(\texttt{MaximumCardinality}(x,y) \wedge \texttt{Integer}(y)))$$
$$\forall(x,y)(\texttt{ReifiedAs}(x,y) \rightarrow \texttt{Relationship}(x) \wedge \texttt{NestedObjectType}(y))$$
$$\forall(x)(\texttt{NestedObjectType}(x) \rightarrow \exists(y)(\texttt{ReifiedAs}(x,y)))$$
$$\forall(x,y,z)(\texttt{ReifiedAs}(x,y) \wedge \texttt{ReifiedAs}(z,y) \rightarrow (x=z))$$
$$\forall(x,y,z)(\texttt{ReifiedAs}(x,y) \wedge \texttt{ReifiedAs}(x,z) \rightarrow (y=z))$$
$$\forall(x,y)(\texttt{ReifiedAs}(x,y) \rightarrow \forall(z,w)(\texttt{Contains}(x,z) \leftrightarrow \texttt{RolePlaying}(z,w,y)))$$
$$\forall(x,y)(\texttt{Identifies}(x,y) \rightarrow (\texttt{IdentificationConstraint}(x) \wedge \texttt{ObjectType}(y)))$$
$$\forall(x)(\texttt{IdentificationConstraint}(x) \rightarrow \exists(y)(\texttt{Identifies}(x,y)))$$
$$\forall(x,y,z)((\texttt{Identifies}(x,y) \wedge \texttt{Identifies}(x,z)) \rightarrow (y=z))$$
$$\forall(x)(\texttt{ObjectType}(x) \rightarrow \exists(y)(\texttt{Identifies}(y,x)))$$
$$\forall(x,y,z)((\texttt{DeclaredOn}(x,y) \wedge \texttt{DeclaredOn}(x,z) \wedge \texttt{IdentificationConstraint}(x) \wedge (\neg(y=z))) \rightarrow (\texttt{ValueProperty}(y) \leftrightarrow \neg\texttt{AttributiveProperty}(z)))$$
$$\forall(x)(\texttt{IdentificationConstraint}(x) \rightarrow \exists(y)(\texttt{DeclaredOn}(x,y)))$$
$$\forall(x,y)((\texttt{DeclaredOn}(x,y) \wedge \texttt{SingleIdentification}(x)) \rightarrow (\texttt{Attribute}(y) \vee \texttt{ValueType}(y)))$$
$$\forall(x)(\texttt{SingleIdentification}(x) \rightarrow \exists(y)(\texttt{DeclaredOn}(x,y)))$$
$$\forall(x,y,z)((\texttt{SingleIdentification}(x) \wedge \texttt{DeclaredOn}(x,y) \wedge \texttt{DeclaredOn}(x,z)) \rightarrow (y=z))$$

Fig. 3. A fragment of the metamodel FOL formalization.

1:1 Mappings. The mappings are those where the elements are the same, and the conversion are straightforward single steps. They are: Relationships (n-ary, with $n \geq 2$), Role, Object type and Associative object type, Subsumption (class and relationship), Disjoint roles, Disjoint entity types, Subset constraint, Object type cardinality, Completeness (classes), Mandatory.

Transformations. Transformations are those where the elements are essentially the same, but not from a syntax viewpoint and therefore require a set of steps that should be treated as one atomic rule. They are: UML (dimensional) Attribute from/to ORM (dimensional) Value Type, UML attribute/ORM value type to and EER (dimensional) Attribute, EER Weak entity type to its ORM version, EER Multivalued Attribute to separate object types in UML and ORM, ORM Value Type to UML and ER attribute, Internal Identifier, Attributive property cardinality, Single identification.

Approximations. The core distinction between transformations and approximations, is that the latter includes a choice point with input from the user or some arbitrary (modifiable) default value may be used; hence, firing the same rule in the same situation twice does not necessarily lead to the same outcome. It is up to the modeller to accept approximations or not. They are: patterns for UML qualified association, EER Weak entity (type with additional relationship), and ORM external identifier when it suits, as depicted in Fig. 4; Role value constraints (with subclasses); UML's composite and shared aggregate with named relationship/fact type; composite attribute; EER (dimensional) Attribute to UML attribute/ORM value type;

Entities for which there are no alternatives. The three families of languages do not have the same expressiveness even at the ontological level, and some of those

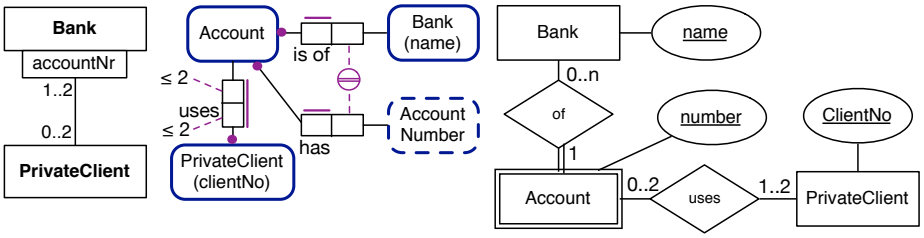

Fig. 4. Patterns for approximations between UML's qualified association, EER weak entity type, and ORM's external identifier.

features cannot be represented or approximated in the other language. They are: UML to ORM: missing inclusive mandatory; EER to ORM: inclusive mandatory; UML to EER: disjunctive mandatory, qualified Identifier, value constraints; ORM to UML: compound cardinality, value comparison, role equality, disjoint relationships, relationship equality, join subset, join disjointness, join equality, all relationship constraints; ORM to EER: disjunctive mandatory, compound cardinality, value comparison, value type constraints, role equality, disjoint relationships, relationship equality, join subset, join disjointness, join equality, all relationship constraints. We do not consider them further in this paper.

4 Interoperability rules

In principle, there are two choices for specifying the interoperability rules: create a mesh between the languages, or do it via the metamodel. If we have n conceptual data modeling languages and assuming there were only simple 1:1 mappings or when one glosses over some details, then the former option will require $n!$ rules while the later only $2n$ rules. We already know there are not only simple mappings, so the lower bound of $2n$ will increase due to the intra-metamodel rules to transform entities, such as an attribute into a value type and vice versa. Overall, while the difference in rules might not be large when considering only three languages, one must note that they come in different flavours and versions. The use of the intermediate metamodel helps to reuse rules while focusing on the real changes. For example, a UML v2.0 attribute to an ORM value type mapping can use the same intra-metamodel transformation, but with mesh-transformations, a new one would have to be added. Therefore, we will use the metamodel-mediated rules. This does require annotations to the entities in the metamodel, to the effect that the algorithm checks the annotation what to do: map straight to the entity in the other language, or perform a transformation or approximation first within the metamodel.

1:1 mapping rules and the metamodel For the 1:1 mappings, this amounts to straightforward rules, and only a few are shown; the rest follows the same pattern. We abbreviate the metamodel as MM in the following rules.

(OT) Object Type

(O1) Class $\xRightarrow{\text{UML to MM}}$ Object Type
in: `Class`
out: `Class` → Object Type

(1O) Object Type $\xRightarrow{\text{MM to UML}}$ Class
in: Object Type
out: Object Type → `Class`

(xOx) Likewise for the other 1:1 mappings between Class, Entity type and Entity type, with (O2) $\xRightarrow{\text{ORM to MM}}$; (2O) $\xRightarrow{\text{MM to ORM}}$; (O3) $\xRightarrow{\text{EER to MM}}$; (3O) $\xRightarrow{\text{MM to EER}}$.

(Rol) Role

(Ro1) Association end $\xRightarrow{\text{UML to MM}}$ Role
in: `AssociationEnd`
out: `AssociationEnd` → Role

(1Ro) Role $\xRightarrow{\text{MM to UML}}$ `AssociationEnd`
in: Role
out: Role → `AssociationEnd`

(xRox) Likewise for the other 1:1 mappings of Role and Relationship component, with (Ro2) $\xRightarrow{\text{ORM to MM}}$; (2Ro) $\xRightarrow{\text{MM to ORM}}$; (Ro3) $\xRightarrow{\text{EER to MM}}$; (3Ro) $\xRightarrow{\text{MM to EER}}$.

(Rel) Relationship

(R1) Association $\xRightarrow{\text{UML to MM}}$ Relationship
in: `Association`(`AssociationEnd` : `Class`, `AssociationEnd` : `Class`)
out: `AssociationEnd` → Role *// i.e., using (Ro1)*
out: `Association` → Relationship
out: `Class` → Object Type *// i.e., using (O1)*
out: Relationship(Role:Object type, Role:Object Type)

(1R) Relationship $\xRightarrow{\text{MM to UML}}$ `Association`
in: Relationship(Role:Object type, Role:Object Type)
out: Role → `AssociationEnd` *// i.e., using (1Ro)*
out: Relationship → `Association`
out: Object Type → `Class` *// i.e., using (1O)*
out: `Association`(`AssociationEnd` : `Class`, `AssociationEnd` : `Class`)

(xRx) Likewise for the other 1:1 mappings of Fact type and Relationship, with (1R) $\xRightarrow{\text{MM o UML}}$; (R2) $\xRightarrow{\text{ORM to MM}}$; (2R) $\xRightarrow{\text{MM to ORM}}$; (R3) $\xRightarrow{\text{EER to MM}}$; (3R) $\xRightarrow{\text{MM to EER}}$.

A mapping $\xRightarrow{\text{UML to ORM}}$ for a class `C` in UML model M_1 and generating an entity type for model transformation to ORM model M_2 then is simply composed of the component-rules. For instance, for an inter-model assertion:

GenOT Class $\xRightarrow{\text{UML to ORM}}$ Entity type
in: `C`
out: (O1)

out: (2O) *// i.e., an ORM* `EntityType` *named* `C`

For a relationship mapping $\stackrel{\text{UML to EER}}{\Longrightarrow}$, with `A` the association in the UML model asserted to be equivalent to some relationship `R` in an EER model, the following set of rules apply when verifying the mapping is correct:

(MapR) Association $\stackrel{\text{UML to ER}}{\Longrightarrow}$ Relationship
in: $\texttt{A}(\texttt{ae}_1 : \texttt{C}_1, \texttt{ae}_2 : \texttt{C}_2)$
out: (R1)
out: (3R)
out: match pattern out(3R) with $\texttt{R}(\texttt{rc}_1 : \texttt{E}_1, \texttt{rc}_2 : \texttt{E}_2)$

To check the validity of the mapping, one also could have started with the EER $\texttt{R}(\texttt{rc}_1 : \texttt{E}_1, \texttt{rc}_2 : \texttt{E}_2)$ and work towards $\texttt{A}(\texttt{ae}_1 : \texttt{C}_1, \texttt{ae}_2 : \texttt{C}_2)$ using (R3) and (1R). The generation of a new model in another language and checking of an asserted inter-model relation for the other 1:1 mappings listed in the previous section follow a similar pattern and is omitted for brevity.

Transformations We describe two transformations, which are arguably the most important for they are used most widely. It follows the same approach as with the 1:1 mappings, but the rules become increasingly more elaborate, and for ease of comprehension, we have changed the type of arrow.

To handle a model generation or mapping for a UML attribute and ORM value type, we first need to declare their respective mappings into the metamodel, and then the transformation at the level of the metamodel. This is described in the next set of rules, where the (xDx) rules for datatypes are specified alike those for object types, with the same naming scheme.

(Att) Attributive property
(A1) Attribute $\stackrel{\text{UML to MM}}{\longmapsto}$ Attribute
in: $\texttt{Attribute}(\texttt{Class}, \texttt{DataType})$
out: (O1)
out: (D1)
out: `Attribute` $\rightarrow$ Attribute
out: Attribute(Object type, Data type)
(1A) Attribute $\stackrel{\text{MM to UML}}{\longmapsto}$ Attribute
.... *// steps in (A1) in reverse order*
(VT) Value type
(V1) Value type $\stackrel{\text{ORM to MM}}{\longmapsto}$ Value type
in: $\texttt{ValueType} \wedge \texttt{mapped\_to}(\texttt{ValueType}, \texttt{DataType})$
out: (D1)
out: `mapped_to` $\rightarrow$ mapped_to
out: `ValueType` $\rightarrow$ Value type
out: ValueType $\wedge$ mapped_to(Value type, Data type)
(1V) Value type $\stackrel{\text{ORM to MM}}{\longmapsto}$ Value type
.... *// steps in (V1) in reverse order*

(Att-VT) Attribute and Value type conversions
(Att-to-VT) Attribute $\overset{\text{MM}}{\longmapsto}$ Value type
in: Attribute(Object type, Data type)
out: (D1)
out: Role
out: Relationship
out: mapped_to
out: Attribute → Value type
out: Relationship(Role:Object type, Role:Value type)
out: mapped_to(Value type, Data type)
(VT-to-Att) Value type $\overset{\text{MM}}{\longmapsto}$ Attribute
in: Value type ∧ mapped_to(Value type, Data type)
out: (D1)
out: Object type
out: ValueType → Attribute
out: Attribute(Object type, Data type)

It is now possible to generate an ORM value type from an attribute in a UML diagram, and vv., and, in a similar fashion, to verify whether an inter-model assertion between a particular UML attribute and ORM value type is correct (at least structurally). This is specified in the following two rules.

GenVT Attribute $\overset{\text{UML to ORM}}{\longmapsto}$ Value type
in: $\mathtt{A}(\mathtt{C},\mathtt{D})$
out: (A1)
out: (Att-to-VT)
out: (2R)
out: (1V) *// i.e., an ORM model with* $\mathtt{F}(\mathtt{r_c}:\mathtt{C},\mathtt{r_v}:\mathtt{V})$, $\mathtt{V}$ *and* $\mathtt{mapped\_to}(\mathtt{V},\mathtt{D})$

MapVTAtt Value type $\overset{\text{ORM to UML}}{\longmapsto}$ Attribute
in: $\mathtt{V} \wedge \mathtt{mapped\_to}(\mathtt{V},\mathtt{D})$
out: (V1)
out: (VT-to-Att)
out: (1A) *// i.e., a UML Class Diagram with* $\mathtt{A}(\mathtt{C},\mathtt{D})$
out: match pattern out(1A) with attribute declaration in the UML diagram

These basic pieces can, in turn, be used for more complex transformations and approximations (illustrated below).

Approximations As mentioned above, approximates contain a 'choice' step that requires input from the user to complete the transformation. Such choice points in the rules are indicated in italics. We select the rules for identifiers in EER and ORM, as they are important in a model, and illustrate it with the case of simple (single attribute) identifier. To be able to do so, the attribute mapping from EER is introduced first; the mandatory and cardinality constraint (a 1:1) are straight-forward mappings into and from the metamodel and have the same naming pattern, i.e., M1, 1M etc, and C1, 1C etc, with MinimumCardinality abbreviated as mic and MaximumCardinality as mac.

(Att) Attribute
  (Ae1) Attribute $\rightsquigarrow_{\text{EER to MM}}$ Attribute
   in: `Attribute(Class, __)`
     out: (O1)
     out: __ → *choose a* DataType
     out: `Attribute` → Attribute
     out: Attribute(Object type, Data type)
  (1Ae) Attribute $\rightsquigarrow_{\text{MM to EER}}$ Attribute
   in: Attribute(Object type, Data type)
     out: (O1)
     out: Attribute → `Attribute`
     out: DataType → __
     out: `Attribute(Class, __)`

With these rules, one can generate, e.g., an EER single attribute identifier from an ORM reference scheme, and vv., and confirm a mapping between the two; one of the four options are declared in the following rule set.

MapSID ORM reference scheme $\rightsquigarrow_{\text{ORM to EER}}$ EER single attribute identifier
 in: $\texttt{FT}(\texttt{r}_\texttt{e} : \texttt{E}_1, \texttt{r}_\texttt{v} : \texttt{V}) \wedge \texttt{mapped\_to}(\texttt{V}, \texttt{D}) \wedge \texttt{M} \wedge \texttt{C}(\texttt{mic} = 1, \texttt{mac} = 1)$
  out: (O2) *// ORM entity type into MM object type*
  out: (V1) *// ORM value type into MM value type*
  out: (M2) *// ORM mandatory into MM mandatory*
  out: (C2) *// ORM cardinality into MM cardinality*
  out: (VT-to-Att) *// MM conversion value type to attribute*
  out: (3O) *// MM object type into entity type* `E` *of EER*
  out: (1Ae) *// generate EER Diagram attribute:* `A(E, __)`
  out: (3M) *// MM mandatory into mandatory of EER*
  out: (3C) *// MM cardinality into cardinality of EER*
  out: match pattern out(1Ae,3M,3C) with single identifier declaration in the EER diagram

Arguably, it looks like one might be able to do this more succinctly by defining the notion of identifier-using-an-attribute and identifier-using-a-value-type and to use that in the transformation and create different versions of (Atto-to-VT) and (VT-to-Att) that include the mandatory 1:1 constraints. However, this also requires duplications that cannot be isolated, and the above option is then the more transparent one.

5 Validating mappings with the metamodel and rules

The metamodel is useful for creating less, and more efficient, mapping and transformation rules, but this is not its only advantage. It can drive the validation of mappings and the generation of model transformations thanks to the constraints declared in the metamodel. Consider again the centre-part of Fig. 2 with its "process mapping assertions using the transformation algorithms". The

approach takes as input two models (M_1 and M_2), an inter-model assertion (e.g., a UML binary association R_1 and an ORM fact type R_2, the look-up list with the mappings, transformation, approximations, and the non-mappable elements (see Section 3.2), and the formalised metamodel (see Fig. 3). Once the model elements of M_1 and M_2 are classified in terms of the metamodel, the mapping validation process start, which goes through several steps, depending on what is asserted to be a mapping. This is illustrated for a R_1 to R_2 mapping.

Step 1. It can be seen from the vocabulary that association and fact type correspond to Relationship in the metamodel, and thus enjoy a 1:1 mapping. The ruleset that will be commenced with are R1 from UML to the metamodel and 2R to OMR's fact type.

Step 2. R1 and 2R refer to Role and Object type of the metamodel. The metamodel states that there must be at least 2 contains relations from Relationship to Role (Fig. 3, line 2). There are 2, which each cause the role-rules to be evaluated, with Ro1 of R_1's two association ends and 2Ro for ORM's roles.

Step 3. The metamodel states that Role must participate in the relationship rolePlaying (Fig. 3, lines 5 and 6), and it has a participating Object type (possibly a subtype thereof) and optionally a Cardinality constraint. They also have 1:1 mappings, which is straight-forward for cardinality (1C and C2).

Step 4. The class participating in R_1 causes its rules to be evaluated, being an O1 to Object type and 2O to ORM's entity type.

Step 5. Each Object type must have at least one Identification constraint (Fig. 3, last 9 lines), be this an internal one or an external one, and involving one or more attributes or value types (which one it is has been determined by the original classification). If it is a Single identification, then a rule similar to MapSID (see previous section) is called and executed (which, in turn, calls the Att-to-VT rule and the use of Data type).

There are no further mandatory constraints from the 'chain' from Relationship to Role to Object type to Single identification (that, in turn, consults Attribute and Data type for the 'UML to ORM' example here). The sequence readily becomes longer if the participating object type is actually one of its subtypes: e.g., Nested object type has a mandatory constraint such that it must be related to the Relationship it objectifies, which causes the verification to go through a new sequence of steps following the chain of mandatory constraints. If the relationship would have been a subtype of Relationship, then the four stages above will have been specified more precisely correspondingly (e.g., adding attributes). Because an object type need not to have non-identifier attributes, a check for the presence of this entity has to be added.

Consider again Fig. 2 and the possible mapping between the UML class `Flower` and ORM's entity type `Flower`, which can be validated from UML to ORM or ORM to UML. If the former then, like Step 4, above, 1O and O2 is called and, like in Step 5, the identifier. The mapping can work, provided one admits to using the UML internal identifier as candidate for single identifier (reference scheme) in ORM. Executing it in the other direction from ORM to UML, one

could include a choice point and add ORM's reference scheme as a UML user-defined identifier (indicated with a {id} after the name of the attribute).

6 Discussion and Conclusions

We have presented a metamodel-driven approach for model transformations and inter-model assertions where the models are represented in different languages. Besides the input model and a mapping table, it uses a formalised metamodel to direct a sequence of the language transformations, and it uses a set of mapping, transformation, and approximation rules to carry it out. We presented a selection of the rules, in particular considering the static structural, components of ER, EER, UML v2.4.1, ORM, and ORM2. The transformations for model elements and patterns, in turn, are used with the metamodel to verify correctness of inter-model assertions across models in different languages. An next step is to implement them and evaluate them with actual conceptual models.

The metamodel-driven approach requires quite an investment upfront, first and foremost in terms of designing the metamodel. Its formalization ironed out some duplications and enabled the capturing also of the textual constraints. While one could have chosen to remain at the term dictionary level for the entities in the conceptual data modelling languages, alike [3], this extra work pays off in increased coverage of features, higher precision of mappings, as well as explicit approximations where asked for by the user. In addition, it makes the whole procedure more transparent, and the rules are usable essentially for both transformations and for validations of inter-model mapping assertions.

The overall fine-grained granular and modular approach with the rules for the transformation and mappings also increase the reusability of the rules across the various larger-sized mappings, and can be used to construct a set of transformation steps or larger 'chunks' of the model, alike for the qualified associations, external uniqueness, and weak entity types in Fig. 4. It does not, however, readily offer a single procedure for testing schema equivalence, which is not only out of scope of the current work, but also extremely unlikely in the case of inter-model assertions, for the simple reasons that that is typically not the aim, and the intersection of entities that are truly the same across the three conceptual modelling language families (UML, EER, and ORM) is small (see also figures 1 and 2 in [17]).

These sets of rules not only contributes to the comprehension of differences between heterogenous conceptual models, they also serve as the formal framework for a tool supporting the design, management and integration of conceptual schemas and ontologies in different modelling languages. Even though it is not always possible to find exact matches between entities in the different models, the approximations rules will help users to find corresponding alternatives.

Acknowledgements This work is based upon research supported by the National Research Foundation of South Africa (Project UID: 90041) and the Argentinian Ministry of Science and Technology.

References

1. See the list of collaborations (2014), http://www.tipharma.com/
2. Atzeni, P., Cappellari, P., Torlone, R., Bernstein, P.A., Gianforme, G.: Model-independent schema translation. VLDB Journal 17(6), 1347–1370 (2008)
3. Atzeni, P., Gianforme, G., Cappellari, P.: Data model descriptions and translation signatures in a multi-model framework. AMAI Mathematics and Artificial Intelligence 63, 1–29 (2012)
4. Banal-Estanol, A.: Information-sharing implications of horizontal mergers. International Journal of Industrial Organization 25(1), 31–49 (2007)
5. Bollen, P.W.L.: A formal ORM-to-UML mapping algorithm (2002), http://arno.unimaas.nl/show.cgi?fid=46, research memo RM 02/016, Faculty of Economics and Business Administration, University of Maastricht
6. Bowers, S., Delcambre, L.M.L.: Using the uni-level description (ULD) to support data-model interoperability. Data & Knowledge Engineering 59(3), 511–533 (2006)
7. Boyd, M., McBrien, P.: Comparing and transforming between data models via an intermediate hypergraph data model. J. on Data Semantics IV, 69–109 (2005)
8. Calvanese, D., Lenzerini, M., Nardi, D.: Unifying class-based representation formalisms. Journal of Artificial Intelligence Research 11, 199–240 (1999)
9. Fill, H.G., Burzynski, P.: Integrating ontology models and conceptual models using a meta modeling approach. In: Proc. of 11th Int. Protégé Conference (2009), amsterdam. 2009
10. Grundy, J., Venable, J.: Towards an integrated environment for method engineering. In: Proceedings of the IFIP TC8, WG8.1/8.2 Method Engineering 1996 (ME'96). vol. 1, pp. 45–62 (1996)
11. Halevy, A.Y., Ashish, N., Bitton, D., Carey, M.J., Draper, D., Pollock, J., Rosenthal, A., Sikka, V.: Enterprise information integration: successes, challenges and controversies. In: Özcan, F. (ed.) SIGMOD Conference. pp. 778–787. ACM (2005)
12. Halpin, T.: Information Modeling and Relational Databases. San Francisco: Morgan Kaufmann Publishers (2001)
13. Hofstede, A.H.M.t., Proper, H.A.: How to formalize it? formalization principles for information systems development methods. Information and Software Technology 40(10), 519–540 (1998)
14. Hovy, E.: Data and knowledge integration for e-government. In: Digital Government, pp. 219–231. Springer (2008)
15. Keet, C.M.: Ontology-driven formal conceptual data modeling for biological data analysis. In: Elloumi, M., Zomaya, A.Y. (eds.) Biological Knowledge Discovery Handbook: Preprocessing, Mining and Postprocessing of Biological Data, chap. 6, pp. 129–154. Wiley (2013)
16. Keet, C.M., Fillottrani, P.R.: Structural entities of an ontology-driven unifying metamodel for UML, EER, and ORM2. In: Proc. of MEDI'13. LNCS, vol. 8216, pp. 188–199. Springer (2013), sept. 25-27, 2013, Amantea, Calabria, Italy
17. Keet, C.M., Fillottrani, P.R.: Toward an ontology-driven unifying metamodel for UML class diagrams, EER, and ORM2. In: Proc. of ER'13. LNCS, vol. 8217, pp. 313–326. Springer (2013), 11-13 Nov., 2013, Hong Kong
18. Louie, B., Mork, P., Martin-Sanchez, F., Halevy, A., Tarczy-Hornoch, P.: Data integration and genomic medicine. J. of Biomedical Informatics 40(1), 5–16 (2007)
19. Mendes Calo, K., Cenci, K.M., Fillottrani, P.R., Estevez, E.C.: Information sharing – benefits. Journal of Computer Science & Technology 12(2), 49–55 (2012)

20. Nelson, E.K., Piehler, B., Eckels, J., et al.: Labkey server: an open source platform for scientific data integration, analysis and collaboration. BMC bioinformatics 12(1), 71 (2011)
21. United Nations Department of Economic and Social Affairs: United Nations E-Government Survey 2010 – Leveraging e-government at a time of financial and economic crisis. Tech. Rep. ST/ESA/PAD/SER.E/131, United Nations (2010), http://unpan3.un.org/egovkb/global_reports/10report.htm
22. Venable, J., Grundy, J.: Integrating and supporting Entity Relationship and Object Role Models. In: Proc. of ER'95. LNCS, vol. 1021, pp. 318–328. Springer (1995)
23. Zhu, N., Grundy, J., Hosking, J.: Pounamu: a metatool for multi-view visual language environment construction. In: IEEE Conf. on Visual Languages and Human-Centric Computing 2004 (2004)

Appendix

Algorithm 1: Overview checking inter-model assertions.

input: model M_1 and model M_2, represented in languages L_1 and L_2; intermodel equivalence assertions
for *each entity* $e \in M_1, M_2$ **do**
  classify e according to metamodel entities in the vocabulary
end
for *each equivalence assertion* $e_1 \equiv e_2, e_1 \in M_1, e_2 \in M_2$ **do**
  if $type(e_1) \xRightarrow{L_1 \text{ to MM to } L_2} type(e_2) \in$ *1:1Mappings* **then**
    `// there is a corresponding 1:1 mapping`
    call relevant Algorithm (set of mapping rules);
  else
    `// then lookup transformations`
    if $type(e_1) \xmapsto{L_1 \text{ to MM to } L_2} type(e_2) \in$ *Transformations* **then**
      call relevant Algorithm;
    else
      `// offer user approximation`
      if $type(e_1) \rightsquigarrow_{L_1 \text{ to } L_2} type(e_2) \in$ *Approximations* **then**
        Data: Ask whether the user would accept an approximation; $a \leftarrow$ answer
        if $a ==$ *yes* **then**
          call relevant Algorithm;
        else
          output: *"There is no accepted approximations from e_1 to e_2. Your asserted link will be removed. "*
        end
      else
        output: *"There is no transformation nor approximation for e_1 and e_2. Your asserted link is invalid, and will be removed."*
      end
    end
  end
end
run reasoner on combined model;
if *reasoner == ok* **then**
  return "The assertions are logically correct."
else
  return "The models together with the assertions resulted in an inconsistency. You must revise and run the procedure again."
end