GeRoMe: A Generic Role Based Metamodel for Model Management

David Kensche[1], Christoph Quix[1], Mohamed Amine Chatti[1], and Matthias Jarke[1,2]

[1] RWTH Aachen University, Informatik V (Information Systems), 52056 Aachen, Germany
[2] Fraunhofer FIT, Schloss Birlinghoven, 53574 St. Augustin, Germany
{kensche, quix, chatti, jarke}@cs.rwth-aachen.de

Abstract. The goal of *Model Management* is the development of new technologies and mechanisms to support the integration, evolution and matching of models. Such tasks are to be performed by means of a set of model management *operators* which work on models and their elements, without being restricted to a particular metamodel (e.g. the relational or UML metamodel).

We propose that generic model management should employ a generic metamodel (GMM) which serves as an abstraction of the features of particular metamodels while preserving the semantics of its different elements. A naive generalization of the elements of concrete metamodels in generic metaclasses would loose some of the specific features of the metamodels, or yield a prohibitive number of metaclasses in the GMM. To avoid these problems, we propose the *Generic Role Based Metamodel GeRoMe* in which each model element is *decorated* with a set of role objects that represent specific properties of the model element. Roles may be added to or removed from elements at any time, which enables a very flexible and dynamic yet accurate definition of models.

Roles constitute to operators different views on the same model element. Thus, operators concentrate on features which affect their functionality but may remain agnostic about other features. Consequently, these operators can use polymorphism and have to be implemented only once using *GeRoMe*, and not for each specific metamodel. We verified our results by implementing *GeRoMe* and a selection of model management operators using our metadata system ConceptBase.

1 Introduction

Design and maintenance of information systems require the management of complex models. Research in *model management* aims at developing technologies and mechanisms to support the integration, merging, evolution, and matching of models. These problems have been addressed for specific modeling languages for a long time. Model management has become an active research area recently, as researchers now address the problem of *generic* model management, i.e. supporting the aforementioned tasks without being restricted to a particular modeling language [3, 4]. To achieve this goal, the definition of a set of *generic* structures representing models and the definition of *generic* operations on these structures are required.

According to the IRDS standard [10], metamodels are *languages* to define models. Examples for metamodels are *XML Schema* or the *UML Metamodel*. Models are the

R. Meersman and Z. Tari (Eds.): CoopIS/DOA/ODBASE 2005, LNCS 3761, pp. 1206–1224, 2005.

description of a concrete application domain. Within an (integrated) information system, several metamodels are used, a specific one for each subsystem (e.g. DB system, application). Thus, the management of models in a generic way is necessary.

1.1 The Challenge: A Generic Mechanism for Representing Models

This paper addresses the first challenge mentioned in [4], the development of a mechanism for representing models. Since the goal is the support of *generic* model management, this has to be done in some generic way. Currently, model management applications often use a generic graph representation but operators have to be aware of the employed metamodel [5, 8, 13]. While a graph representation is often sufficient for the purpose of finding correspondences between schemas, it is not suitable for more complex operations (such as merging of models) as it does not contain detailed semantic information about relationships and constraints. For example, in [15] a generic (but yet simple) metamodel is used that distinguishes between different types of associations in order to merge two models. A more detailed discussion about the related work on the representation of models is given in section 2.

The intuitive approach to develop a truly generic metamodel (GMM) identifies abstractions of the metaclasses of different metamodels. Its goal is to define a comprehensive set of generic metaclasses organized in an inheritance lattice. Each metaclass in a given concrete metamodel then has to be mapped to a unique metaclass of the GMM.

The sketched approach exhibits a prohibitive weak point: elements of particular metamodels often have semantics that overlap but is neither completely different nor equivalent. For example, a generic Merge operator has to merge elements such as classes, relations, entity types and relationship types. All of these model elements can have attributes and should therefore be processed by the same implementation of an operator. In this setting, such polymorphism is only possible if the given model elements are represented by instances of the same metaclass in the GMM, or at least by instances of metaclasses with a common superclass. Thus, one has to choose the features of model elements which are combined in one metaclass.

Actually, in each metamodel there may be elements incorporating an entirely new combination of such aspects. One approach to cope with this problem is to focus on the "most important" features of model elements while omitting such properties which are regarded as less important. But to decide which properties are important and which are not results in loss of information about the model.

All properties of model elements could be retained if the GMM introduced a set of metaclasses as comprehensive as possible and combined them with multiple inheritance such that any combination of features is represented by a distinct metaclass. Despite the modeling accuracy of such a GMM, it will suffer from another drawback, namely that it leads to a combinatorial explosion in the number of sparsely populated intersection classes which add no new state.

1.2 Our Solution: Role Based Modeling

In such cases, a role based modeling approach is much more promising. In role based modeling, an object is regarded as playing roles in collaborations with other objects.

Applied to generic metadata modeling this approach allows to *decorate* a model element with a combination of multiple predefined aspects, thereby describing the element's properties as accurately as possible while using only metaclasses and roles from a relatively small set. In such a GMM, the different features of a model element (e.g. it is not only an *Aggregate* but also an *Association*) are only different views on the same element. During model transformations, an element may gain or lose roles, thereby adding and revoking features. Thus, the combinatorial explosion in the number of metaclasses is avoided but nevertheless most accurate metadata modeling is possible.

Therefore, the GMM proposed in this work retains these characteristics by employing the *role based* modeling approach, resulting in the *Generic Role Based Metamodel GeRoMe*. Implementations of model management operators can assure that model elements have certain properties by checking whether they play the necessary roles. At the same time the operator remains agnostic about any roles which do not affect its functionality. Thus, while role based metamodeling allows to formulate accurate models, the models appear to operators only as complex as necessary. *GeRoMe* will be used only by model management applications; users will use their favorite modeling language.

The definition of the GMM requires a careful analysis and comparison of existing metamodels. Since it has to be possible to represent schemata in various metamodels in order to allow generic model management, we analyzed five popular yet quite different metamodels (Relational, EER, UML, OWL DL, and XML Schema). We identified the common structures, properties, and constraint mechanisms of these metamodels. This part of our work can be seen as an update to the work in [9], in which several semantic database modeling languages have been compared.

The paper is structured as follows. Section 2 provides some background information on model management and role based modeling, and presents a motivating scenario. In section 3, we analyze and compare existing metamodels and derive the *Generic Role Based Metamodel GeRoMe*. Section 4 shows several examples of models in different metamodels represented in *GeRoMe*. Section 5 explains how model management operations can be performed using *GeRoMe*. As an example, we describe some atomic operations necessary for the transformation of an EER model into a a relational model. The implementation of our model management prototype is discussed in section 6. Finally, section 7 summarizes our work and points out future work.

2 Background and Motivation

2.1 Model Management

Model management aims at providing a formalization for the definition and modification of complex models [4]. To achieve this goal, a model management system has to provide definitions for *models* (i.e. schemas represented in some metamodel), *mappings* (i.e. relationships between different models), and *operators* (i.e. operations that manipulate models and mappings). There have been earlier approaches to model management [1, 12], which did address especially the transformation of models between different metamodels. Model management has become more important recently, as the integration of information systems requires the management of complex models. The most important operations in model management are Merge (integration of two models),

Match (creating a mapping between two models), Diff (finding the differences between two models), and ModelGen (generating a model from another model in a different metamodel representation).

Rondo [13] is the first complete prototype of model management. It represents models as directed labeled graphs. Each node of such a graph denotes one model element, e.g. an XML complex type or relational table. A model is represented by a set of edges between these nodes. A model element's type (Table, Column, Class, ...) is also specified by such an edge with the label *type*. Furthermore, types of attributes are specified by other dedicated edges, e.g. *SQLtype*. For each of the supported metamodels a different set of types is available. Although the models are represented in a generic graph structure, the implementation of the operators is not truly generic. For example, the implementation of the match operator requires two models of the same type as input, and some operators (such as Extract) have specific implementations for each metamodel.

Clio [8] is a tool for creating schema mappings. Whereas schema matching algorithms just discover correspondences between schemas, Clio goes one step further and derives a mapping from a set of correspondences. The mapping is a query that transforms the data from one schema into another schema. However, Clio supports only XML and relational schemas.

More sophisticated model management operators such as Merge (integration of two models according to a given mapping, resulting in a new model) require even more semantic information about the models involved. For example, in [15] a meta model with several association types (e.g. has-a, is-a) is used.

The various approaches to model management show that each operator requires a different view on a model. Schema matching focuses on labels and structure of schema elements, whereas merging and transformation of models require more detailed information about the semantics of a model (e.g. association types, constraints). These different views are supported by our role based approach, as operators will see only those roles which are relevant in their context.

2.2 Scenario

Complex information systems undergo regular changes due to changes of the requirements, of the real world represented by the information system, or of other systems connected to the information system. As an example, we consider the following eBusiness scenario: a supplier of an automotive manufacturer receives orders from a business partner in some XML format. The orders are entered into the ERP system of the supplier by a transformation program, which uses a mapping between the XML schema and the relational DB of the ERP system.

In order to generate this mapping, the two models are represented as models in a generic metamodel (GM1 and GM2). A Match operator can then be used to create a mapping GM1_GM2 between the two models, which can be further translated into a mapping XS1_RM2 between the original models.

Due to a change in the system of the manufacturer, the schema of the orders has changed. Then, this change has to be applied to the mapping between the XML schema and relational DB. Focusing on the models, this scenario can be seen as an example of schema evolution (Fig. 1). The original XML schema XS1 is mapped to the relational

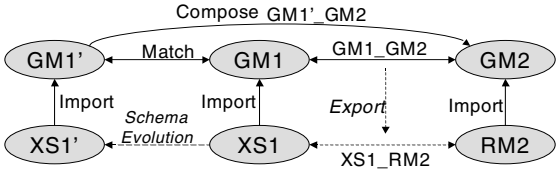

Fig. 1. Schema evolution using *GeRoMe* and Model Management

model (RM2) of the DB using the mapping XS1_RM2. The schema evolution generates a new version of the XML schema, namely XS1'.

Again, instead of applying the model management operators to the level of specific schemas, we will first generate a corresponding representation of the specific model in our GMM (GM1'). Then, we have to apply the Match operator to GM1 and GM1', resulting in a mapping GM1'_GM1 between these models. This match operation should be simpler than matching the new version GM1' with GM2 directly, as two versions of the same model should be quite similar. Then, we can compose the mappings GM1'_GM1 and GM1_GM2 to a new mapping GM1'_GM2. Note, that this operation has just to consider mappings between models represented in *GeRoMe*, which should simplify the implementation of such an operator. The result of this step is a mapping from GM1' to GM2 of those elements which are also present in GM1.

In order to map elements which have been added during the schema evolution a Diff operator has to be used on GM1' and GM1 which takes into account the mapping GM1'_GM1. The difference then has to be mapped individually.

The important difference to other approaches is that the operations in *GeRoMe* are truly generic, they do not have to take into account different representations of models. Therefore, the operators have to be implemented only once, namely for the *GeRoMe* representation.

2.3 Role Based Modeling

The concept of role (or aspect) based modeling has first been described in detail in the context of the network model [2] and later on in several works on object-oriented development and object-oriented databases [6, 16, 17].

Different formalizations have been proposed, which exhibit significant differences, but all have in common that a role or aspect extends the features of an existing object while being a view on the object and *not* an object in its own right. In [6] *multiple direct class membership* is considered as a solution to the problem of artificial intersection classes. That is, instead of defining an intersection class, the combination of state and behavior is achieved by defining an object to be instance of several classes at the same time, which are not necessarily on the same specialization path.

In [16] the notion of aspects of objects is discussed. It is stated that at any given moment an entity may have many different types that are not necessarily related. Often this issue cannot be handled by multiple inheritance since this would lead to a large number of sparsely populated "intersection classes" which add no new state. This approach is different from multiple direct class membership in that each object can have multiple

aspects of the same type, e.g. a person can at the same time be a student at more than one university while still being the same individual.

Other approaches, such as the one considered in [17], treat the different features of an object as roles, which are themselves instances of so called *role classes* and have identity by state. This representation also allows model elements to play directly or implicitly more than one instance of the same role. In addition, [17] introduces the concept of *role player qualification* which means that not every object may play every role but that certain conditions have to hold.

3 The Generic Role Based Metamodel *GeRoMe*

In this section, we will first explain the role model which we have employed to define *GeRoMe*. Based on our analysis of existing metamodels (section 3.2), we have derived the generic role based metamodel, which is described in detail in section 3.3.

3.1 Description of the Role Model

GeRoMe employs the following role model. A model element is represented by an object which has no characteristics in its own right. Roles can be combined to make up a model element encompassing several properties. Therefore, the model element is *decorated* with its features by letting it *play* roles. A role maintains its own identity and may be player of other roles itself. Every model element has to play at least one role and every role object has exactly one player. In our model, some role classes may be used more than once by a model element, e.g. an *Attribute* may play the role of a *Reference* to more than one other *Attribute*. Thus, the complete representation of a model element and its roles forms a tree with the model element as its root.

We used three different relationships between role classes, namely *inheritance*, *play*, and *precondition*. The *play* relationship defines which objects may be player of certain roles. In addition, a role may be a precondition of another role. Thus, in order to be qualified to play a role of a certain class, the player must be already the player of another role of a certain other class. Except for namespaces, all links between model elements are modeled as links between roles played by the elements.

To tap the full power of role modeling, we have to define role classes in such a way that each of them represents an "atomic" property of a model element. Then roles can be combined to yield the most accurate representation of an element.

3.2 Role Based Analysis of Concrete Metamodels

A generic metamodel should be able to represent both the structures and constraints expressible in any metamodel. Thus, to define such a metamodel it is necessary to analyze and compare the elements of a set of metamodels. Our choice of metamodels comprises the relational model (RM) [7] and the enhanced entity relationship model (EERM) [7] because these two models are rather simple and are in widespread use. The metamodel of the Unified Modeling Language (UML, version 1.5) has been analyzed as an example for object-oriented languages. The description logics species of the Web Ontology

Table 1. Roles played by concrete metaclasses

Role	EER	Relational	OWL DL	XML Schema
Domain	domain	domain	xsd datatype	any simple type
Aggregate	entity/rel.-ship type, composed attribute	relation	class	complex type
Association	relationship type	-	a pair of inverse object properties	element
Identified	entity/rel.-ship type	-	class	complex type, schema
BaseElement	base of anon. domain, supertype in isA, comp. type in Union	base of anonymous domain	superclass, superproperty	base simple / complex type
DerivedElement	subtype in isA or union type	anonymous domain constraint	subclass, subproperty	derived simple / complex type
Union	derivation link of union type	-	derivation link of union class	derivation link of union type
IsA	isA derivation link	-	subclassing derivation link	restriction / extension derivation link
Enumeration	enumerated domain restriction	enumerated domain restriction	enumeration	enumeration
Attribute	(composite / multivalued) attribute	column	data type property	attribute, element with simple type
AssociationEnd	link between relationship type and its participator	-	object property	links between two elements one of which is element of the other's complex type
Value	any instance	domain value, tuple	data type value, individual	xsd value, valid XML
Visible	entity type, relationship type, attribute	relation, column	named class, property	named type, attribute element
Reference	-	foreign key	-	keyref
Disjointness	constraint on subtypes	-	constraint on classes	-
Injective	primary/partial key	unique, primary key	inverse functional	unique, key
Identifier	primary/partial key	primary key	-	key
Universal	anonymous domain of attribute	anonymous domain constraint of column	allValuesFrom	restriction of complex type
Existential	-	-	someValuesFrom	-
Default	-	default value	-	default value

Language (OWL DL, `http://www.w3.org/2004/OWL/`) has been included since it follows different description paradigms due to its purpose. For example, properties of concepts are not defined within the concepts themselves but separately. Finally, XML Schema (`http://www.w3.org/XML/Schema`) has been analyzed as it is the most important metamodel for semistructured data.

We analyzed the elements and constraints available in these five metamodels and made out their differences and similarities. In doing so, we identified the role classes, which make up our role based metamodel. In total, we compared about seventy structural properties and elements and twenty types of constraints. Some of them are very easily abstracted, such as data types or aggregates. Others, such as the XML Schema *element* or OWL object properties, are rather intricate and need closer inspection. The XML Schema element is an association (associating a parent element with its children). The root element of a document is a special element which does not have a parent. Furthermore, an XML Schema may allow different types of root elements for a document.

Another problematic example are object properties in OWL DL: the *Association* role is played by a "pair of properties" and the *AssociationEnd* role is played by object properties. Thus, it is difficult to represent such specific model elements in a GMM.

In section 4, we describe some of the representation problems in more detail. Furthermore, some metamodels provide redundant options for representing the same semantics, e.g. there is no semantic difference between an XML Schema attribute and a simple-typed XML Schema element with a maximum cardinality of 1.

Table 1 shows a selection of role classes and states the related model elements in the considered metamodels. The table contains roles which are used to define structural model elements (e.g. relation, class) and roles to define relationships and constraints (e.g. association, disjointness). Due to space constraints, the table does not embody all metamodel elements and correspondences in the different metamodels. Furthermore, we omitted the UML metamodel for the same reason.

3.3 Description of *GeRoMe*

Figure 2 presents the Generic Role Based Model *GeRoMe* at its current state, based on the analysis of the previous section. Below, we will describe the elements of *GeRoMe* according to their basic characteristics: structural elements, derivations, and constraints.

Structural Elements. Every model element representing a primitive data type plays the role of a *Domain*. *GeRoMe* contains a collection of predefined domains such as *int*

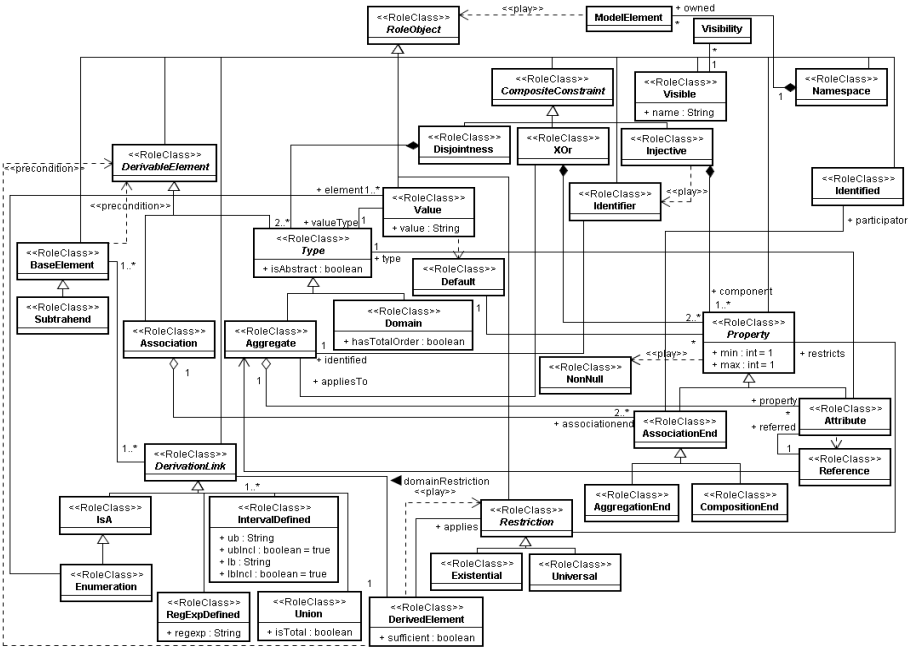

Fig. 2. The Generic Role Based Metamodel (*GeRoMe*)

and *string*. In contrast, model elements which may have attributes play an *Aggregate* role (e.g. entity and relationship types, composite attributes in EER; relations, classes and structs in other metamodels).

Thus, the *Aggregate* role is connected to a set of *Attribute* roles. Each of these *Attribute* roles is part of another tree-structured model element description. An *Attribute* role is a special kind of *Property* and has therefore the *min* and *max* attributes which can be used to define cardinality constraints. Every attribute has a *Type*, which may be a primitive type or an *Aggregate* for composite attributes. Furthermore, an *Attribute* role may itself play the role of a *Reference*, which defines a referential constraint to another *Attribute* of the same type.

The *Aggregate* role and the *Domain* role are specializations of *Type*. *Type* is a specialization of *DerivableElement* which is the abstract class of roles to be played by all model elements which may be specialized. Another kind of *DerivableElement* is the *Association* role. Properties of associations are *AssociationEnd* roles. For example, association roles are played by EER relationship types, UML associations, or UML association classes. A model element which provides object identity to its instances may participate in one or more associations. This is modeled by specifying the element's *Identified* role to be the participator of one or more *AssociationEnd* roles. Thus, an association end is a model element in its own right, and the association is a relationship between objects with identity. In addition, the roles *AggregationEnd* and *CompositionEnd* can be used to model the special types of associations available in UML.

The *Association* and *Aggregate* roles are an intuitive example of two role classes that can be used in combination to represent similar concepts of different metamodels. If the represented schema is in a concrete metamodel which allows relationship types to have attributes, such as the EER metamodel, then every model element playing an *Association* role may play additionally an *Aggregate* role. If associations may not have attributes, which is the case in OWL, a model element may only play either of both roles. On the other hand, the representation of a relational schema may not contain *Association* roles at all. Thus, these two roles can be combined to represent the precise semantics of different metamodel elements. Of course any of these combinations can be further combined with other roles, such as the *Identified* role, to yield even more description choices.

Finally, model elements can be *Visible*, i.e. they can be identified by a name. The *name* attribute of a *Visible* role has to be unique within the *Namespace* it is defined in. A model's root node is represented by a model element which plays a *Namespace* role.

Derivation of New Elements. A *BaseElement* role is played by any model element used in the definition of a derived element. Thus, a *DerivedElement* can have more than one *BaseElement* and vice versa. The type of base element determines the properties of the derived element. A *Subtrahend* is an element whose instances are never instances of the derived element (e.g. a `complementOf` definition in OWL).

BaseElement and *DerivedElement* roles are connected via dedicated model elements representing the *DerivationLink*. Each *DerivationLink* connects one or more *BaseElement*s to one *DerivedElement*. For example, new types can be defined by *Enumeration*, *IsA*, or *Union* definitions. The *IsA* role can be used to define specialization relationships. It extends the definition of a superclass by adding new properties (e.g.

inheritance in UML). A *DerivedElement* role which is connected to an *isA* role with more than one *BaseElement* role can be used to define a type to be the intersection of its base elements.

We identified two different kinds of *isA* relationships which are often not distinguished from each other. All metamodels allow *extension* (i.e. the subtype defines additional attributes and associations) if they allow specialization at all. In EER and OWL, model elements can specialize base elements also by constraining the ranges of inherited properties. In EER, this is called *predicate defined specialization* [7–p.80], whereas in OWL it is called *restriction* and comprises a very important description facility for inheritance. Such derivations can be expressed in our metamodel by connecting a *Universal* or *Existential* role played by the restricted range to the *DerivedElement* role. This *Restriction* role has to define a property, which it constrains.

Furthermore, there are several different ways to define new domains based on existing ones. In XML Schema, named domains can be derived from others whereas in the relational metamodel derived domains occur only as an anonymous type of attributes with enumeration or interval domains.

Constraints. Constraints are represented by separate model elements. For example, a disjointness constraint on a set of derived elements (or any other types) has to be defined by a model element representing this constraint. The element has to play a *Disjointness* role which references the types to be disjoint. In the case of OWL or UML, any collection of classes can be defined to be disjoint; in EER this constraint can be used to define a disjoint *isA* relationship by referencing at least all of the derived elements.

Another constraint is the *Injective* constraint which can be defined on a set of properties. Such an *Injective* role is equivalent to a uniqueness constraint. It can also define a composite key by being connected to multiple properties. An injective constraint playing an *Identifier* role defines a primary key. This reflects the fact that a primary key is only a selected uniqueness constraint, and thereby one of multiple candidate keys.

The *XOr* constraint is a modeling feature that is available in UML metamodels. It can be defined on association ends and states that an object may participate only in one of these associations. Thus, in *GeRoMe* an *XOr* constraint is related to at least two properties. *GeRoMe* can be extended with new role classes representing other features of constraints and structures while existing models and operators still remain correct.

4 Representation Examples

This section presents some example models based on a small airport database in [7–p.109] (see fig. 3). We represented EER, XML Schema and OWL DL models for this example. The model contains simple entity types composed of attributes as well as some advanced features, which are not supported by all metamodels (e.g. composite attributes, *isA* relationship).

4.1 Representation of an EER Schema

Fig. 4 shows a part of the representation of the airport model in *GeRoMe*. For the sake of readability, we refrain here from showing the whole model and omitted repeating

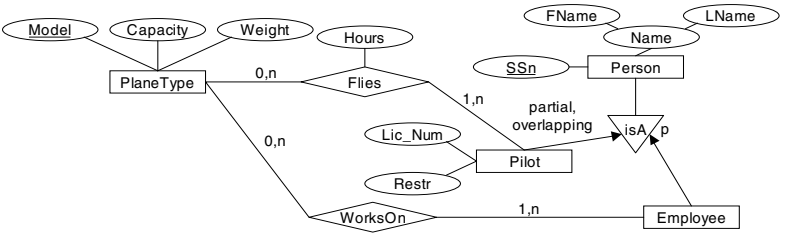

Fig. 3. Part of an airport EER schema

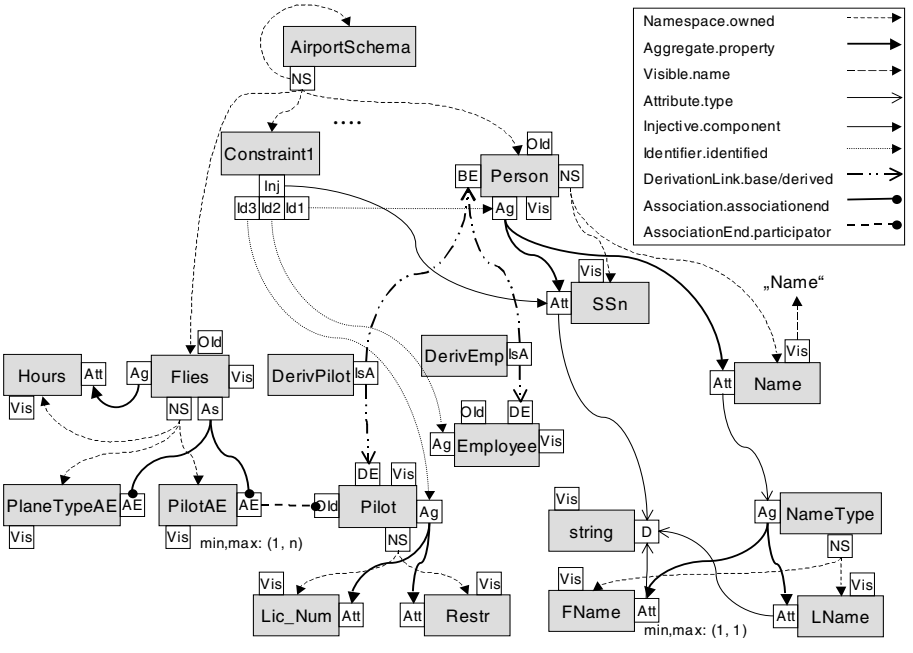

Fig. 4. *GeRoMe* representation of an EER schema

structures with the same semantics such as literal values or links from namespaces to their owned elements. The *GeRoMe* representation shows each model element as a *ModelElement* object (gray rectangle) which plays a number of roles (white squares) directly or by virtue of its roles playing roles themselves. Each such role object may be connected to other roles or literals, respectively. Thus, the roles act as interfaces or views of a model element. The links between role objects connect the model element descriptions according to the semantics of the represented schema.

The root model element of the airport schema is a model element representing the schema itself (*AirportSchema*). It plays a *Namespace* role (NS) referencing all model elements directly contained in this model.

The *Name* attribute is a visible model element and therefore its model element object plays the *Visible* role (Vis). The role defines a name of the element as it could

be seen in a graphical EER editor (note that we omitted the names for other *Visible* roles).

Since entity types are composed of attributes, every object representing an entity type plays an *Aggregate* role (Ag). Furthermore, instances of entity types have object identity. Consequently, representations of entity types also play an *Identified* role (OId). The *Aggregate* role is again connected to the descriptions of the entity type's attributes.

The EER model defines a primary key constraint on the *SSn* attribute. Therefore, a model element representing the constraint (*Constraint1*) and playing an *Injective* role (Inj) is connected to this attribute. This is a uniqueness constraint which is special in the sense that it has been chosen to be a primary key for the entity type *Person*. This fact is represented by the constraint playing an *Identifier* role (Id1) connected to the identified aggregate. Since *Person*'s subtypes must have the same identifier, the injectiveness constraint plays also *Identifier* roles (Id2, Id3) with respect to these model elements.

Specification of domain constraints is usually not possible in the EER model, but the addition of default domains does not hurt. Therefore, attributes always have a type in *GeRoMe*. Domains are themselves represented as model elements playing domain roles (D) (e.g. string). It is also possible to derive new types from existing ones as this is also possible in most concrete metamodels.

In addition, note that the composite attribute *Name* has not a domain but another *Aggregate* as type. Unlike the representation of an entity type, *Name_Type* is not player of an *Identified* role. Consequently, this element cannot be connected to an *AssociationEnd*, which means that it cannot participate in associations. Furthermore, *Name_Type* is not visible as it is an anonymous type. However, the representation is very similar to that of entity types and this eases handling both concepts similarly. For example, in another schema the composite attribute could be modeled by a weak entity type. If these two schemata have to be matched, a generic Match operator would ignore the *Identified* role. The similarity of both elements would nevertheless be recognized as both elements play an *Aggregate* role and have the same attributes.

Furthermore, the figure shows the representation of the *isA* relationship. Since every instance of *Pilot* and *Employee* is also an instance of *Person*, the *Person* model element plays a *BaseElement* role (BE) referenced by two *IsA* roles (IsA). These roles define two children, namely the *DerivedElement* roles (DE) which are played by the respective subtypes *Employee* and *Pilot*. Any attribute attached to the *Aggregate* roles of the subtypes defines an extension to the supertype. The children could also be defined as predicate-defined subtypes by associating to the *DerivedElement* roles a number of *Restriction* roles.

The subtype *Pilot* participates in the relationship type *Flies*. The representation of this relationship contains an *Association* role (As) which is attached to two *AssociationEnd*s (AE) (i.e. a binary relationship). Furthermore, the relationship has an attribute, and consequently, it plays the role of an *Aggregate*. The representations of the two association ends define cardinality constraints and are linked to the *Identified* roles (OId) of their respective participators. They also may play a *Visible* role which assigns a name to the association end.

4.2 Representation of an XML Schema

Fig. 5 shows the XML Schema representation of the example model in *GeRoMe*. The XML Schema element is a relationship between the type defined for the element and the complex type of the nested element. But it is always a 1:n relationship since an XML document is always tree structured. Cross links between elements in different subtrees have to be modeled by references. But what about root elements in a schema? These elements are related to the schema itself which in our role based model is represented by the *AirportSchema* model element. This is just one example of a concrete model element which is not obviously mapped to a generic metamodel.

An XML document conforming to an XML Schema can have any element as root element which is defined in the schema file as a direct son of the `schema` element. Consequently, any such element is represented in *GeRoMe* as a model element playing an association role with its complex type as one participator and the schema node as the other participator. In the example, *Airport* is the only such element. This element is visible and its name is "airport". *AssociationEnd*s of XML elements have no names attached and therefore are anonymous. Complex types may be anonymously nested into an element definition. In the example, this is the case for *AirportType*. Since definitions of keys have labels in XML Schema, the identifier of *Person* plays a *Visible* role with its label "personKey" assigned to it.

Model elements defined within other model elements such as attributes and XML elements are referenced by the *Namespace* role of the containing element. For example,

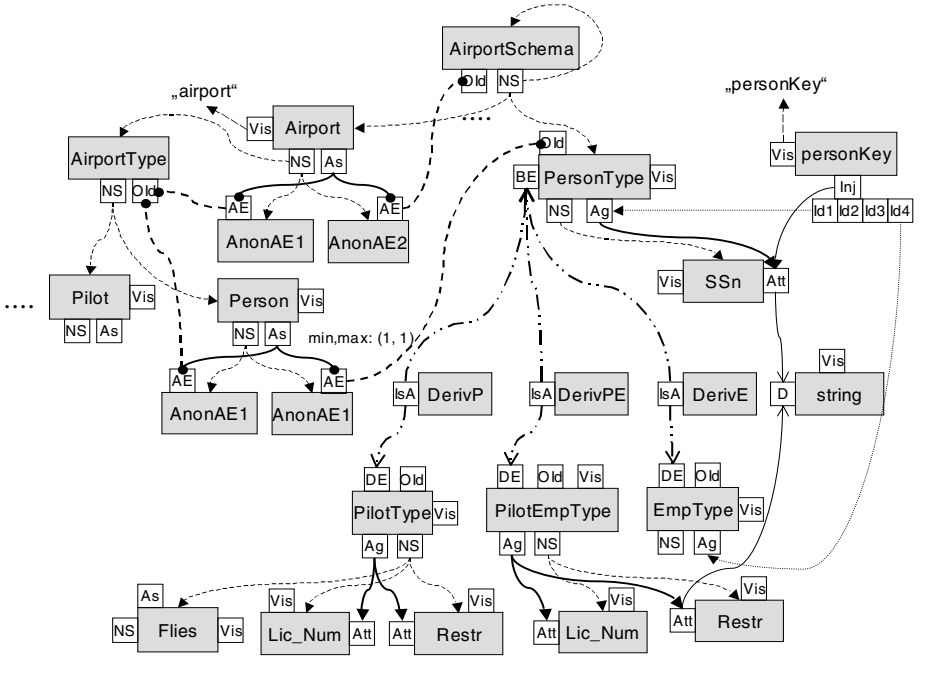

Fig. 5. Representation of a similar XML Schema

the element *Flies* is owned by the *Namespace* role of *PilotType*. Another consequence of the structure of semistructured data is that the *AssociationEnd* of the nested type always has cardinality (1,1), i.e. it has exactly one parent. Finally, the model element *PilotEmpType* has been introduced as it is not possible to represent overlapping types in XML Schema.

4.3 Representation of an OWL DL Ontology

In table 1, we stated that OWL DL object properties are represented by model elements playing *AssociationEnd* roles and that a pair of these model elements is connected by an *Association*. This is another good example for the problems which occur when integrating heterogenous metamodels to a GMM. The reasons for the sketched representation can be explained with the semantics of the relationship type *WorksOn* in fig. 3.

Intuitively and correctly, one represents *WorksOn* as a model element playing an *Association* role. *WorksOn* has two *AssociationEnd*s: one with cardinality (0,n) pointing on *PlaneType* and one with cardinality (1,n) pointing on *Employee*. This is represented analogous to *Flies* in fig. 4. Now what are the problems if you would regard an object property *WorksOn* as corresponding to the given relationship type?

Firstly, an object property always has domain and range. Thus, it has a direction. But the direction of a relationship type is only suggested by its name. On the other hand, an association end has a direction. The role name describes the role which the participator plays in the relationship type w.r.t. the participator at the opposite end. Furthermore, these role names are often phrasal verbs as are the names of object properties in OWL. Actually, in description logics object properties are often called *roles*. Thus, "WorksOn" should be the role name assigned to the link between the relationship type and the entitiy type *PlaneType*.

Secondly, an object property may have one cardinality restriction, whereas a relationship type has at least two (one for each participating entity). This shows that an object property corresponds to an association end, and that a pair of object properties (of which one is the inverse of the other) is a correct representation of a binary association. Note that OWL DL allows only binary relationships.

In order to allow other constraints, such as *Symmetric*, new roles can be added to *GeRoMe*. Adding a new role to the metamodel will render existing models and operator implementations valid and correct. Thus, it is also easy to extend *GeRoMe* if this is necessary in order to include new modeling constructs.

5 Model Management Using *GeRoMe* Models

In this section, we show how model management operators can make use of *GeRoMe*. Transformation of models is a typical task for model management applications. We will explain the transformation of the EER model of fig. 4 into a relational schema. Therefore, the original representation has to undergo several transformations in order to become a representation of a relational schema. Fig. 6 shows the final result of the transformation steps which will be discussed in detail in the following paragraphs.

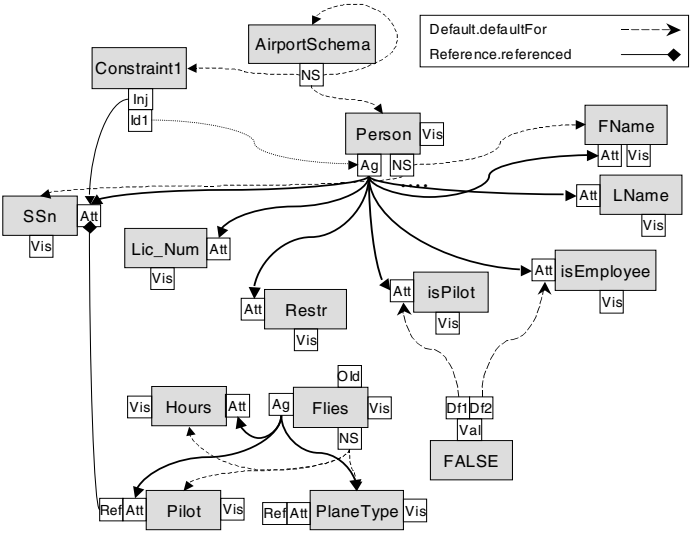

Fig. 6. Representation of the resulting relational schema

In model management, transformation of models is performed by a ModelGen operator, i.e. the operator generates a model from another existing model. We have implemented the transformation of constructs such as composite attributes or inheritance from an EER schema by several ModelGen_X operators. Each operator transforms the modeling constructs not allowed in the relational model into equivalent modeling elements of the relational model. The decomposition of the operators into several "atomic" operators has the advantage that they can be reused in combination with other operators to form new operators. Note that the following operators are not aware about the original representation of the models, i.e. the operators just use the *GeRoMe* representation. Thus, these operators could also be used to transform a UML model into XML Schema if similar transformation tasks are required (e.g. transformation of associations to references).

It has to be emphasized that mapping of models from one metamodel to another is just one popular example application of model management. The goal of our generic metamodel is *not only* to provide a platform for schema translation but to provide a generic model representation that serves as a foundation for the polymorphic usage of *any* model management operator. Thereby, other applications of model management, such as schema evolution, are also supported in a generic way.

Transformation of Relationship Types. Relationship types are not allowed in the relational metamodel. According to properties such as cardinality constraints, they have to be transformed to relations by executing the operator ModelGen_AssocToRef for each *Association* role. First, it looks for attached *AssociationEnd* roles, the arity of the association, and cardinality constraints. Depending on these constraints the transformation is either performed automatically or the user is asked for a decision before the operator can proceed. Copies of all attributes in the participators' identifiers are at-

tached to the relationship's *Aggregate* role. An *Aggregate* role has to be created first, if not yet available. Furthermore, these copies play *Reference* roles (Ref) referencing the original attributes, and thereby defining referential constraints. After performing all these transformations, the association ends and the relationship's *Association* role are deleted.

This yields an intermediate result which cannot be interpreted as a valid schema in the EER or relational metamodel, since it now contains constructs disallowed in both metamodels. An Export operator to the Relational or EER metamodel would have to recognize this invalidity and reject to export.

Transformation of IsA Relationships. The *isA* relationships also have to be removed depending on their characteristics (partial and overlapping), the attributes of the extensions *Pilot* and *Employee* thereby become attributes of the supertype.

The operator *ModelGen_FlattenIsA* fulfills this task by receiving a *BaseElement* role as input. It first checks for disjointness of connected isA relationships and whether they are total or not. Depending on these properties, the user is presented a number of choices on how to flatten the selected isA relationships. In the example, the base type *Person* and its subtypes *Pilot* and *Employee* have been selected to be transformed to one single aggregate due to the fact that the isA relationship is neither total nor disjoint. The resulting aggregate contains all attributes of the supertype and of the subtypes. Additionally, the boolean attributes *isPilot* and *isEmployee* (including *Default* roles Df1 and Df2 related to these attributes) have been introduced.

Transformation of Composite Attributes. The result yet contains a composite attribute and *Identified* roles (OId), which are not allowed in a relational model. The *Identified* roles can be removed directly, as the associations have been transformed to attribute references (earlier by the operator ModelGen_AssocToRef). The transformation of composite attributes is done by another atomic operator. First, it collects recursively all "atomic" attributes of a nested structure. Then, it adds all these attributes to the original *Aggregate* and removes all the structures describing the composite attribute(s) (including the anonymous type). This operator also needs to consider cardinality constraints on attributes, since set-valued attributes have to be transformed into a separate relation.

In this way, the whole EER schema has been transformed to a corresponding relational schema. Of course, more operators are needed to handle other EER features, such as *Union* derivations of new types.

Please note that the differences in the representations stem from the constraints and semantics of the concrete metamodels. Nevertheless the representations use the same role classes in all models, while accurately representing the features of the constructs in the concrete modeling languages. For example, the XML Schema *PersonType* plays the same roles as the EER *Person*, since entity types have the same semantics as XML Schema complex types. Furthermore, the relational *Person* does not play the *Identified* and *BaseElement* roles since these are not allowed in the relational model. On the other hand, all these roles play an *Aggregate* role, and therefore they look the same to an operator which is only interested in this role.

6 Implementation

Our implementation of a model management platform is based on a multi-layered architecture. The lowest layer provides facilities to store and retrieve models in the *GeRoMe* representation and is implemented using the deductive metadatabase system ConceptBase [11]. ConceptBase uses Telos as modeling language [14], which allows to represent multiple abstraction levels and to formulate queries, rules and constraints. Objects are represented using a frame-like or graphical notation on the user side, and a logical representation (triples similar to RDF) based on Datalog$^{\neg}$ internally. Using a logical foundation for the implementation of *GeRoMe* gives the advantage that some of the operators can be implemented in a declarative way. Furthermore, the semantics of the concrete metamodels can also be encoded in logical rules (e.g. inheritance of attributes).

Models represented in *GeRoMe* can also be stored as XMI documents (XML Metadata Interchange) to ease the exchange of metadata. Import and export operators to the native format of the various modeling languages are currently being implemented. The implementation of import and export operators is also a validation of *GeRoMe* since it proves that the modeling constructs from various metamodels can be represented. Before a model can be exported to a concrete metamodel, the export operator has to check whether all roles used can be represented in the target metamodel. If not, the problematic roles have to be transformed into different elements as described, for example, in the previous section. Note that an import to and an export from *GeRoMe* will result in a different model than at the beginning, as there are redundant ways to represent the same modeling construct in specific metamodels. For example, consider a simple typed XML Schema element with a maximum occurrence of one; this could also be modeled as an attribute.

On top of the storage layer, an abstract object model corresponding to the model in fig. 2 has been implemented as a Java library. It uses ConceptBase as storage mechanism and to evaluate rules and queries over a *GeRoMe* representation. The next layer is formed by atomic operators. Operators have to be implemented "as atomically as possible" in order to allow maximum reuse. These atomic operators are not aware of the original metamodel, i.e. their implementations should use only roles and structures in *GeRoMe*.

The operator layer is used by a scripting engine which enables the definition of more complex operators as scripts. A ModelGen_RM operator for transformation of schemata to the relational model could then be defined as a script which reuses operators as described in section 5. This scripting facility is analogous to the scripts which can be defined in the model management application *Rondo* [13].

7 Conclusion

Generic model management requires a generic metamodel to represent models defined in different modeling languages (or metamodels). The definition of a generic metamodel is not straightforward and requires the careful analysis of existing metamodels. In this paper, we have presented the generic role based metamodel *GeRoMe*, which is based on our analysis and comparison of five popular metamodels (Relational, EER, UML, OWL, and XML Schema).

We recognized that the intuitive approach of identifying generic metaclasses and one-to-one correspondences between these metaclasses and the elements of concrete metamodels is not appropriate for generic metamodeling. Although classes of model elements in known metamodels are often similar, they also inhibit significant differences which have to be taken into account. We have shown that role based metamodeling can be utilized to capture both, similarities and differences, in an accurate way while avoiding sparsely populated intersection classes. In addition, the role based approach enables easy extensibility and flexibility as new modeling features can be added easily. Implementations of operators access all roles they need for their functionality but remain agnostic about any other roles. This reduces the complexity of models from an operator's point of view significantly. Furthermore, the detailed representation of *GeRoMe* models is used only by a model management application, users will still use their favorite modeling language.

Whereas role based modeling has yet only been applied to the model level, we have shown that a generic metamodel can benefit from roles. In particular, *GeRoMe* enables *generic* model management. As far as we know, the role based approach to the problem of generic metadata modeling is new.

Using *GeRoMe*, model management operators can be implemented polymorphically, i.e. they just have to be implemented only once using the GMM. The role based approach has been validated by representing several models from different metamodels in *GeRoMe*. We are implementing *automatic* import/export operators in order to verify that the model elements of different metamodels can be represented accurately and completely in *GeRoMe*. Furthermore, we have implemented *GeRoMe* and some ModelGen operators using our metadatabase system ConceptBase.

Future work will concentrate on evaluating and refining *GeRoMe*. The approach has to be further validated by implementing model management operators that make use of the GMM. While it might be necessary to integrate new modeling features of other languages, or features which we did not take into account so far, we are confident that our work is a basis for a generic solution for model management.

Acknowledgements. This work is supported in part by the EU-IST project SEWASIE (`www.sewasie.org`) and the EU Network of Excellence ProLearn (`www.prolearn-project.org`).

References

1. P. Atzeni, R. Torlone. Management of Multiple Models in an Extensible Database Design Tool. *Proc. EDBT'96*, pp. 79–95. Springer, Avignon, 1996.
2. C. W. Bachman, M. Daya. The Role Concept in Data Models. *Proc. VLDB Conf.*, pp. 464–476. Tokyo, 1977.
3. P. A. Bernstein. Applying Model Management to Classical Meta Data Problems. *1st Biennial Conf. on Innovative Data Systems Research (CIDR2003)*. Asilomar, CA, 2003.
4. P. A. Bernstein, A. Y. Halevy, R. Pottinger. A Vision for Management of Complex Models. *SIGMOD Record*, **29**(4):55–63, 2000.
5. P. A. Bernstein, S. Melnik, M. Petropoulos, C. Quix. Industrial-Strength Schema Matching. *SIGMOD Record*, **33**(4):38–43, 2004.
6. E. Bertino, G. Guerrini. Objects with Multiple Most Specific Classes. *Proc. European Conference on Object-Oriented Programming (ECOOP)*, pp. 102–126. Springer, 1995.

7. R. A. Elmasri, S. B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, Reading, MA, 3rd edn., 1999.
8. M. A. Hernández, R. J. Miller, L. M. Haas. Clio: A Semi-Automatic Tool For Schema Mapping. *Proc. ACM SIGMOD Conf.* Santa Barbara, CA, 2001.
9. R. Hull, R. King. Semantic Database Modeling: Survey, Applications, and Research Issues. *ACM Computing Surveys*, **19**(3):201–260, 1987.
10. ISO/IEC. Information technology – Information Resource Dictionary System (IRDS) Framework. *Tech. Rep. ISO/IEC 10027:1990*, 1990.
11. M. A. Jeusfeld, M. Jarke, H. W. Nissen, M. Staudt. ConceptBase – Managing Conceptual Models about Information Systems. P. Bernus, K. Mertins, G. Schmidt (eds.), *Handbook on Architectures of Information Systems*, pp. 265–285. Springer, 1998.
12. M. A. Jeusfeld, U. A. Johnen. An Executable Meta Model for Re-Engineering of Database Schemas. *Proc. ER94*, pp. 533–547. Springer, Manchester, 1994.
13. S. Melnik, E. Rahm, P. A. Bernstein. Rondo: A Programming Platform for Generic Model Management. *Proc. ACM SIGMOD Conf.*, pp. 193–204. San Diego, 2003.
14. J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis. Telos: Representing Knowledge About Information Systems. *ACM Transactions on Information Systems*, **8**(4):325–362, 1990.
15. R. Pottinger, P. A. Bernstein. Merging Models Based on Given Correspondences. *Proc. VLDB*, pp. 862–873. Berlin, 2003.
16. J. Richardson, P. Schwarz. Aspects: extending objects to support multiple, independent roles. *Proc. ACM SIGMOD Conf.*, pp. 298–307. Denver, 1991.
17. R. K. Wong, H. L. Chau, F. H. Lochovsky. A Data Model and Semantics of Objects with Dynamic Roles. *Proc. ICDE*, pp. 402–411. Birmingham, UK, 1997.