Electronic Notes in Theoretical Computer Science 72 No. 3 (2003)
URL: http://www.elsevier.nl/locate/entcs/volume72.html 15 pages

Transforming Object Oriented Models with BOTL

Peter Braun and Frank Marschall [1,2]

Institut für Informatik
Technische Universität München
D-85748 Garching b. München, Germany

Abstract

In many application domains, like for example B2B, B2C, or CASE, a lot of heterogeneous applications exist which have to cooperate and exchange data. A major task of system integration is the mapping of different data models, on which the applications are built upon. In this paper, the Basic Object-oriented Transformation Language (BOTL) is introduced as a mathematically founded approach for the integration of data models. Class diagrams of the Unified Modelling Language (UML) are today established as a technique for meta modelling in the software engineering domain. Therefore, BOTL uses a UML-based notation for the definition of declarative mapping rules that allow reasoning about properties like applicability, metamodel conformance and bidirectionality of transformations.

1 Introduction

The ever increasing popularity of themes like B2B, B2C, and Web-Services shows that the integration of heterogeneous software systems is one of the most challenging tasks in software engineering today. Beyond the integration of different workflows, the major task is the transformation of data among different representations.

In today's heterogeneous environments, technologies like SOAP [4] are used for communication, relying on XML [7] as a common markup language. Standards like XMI [2] or XML Schema [5], which define a schematic mapping from class diagrams to DTDs, further ease the task of integration.

Yet, it turns out that standardized generated exchange formats cannot finally solve the dilemma of system integration. Different models lead to different exchange formats, thus preventing compatibility. Either the exchanged

[1] Email: braunpe@in.tum.de
[2] Email: marschal@in.tum.de

data has to be transformed, e.g. with XSL [3] transformations or enhanced methods like XAS [10], or the models have to be integrated, before they are encoded in exchange formats like XML.

The drawback of XSL-like approaches is that these languages are only unidirectional and lack an intuitive, graphical representation. Moreover the increased number of transformation steps implies a lot of overhead which often leads to consistency problems.

What is actually needed is the possibility to transform object models, not their textual representations. Unfortunately the common way of doing this by manually coding, maybe using textual script languages, is naturally fault-prone, and time and cost expensive.

At this point graph grammars [12,14] seem to offer a much more elegant solution at the first glance. Graph grammars are well known and examined in the academic domain and can be easily supported with graphical modelling techniques. In order to apply graph grammar transformations to object models, objects and attributes are treated as nodes connected by edges. Graph grammar rules determine how subgraphs can be replaced.

Thus objects have to be resolved into rather fine-grained subgraphs which leads to very big and confusing models that are, even for tools, hard to handle. Furthermore, this approach does (basically) not address a set of features that are of fundamental importance for object orientation. So there is no native concept to deal with object identities; moreover graph grammars do not know metamodels, i.e. class models, that define the structure of valid object models. Finally, it is difficult to transform values of attributes with this approach, e.g. transforming a temperature value from degree Fahrenheit to Celsius is much easier with an algorithmic language than with graph grammars.

It turns out that other approaches to model transformation that originate from database domain like mediators [8] or schema evolution [9,11,6,16] have similar deficits when applied to object-oriented model transformations.

Hence we are convinced that neither algorithmic languages nor graph grammars are appropriate and ergonomic languages to define object model transformations. Therefore we propose BOTL, the Basic Object-oriented Transformation Language, as a trade-off between these two approaches.

This paper introduces the very basic concepts of the BOTL approach currently developed by the authors. BOTL comes with a sound, mathematical founded description of the language and its transformation mechanisms. The language offers the ability to use graphical description techniques and integrated algorithmic descriptions to graphically define a set of mapping rules.

In Section 2 a small scenario is introduced that serves as a running example throughout the paper. Section 3 gives a rough overview over the core BOTL concepts; an extensive formal documentation is currently developed and will be available soon. In Section 4 the application of BOTL rules for the given example is described in detail. The paper is rounded up by a short discussion and outlook on future work.

2 Example

In this section a small example is introduced to illustrate the proposed approach. In the scenario there are two applications, the Alpha Information System and the Beta Application, that both handle data about employees and offices. The structure of the object models that the two applications use is internally determined by UML class diagrams.

The class diagram for the first application, the Alpha Information System (AIS), is depicted in Figure 1.

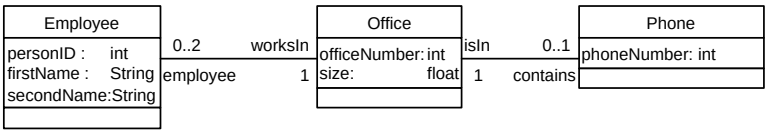

Figure 1. The metamodel of the Alpha Information System

As one can see, every employee works in exactly one office, while an office offers space for up to two employees. Moreover, every office may have a phone.

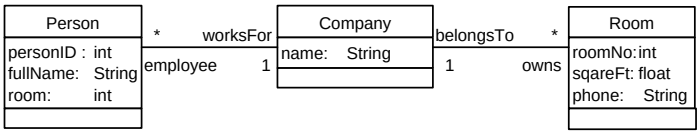

Figure 2. The metamodel of the Beta Application

Figure 2 shows the Beta Application's class diagram for dealing with employees and offices. Obviously the Beta Application has no extra class for phones; on the contrary the phone number is stored in an additional attribute of the class Room.

Figure 3 shows a sample object model of the Alpha Information System that should be transformed to correspond to the Beta Application's metamodel. We will refer to this model as the alpha model m_α.

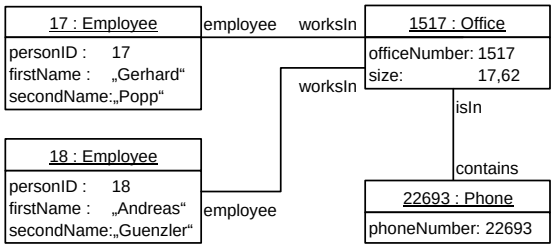

Figure 3. The alpha object model m_α that is conform to the AIS metamodel

Intuitively, one can sketch a solution by interpreting the association names. After transforming the model from Figure 3 to an instance m_β of the metamodel shown in Figure 2 m_β should obviously look like depicted in Figure 4. As one can see, the resulting model had to be enriched with static informa-

tion, like e.g. the company name and the area code, that is not available in the Alpha Information System but required within the Beta Application.

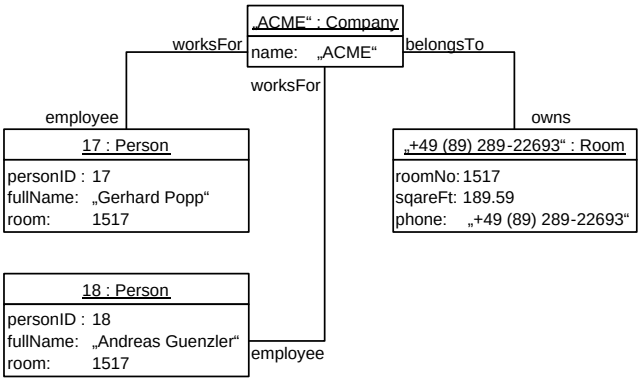

Figure 4. The transformed beta object model m_β originating from the AIS model m_α

The goal of the proposed approach is to provide a language that allows a developer to easily express this intuitive knowledge about how two models relate. Further the language needs a well defined semantics that allows the generation of model transformers.

Thus the next section exposes the basic concepts of such a language that is applied in Section 4 to this scenario.

3 The BOTL Formalism

In this section the most important core concepts of the BOTL formalism are described. Since this paper serves only as an overview over BOTL, just a brief sketch of those concepts is given here.

We use a metamodel enabled approach for the transformation of models. A metamodel describes structural constraints respectively common properties of models. As the UML [1] and the MOF [13] are nowadays wide spread and commonly used in practice, the UML oriented approach of using class diagrams for meta modelling is becoming more and more popular. Since it's our goal to use commonly accepted description techniques, BOTL is based upon class diagrams for the description of metamodels. Unfortunately, class diagrams are not formally defined within the UML. Thus BOTL is based upon a straightforward formalization. In favor of clarity, our formalization of class diagrams is not presented in detail within this paper. It is assumed that there is a common agreement on the basic concepts of classes, objects, relations, and types in object orientation.

Definition 3.1 A *metamodel* mm is a tuple of a set of *types*, a *class allocation*, and a set of *class associations*. All classes have unique names. The attribute names within each class are unique, too. All class associations connect classes

of the class allocation. The types of all attributes are elements of the set of types.

A sample for a metamodel is shown in Figure 1. In the example the class allocation consists of three *classes*, which are tuples of an *identifier* and a set of *attributes*. Attributes are tuples consisting of an *identifier* and a *type*. In Figure 1 the class with the *id* Phone has one attribute PhoneNumber. Classes may be connected by class associations of the usual UML types. A class association as shown in Figure 1 is a set with one or two association ends that consist of a role name, a class, and a multiplicity range. Class associations with just one association end are symmetric class associations.

Definition 3.2 Similar to the definition of metamodels, a *model* m is a tuple (mm, OS, OA) consisting of a metamodel mm, an *object allocation* OS, and a set OA of *object associations*. OS and OA must be conform to the metamodel mm, which means that:

(i) All objects and all object associations must be of a type defined in mm.
(ii) All ends of object associations must refer to an object of OS.
(iii) For every object in OS the sum of the cardinalities of all outgoing object associations according to a class association has to be in the range of the multiplicity defined by the class association.

Figure 3 shows an object diagram that describes a model according to the metamodel of Figure 1. Similar to classes, objects have an *identifier* and an *object type* which is a class of the metamodel. *Attributes* of objects have assigned *values*. Objects may be connected by *object associations*. Object associations have a *cardinality* assigned, which has value one for all object associations in the example of Figure 3.

BOTL uses rules for the definition of a model transformation. Each rule defines a mapping of a clipping of a source model onto a clipping of a target model. Each rule consists of a left and a right hand side being model variables.

Definition 3.3 A *model variable* mv is a tuple (mm, OVS, OVA) consisting of a metamodel mm, an *object variable allocation* OVS, and a set OVA of *object variable associations*. Object variables and object variable associations refer to classes and class associations of the metamodel mm. All object variable associations connect object variables of OVS.

In Figure 7 four model variables in two rules are shown. An *object variable* is similar to an object, but instead of concrete values terms are assigned to the identifier and to the attributes. An *object variable association* is like an object association except it connects object variables. Concrete values for the cardinalities are assigned to object variable associations. In the example of this paper, the value one is used as the cardinality of all object variable associations, and is not shown explicitly. A model variable may be inconsistent with respect to the metamodel, since the cardinalities of a model variable's

association may conflict with the allowed range.

Definition 3.4 A *model transformation* rule r_i is a tuple (mv_0, mv_1) consisting of two model variables. A *model transformation rule set* r is a finite sequence of rules between the same two metamodels.

All variables used within terms of a rule have to appear on both sides of the rule. The special value $\diamondsuit$ can be used instead of a "free´´ variable.

Definition 3.5 A *model fragment* mf is a tuple (OS, OA) consisting of an object allocation OS, and an object association set OA. All object associations connect objects of OS.

A model fragment is similar to a model (c.f. Definition 3.2), but it is not necessarily consistent with respect to the multiplicities of class associations. Furthermore, within model fragments, attributes may be assigned the special value $\diamondsuit$, which may also be used within terms marking unset values.

Definition 3.6 A *model fragment match* mfm_i is a tuple $(mv, mf, match_o, match_a)$ consisting of a model variable mv, a model fragment mf, and two bijective functions $match_o : OVS \rightarrow OS$ and $match_a : OVA \rightarrow OA$. $match_o$ takes an object variable and returns an object, so that every object variable of OVS is bijectively mapped to an object of OS of the same type. $match_a$ takes an object variable association and returns an object association, so that all object variable associations of OVA are bijectively mapped to an object association of OA with the same association ends and the same cardinality. Furthermore, for the result of $match_o$ and $match_a$ it has to hold that if an object variable and an object variable association are connected, then also their matches are connected.

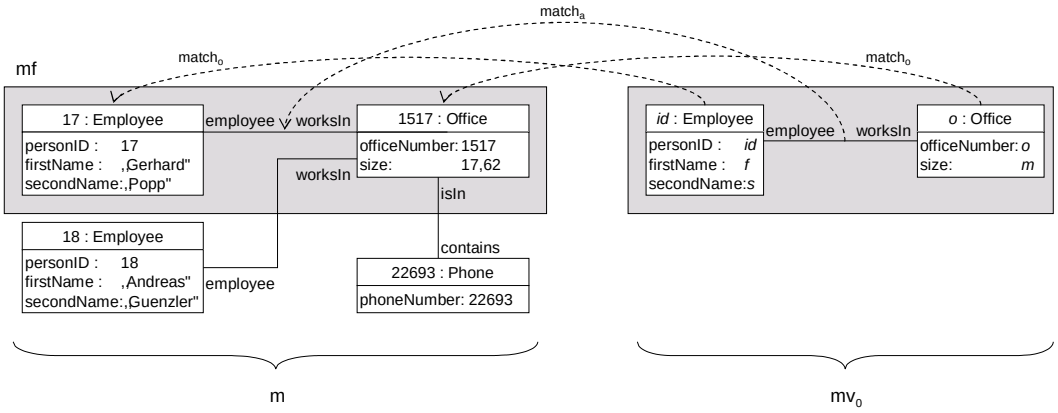

Figure 5. A model fragment match $(mv, mf, match_o, match_a)$

Model fragment matches are needed to find those parts of a given source model which match to the left hand side of a rule. They are also used to construct model fragments of the target. In the example of Figure 5, one of

two possible model fragment matches mfm_0 and mfm_1 is shown (c.f. Figure 8).

Definition 3.7 For a given model m and a model variable mv $MFM(m, mv)$ is the set of all possible (finite) *model fragment match sequences* mfm of model fragment matches mfm_i according to mv. It holds that [3]:

(i) $\forall i, j : i \neq j : mfm_i \neq mfm_j$
(ii) $\forall i : mfm_i|_{mf}|_{OS} \subseteq m|_{OS} \wedge mfm_i|_{mf}|_{OA} \subseteq m|_{OA} \wedge mv = mfm_i|_{mv}$
(iii) $\forall mfm_\tau$ with (ii) holds for mfm_τ
$\Rightarrow \exists mfm : mfm_\tau \in mfm \in MFM(m, mv)$

The number of model fragment matches to a given finite model and a given finite model variable is finite. For the application of a rule an arbitrary ordered sequence of model fragment matches is needed.

Definition 3.8 A *model fragment relation* is a relation

$$mfr : (mfm_i^0, r_j) \times mf_1$$

where $r_j = (mv_0, mv_1)$ and $mfm_i^0|_{mv} = mv_0$. Further let $mf_0 = mfm_i^0|_{mf}$. For mfr we require two postulations that have to hold:

(i) $\exists mfm_k^1$, with $mfm_k^1|_{mv} = r_j|_{mv_1} \wedge$
$mfm_k^1|_{mf} = mf_1$

(ii) Regard the following sets of equations:
- For every object variable in mv_0 there is a set of equations ES_0 between the terms of the attributes, resp. the identifiers, in mv_0 and the corresponding values of the matching objects in mf_0.
- For every object variable in mv_1 there is a set of equations ES_1 between the terms of the attributes, resp. the identifiers, in mv_1 and the corresponding values of the objects in mf_1 that match accordingly to the model fragment match mfm_k^1.

Postulation (ii) holds iff there is a mfm_k^1 so that the equational system $ES = ES_0 \cup ES_1$ can be solved. Therefore every equation with a $\diamondsuit$ at one side is ignored, because we use $\diamondsuit$ to state that the value of a given attribute is arbitrary.

In Figure 6 an example for a valid model fragment relation is given. There is a model fragment match mfm_i^0 of the left rule variable mv_0 to a model fragment mf_0 that is indicated by the dashed pointers between the elements of the model variable and the model fragment. As required in Definition 3.8 (i) there exists also a match mfm_k^1 for mv_1 that leads to another model fragment mf_1. According to statement (ii) of the definition, we get the following equational system:

[3] we write $u|_x$ if we mean the tuple element x of $u = (x, y)$

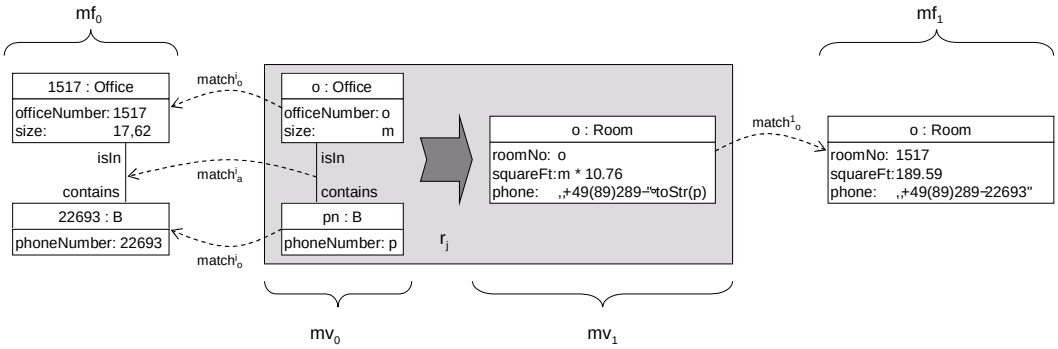

Figure 6. A rule r_j with a model fragment match for the left side and a model fragment mf_1 where $mfr((mfm_i^0, r_j), mf_1)$ does hold.

$$\begin{aligned} ES_0: \quad 1517 &= o \\ 17.62 &= m \\ 22693 &= p \\ ES_1: \quad o &= 1517 \\ m * 10.76 &= 189.59 \\ phone &= \text{"} + 49(89)289 - 22693\text{"} \end{aligned}$$

As one can see this is a solvable equational system with valid solutions for all variables ($o = 1517, m = 17.62, p = 22693$). Thus $mfr((mfm_i^0, r_j), mf_1)$ does hold. [4]

There may be several (or none) model fragments mf_1^l that are in the relation $mfr((mfm_i^0, r_j), mf_1^l)$ for given mfm_i^0 and r_j. Those relations that hold only for one model fragment mf_1 are of special interest, because then attribute values for model fragments can be computed deterministically.

Definition 3.9 A *model fragment transformation* mft is a function such that

$$mft(mfm_i^0, r_j) \mapsto \begin{cases} mfr(mfm_i^0, r_j) & \text{if } mfr \text{ is a function} \\ \perp & \text{otherwise} \end{cases}$$

A model fragment transformation transforms one match of the source model into exactly one model fragment of the target model. To transform a complete model all matches are transformed and resulting fragments are merged.

Definition 3.10 $\cup_{\mathfrak{m}}$ is the strict function which merges two model fragments mf_0, mf_1 and returns a model fragment mf_2 so that:

$$mf_0|_{OS}|_{id} \cup mf_1|_{OS}|_{id} = mf_2|_{OS}|_{id}$$

Two objects with the same id are merged into one object. Corresponding attributes must have the same value or at least one value is $\diamondsuit$ which will be

[4] Please note that in this case also types like strings are used, that do not form a mathematical group.

overwritten. Otherwise the resulting model fragment is $\perp$. Further all object associations are preserved. The cardinality of the resulting object associations is the maximum cardinality of the corresponding object associations in the source model fragments. Merging $\perp$ with any fragment results in $\perp$.

Theorem 3.11 $\cup_{\mathfrak{m}}$ *is commutative and associative.*

The proof is based upon the commutativity and associativity of $\cup$.

Definition 3.12 A *rule application* is a function

$$apply(m, r_i, mf_0) \mapsto mf_1$$

Let mfm be an arbitrary finite model fragment match sequence with respect to the model m and the model variable $r_i|_{mv_0}$.

$$\begin{aligned} mf_{1,0} &= mf_0 \\ mf_{1,j} &= mf_{1,j-1} \cup_{\mathfrak{m}} mft(mfm_{j-1}, r_i) \text{ , for } 1 \leq j \leq |mfm| \\ mf_1 &= mf_{1,|mfm|} \end{aligned}$$

A rule application applies a rule r_i to the source model m and merges the resulting model fragments into the existing fragment mf_0.

Theorem 3.13 *The result of a rule application is invariant with respect to different possible model fragment match sequences.*

The proof of Theorem 3.13 is a direct consequence of Theorem 3.11.

Theorem 3.13 states an important property of the given formalism: within a rule application we extend the target model by finding matches for a rule's model variable on the left-hand side in the source model, creating new model fragments for the new target model, and merging them successively into the target model. According to Theorem 3.13 the (chronological) order in which matches are found does not matter. Consequently the order in which new fragments are merged doesn't matter, too. Thus a rule application always yields to a deterministic result, independent of the chosen pattern matching strategy.

Definition 3.14 A *rule set application* is a function

$$\begin{aligned} transform(m, r) \mapsto mf_{|r|} \quad & \text{where} \\ mf_0 = (\emptyset, \emptyset) \quad & \text{and} \\ mf_i = apply(m, r_{i-1}, mf_{i-1}) \quad & \text{for } i \in \{1...|r|\} \end{aligned}$$

$transform$ is a *model transformation* if $mf_{|r|}$ is a valid model wrt. $r|_{mv_1}|_{mm}$.

Definition 3.15 A rule r_i is *applicable* iff for all possible models m_0 the following holds:

$$apply(m_0, r_i, (\emptyset, \emptyset)) \neq \perp$$

A rule set r is *applicable* iff it holds for all possible models m_0:

$$transform(m_0, r) \neq \perp$$

We can give some heuristics to determine when a rule is applicable. However these heuristics are only sufficient, but not always necessary, postulations:

Theorem 3.16 *A rule r_i is applicable if the following three statements hold:*

- (i) *All equations according to Definition 3.8 have a unique solution.*
- (ii) *We can determine for every object variable $ov \in r_i|mv_1|OVS$ that at least one of the following statements holds:*
  - *If the identity of ov is one-to-one dependent on a set of object variables from the rule's left-hand side we call this set OV. If the elements of OV are matched to the same objects of the left-hand model then every attribute of ov gets assigned the same value by the model fragment transformation.*
  - *It holds for the identity of the object variable that $ov|_{oiv} = \diamondsuit$.*
  - *All attribute values of ov are $\diamondsuit$.*
- (iii) *For any two object variables in $r_i|mv_1$ of the same type with an identity different from $\diamondsuit$, it has to hold that their pairwise identical attributes have either equal values or at least one of them has the value $\diamondsuit$.*

Please note that the postulations in Theorem 3.16 are sufficient, but postulations (ii) and (iii) are not necessary, i.e. there may be rules that are applicable according to Definition 3.15 but this property cannot be proofed with 3.16.

Obviously there are two possibilities to obtain $\perp$ as the result of a rule application: either a model fragment transformation returns $\perp$, or a merge operation returns $\perp$ because different attribute values could not be merged. Thus postulation (i) ensures that no model fragment transformation within the rule application results in $\perp$. Further (ii) and (iii) ensure that no attribute values that are different from $\diamondsuit$ conflict in the target model. Such a conflict couldn't be solved deterministically.

To prove the properties of Theorem 3.16 (ii) BOTL comes with a set of mechanisms to decide if an object may be created twice within one rule. If an object variable has an *id* equal to $\diamondsuit$, a unique value for this *id* is created (and no other object variable may randomly create this *id*). Thus all objects "created" from this object variable are mutually different. 3.16 (ii) states that if the *id* term of an object variable is one-to-one dependent on some source *id*'s, then this *id* is considered unique (if there is no other object variable with the same type).

Of course one may develop more sophisticated and powerful heuristics that allow the proof of applicability for a greater set of rules than this deliberations allow.

Theorem 3.17 *A sufficient but not necessary criterion for a rule set r to be applicable is that the following holds for all rules r_i:*

(*i*) *r_i is applicable*
(*ii*) *for any two different rules there are no terms that lead to mutually contradictory attribute values of one object.*

Beyond applicability, it is further possible to verify that a set of rules is:

- *valid*, i.e. all created target models comply to a given target metamodel
- *bijective*, i.e. the result of the application and the reverse application with interchanged source and target is isomorphic to source.

Furthermore, the structure of the according metamodel and the model variable can be used to compute an upper bound of possible matches for which the above mentioned objects with unique *id*'s are considered as fix. From this information some further propositions can be made. E.g. this information is necessary to compute an upper bound of the cardinalities of associations, while the lower bound can be seen directly from every rule. With this information a sufficient but not necessary criterion for the validity of rules (or rule sets) is given within the BOTL formalism.

Due to the formalization of BOTL it is possible to check these properties for a set of given rules and eventually generate programs that transform models according to a given rule set.

4 Application

The BOTL is now used to define a transformation from models of the Alpha Information System to those of the Beta Application. The application of this transformation is illustrated with the models m_α and m_β.

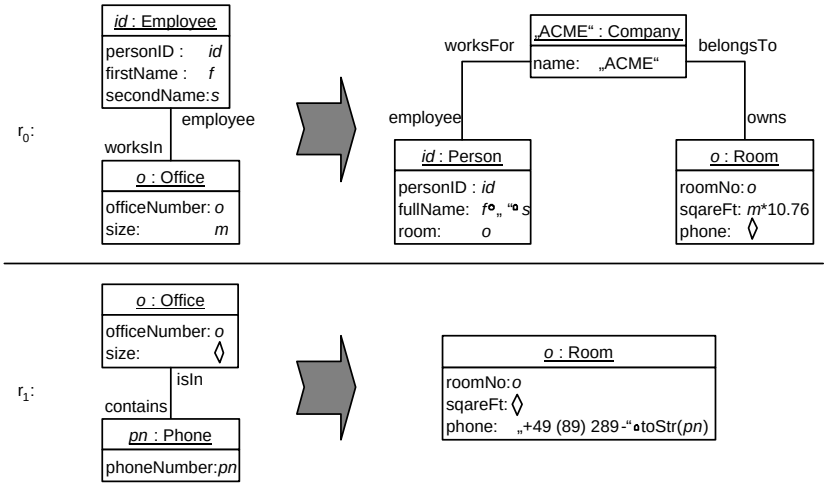

Figure 7. Sample BOTL rule set $r = (r_0, r_1)$

To define the relation among alpha and beta models we use a rule set

$r = (r_0, r_1)$, shown in Figure 7, that consists of two BOTL rules. Informally speaking, the first rule r_0 identifies employees and their offices and relates them to a pair of Person and Room objects in the beta model. Thereby the Company object "ACME" of the beta model remains always constant. As already mentioned in Section 2 the author of the rules can add static information to the rules. Thereby he can determine that the company is always ACME and provide the full phone number for the Beta Application, since the AIS does not have this information.

Note that the identity of objects is determined by relating them with an attribute value. This allows one to access generated objects in the target model.

In terms of our formalism, a model transformation is a rule set application as defined in 3.14. Accordingly, the result can be obtained from the function $transform(m_\alpha, r)$. Inserting the example's values leads to

$$transform(m_\alpha, r) \mapsto mf_2 = apply(m_\alpha, r_1, apply(m_\alpha, r_0, (\emptyset, \emptyset)))$$

From Definition 3.6 and 3.7 we can figure out an arbitrary model fragment

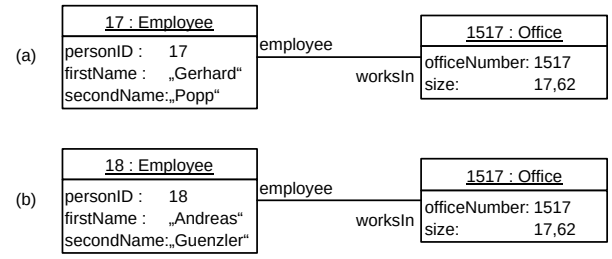

Figure 8. The model fragments (a) $mfm_0(r_0|_{mv_0})$ and (b) $mfm_1(r_0|_{mv_0})$

match sequence $mfm = (mfm_0, mfm_1)$ of all (in our case two) possible model fragment matches in m_α and the model variable $r_0|_{mv_0}$. Figure 8 shows the model fragment matches mfm_0 and mfm_1. This model fragment match sequence is needed to resolve the inner *apply* operator according to Definition 3.12:

$$apply(m_\alpha, r_0, (\emptyset, \emptyset)) = (mf_0 \cup_{\mathfrak{m}} mft(mfm_0, r_0)) \cup_{\mathfrak{m}} mft(mfm_1, r_0)$$

Below, the set of equations for the first model fragment transformation mft is depicted. The left column shows the equations descending from the source model fragment $mfm_0(r_0|_{mv_0})$ and the left side of the rule $r_0|_{mv_0}$. The right

column shows the equations from the right side $r_0|_{mv_1}$.

$$\begin{aligned} & name = \text{“ACME”} \\ 17 = id \qquad & personID = id \\ \text{“Gerhard”} = f \qquad & fullName = f \circ \text{“ ”} \circ s \\ \text{“Popp”} = s \qquad & room = o \\ 1517 = o \qquad & roomNo = o \\ 17,62 = m \qquad & squareFt = m * 10.76 \\ & phone = \diamondsuit \end{aligned}$$

In general not all systems of equations can be resolved as easily as in this example. The example already shows that data types like strings together with the concatenation operator “$\circ$” may occur that do not form a mathematical group. Hence a useful tool support for the presented approach must provide the ability to plug-in user defined solution strategies.

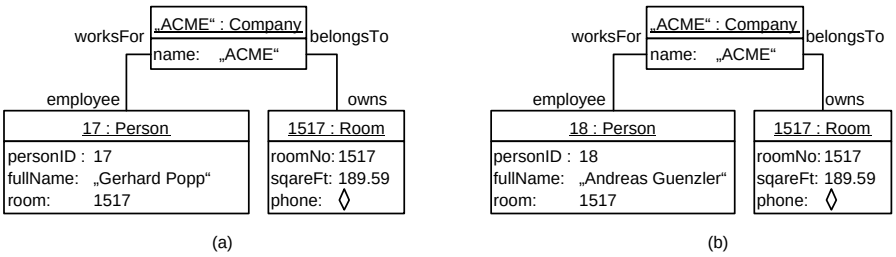

Figure 9. The created model fragments (a) $gmf_{0,0}$ and (b) $gmf_{0,1}$

Since all the values can be derived deterministically, mft is a model fragment transformation with respect to Definition 3.9. The generated model fragment $gmf_{0,0} := mft(mfm_0, r_0)$ is depicted in Figure 9 (a). Likewise, we get the second generated model fragment $gmf_{0,1}$ depicted in Figure 9 (b).

When the first rule r_0 is applied it holds that $mf_0 = (\emptyset, \emptyset)$. Thus,

$$mf_0 \cup_{\mathfrak{m}} mft(mfm_0, r_0) = (\emptyset, \emptyset) \cup_{\mathfrak{m}} gmf_{0,0} = gmf_{0,0}$$

i.e. the model fragment $gmf_{0,0}$ is merged into an empty model fragment. Resolving the *apply* function according to 3.10 we retrieve gmf_0 (c.f. Fig. 10):

$$apply(m_\alpha, r_0, (\emptyset, \emptyset)) = gmf_{0,0} \cup_{\mathfrak{m}} gmf_{0,1} = gmf_0$$

The value for the attribute phone will be generated by the application of the second rule. According to Theorem 3.11 we could also permute the model fragment matches mfm_0 and mfm_1 and would still get the same result.

The application of the second rule r_1 is performed analogously, but it starts with gmf_0. Since the merge operator ensures that already created objects are merged with those that are newly created, the phone attribute is set correctly in all existing Room objects. The result of the transformation is shown in Figure 4.

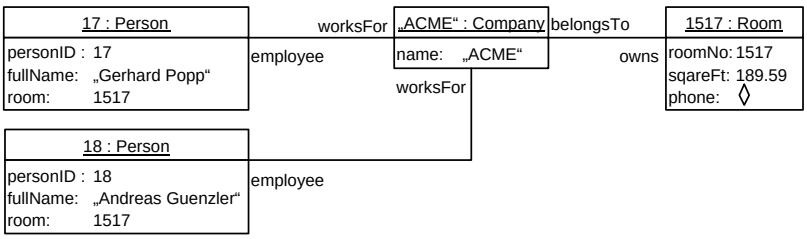

Figure 10. The result of $apply(m_\alpha, r_0, (\emptyset, \emptyset)) = gmf_0$: The Person and Room objects are already created, but there is no value for the phone attribute yet.

Finally we can state that our rule set is applicable, because the two rules comply with the two postulations of Theorem 3.17. First, all equations have a unique solution. Second, there are no different object variables of the same type on the right side of any rule and all terms for attributes stem only from object variables which determine the identity of an object. Note that the term o within the object variable "Person" of rule r_0 is no problem as an Employee worksIn exactly one Office. Also the use of $\diamondsuit$ terms ensure that there are no contradictory terms for any two different rules. Thus the rule set is applicable.

5 Conclusion

We have shown that the BOTL approach is helpful for transforming models according to two different metamodels. Besides the illustrated example, BOTL is used within the FORSOFT II project Automotive [15] for the transformation of models between three different CASE tools: DOORS, The UML Suite, and ASCET-SD. As already stated in this paper, we have a mathematical model of restricted UML class diagrams as a formal base for BOTL. In the paper we have informally shown theorems which deal with the applicability of rules. Even more interesting are formally shown heuristics when a transformation generates models for arbitrary source models. We are currently working on improved statements to recognize that a given rule set is bijective. Bijective means that the result of the reverse transformation of the transformation of a source model is isomorphic to the source model. Also we are currently developing some tool support for the BOTL.

Finally we would like to thank the Bayerische Forschungsstiftung for funding our work in FORSOFT II. We also thank Bernhard Schätz and Martin Rappl for many fruitful discussions and suggestions.

References

[1] *OMG Unified Modeling Language Specification* (1999).

[2] *XML Metadata Interchange (XMI)*, OMG Document ad/99-10-02 (1999).

[3] *XSL Transformations (XSLT) Version 1.0*, W3C Recommendation (1999).

[4] *Simple Object Access Protocol (SOAP) 1.1* (2000).

[5] *XML Schema Home Page* (2002), http://www.w3.org/XML/Schema.

[6] Bouneffa, M. and N. Boudjlida, *Managing Schema Changes in Object-Relationship Databases*, in: M. Papazoglou, editor, *OOER 95 Object-Oriented and Entity-Relationship Modeling*, number 1021 in LNCS (1995).

[7] Bray, T. and J. Paoli, "Extensible Markup Language (XML) 1.0," W3C, 1998, available at http://www.w3.org/TR/1998/REC-xml-19980210.

[8] Busse, S., *A Specification Language for Model Correspondence Assertions*, Technical report, Technische Universität Berlin, Fachbereich 13 Informatik, Computergestützte Informationssysteme (1999).

[9] Casais, E., "Managing Class Evolution in Object-Oriented Systems," The Object-Oriented Series **Object-Oriented Software Composition**, Prentice Hall, 1995 .

[10] Chen, S.-K., M.-L. Lo, S. Padmanabhan and J.-Y. Chung, *XAS: A system for accessing componentized, virtual XML documents*, in: *Proc. 23rd Int'l Conf. Software Engineering* (2001), pp. 493–502.

[11] Liu, C.-T., P. K. Chrysanthis and S.-K. Chang, *Database Schema Evolution through the Specification and Maintenance of Change on Entities and Relationships*, in: P. Lucopoulos, editor, *Entity-Relationship Approach - ER'94*, number 881 in LNCS (1994).

[12] Nagl, M., editor, "Building Tightly Integrated Software Development Environments: The IPSEN Approach," Springer, 1996.

[13] OMG, *OMG Meta Object Facility (MOF)*, Technical Report 1.3.1, formal/01-11-02, Object Management Group (OMG), www.omg.org (2001).

[14] Schürr, A., *Specification of Graph Translators with Triple Graph Grammars*, in: G. Tinhofer, editor, *Proc. WG'94 20th Int. Workshop on Graph-Theoretic Concepts in Computer Science*, number 903 in LNCS (1994), pp. 151–163.

[15] v.d. Beeck, M., P. Braun, M. Rappl and C. Schröder, *Automotive Software Development: A Model Based Approach*, number 2002-01-0875 in SAE Technical Papers Series, Society of Automotive Engineers (2002).

[16] Vidal, V. M. and M. Winslett, *A Rigorous Approach to Schema Restructuring*, in: M. Papazoglou, editor, *OOER 95 Object-Oriented and Entity-Relationship Modeling*, number 1021 in LNCS (1995).