Innovations in Systems and Software Engineering
DOI 10.1007/s11334-011-0160-5

Coupling-based Transformations of Z Specifications into UML Diagrams

Andreas Bollin

Received: 30 April 2011 / Accepted: 1 September 2011

Abstract Due to their accuracy in describing systems, formal specifications can play an important role during forward as well as reverse engineering activities. However, besides dense mathematical expressions, their lack in visually appealing notations impedes their use and exchange among different stakeholders. One solution to this problem is to enrich the specification by other views, in most cases UML diagrams. But the mapping is not trivial, and existing approaches have their impediments, among them the assignment of methods to classes – which has to be re-done by hand quite often.

By the example of Z, this paper demonstrates that the situation can be improved. The new approach combines existing mapping strategies, but additionally lets the assignment of methods rest on quality-related measures. The basic idea is to balance the values of coupling for all methods within and between the UML classes. With that, two issues are addressed: firstly, the mapping of sets, types, and operations (to UML classes and UML methods) is based on reproducible measures that are intuitively comprehensible. Secondly, implementations based on the resulting UML class diagrams very likely also have comparable quality-related properties.

Keywords Formal Specification · Transformation · Slice-based Coupling Measures

1 Introduction

Today's systems become more and more software-intensive which typically means that software is the major component that provides the needed functionality. With that, reliability and dependability considerations gain in importance, and, by following the arguments in [14, p.55], formal methods hence lend themselves back to the software engineering community. Several stories of success and myths are around (and might balance each other [11,5,28]), but when used appropriately, they form the basis for ongoing software engineering steps like code- and test-case generation.

Alpen-Adria Universität Klagenfurt
Universitätsstrasse 65, 9020 Klagenfurt
Tel.: +43-463-27003516
Fax: +43-463-27003599
E-mail: Andreas.Bollin@uni-klu.ac.at

But there is a drawback. Even if the system has been refined correctly, there is another issue to be solved: the validation problem. What sounds like a requirements elicitation problem also has to do with the question of choosing a suitable notation (as several stakeholders in the project will have to agree upon the specification). Here, the mathematically dense notations might be an impediment. A pragmatic solution is therefore to combine formal specifications and other (semi-formal) graphical notations and thus to provide an additional view onto the specification.

As illustrated in Section 2.1, there are already approaches dealing with the mapping between graphical notations (like UML) and formal specifications. However, these transformations are not without restrictions. Formal specifications are usually not object-oriented and they are not necessarily concerned with classes. This contribution makes an attempt to resolve one of the stumbling blocks in the mapping of specifications to UML diagrams: that of a "pragmatic" layout of the diagrams. But what is the exact problem?

Well, apart from a possible loss in semantic expressiveness, the resulting diagrams quite often have to be re-formatted by hand so that developers and managers are satisfied with them and perceive them as useful. Especially the assignment of operations to classes is a problem that has to be resolved by human beings. Existing solutions do their best in finding pertinent classes for methods, but the mapping is not necessarily recognized as useful, and when the specification scales, classes with unbalanced design might be created.

Here, by the example of Z specifications [29], the paper suggests an alternative way in finding an improved mapping. The basic idea is to focus on the values of coupling between the operations in the specification and to use this information to find class assignments that are "optimal". Basically, the objective is to find a mapping such that the coupling between the methods within a class is at a maximum, and the coupling to methods in other classes is at a minimum. The optimization process will be discussed in more details in Section 4.

The contribution is structured as follows. Section 2 explains the need for mapping strategies in more detail and presents approaches transforming UML diagrams to formal specifications and vice-versa. It also gives attention to some limitations of existing approaches. Section 3 discusses the transformation process for Z specifications. It also provides the necessary background for the calculation of slice-based coupling values. Section 4 then explains the approach by making use of a small Z specification. Finally, the paper concludes with a short summary and an outlook.

Type	**Symbol**	**A**	**B**
Relation	$A \leftrightarrow B$	$*$	$*$
Partial	$A ⇸ B$	$*$	$0\,..\,1$
Total	$A \rightarrow B$	$*$	1
Part. Inj.	$A ⤔ B$	$0\,..\,1$	$0\,..\,1$
Part. Surj.	$A ⤀ B$	$1\,..\,*$	$0\,..\,1$
Total Surj.	$A \twoheadrightarrow B$	$1\,..\,*$	1
Total Bij.	$A ⤖ B$	1	1
Total Inj.	$A \rightarrowtail B$	$0\,..\,1$	1

Table 1 As defined in [16], relations between sets A and B are mapped to associations with the given multiplicities.

2 Formal Specification Transformations

The idea of combining formal specifications with other notations is not new, and the existing approaches help to focus on orthogonal properties of the underlying system. Understandability is gained due to the different views, and decisions are alleviated. Dick argues in [7] that confidence and acceptability is raised and changes of the system are alleviated. As the transformation is possible in two directions, also some weaknesses of purely graphical notations can be eliminated. The following section summarizes existing approaches briefly and then moves on to the issue of the improved mapping strategy.

2.1 Related Work

Besides formal extensions to existing graphical notations (e.g. Petri-nets with Z extensions [13] or VDM-link to UML [8]), two classes of approaches for specification transformations are existing.

The first class comprises approaches that map graphical notations to formal specifications. UML is wide-spread, so most of them take static UML diagrams and generate formal state descriptions from it (e.g. UML to Z [9,17], or UML to Z++ [20,27]). The approaches have in common that formal specification skeletons are generated which then have to be completed by the designer/developer. After completion the resulting predicates are simplified, resulting in a compact formal specification. So, semantics has to be added by the designer, but the specification can then be taken to prove properties of the system and the results can then be mapped back to the design documents in order to eliminate deficiencies.

The second class comprises approaches that map formal specification to some graphical notation. One early approach is the visualization of Z defined by Kim [19], who makes use of constraint diagrams [18]. The notation is able to express predicate logic, but, unfortunately, there is no integration into existing frameworks. In addition, constraint diagrams look differently from UML diagrams, and so the understanding among different stakeholders is impeded again.

When not the whole semantics of a specification at hand has to be mapped, then UML, being state-of-the-practice, is a possible candidate. With the involvement of members of the precise UML group[1] in the standardization process, it also gets an interesting target for the transformation process. The approach of Fekih et.al maps B specifications to UML [10]. It takes the state space of the specification and creates an UML class for every abstract set that is element in the domain of relations. The transformation rules are simple and lead to incomplete class diagrams as operations are not regarded. In addition to that the generated classes are not associated. Idani and Ledru improve the approach by mapping occurring relations to UML associations [16] (as summarized in Table 1). Furthermore they take operations into account and provide an algorithm for mapping an operation as a method to the most suitable class (called pertinent class). Altogether this leads to a more complete static UML diagram, though their approach neglects the dynamics behind operations.

In [2] the approach of Idani and Ledru is mapped to Z and extended by rules to cover also activity diagrams. This is done by regarding control and data dependencies that have been recalculated beforehand (via a specification transformation that is explained in more details in [24]). The approach has been refined and integrated into a Java-based environment called *ViZ* by Lessacher in 2007 [21]. Neverthe-

[1] The precise UML group, created in 1997, tries to bring international researchers and practitioners together in order to develop the Unified Modelling Language (UML) as a well defined modeling language. See http://www.cs.york.ac.uk/puml/index.html for more details.

less, also this set of rules has its limitations as the mapping of operations is handled in such a way that all operations are part of a root class with the stereotype “⟪system⟫” (as will be shown in Section 3.2).

2.2 Mapping Strategies

Algorithms that map formal specifications to UML diagrams follow a pragmatic approach: sets correspond to classes and relations correspond to associations. This mapping is quite natural as programmers often use classes to represent abstract types. Their interrelation is then expressed by various forms of associations between them. The mapping of operations to classes is more sophisticated. One way in dealing with the situation is to take a look at the number of uses (and references) of these sets of types in the operations and to assign them to the most frequently used classes (as done by Idani and Ledru in [16] and extended later on in [15]). A contrary approach (as used in [2]) is to collect all of the operations and to put them into a separate class. However, both strategies have drawbacks:

– For the first approach it might happen that more than one class is pertinent for an operation. It is then up to the user to decide where to put the method to. While this is not a problem for the second approach, putting all operations into one class definitely does not scale-up very well.
– Besides abstract types, also the state-space is relevant. The first approach does not deal with this information. The second approach creates a system class for every state, but whenever several state spaces are included it is again not defined where a method has to be mapped to.
– None of the transformation rules do take implementation related issues into account. While implementation details are not an issue for formal specifications, this point gets important when (a) communicating them to different stakeholders and (b) using them as the basis for the ongoing development process.

An improved mapping strategy should take these issues into account, and especially the third aspect can be used to improve the existing mapping strategies. When a specification is the basis for an implementation (especially when specifications are refined) then it is very likely that the dependencies between operations are still prevalent and affect implementation-specific properties. This includes the dual properties of coupling and cohesion and operations should be created in such a way that the values for coupling and cohesion are minimized (maximized) whenever possible.

The idea now is quite simple: one can use specification measures to decide where to map operations to. In [4] it is shown that slice-profiles [26,23,30] can be computed for Z specifications and that they can then be used to calculate specification-based coupling and cohesion measures. It is also demonstrated that the behavior of these measures can be compared to their corresponding measures in the field of ordinary programming languages. So, based on these values, it is suggested to add another rule set that regards the average values of coupling for every operation. Whenever necessary, it moves the operations to other (pertinent) classes in such a way that at the end of this process all values have reached an optimum (which means high values for coupling between methods in one class and lower values for coupling between methods of different classes).

Measure	*Definition*
Inter-Schema Flow $\mathcal{F}(\psi_s, \psi_d)$ measures the number of primes of the slices in ψ_d that are in ψ_s	$\frac{\lvert(SU(\psi_d)\cap\psi_s)\rvert}{\lvert\psi_s\rvert}$
Inter-Schema Coupling $C(\psi_s, \psi_d)$ computes the normalized ratio of the flow $\mathcal{F}$ in both directions	$\frac{\mathcal{F}(\psi_s,\psi_d)\ \lvert\psi_s\rvert+\mathcal{F}(\psi_d,\psi_s)\ \lvert\psi_d\rvert}{\lvert\psi_s\rvert+\lvert\psi_d\rvert}$
Schema Coupling $\chi(\psi_i)$ is the weighted measure of Inter-Schema Coupling C of ψ_i and all n other schemata	$\frac{\sum_{j=1}^{n} C(\psi_i,\psi_j)\ \lvert\psi_j\rvert}{\sum_{j=1}^{n}\lvert\psi_j\rvert}$

Table 2 Slice-based measures for Inter-Schema Flow and Coupling as introduced for Z specifications in [4].

Section 3 now introduces the necessary background for the calculation of slice-based coupling measures and the refined set of transformation rules.

3 Slice-based Transformation Process

The calculation of slice-based measures goes back to the notion of a static specification slice as introduced by Oda and Araki [25] and Chang and Richardson [6]. Their idea is to look for predicates that are part of pre- and postconditions and to introduce “control” dependencies between them. Their idea has been refined and extended by Bollin [1] which also led to the development of an environment called *ViZ* that now supports reverse engineering of formal Z specifications by slicing, chunking and clustering techniques [3].

3.1 Slice-based Coupling

For the definition of the measure of coupling we first need to introduce the notion of specification slice profiles and the union of all its slices. The basic idea is quite simple: for every post-condition prime in a schema ψ one has to calculate the corresponding slices. The set of all possible specification slices is called *Slice Profile* ($SP(\psi)$). The union of all slices in $SP(\psi)$ is called *Slice Union* ($SU(\psi)$).

Nr.	*Rule*
1	Every section in a Z specification corresponds to one UML class diagram.
2	Every state schema corresponds to a root class with the stereotype ⟨⟨system⟩⟩ and the name of the schema.
3*	Every given set A corresponds to a class in the UML specification, getting the name of the basic type.
4	Every inclusion of a state A in the declaration part of a schema B corresponds to an aggregation of the classes A and B (where B is the whole class).
5	Every use of a given set or free-type A in a state schema B corresponds to a ⟨⟨use⟩⟩ association between the classes A and B and with multiplicities $(*, 1)$.
6*	Every variable representing relationships between entities in a state schema is translated to associations. It holds that (i) multiplicity is resolved by the mapping rules presented in Table 1, (ii) subsets between relations are resolved by a subset constraint, and (iii) an identifier A representing a set of a type B is resolved by a generalization between class A and super-class B. Associations do get role-names. They are built by combining the first characters of the source class and association name.
7	Every use of a given set or free-type A in an operation schema B that has been assigned to a class C is mapped to a ⟨⟨use⟩⟩ association with multiplicities $(*, 1)$ between A and C.
8	Every identifier A in the declaration part of an operation schema is mapped to a parameter of the corresponding method B, annotated by "In" for input and "Out" for output. When the output is a set, then a Vector of the type is returned.
9	Every operation schema A is added as a method to those system root class which has been included in the operations declaration part. When there are several root classes possible, then A is added to a system class called "Operations". In this case a ⟨⟨use⟩⟩ association with multiplicities $(1, 1)$ is introduced between these classes. The initialization schema is mapped as a constructor to its corresponding system class.

Table 3 (Part I) Z mapping rules for the static part of a specification as defined by Bollin and Lessacher [2,21]. Rules following the approach of Idani and Ledru [17] are marked by an asterisk.

The calculation of coupling follows the definitions to be found in [12]. First, an Inter-Schema Flow F is specified. It describes how many primes of the slices in the slice union are outside of the schema. Inter-Schema Coupling C is then computed by the normalized ratio of this flow in both directions. Finally, Schema Coupling χ is calculated by considering the Inter-Schema Coupling values to all other schemata. The definitions of the measures are summarized in Table 2.

3.2 Transformation Rules

The mapping of Z specifications to static class diagrams as introduced in [2] is based on the idea of Idani and Ledru [17]. However, the approach omits assigning the operations to derived classes. Instead, it introduces one or several system classes and assigns the operations to them. When an operation can be assigned to several system classes, then a helper class called "Operations" is created and the operation is assigned to it.

Nr.	*Rule*
10	Every free-type corresponds to a UML class with the stereotype ⟨⟨datatype⟩⟩. Every constant of a free-type A is mapped to an attribute of the corresponding UML class. Every dataset of the constructor of a free-type A corresponds to an instance variable in the corresponding class A. Every link-set of the constructor of a free-type A corresponds to a recursive association of the corresponding class A with the name of the constructor and multiplicities $(0\,..\,1, 1)$.
11	Every global constant A representing a subset of a free-type B is mapped to a UML class with the name of A and the stereotype ⟨⟨datatype⟩⟩. Additionally, a generalization between the classes A and B is introduced. Every element of a subset A corresponds to a class attribute of class A and gets the name of the element and the type of A.
12	Every identifier in the declaration part of a schema A representing a sequence of a state schema or a type B is mapped to an association between classes A and B. For non-empty sequences the multiplicities are $(0\,..\,1, 1\,..\,*)$, otherwise $(0\,..\,1, *)$.
13	Every identifier in the declaration part of a schema representing a relation between a type A and a sequence of type B corresponds to an association between classes A and B. Multiplicity is resolved by the mapping rules presented in Table 3.2 where the multiplicity for class B is $1\,..\,*$ for non-empty sequences, and $*$ otherwise.

Table 4 (Part II) Extended set of rules (compared to the rule-set presented in [2] and [21]) for the static part of a Z specification, now also dealing with sequences and free-types.

Experiments with larger specifications showed that free-types and sequences are used quite often. As they are not covered by the rules defined in [2], the existing set of rules had to be extended again. The resulting mapping strategy[2] can be found in Tables 3 and 4.

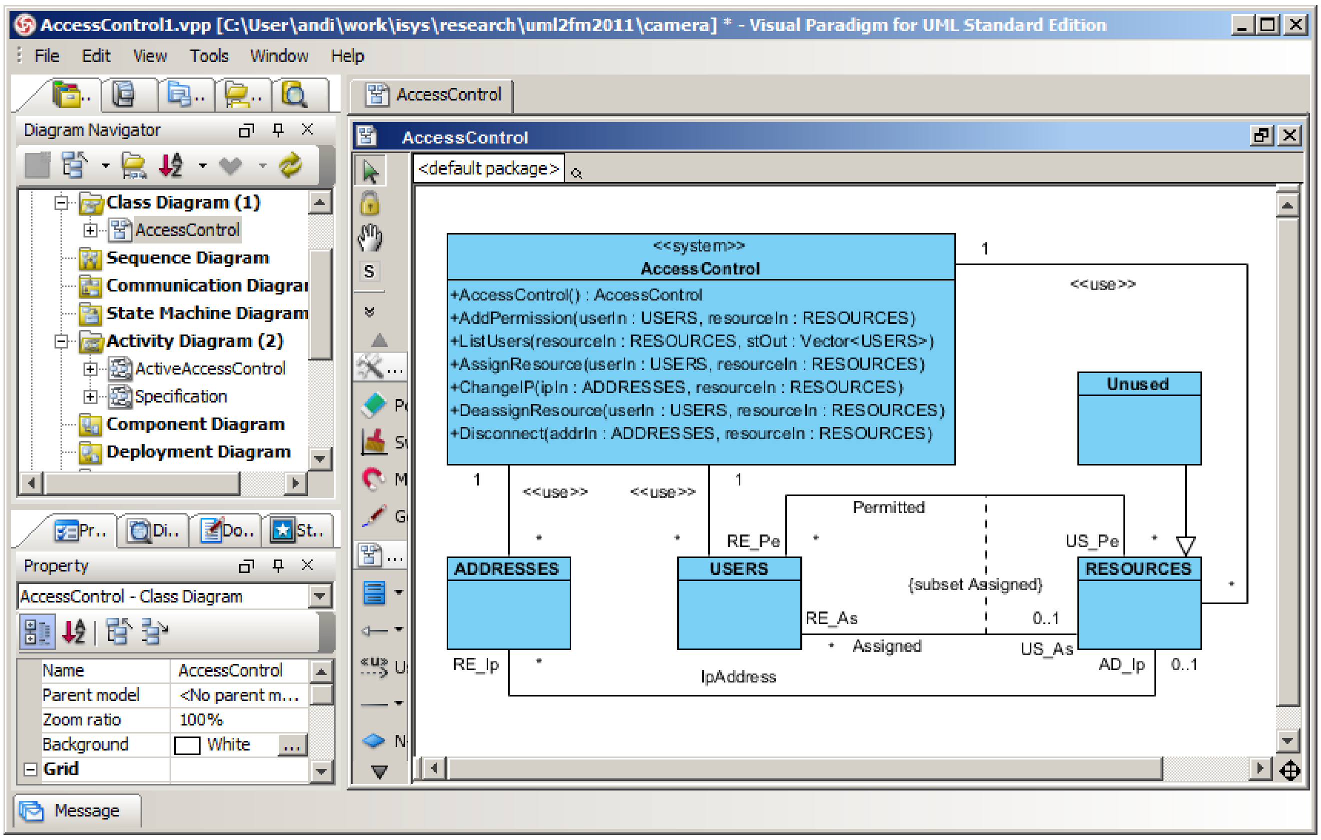
Fig. 1 presents the result for the transformation of the "Access Control Specification" as done by the *ViZ* environment[3]. *ViZ* generates an *XML* file that can be imported by UML modeling tools like *VisualParadigm*[4]. According to the mapping rules the system class contains all operations and the initialization schema as a constructor. As there are three given set definitions (called *USERS*, *RESOURCES*, and *ADDRESSES*), three classes are introduced and connected to this system class. Additionally, the identifier *Unused* is modeled as a subset of *Resources*, and *Assigned* and *Permitted* are enriched by a subset constraint.

In fact, the set of rules operates well when there is only a small number of given sets and a few operations. With larger specifications that contain a lot of operations the approach of taking given sets as classes and putting them into the system

[2] A description of the mapping rules (also covering activity diagrams) including Java-like pseudo-code can be found in [21].

[3] The specification is also used by other authors to demonstrate their mapping strategies and has thus been selected for this contribution.

[4] Visual Paradigm is part of the Visual Paradigm Suite and is free for academic sites. For more information see http://www.visual-paradigm.com. Page last visited: August 2011.


AccessControl1.vpp [C:\User\andi\work\isys\research\uml2fm2011\camera] * - Visual Paradigm for UML Standard Edition
File Edit View Tools Window Help
Diagram Navigator
Class Diagram (1)
AccessControl
Sequence Diagram
Communication Diagram
State Machine Diagram
Activity Diagram (2)
ActiveAccessControl
Specification
Component Diagram
Deployment Diagram
Property
AccessControl - Class Diagram
Message
AccessControl
<default package>
<<system>>
AccessControl
+AccessControl() : AccessControl
+AddPermission(userIn : USERS, resourceIn : RESOURCES)
+ListUsers(resourceIn : RESOURCES, stOut : Vector<USERS>)
+AssignResource(userIn : USERS, resourceIn : RESOURCES)
+ChangeIP(ipIn : ADDRESSES, resourceIn : RESOURCES)
+DeassignResource(userIn : USERS, resourceIn : RESOURCES)
+Disconnect(addrIn : ADDRESSES, resourceIn : RESOURCES)
Unused
ADDRESSES
USERS
RESOURCES
<<use>>
<<use>>
<<use>>
1
1
1
*
*
*
*
*
Permitted
RE_Pe
US_Pe
{subset Assigned}
RE_As
0..1
Assigned
US_As
RE_Ip
IpAddress
AD_Ip
0..1

Name	AccessControl
Parent model	<No parent m...
Zoom ratio	100%
Background	White
Grid	

Fig. 1 Applying rules 1 to 13 leads to a static UML Diagram for the Access Control system specification (see Appendix). The UML diagram has been exported by the *ViZ* environment and imported by Visual Paradigm 8.0 (and making use of the "Auto Layout" algorithm).

$C(\psi_s, \psi_d)$	S1	S2	S3	S4	S5	S6	S7	S8	$\chi(\psi_i)$
S1 (AccessControl)	1.000	0.250	0.444	0.800	0.400	0.364	0.400	0.400	0.383
S2 (InitAccessControl)	0.250	1.000	0.154	0.222	0.143	0.133	0.143	0.143	0.215
S3 (AddPermission)	0.444	0.154	1.000	0.400	0.267	0.250	0.267	0.267	0.248
S4 (ListUsers)	0.800	0.222	0.400	1.000	0.364	0.333	0.364	0.364	0.343
S5 (AssignResource)	0.400	0.143	0.267	0.364	1.000	0.235	0.250	0.250	0.233
S6 (ChangeIP)	0.364	0.133	0.250	0.333	0.235	1.000	0.235	0.235	0.220
S7 (DeassignResource)	0.400	0.143	0.267	0.364	0.250	0.235	1.000	0.125	0.233
S8 (Disconnect)	0.400	0.143	0.267	0.364	0.250	0.235	0.125	1.000	0.233

Table 5 Values for Inter-Schema Coupling C and schema coupling $\chi(\psi_i)$ for the Z schemata of the *Access Control specification*.

class (as can be seen in Fig. 1) leads to huge static UML diagrams no developer would generate by hand. However, making use of coupling measures mitigates this situation. It also leads to an assignment of operations to classes such that the values for coupling within a class are maximized and the values for coupling to other classes are minimized.

4 Coupling-based Mapping

Contrary to existing approaches where pertinent classes are identified by the number of use or references to identifiers, the strategy for identifying pertinent classes is now based on the analysis of the values of Inter-Schema Coupling C. The objective is to "optimize" their values for the whole specification:

Definition 1 *The value for Inter-Schema Coupling of the transformed specification is optimal, when there is no other assignment of operations to classes such that the average value of all Inter-Schema Coupling values within the classes can be increased.*

When taking a look at the values of schema coupling $\chi(\psi_i)$ for the Access Control specification in Table 5 then we can see that the values are quite evenly distributed. Nevertheless, there are variations. The state space *AccessControl* has the highest values as it is connected to all the other schemas, and *InitAccessControl* has the lowest relation to all the other schemas. For an optimal distribution (where the values for coupling are maximized for methods within classes) the following additional rules are pursued:

Rule 14 *For every system class select all operations that have an Inter-Schema Coupling value lower than a given threshold ρ, and for every selected operation determine the set of possible (pertinent) class candidates.*

Rule 15 *Move every operation to one class-candidate by the following strategy: (a) select a class from the set of candidates that has no method at all, and (ii) if there is no such class select a class so that the average value for Inter-Schema Coupling for all class candidates stays at a maximum.*

Rule 16 *Look at every class and check whether a swap or delegation of methods between two classes increases the average values for Inter-Schema Coupling. If so, then delegate or swap the methods. Continue the steps till there are no more improvements.*

The problem of finding the optimal solution can be compared to the knapsack problem [22]. However, we know a bit about the preferred properties of the resulting classes and the optimal solution is approximated by the rules due to a simple strategy: at first it tries to avoid empty classes (the value for inter-class coupling would be zero). The first class-candidate that matches gets the method assigned to it, yielding a value for intra-schema coupling equal to 1 (per default). Secondly, when there is no class-candidate that is empty, then the method is assigned to one of the classes in such a way that the average value for schema coupling does not decrease too much. The last step is to try to find a swapping of the methods such that the average value for coupling increases a bit.

With the suggested procedure, one is able to calculate a balanced distribution of the operation schemas to the classes. In fact, the first couple of steps are quite straight forward and efficient. Choosing between all operations according to a given threshold can be done in linear time (when the Inter-Schema Coupling matrix is already given). The identification of the set of class candidates per method can also be conducted in one run when parsing the parameter lists of the methods. The same holds for the next step, the assignment of the methods to the classes. Empty classes are selected easily, and only in the case of already occupied classes one has to take a look at the matrix again and compute the average of the related Inter-Schema Coupling values.

Definitely more complex is the implementation of the last two steps, the delegation and swapping of methods. In the worst case, all possible combinations of methods per class have to be considered till the algorithm stops. In fact, it might be that there is more than one optimal solution[5], but the algorithm stops when no further improvement of the average value of the Inter-Schema Coupling values can be found.

This process of assigning methods to classes – according to a balanced distribution of Inter-Schema Coupling values – is now demonstrated by the example of the Access Control specification (see Appendix for the Z specification).

4.1 Example

The starting point of our example is the UML layout as presented in Fig. 1 (so rules 1 to 13 have already been applied). There, all the methods are element of one system class. Rule 14 now tells us that (for a given value ρ) we have to decide upon the methods we want to delegate to associated classes. For ρ let us assume to start with a value of 0.7, which means that we want to delegate those methods that have a value of Inter-Schema Coupling lower that 0.7. The relevant methods (see Table 5, rows $C(\psi_s, \psi_d)$ and $S1$) are therefore: *AddPermission*, *AssignResource*, *ChangeIP*, *DeassignResource*, and *Disconnect*. (*ListUsers* has a value $C(\mathit{ListUsers}) = 0.800$ within the system class, so a value higher than ρ and it is skipped. The initialization schema is also skipped as it becomes the constructor of the system class). For the selected methods we have to calculate the class-candidates (so the pertinent classes that are associated with the system class and that are referred to or used in the methods). The class-candidates and their pertinent classes are as follows:

- *AddPermission* → *USERS* or *RESOURCES*.
- *AssignResource* → *USERS* or *RESOURCES*.
- *ChangeIP* → *ADDRESS* or *RESOURCES*.
- *DeassignResource* → *USERS*, *RESOURCES*, or *Unused*.
- *Disconnect* → *ADDRESS*, *RESOURCES*, or *Unused*.

We now apply rule 15 to the specification. In this first step we are taking a look at the methods one by one. At first, *AddPermission* could be delegated to either *USERS* or *RESOURCES* (as both of the classes are still empty). In our case let us take the *USERS* class, which also means that the value for coupling within the USERS class increases from 0 to 1. Next is *AssignResource* which goes to the *RESOURCES* class as *USERS* already contains one method. *ChangeIP* is moved to *ADDRESSES*, *DeassignResource* is moved to class *Unused*. For the *Disconnect* operation we have the choice between a delegation to three classes (either *RESOURCES*, *ADDRESSES*, or *Unused*) that already contain operations. Thus the value of C has to be calculated for these three options:

[5] This might be the case as the initial distribution of methods to empty classes is non-deterministic and the values of Inter-Schema Coupling between some of the operation schemas can be the same.

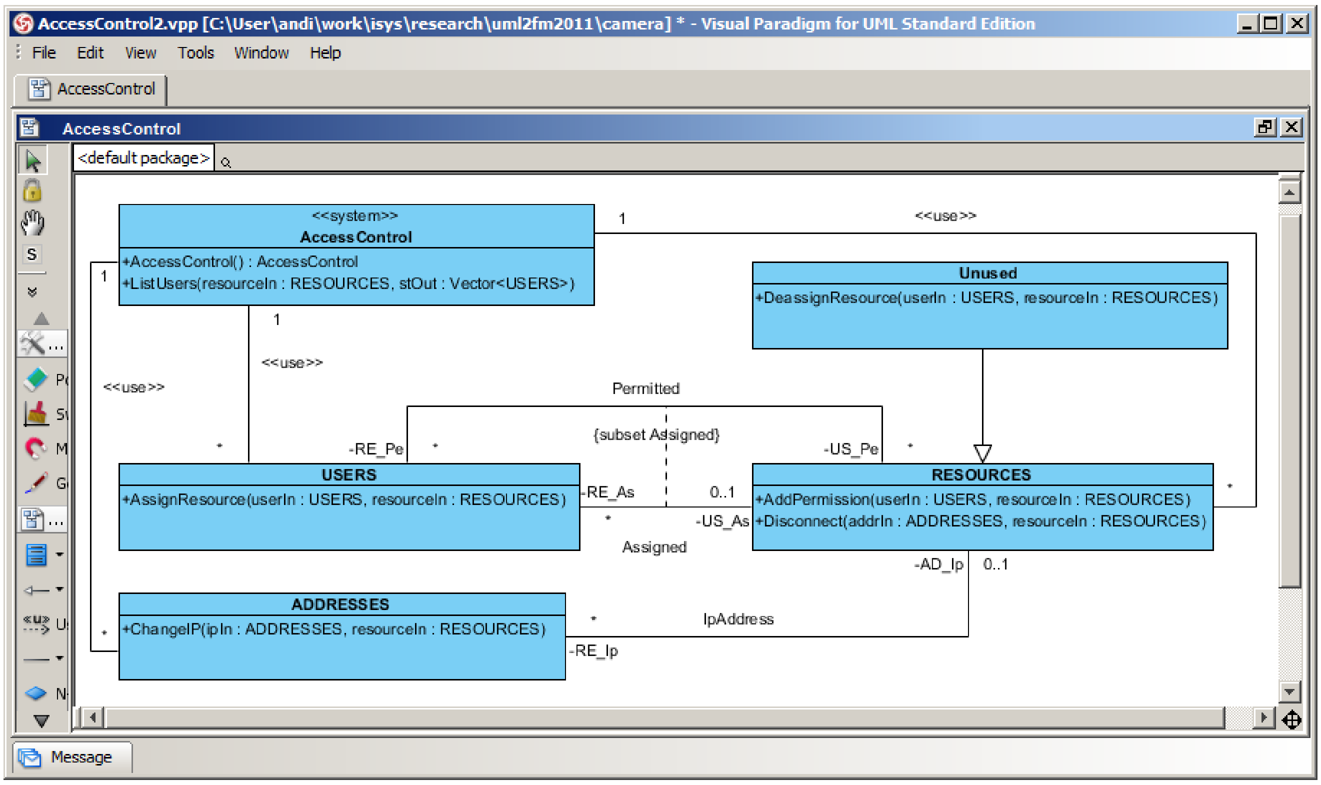

AccessControl2.vpp [C:\User\andi\work\isys\research\uml2fm2011\camera] * - Visual Paradigm for UML Standard Edition
File Edit View Tools Window Help
AccessControl
AccessControl
<default package>
<<system>>
AccessControl
+AccessControl() : AccessControl
+ListUsers(resourceIn : RESOURCES, stOut : Vector<USERS>)
Unused
+DeassignResource(userIn : USERS, resourceIn : RESOURCES)
USERS
+AssignResource(userIn : USERS, resourceIn : RESOURCES)
RESOURCES
+AddPermission(userIn : USERS, resourceIn : RESOURCES)
+Disconnect(addrIn : ADDRESSES, resourceIn : RESOURCES)
ADDRESSES
+ChangeIP(ipIn : ADDRESSES, resourceIn : RESOURCES)
<<use>>
<<use>>
<<use>>
Permitted
{subset Assigned}
Assigned
IpAddress
-RE_Pe
-US_Pe
-RE_As
-US_As
-AD_Ip
-RE_Ip
1
1
1
0..1
0..1
*
Message

Fig. 2 Applying rules 14 to 16 leads to a more balanced static UML Diagram for the Access Control system specification (see Appendix). Please note that the annotations containing the predicates, even though exported by the *ViZ* environment, have been omitted for reasons of readability.

- $C(\mathit{Disconnect}, \mathit{AssignResource}) = 0.250$.
- $C(\mathit{Disconnect}, \mathit{ChangeIP}) = 0.235$.
- $C(\mathit{Disconnect}, \mathit{DeassignResource}) = 0.125$.

In order to keep coupling for the classes at a maximum, *Disconnect* is moved to *RESOURCES* as a second method (as it already contains *AssignResource*). After applying rule number 15, the assignment of methods to classes looks as follows:

- *USERS* contains $\{\mathit{AddPermission}\}$.
- *RESOURCES* contains $\{\mathit{AssignResource}, \mathit{Disconnect}\}$.
- *ADDRESSES* contains $\{\mathit{ChangeIP}\}$.
- *Unused* contains $\{\mathit{DeassignResource}\}$.

We are now able to apply rule 16 to the UML diagram. At first, we try to swap the methods so that we manage to increase the average values of coupling. Swapping and delegation only makes sense for classes that contain more than one method, so we are looking at the *RESOURCE* class.

- The method *AssignResource* could be swapped with the method *AddPermission* of the *USERS* class. As the values for coupling are $C(\mathit{Disconnect}, \mathit{AssignResource}) = 0.250$ and $C(\mathit{Disconnect}, \mathit{AddPermission}) = 0.267$, a swap of the methods would indeed increase the value of coupling in total a bit.
- The method *Disconnect* could be either swapped with *ChangeIP* or with the method *DeassignResource*. However, there is no variation of the swaps that does increase the values of coupling anymore.

The next step of Rule 16 is to try to delegate some of the methods to other classes and thus to increase the value of coupling on class level.

- *Disconnect* could be either moved to the *ADDRESSES* or *Unused* class. Moving it to *ADDRESSES* would mean that C decreases from 1.0 to 0.235 for the class named *ADDRESSES* and that C increases from 0.267 to 1.0 for the class named *RESOURCES*. The difference is $\Delta = -0.032$. The delegation does not improve the situation as a whole. Moving *Disconnect* to *Unused* would decrease C from 1.0 to 0.125 for the *Unused* class and increase it from 0.267 to 1.0 for the *RESOURCES* class. The difference is $\Delta = -0.142$, and a delegation is of no use again.
- Delegating *AssignResource* to the *USERS* class as a alternative step would be possible, but is does not change the value for coupling (as $\Delta = \pm 0$).

So, in both cases, the delegation does not maximize the values of coupling in the mean. With no more swaps or delegations possible we are done. The resulting diagram is displayed in Fig. 2.

4.2 Discussion and Improvements

By following the rules 14 to 16 we end up in a transformation that maximizes the values for schema coupling in every class. The approach is based on the assumption that the structure of the specification and the structure of the resulting implementation are, at least along general lines, comparable. Though this might not be the case in all situations, there are empirical evidences that at least the measures are correlated [4]. So, the approach presents a heuristic that is worth to be applied. Also the problems of scalability are influenced positively.

With the introduction of a threshold value ρ we are additionally able to control how many methods in the system class are to be delegated. Thus, an increase in the number of schema operations is not so much a problem anymore. Two other aspects of scalability are not addressed in this paper, but they can also be dealt with easily:

– When the specification is big and when it contains a lot of given sets or free-types, then the number of classes is very high. One solution to this problem is to extend Rules 3 and 10 by a simple manual step: as the mapping strategy uses classes as types, the user should decide about which of the given sets or free-types to map to classes and which to ignore (treating them as basic types). This extension could keep the resulting UML diagram smaller and also help shifting the view onto the specification a bit (depending at the situation at hand).
– When there are a lot of associations, then it would be possible to introduce association classes (between the classes representing given sets or free-types) to the UML model. This increases the number of classes, but in several situations this eases the mapping process of operations. Schema operations quite often only modify one state identifier in the state space, and here an association class could easily be used to hold such (getter and setter) methods.

Complex specifications still lead to complex diagrams, but as the classes are constructed in such a way that their internal connectivity is at a maximum, the resulting diagrams are, at least from a measurement perspective, not the worst ones.

One final limitation of the approach remains: when the values for coupling are all the same or when there are a lot of empty class fragments, then the assignment (due to rule 16) is at random first (in our case this happened with the assignment of *DeassignResource* to the *Unused* class). So, it is up to the user to reformat the resulting UML diagram - a drawback developers have to deal with anyway in situations when there are several class-candidates.

5 Conclusion and Outlook

This paper presents a set of rules for transforming formal Z specifications to UML in order to open the documents to a wider range of stakeholders. Existing approaches produce very useful UML diagrams, but the assignment of operations to classes still follows a simple heuristic, namely that of looking how often the operations refer to specific state-identifiers. As an extension to this strategy this contribution recommends to make use of slice-based measures. It introduces a coupling-based measurement method for the related schema operations and suggests to map the operations to classes in such a way that the average values for Inter-Schema Coupling stay at a, on the class level, high value.

The approach enables a wider range of analysis techniques and extensions that will be looked at in the future (and that will eventually be integrated into the *ViZ* environment[6] in one of the next releases). At first, the technique can be combined with existing techniques (like that of Idani in [15]) and thus help in producing additional variations of the resulting UML diagrams. Secondly, the generated UML diagrams can be assessed, either by looking at code smells (like large or lazy classes), by looking at exceptionally high values of coupling between some of the classes, or by combining the approach with a high-level treatment of object-oriented analysis and design. The results can then be used to reflect on the original specification itself.

There are still limitations that should not be concealed. As with other approaches the issue of inherent complexity is hard to solve automatically. Some improvements are possible, e.g. by different views onto parts of the specification or by making use of association classes whenever possible. The transformation rules do not necessarily lead to UML representations as generated by experienced developers by hand, but the approach provides a good picture of *what* is *in* the specifications and it puts the rules for the generation on a measurement-based solid ground. With that, it enables external validation steps and it supports the comprehension process of the involved stakeholders.

References

1. Bollin, A.: Specification comprehension – reducing the complexity of specifications. Ph.D. thesis, University of Klagenfurt (2004)
2. Bollin, A.: Crossing the Borderline - from Formal to Semi-Formal Specifications. In: Software Engineering Techniques: Design for Quality, pp. 73–84. Springer (2006)
3. Bollin, A.: Concept Location in Formal Specifications. Journal of Software Maintenance and Evolution – Research and Practice **20**(2), 77–105 (2008)

[6] A description of the environment and the software can be found at http://viz.uni-klu.ac.at. Page last visited: September 2011.

4. Bollin, A.: Slice-based Formal Specification Measures - Mapping Coupling and Cohesion Measures to Formal Z. In: C. Muñoz (ed.) Proceedings of the Second NASA Formal Methods Sysmposium. NASA Center for AeroSpace Information (CASI) (2010)
5. Bowen, J.P., Hinchey, M.G.: Seven more myths of formal methods. IEEE Software **12**(4), 34–41 (1995)
6. Chang, J., Richardson, D.J.: Static and Dynamic Specification Slicing. Tech. rep., Department of Information and Computer Science, University of California (1994)
7. Dick, J.: Formalising the Informal: Linking Formal Methods to Informal Requirements. Inited Talk at FMICS'2004 as a co-located workshop of ASE 2004, Johannes Kepler Universitt Linz, Austria (2004)
8. Dick, J., Loubersac, J.: A visual approach to VDM: Entity-structure diagrams. Technical Report DE/DRPA/91001, Bull, 68, Route de Versailles, 78430 Louveciennes (France) (1991)
9. Dupuy, S., Ledru, Y., Chabre-Peccoud, M.: An overview of RoZ: A tool for integrating UML and Z specifications. In: Proceedings of CAiSE'00, pp. 417–430 (2000)
10. Fekih, H., Jemni, L., Merz, S.: Transformation des spècifications B en des diagrammes UML. In: Approches Formelles dans l'Assistance au Dveloppement de Logiciels, AFADL'04, pp. 131–148 (2004)
11. Hall, A.: Seven Myths of Formal Methods. IEEE Software **7**(5), 11–19 (1990)
12. Harman, M., Okulawon, M., Sivagurunathan, B., Danicic, S.: Slice-based measurement of coupling. In: Proceedings of the IEEE/ACM ICSE workshop on Process Modelling and Empirical Studies of Software Evolution. Boston, Massachusetts, pp. 28–32. IEEE Computer Society, Los Alamitos, CA, USA (1997)
13. He, X.: PZ Nets - a formal method integrating Petri Nets with Z. Information and Software Technology **43**(1), 1–18 (2001)
14. Hinchey, M., Jackson, M., Cousot, P., Cook, B., Bowen, J.P., Margaria, T.: Software engineering and formal methods. Communications of the ACM **51**(9), 54–59 (2008)
15. Idani, A.: UML Models Engineering from Static and Dynamic Aspects of Formal Specifications. In: T.A. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, P. Soffer, R. Ukor (eds.) Enterprise, Business-Process and Information Systems Modeling, 10th International Workshop, BPMDS 2009, and 14th International Conference, EMMSAD 2009, held at CAiSE 2009, Amsterdam, The Netherlands, June 8-9, 2009. Proceedings, *Lecture Notes in Business Information Processing*, vol. 29, pp. 237–250 (2009)
16. Idani, A., Ledru, Y.: Object oriented concepts identification from formal B specifications. In: Formal Methods in Industrial Critical Applications, FMICS'04 (2004)
17. Idani, A., Ledru, Y., Bert, D.: Derivation of UML class diagrams as static views of formal B developments. In: 7th International Conference on Formal Engineering Methods, ICFEM 2005, pp. 37–51 (2005)
18. Kent, S.: Constraint diagrams: Visualising invariants in object-oriented models. In: In Proceedings of OOPSLA'97. ACM Press (1997)
19. Kim, S.K., Carrington, D.: Visualization of formal specifications. In: In Proceedings Sixth Asia Pacific Software Engineering Conference (ASPEC'99), pp. 102–109. IEEE Computer. Society Press, Los Alamitos, CA, USA (1999)
20. Kim, S.K., Carrington, D.: A formal mapping between UML models and Object-Z specifications. Lecture Notes in Computer Science **1878**, 2–21 (2000)
21. Lessacher, J.: Visualisierung von Spezifikationen – Transformation von formalen Z-Spezifikationen in UML-Diagramme (in German). Master's thesis, Alpen-Adria Universität Klagenfurt (2007)
22. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. John Wiley and Sons Ltd. (1990)
23. Meyers, T.M., Binkley, D.: An Empirical Study of Slice-Based Cohesion and Coupling Metrics. ACM Transactions on Software Engineering and Methodology **17**(1), 2:1–2:27 (2007)
24. Mittermeir, R.T., Bollin, A.: Demand-driven specification partitioning. In: Proceedings of the 5th Joint Modular Languages Conference, JMLC'03 (2003)
25. Oda, T., Araki, K.: Specification slicing in a formal methods software development. In: 17th Annual International Computer Software and Applications Conference, IEEE Computer Socienty Press, pp. 313–319 (1993)
26. Ott, L.M., Thuss, J.J.: Slice based metrics for estimating cohesion. In: In Proceedings of the IEEE-CS International Metrics Symposium, pp. 71–81. IEEE Computer Society, Los Alamitos, CA, USA (1993)
27. Roe, D., Broda, K., Russo, A.: Mapping UML models incorporating OCL constraints into Object-Z. Technical Report ISBN/ISSN: 1469-4174, Imperial College of Science, Technology and Medicine, Department of Computing (2003)
28. Ross, P.E.: The exterminators. IEEE Spectrum pp. 36–41 (2005)
29. Spivey, J.: The Z Notation. C.A.R. Hoare Series. Prentice Hall (1989)
30. Weiser, M.: Program slices: formal, psychological, and practical investigations of an automatic program abstraction method. Ph.D. thesis, University of Michigan (1979)

Appendix - Access Control Specification

The Access Control Specification is a sample specification (after Kim in [19]) for demonstrating the transformation process. Please note that the specification is not meant to be complete (e.g. an operation for creating resources would be needed, also the initialization could be handled in a different manner).

$$[\mathit{USERS}, \mathit{RESOURCES}, \mathit{ADDRESSES}]$$

AccessControl

$$\begin{array}{l}
\mathit{Permitted} : \mathit{USERS} \leftrightarrow \mathit{RESOURCES} \\
\mathit{Assigned} : \mathit{USERS} \nrightarrow \mathit{RESOURCES} \\
\mathit{IpAddress} : \mathit{ADDRESSES} \nrightarrow \mathit{RESOURCES} \\
\mathit{Unused} : \mathbb{P}\,\mathit{RESOURCES} \\
\hline
\mathit{Assigned} \subseteq \mathit{Permitted} \\
\mathit{Unused} \cap (\mathrm{ran}\,\mathit{Assigned}) = \varnothing
\end{array}$$

InitAccessControl

$$\begin{array}{l}
\mathit{AccessControl} \\
\hline
\mathit{Permitted} = \varnothing \\
\mathit{Assigned} = \varnothing \\
\mathit{IpAddress} = \varnothing \\
\mathit{Unused} = \varnothing
\end{array}$$

$$\begin{array}{l}
\textit{AddPermission} \\
\hline
\Delta \textit{AccessControl} \\
\textit{user}? : \textit{USERS} \\
\textit{resource}? : \textit{RESOURCES} \\
\hline
(\textit{user}? \mapsto \textit{resource}?) \notin \textit{Permitted} \\
\textit{Permitted}' = \textit{Permitted} \cup \{\textit{user}? \mapsto \textit{resource}?\} \\
\textit{Assigned}' = \textit{Assigned} \\
\textit{IpAddress}' = \textit{IpAddress} \\
\textit{Unused}' = \textit{Unused} \\
\hline
\end{array}$$

$$\begin{array}{l}
\textit{ListUsers} \\
\hline
\Xi \textit{AccessControl} \\
\textit{resource}? : \textit{RESOURCES} \\
\textit{st}! : \mathbb{P}\, \textit{USERS} \\
\hline
\textit{st}! = \operatorname{dom}(\textit{Permitted} \rhd \{\textit{resource}?\}) \\
\hline
\end{array}$$

$$\begin{array}{l}
\textit{AssignResource} \\
\hline
\Delta \textit{AccessControl} \\
\textit{user}? : \textit{USERS} \\
\textit{resource}? : \textit{RESOURCES} \\
\hline
(\textit{user}? \mapsto \textit{resource}?) \in \textit{Permitted} \\
\textit{user}? \notin \operatorname{dom} \textit{Assigned} \\
\textit{Assigned}' = \textit{Assigned} \cup \{\textit{user}? \mapsto \textit{resource}?\} \\
\textit{Permitted}' = \textit{Permitted} \\
\textit{IpAddress}' = \textit{IpAddress} \\
\textit{Unused}' = \textit{Unused} \\
\hline
\end{array}$$

$$\begin{array}{l}
\textit{ChangeIP} \\
\hline
\Delta \textit{AccessControl} \\
\textit{ip}? : \textit{ADDRESSES} \\
\textit{resource}? : \textit{RESOURCES} \\
\hline
\textit{ip}? \notin \operatorname{dom} \textit{IpAddress} \\
\textit{resource}? \in \operatorname{ran} \textit{IpAddress} \\
\textit{resource}? \in \operatorname{ran} \textit{Permitted} \\
\textit{IpAddress}' = \\
\qquad\qquad \{i : \textit{ADDRESSES};\ r : \textit{RESOURCES} \mid \\
\qquad\qquad (i, r) \in \textit{IpAddress} \wedge r \neq \textit{resource}?\} \\
\qquad\qquad \cup \{\textit{ip}? \mapsto \textit{resource}?\} \\
\textit{Permitted}' = \textit{Permitted} \\
\textit{Assigned}' = \textit{Assigned} \\
\textit{Unused}' = \textit{Unused} \\
\hline
\end{array}$$

$$\begin{array}{l}
\textit{DeassignResource} \\
\hline
\Delta \textit{AccessControl} \\
\textit{user}? : \textit{USERS} \\
\textit{resource}? : \textit{RESOURCES} \\
\hline
\textit{resource}? \notin \textit{Unused} \\
(\textit{user}?, \textit{resource}?) \in \textit{Assigned} \\
\textit{Assigned}' = \textit{Assigned} \setminus \{\textit{user}? \mapsto \textit{resource}?\} \\
\textit{Unused}' = \textit{Unused} \cup \{\textit{resource}?\} \\
\textit{Permitted}' = \textit{Permitted} \\
\textit{IpAddress}' = \textit{IpAddress} \\
\hline
\end{array}$$

$$\begin{array}{l}
\textit{Disconnect} \\
\hline
\Delta \textit{AccessControl} \\
\textit{addr}? : \textit{ADDRESSES} \\
\textit{resource}? : \textit{RESOURCES} \\
\hline
\textit{resource}? \in \textit{Unused} \\
(\textit{addr}?, \textit{resource}?) \in \textit{IpAddress} \\
\textit{IpAddress}' = \textit{IpAddress} \setminus \{\textit{addr}? \mapsto \textit{resource}?\} \\
\textit{Permitted}' = \textit{Permitted} \\
\textit{Assigned}' = \textit{Assigned} \\
\textit{Unused}' = \textit{Unused} \\
\hline
\end{array}$$