Using Bonds for Method Dispatch in Role-Oriented Software Models

Concept Lattices and their Applications
Sevilla, Spain

Henri Mühle

Department of Mathematics
University of Vienna

21.10.2010

Table of Contents

1. Introduction

Late Binding and Method Dispatch

- *late binding* is a basic feature of object-oriented software modeling
- due to type polymorphism the runtime type of an object is not determined until runtime
- the runtime type may even change according to type casts
- however, exact knowledge of the runtime type is crucial, in order to invoke the correct piece of code when using an object
- the process of redirecting method calls along the inheritance hierarchy according to the runtime type is referred to as *method dispatch*

Role-Oriented Software Modeling

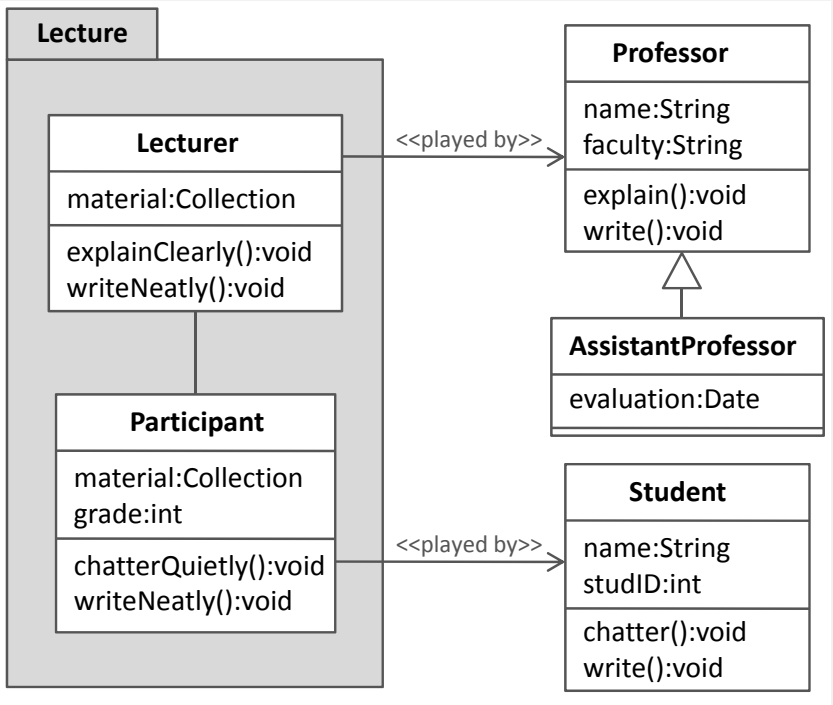
- role-oriented software modeling is an extension to object-oriented software modeling
- it introduces a second, orthogonal hierarchy of *role types*
- role types encapsulate common behavior and provide it to respective *base types*
- a role-play relation between base and role types determines which role behavior can be accessed by which base types
- thus, role modeling introduces new dimensions of method dispatch
  - along the hierarchy of role types
  - along the role-play relation

Example

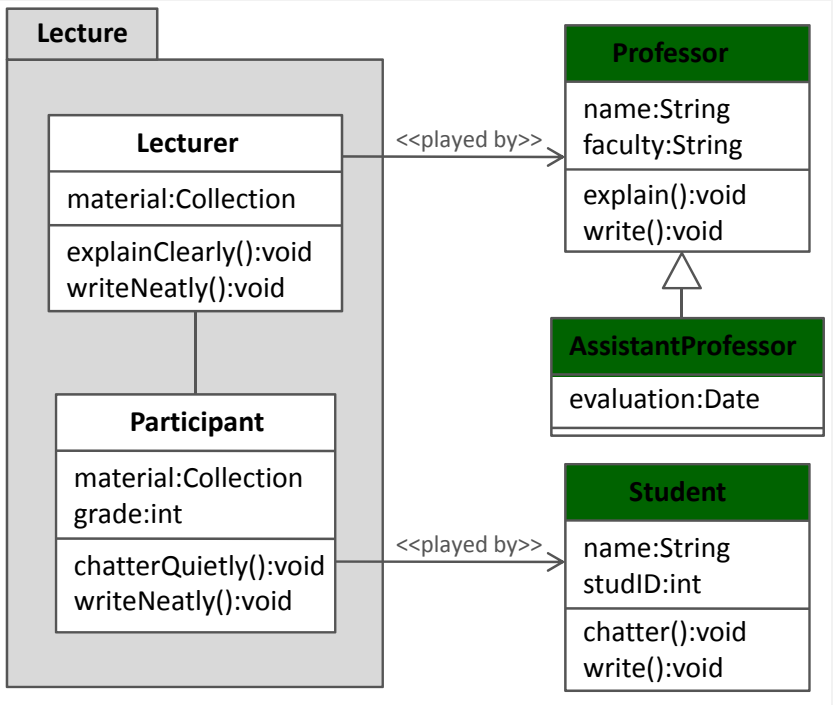
- a `Professor` class surely comes equipped with methods `write()` and `explain()`
- thus, an object of runtime type `Professor` calls these own methods
- in order to alter the fixed behavior encoded in these methods, e. g. a role type `Lecturer` can be defined, which provides methods `writeNeatly()` and `explainClearly()`
- in order to perform the altering of the behavior a connection between either methods has to be established at modeling time
- at runtime, it has to be guaranteed that calls towards the own methods of the `Professor` class are redirected towards the corresponding methods of the `Lecturer` type

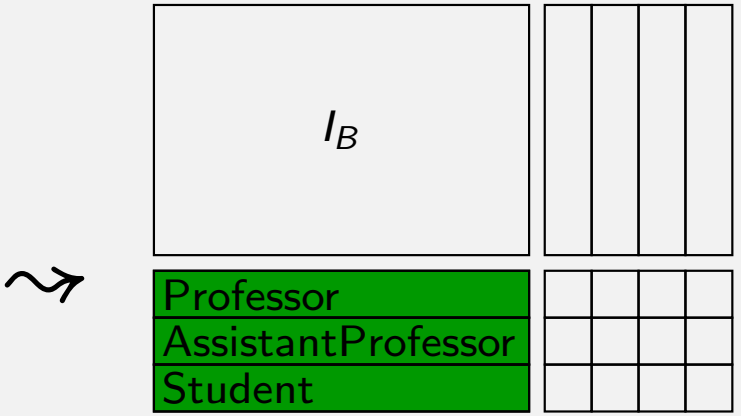
2. Construction

Base/Role Type Context

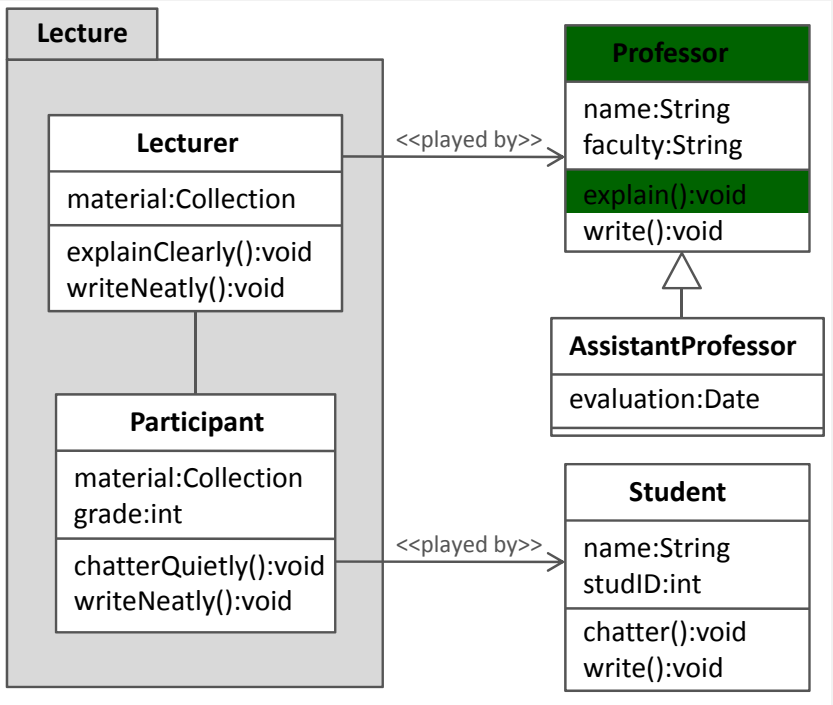


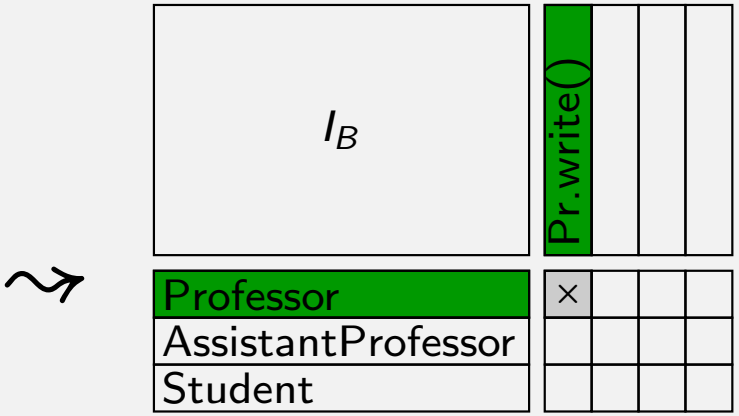
Base/Role Type Context



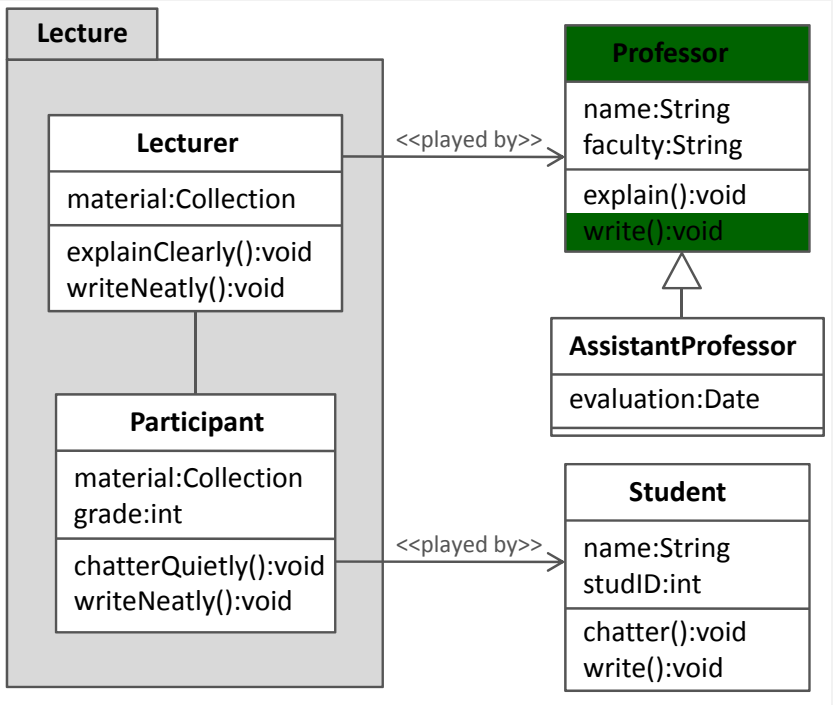


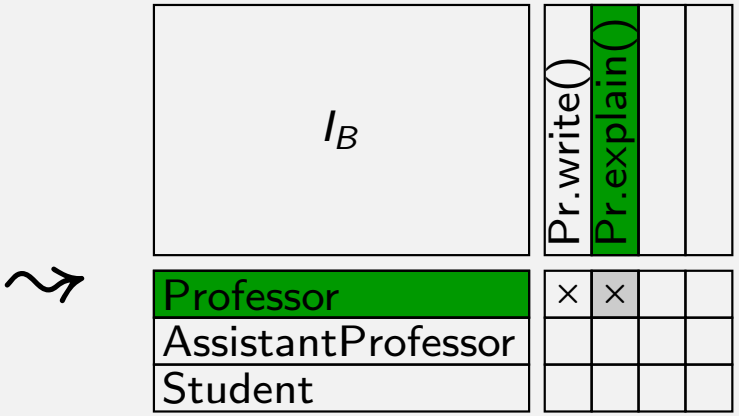
Base/Role Type Context



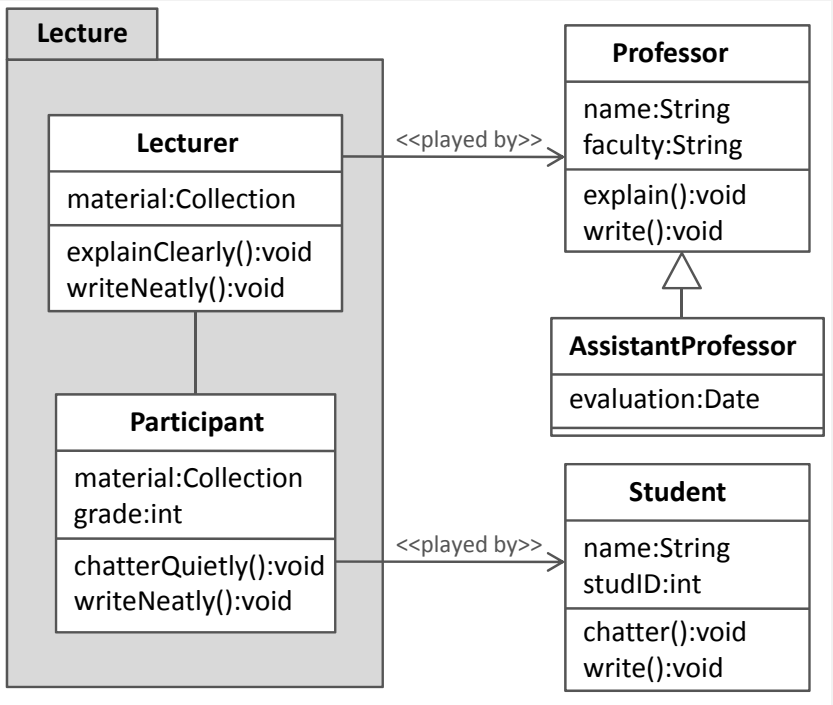


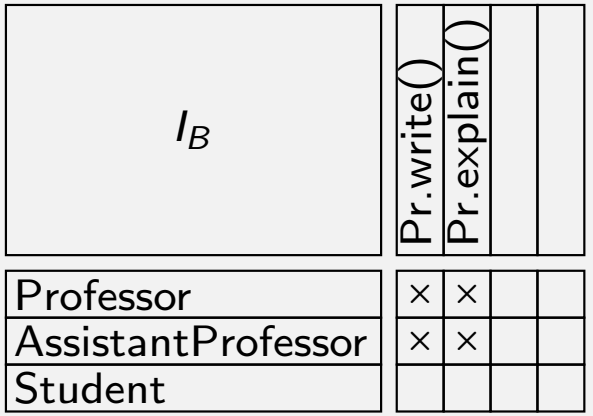
Base/Role Type Context



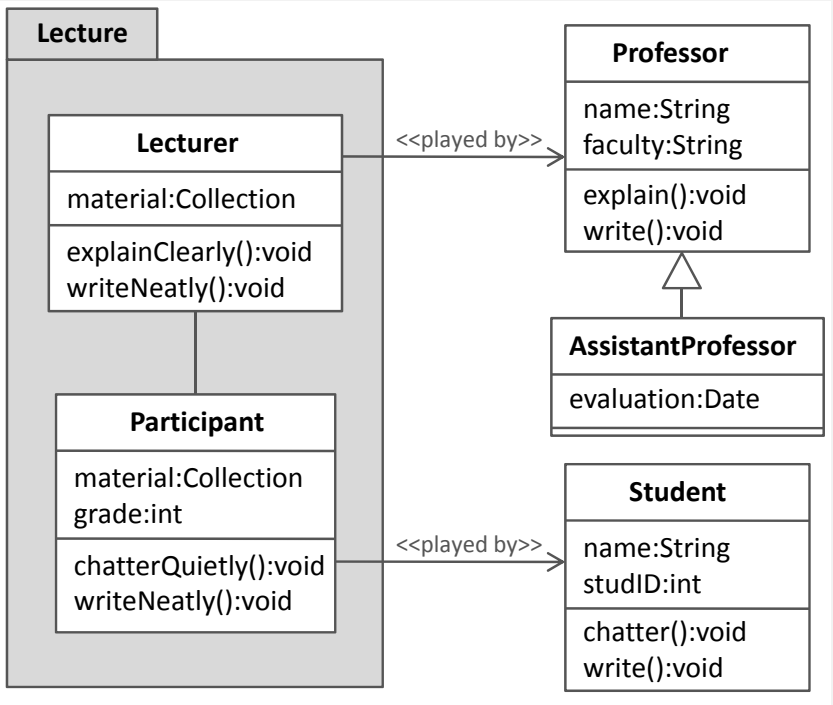


Base/Role Type Context





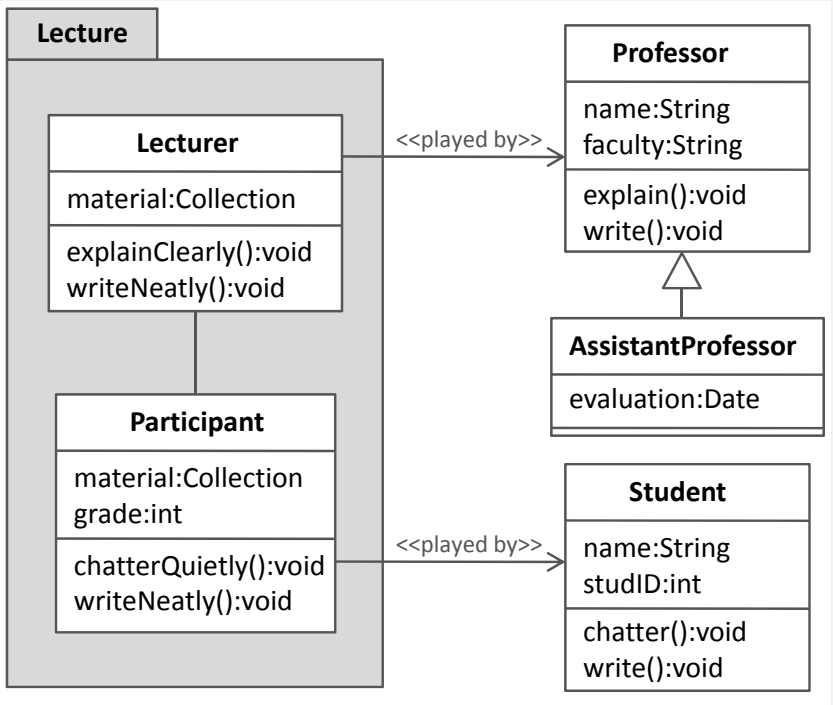
Base/Role Type Context



$\rightsquigarrow$

I_B	Pr.write()	Pr.explain()	St.write()	St.chatter()
Professor	×	×		
AssistantProfessor	×	×		
Student			×	×

Base/Role Type Context



$\rightsquigarrow$

I_B	Pr.write()	Pr.explain()	St.write()	St.chatter()
Professor	×	×		
AssistantProfessor	×	×		
Student			×	×

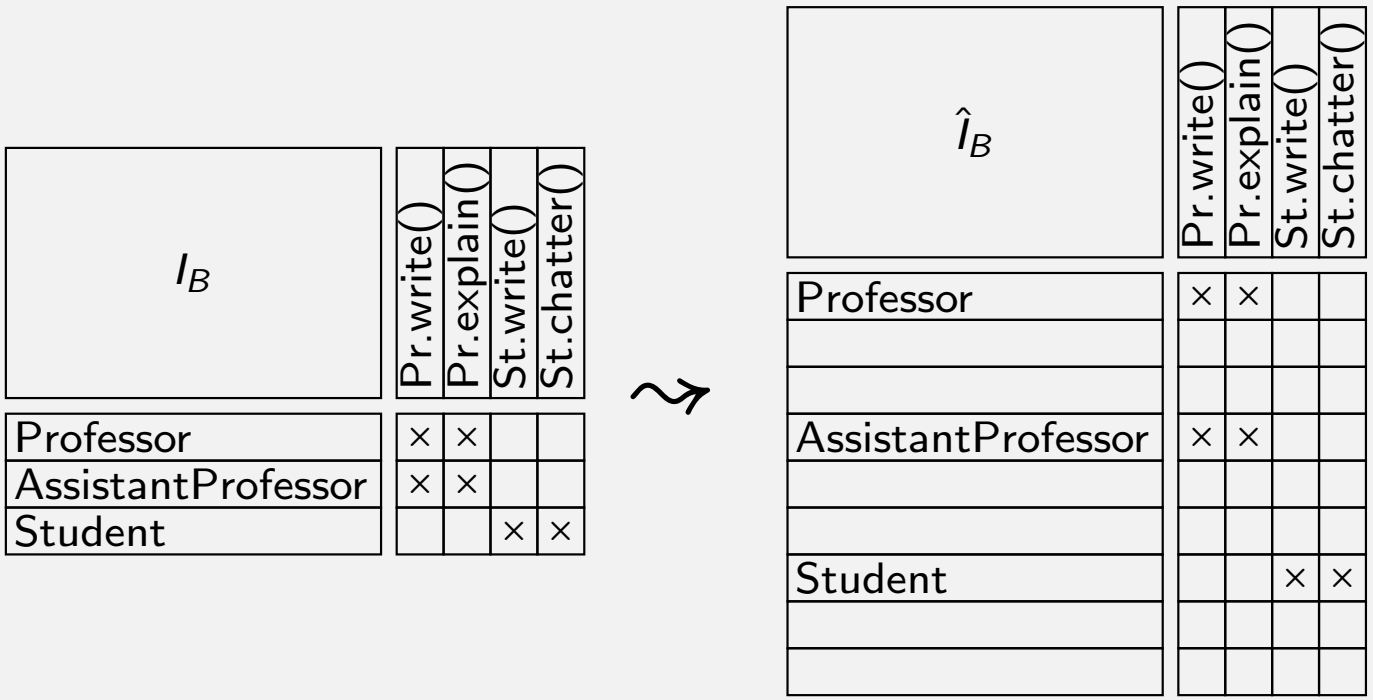
$\rightsquigarrow$

I_R	Le.writeNeatly()	Le.explainClearly()	Pa.writeNeatly()	Pa.chatterQuietly()
Lecturer	×	×		
Participant			×	×

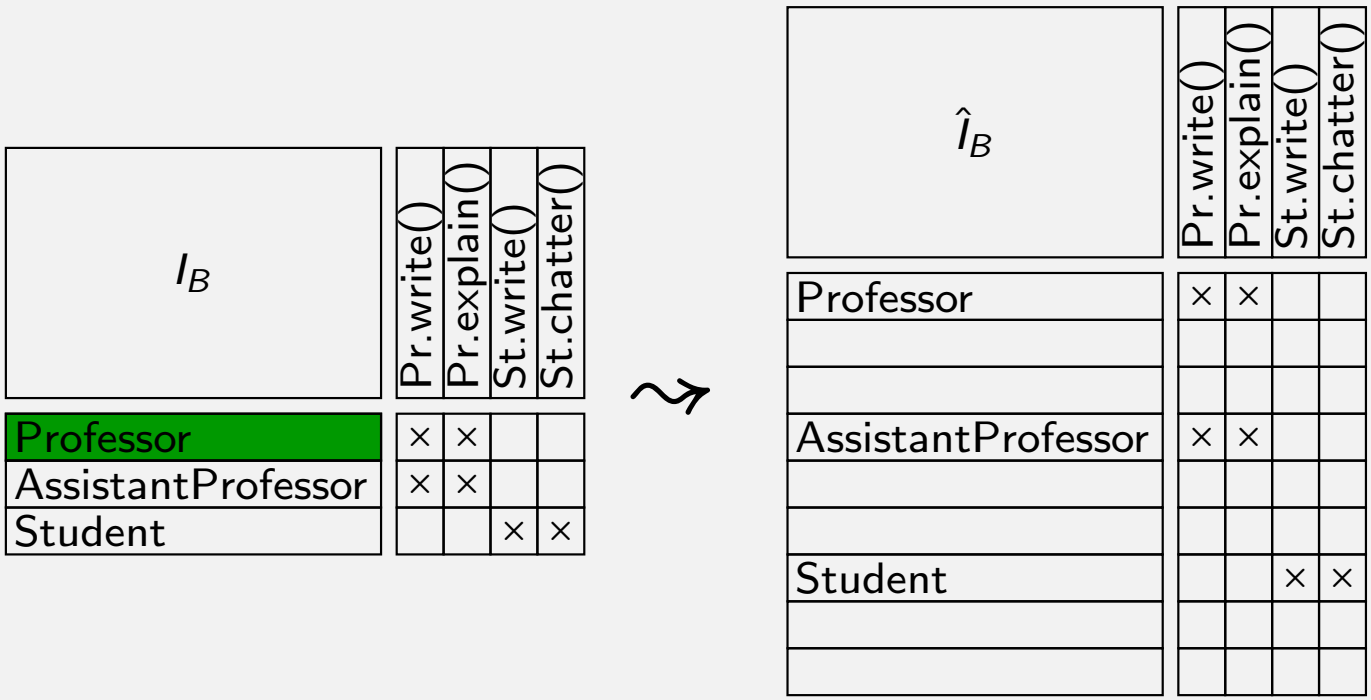
Extended Base/Role Type Context

I_B	Pr.write()	Pr.explain()	St.write()	St.chatter()
Professor	×	×		
AssistantProfessor	×	×		
Student			×	×

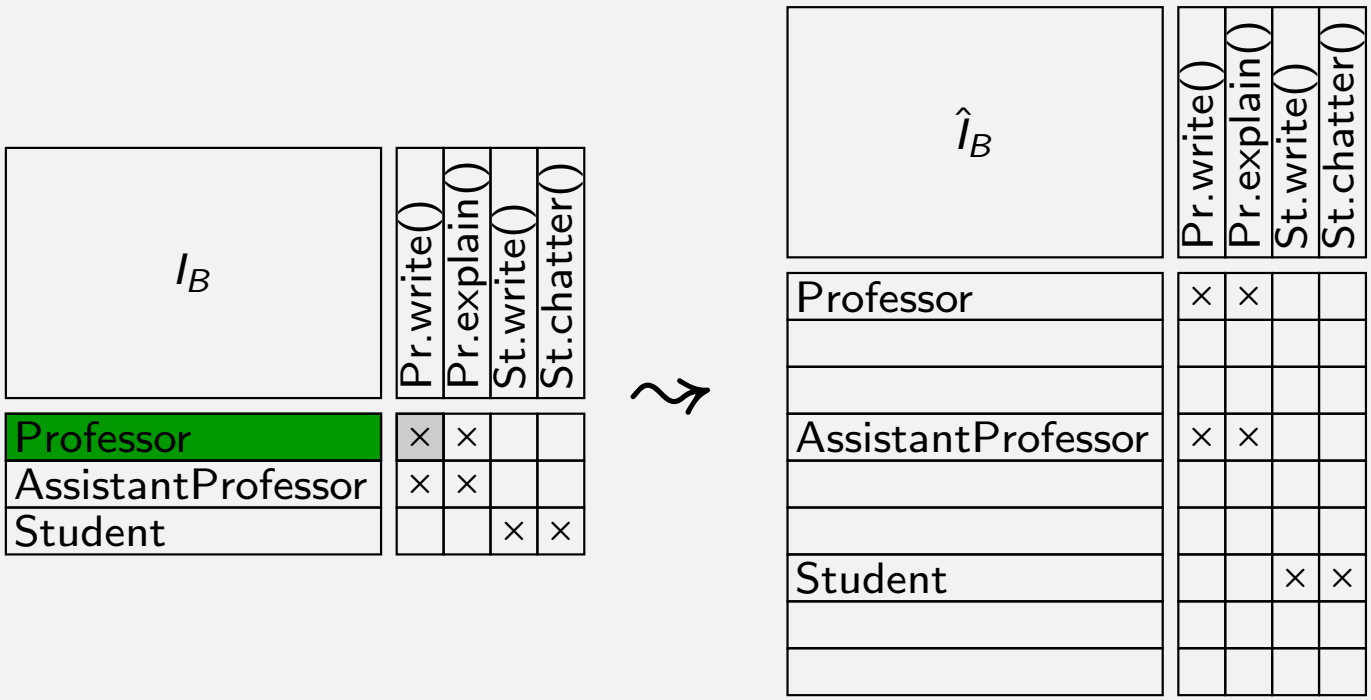
Extended Base/Role Type Context



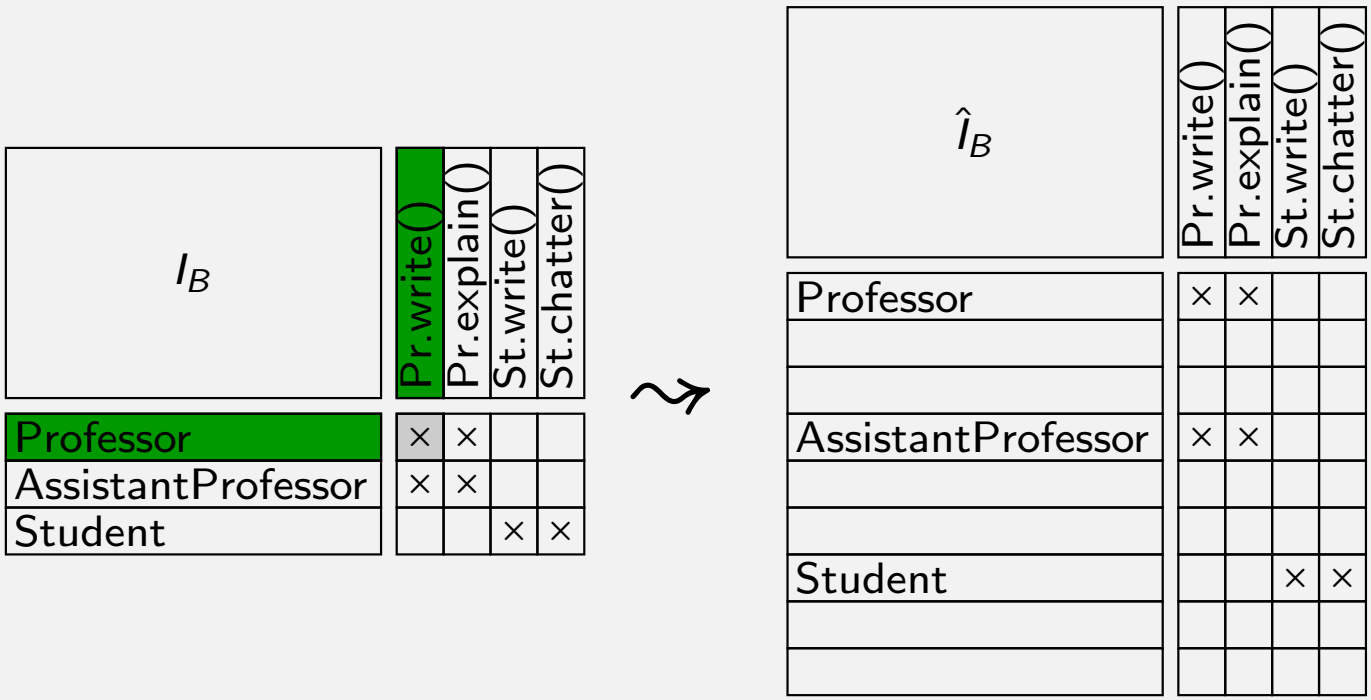
Extended Base/Role Type Context



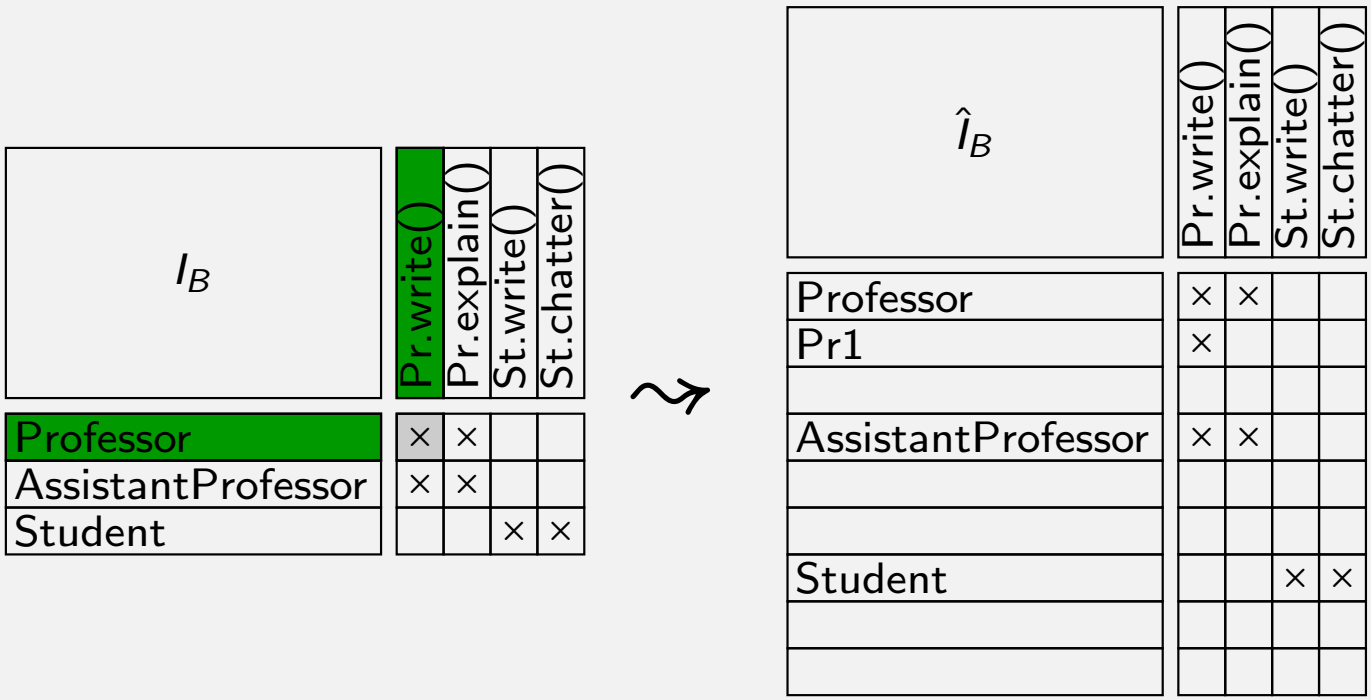
Extended Base/Role Type Context



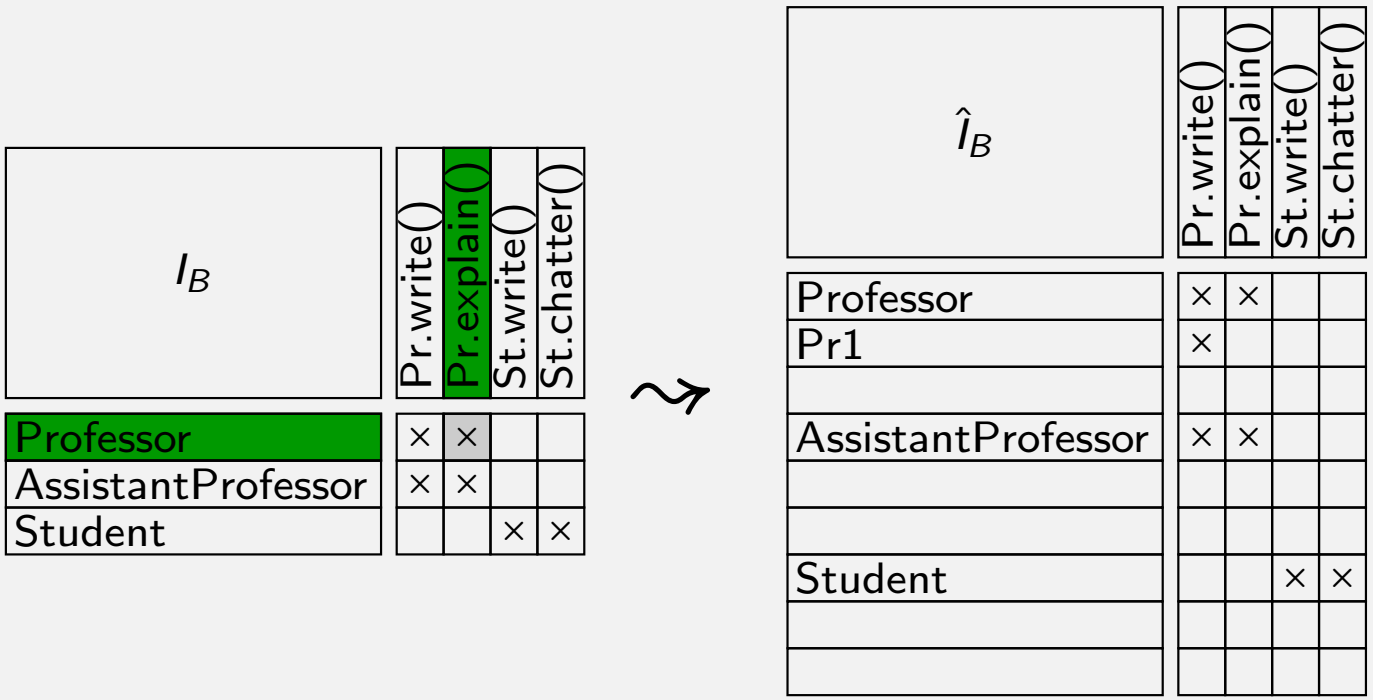
Extended Base/Role Type Context



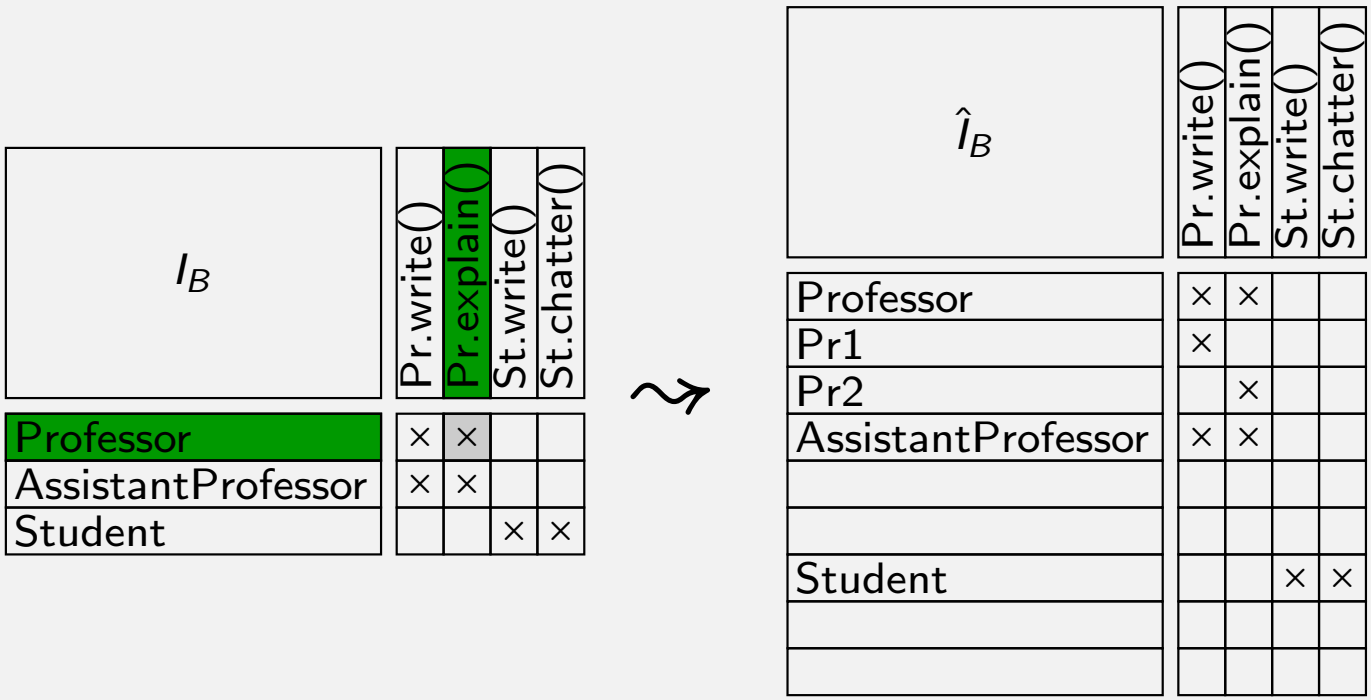
Extended Base/Role Type Context



Extended Base/Role Type Context



Extended Base/Role Type Context



Extended Base/Role Type Context

I_B	Pr.write()	Pr.explain()	St.write()	St.chatter()
Professor	×	×		
AssistantProfessor	×	×		
Student			×	×

$\rightsquigarrow$

$\hat{I}_B$	Pr.write()	Pr.explain()	St.write()	St.chatter()
Professor	×	×		
Pr1	×			
Pr2		×		
AssistantProfessor	×	×		
As1	×			
As2		×		
Student			×	×

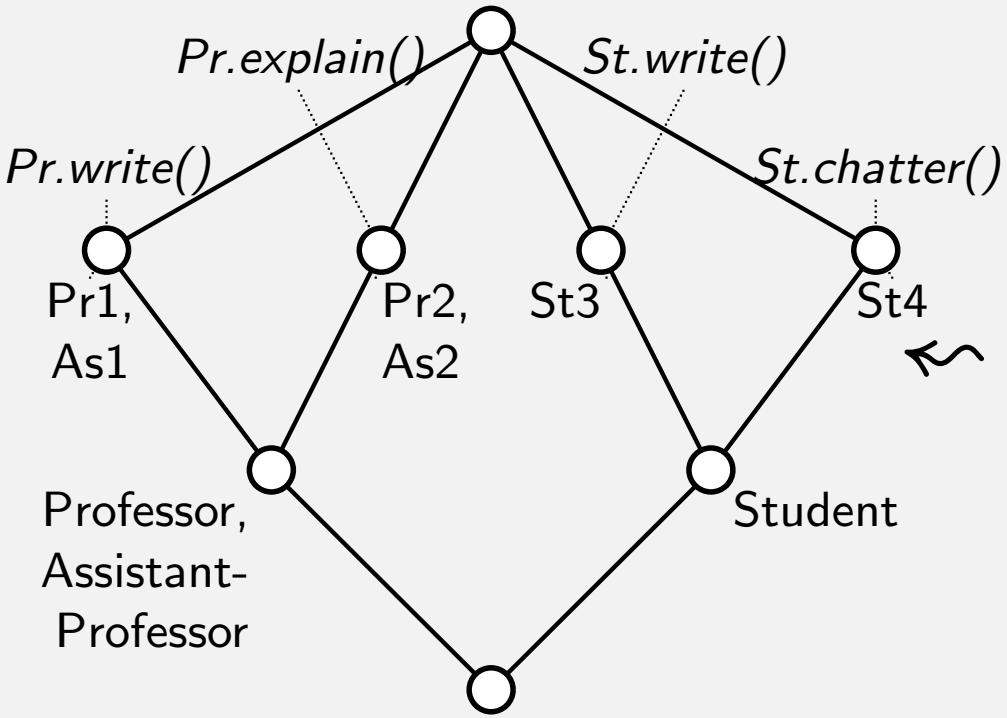
Extended Base/Role Type Context

I_B	Pr.write()	Pr.explain()	St.write()	St.chatter()
Professor	×	×		
AssistantProfessor	×	×		
Student			×	×

$\rightsquigarrow$

$\hat{I}_B$	Pr.write()	Pr.explain()	St.write()	St.chatter()
Professor	×	×		
Pr1	×			
Pr2		×		
AssistantProfessor	×	×		
As1	×			
As2		×		
Student			×	×
St3			×	
St4				×

Extended Base/Role Type Context



$\hat{I}_B$	Pr.write()	Pr.explain()	St.write()	St.chatter()
Professor	×	×		
Pr1	×			
Pr2		×		
AssistantProfessor	×	×		
As1	×			
As2		×		
Student			×	×
St3			×	
St4				×

Dynamic Construction

- since method dispatch needs to be performed at runtime, it is necessary to model the runtime state
- this is, however, only necessary for base types, since role types do not provide their own instances
- for each point of runtime $t \in T$ we define a method call context that describes the actual behavior of the (base) type instances, denoted by $\mathcal{I}^t$
- we assume a sequential execution of the model, i. e. each type instance can be target of at most one method call at a time

Method Call Context

```
public static void main(String[] args){
    Professor ganter;
    Student muehle;
    Student wende;
    ...
    ganter.explain();
    ...
    muehle.chatter();
    ...
    wende.write();
    ...
}
```

$\hat{I}_B$	Pr.write()	Pr.explain()	St.write()	St.chatter()
Professor	×	×		
Pr1	×			
Pr2		×		
AssistantProfessor	×	×		
As1	×			
As2		×		
Student			×	×
St3			×	
St4				×

Method Call Context

```
public static void main(String[] args){
    Professor ganter;
    Student muehle;
    Student wende;
    ...
    ganter.explain();
    ...
    muehle.chatter();
    ...
    wende.write();
    ...
}
```

$\hat{I}_B^1$	Pr.write()	Pr.explain()	St.write()	St.chatter()
Professor	×	×		
Pr1	×			
Pr2		×		
ganter				
AssistantProfessor	×	×		
As1	×			
As2		×		
Student			×	×
St3			×	
St4				×

Method Call Context

```
public static void main(String[] args){
    Professor ganter;
    Student muehle;
    Student wende;
    ...
    ganter.explain();
    ...
    muehle.chatter();
    ...
    wende.write();
    ...
}
```

$\hat{I}_B^2$	Pr.write()	Pr.explain()	St.write()	St.chatter()
Professor	×	×		
Pr1	×			
Pr2		×		
ganter				
AssistantProfessor	×	×		
As1	×			
As2		×		
Student			×	×
St3			×	
St4				×
muehle				

Method Call Context

```
public static void main(String[] args){
    Professor ganter;
    Student muehle;
    Student wende;
    ...
    ganter.explain();
    ...
    muehle.chatter();
    ...
    wende.write();
    ...
}
```

$\hat{I}_B^3$	Pr.write()	Pr.explain()	St.write()	St.chatter()
Professor	×	×		
Pr1	×			
Pr2		×		
ganter				
AssistantProfessor	×	×		
As1	×			
As2		×		
Student			×	×
St3			×	
St4				×
muehle				
wende				

Method Call Context

```
public static void main(String[] args){
    Professor ganter;
    Student muehle;
    Student wende;
    ...
    ganter.explain();
    ...
    muehle.chatter();
    ...
    wende.write();
    ...
}
```

$\hat{I}_B^i$	Pr.write()	Pr.explain()	St.write()	St.chatter()
Professor	×	×		
Pr1	×			
Pr2		×		
ganter		×		
AssistantProfessor	×	×		
As1	×			
As2		×		
Student			×	×
St3			×	
St4				×
muehle				
wende				

Method Call Context

```
public static void main(String[] args){
    Professor ganter;
    Student muehle;
    Student wende;
    ...
    ganter.explain();
    ...
    muehle.chatter();
    ...
    wende.write();
    ...
}
```

$\hat{I}_B^j$	Pr.write()	Pr.explain()	St.write()	St.chatter()
Professor	×	×		
Pr1	×			
Pr2		×		
ganter				
AssistantProfessor	×	×		
As1	×			
As2		×		
Student			×	×
St3			×	
St4				×
muehle				×
wende				

Method Call Context

```
public static void main(String[] args){
    Professor ganter;
    Student muehle;
    Student wende;
    ...
    ganter.explain();
    ...
    muehle.chatter();
    ...
    wende.write();
    ...
}
```

$\hat{I}_B^k$	Pr.write()	Pr.explain()	St.write()	St.chatter()
Professor	×	×		
Pr1	×			
Pr2		×		
ganter				
AssistantProfessor	×	×		
As1	×			
As2		×		
Student			×	×
St3			×	
St4				×
muehle				
wende			×	

Combining both Hierarchies

- to represent the method dispatch, it is necessary to establish a mapping between role and base type methods
- this mapping needs to be connected to the runtime instances and their runtime types
- the method binding is (freely) determined at modeling time
- the presented contexts allow for modeling runtime instances and runtime types
- but how to combine the hierarchies?

Combining both Hierarchies

- to represent the method dispatch, it is necessary to establish a mapping between role and base type methods
- this mapping needs to be connected to the runtime instances and their runtime types
- the method binding is (freely) determined at modeling time
- the presented contexts allow for modeling runtime instances and runtime types
- but how to combine the hierarchies?

Combining both Hierarchies

- to represent the method dispatch, it is necessary to establish a mapping between role and base type methods
- this mapping needs to be connected to the runtime instances and their runtime types
- the method binding is (freely) determined at modeling time
- the presented contexts allow for modeling runtime instances and runtime types
- but how to combine the hierarchies?
  $\rightsquigarrow$ bonds seem to be an intuitive tool for this purpose

Combining both Hierarchies

Definition

Given two contexts $\mathbb{K}_s := (G_s, M_s, I_s), \mathbb{K}_t := (G_t, M_t, I_t)$, a relation $J_{st} \subseteq G_s \times M_t$ is called **bond**, iff $g^{J_{st}}$ is an intent of $\mathbb{K}_t$ for each object $g \in G_s$ and $m^{J_{st}}$ is an extent of $\mathbb{K}_s$ for each attribute $m \in M_t$.

Combining both Hierarchies

Definition

Given two contexts $\mathbb{K}_s := (G_s, M_s, I_s), \mathbb{K}_t := (G_t, M_t, I_t)$, a relation $J_{st} \subseteq G_s \times M_t$ is called **bond**, iff $g^{J_{st}}$ is an intent of $\mathbb{K}_t$ for each object $g \in G_s$ and $m^{J_{st}}$ is an extent of $\mathbb{K}_s$ for each attribute $m \in M_t$. Each bond J_{st} induces two morphisms

$$\varphi_{st} : \underline{\mathfrak{B}}(G_s, M_s, I_s) \to \underline{\mathfrak{B}}(G_t, M_t, I_t),$$
$$\psi_{st} : \underline{\mathfrak{B}}(G_t, M_t, I_t) \to \underline{\mathfrak{B}}(G_s, M_s, I_s)$$

by

$$\varphi_{st}(A, A^{I_s}) := (A^{J_{st} I_t}, A^{J_{st}}), \quad \psi_{st}(B^{I_t}, B) := (B^{J_{st}}, B^{J_{st} I_s})$$

(Proper) Binding Context

J	Pr.write()	Pr.explain()	St.write()	St.chatter()	Le.writeNeatly()	Le.explainClearly()	Pa.writeNeatly()	Pa.chatterQuietly()
Professor	×	×						
Pr1	×							
Pr2		×						
AssistantProfessor	×	×						
As1	×							
As2		×						
Student			×	×				
St3			×					
St4				×				
Lecturer					×	×		
Le1					×			
Le2						×		
Participant							×	×
Pa1							×	
Pa2								×

Restrictions:
1) connect base objects and role attributes only if the according base and role types are connected under the played-by relation
2) connect virtual objects only to such role attributes that are connected with the according base object
3) connect each virtual object with exactly one role attribute
4) connect each role attribute to some base object

(Proper) Binding Context

J	Pr.write()	Pr.explain()	St.write()	St.chatter()	Le.writeNeatly()	Le.explainClearly()	Pa.writeNeatly()	Pa.chatterQuietly()
Professor	×	×						
Pr1	×							
Pr2		×						
AssistantProfessor	×	×						
As1	×							
As2		×						
Student			×	×				
St3			×					
St4				×				
Lecturer					×	×		
Le1					×			
Le2						×		
Participant							×	×
Pa1							×	
Pa2								×

Restrictions:
1) connect base objects and role attributes only if the according base and role types are connected under the played-by relation
2) connect virtual objects only to such role attributes that are connected with the according base object
3) connect each virtual object with exactly one role attribute
4) connect each role attribute to some base object

(Proper) Binding Context

J	Pr.write()	Pr.explain()	St.write()	St.chatter()	Le.writeNeatly()	Le.explainClearly()	Pa.writeNeatly()	Pa.chatterQuietly()
Professor	×	×						
Pr1	×							
Pr2		×						
AssistantProfessor	×	×						
As1	×							
As2		×						
Student			×	×				
St3			×					
St4				×				
Lecturer					×	×		
Le1					×			
Le2						×		
Participant							×	×
Pa1							×	
Pa2								×

Restrictions:
1) connect base objects and role attributes only if the according base and role types are connected under the played-by relation
2) connect virtual objects only to such role attributes that are connected with the according base object
3) connect each virtual object with exactly one role attribute
4) connect each role attribute to some base object

(Proper) Binding Context

J	Pr.write()	Pr.explain()	St.write()	St.chatter()	Le.writeNeatly()	Le.explainClearly()	Pa.writeNeatly()	Pa.chatterQuietly()
Professor	×	×						
Pr1	×							
Pr2		×						
AssistantProfessor	×	×						
As1	×							
As2		×						
Student			×	×				
St3			×					
St4				×				
Lecturer					×	×		
Le1					×			
Le2						×		
Participant							×	×
Pa1							×	
Pa2								×

Restrictions:
1) connect base objects and role attributes only if the according base and role types are connected under the played-by relation
2) connect virtual objects only to such role attributes that are connected with the according base object
3) connect each virtual object with exactly one role attribute
4) connect each role attribute to some base object

(Proper) Binding Context

J	Pr.write()	Pr.explain()	St.write()	St.chatter()	Le.writeNeatly()	Le.explainClearly()	Pa.writeNeatly()	Pa.chatterQuietly()
Professor	×	×			×			
Pr1	×				×			
Pr2		×						
AssistantProfessor	×	×			×			
As1	×				×			
As2		×						
Student			×	×				
St3			×					
St4				×				
Lecturer					×	×		
Le1					×			
Le2						×		
Participant							×	×
Pa1							×	
Pa2								×

Restrictions:
1) connect base objects and role attributes only if the according base and role types are connected under the played-by relation
2) connect virtual objects only to such role attributes that are connected with the according base object
3) connect each virtual object with exactly one role attribute
4) connect each role attribute to some base object

(Proper) Binding Context

J	Pr.write()	Pr.explain()	St.write()	St.chatter()	Le.writeNeatly()	Le.explainClearly()	Pa.writeNeatly()	Pa.chatterQuietly()
Professor	×	×			×			
Pr1	×				×			
Pr2		×						
AssistantProfessor	×	×			×			
As1	×				×			
As2		×						
Student			×	×				
St3			×					
St4				×				
Lecturer					×	×		
Le1					×			
Le2						×		
Participant							×	×
Pa1							×	
Pa2								×

Restrictions:
1) connect base objects and role attributes only if the according base and role types are connected under the played-by relation
2) connect virtual objects only to such role attributes that are connected with the according base object
3) connect each virtual object with exactly one role attribute
4) connect each role attribute to some base object

(Proper) Binding Context

J	Pr.write()	Pr.explain()	St.write()	St.chatter()	Le.writeNeatly()	Le.explainClearly()	Pa.writeNeatly()	Pa.chatterQuietly()
Professor	×	×			×	×		
Pr1	×				×			
Pr2		×				×		
AssistantProfessor	×	×			×	×		
As1	×				×			
As2		×				×		
Student			×	×				
St3			×					
St4				×				
Lecturer					×	×		
Le1					×			
Le2						×		
Participant							×	×
Pa1							×	
Pa2								×

Restrictions:
1) connect base objects and role attributes only if the according base and role types are connected under the played-by relation
2) connect virtual objects only to such role attributes that are connected with the according base object
3) connect each virtual object with exactly one role attribute
4) connect each role attribute to some base object

(Proper) Binding Context

J	Pr.write()	Pr.explain()	St.write()	St.chatter()	Le.writeNeatly()	Le.explainClearly()	Pa.writeNeatly()	Pa.chatterQuietly()
Professor	×	×			×	×		
Pr1	×				×			
Pr2		×				×		
AssistantProfessor	×	×			×	×		
As1	×				×			
As2		×				×		
Student			×	×			×	
St3			×				×	
St4				×				
Lecturer					×	×		
Le1					×			
Le2						×		
Participant							×	×
Pa1							×	
Pa2								×

Restrictions:
1) connect base objects and role attributes only if the according base and role types are connected under the played-by relation
2) connect virtual objects only to such role attributes that are connected with the according base object
3) connect each virtual object with exactly one role attribute
4) connect each role attribute to some base object

(Proper) Binding Context

J	Pr.write()	Pr.explain()	St.write()	St.chatter()	Le.writeNeatly()	Le.explainClearly()	Pa.writeNeatly()	Pa.chatterQuietly()
Professor	×	×			×	×		
Pr1	×				×			
Pr2		×				×		
AssistantProfessor	×	×			×	×		
As1	×				×			
As2		×				×		
Student			×	×			×	×
St3			×				×	
St4				×				×
Lecturer					×	×		
Le1					×			
Le2						×		
Participant							×	×
Pa1							×	
Pa2								×

Restrictions:
1) connect base objects and role attributes only if the according base and role types are connected under the played-by relation
2) connect virtual objects only to such role attributes that are connected with the according base object
3) connect each virtual object with exactly one role attribute
4) connect each role attribute to some base object

(Proper) Binding Context

J	Pr.write()	Pr.explain()	St.write()	St.chatter()	Le.writeNeatly()	Le.explainClearly()	Pa.writeNeatly()	Pa.chatterQuietly()
Professor	×	×			×	×		
Pr1	×				×			
Pr2		×				×		
AssistantProfessor	×	×			×	×		
As1	×				×			
As2		×				×		
Student			×	×			×	×
St3			×				×	
St4				×				×
Lecturer					×	×		
Le1					×			
Le2						×		
Participant							×	×
Pa1							×	
Pa2								×

Restrictions:
1) connect base objects and role attributes only if the according base and role types are connected under the played-by relation
2) connect virtual objects only to such role attributes that are connected with the according base object
3) connect each virtual object with exactly one role attribute
4) connect each role attribute to some base object

The Algorithm

The Algorithm

choose a relevant instance $i \in \mathcal{I}^t$

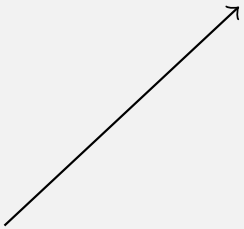
The Algorithm

determine the base method $m \in M_B$, calling i via $m = i^{\hat{I}_B^t}$

choose a relevant instance $i \in \mathcal{I}^t$

The Algorithm

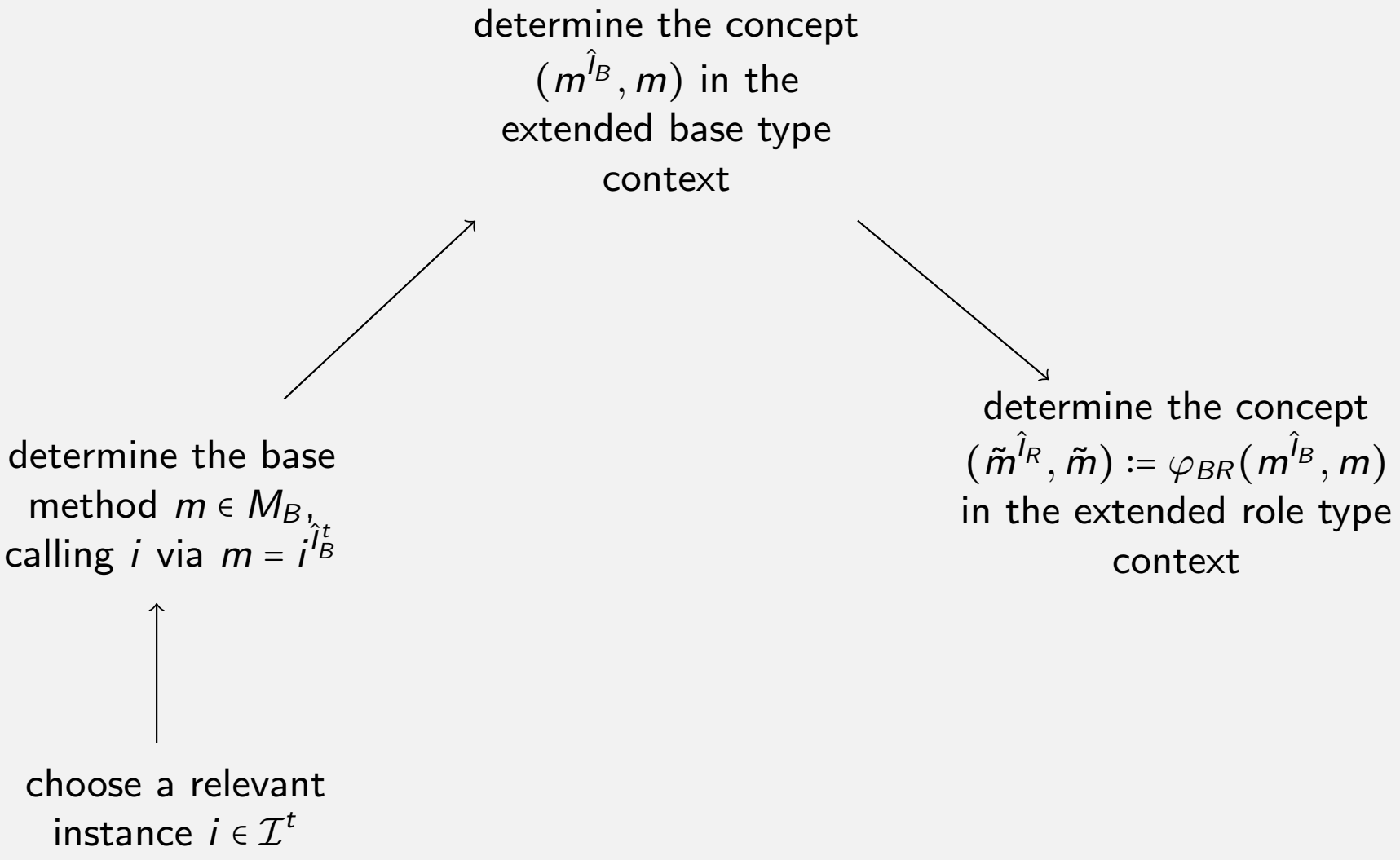
determine the concept $(m^{\hat{I}_B}, m)$ in the extended base type context



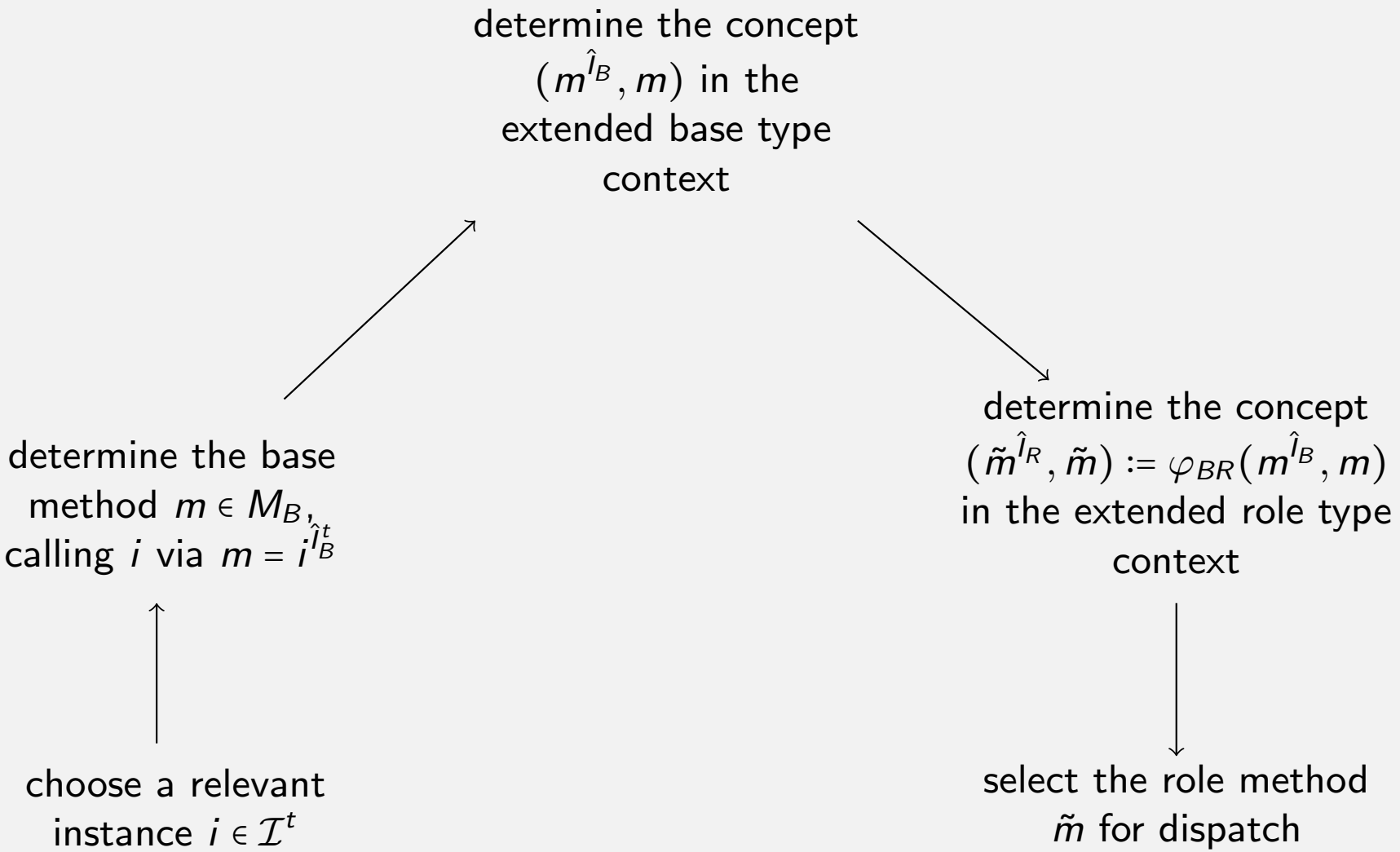
determine the base method $m \in M_B$, calling i via $m = i^{\hat{I}^t_B}$

choose a relevant instance $i \in \mathcal{I}^t$

The Algorithm



The Algorithm



Justification

- the extended role and base type contexts are co-atomistic
- the concept lattice of the method call context is isomorphic to the concept lattice of the extended base type context
- since the attribute set of the method call context is contained in the attribute set of the binding context, the second step delivers a valid concept
- the bond maps attribute concepts of $(\hat{B}, M_B, \hat{I}_B)$ to attribute concepts of $(\hat{R}, M_R, \hat{R})$ by construction
- thus, we can uniquely determine the role method that is bound to a certain base method

3. Conclusion

Conclusion

- we provided a concept-based model for representing role-oriented software models
- this model allows for modeling method bindings between base and role type hierarchy as well as describing the method dispatch
- we are thus able to support the modeling process as well as checking the model for suitability
- since method dispatch is realized by a bond between two appropriate contexts, this approach can be used for realizing classical method dispatch between base types as well

Conclusion

- we provided a concept-based model for representing role-oriented software models
- this model allows for modeling method bindings between base and role type hierarchy as well as describing the method dispatch
- we are thus able to support the modeling process as well as checking the model for suitability
- since method dispatch is realized by a bond between two appropriate contexts, this approach can be used for realizing classical method dispatch between base types as well

Some Remarks

- the introduced axioms appear pretty restricting
  $\rightsquigarrow$ however, they follow immediately from technical requirements
- the simplification towards a sequential model can easily be realized in practice
- using a mapping between lattices may seem exceedingly complicated
  $\rightsquigarrow$ the actual mapping can be computed on demand

Thank you.