Using Bonds for Describing Method Dispatch in Role-Oriented Software Models

Henri Mühle

Technische Universität Dresden
Institut für Algebra
Henri.Muehle@tu-dresden.de

Abstract. Role-oriented software modeling is an approach to object-oriented software engineering which provides a stricter encapsulation by separating the type behavior from the object into so-called *roles*. This role behavior can dynamically be accessed in certain situations and extends or alters the original type behavior. The process of extending or altering type behavior in object-oriented systems is realized by so-called method dispatch which controls message sending and routing. It is thus essential to guarantee the correct execution of the model.
In this paper we present a context-based construction to describe the method dispatch via special formal contexts containing bonds. It turns out that the bond-induced morphisms serve well for determining the role method which is bound to a certain base method during runtime. This formal context can also be used to check the role model and determine whether base and role methods are bound correctly.

Keywords: Formal Concept Analysis, Role-Oriented Software Modeling, Method Dispatch, Concept-Driven Framework

1 Introduction

Role orientation is an approach to object-oriented software modeling that relies on separating the behavior from the object. It was introduced in the 1990s by Trygve Reenskaug [7] and later investigated by Friedrich Steimann [8], who also gave a first formalisation of this approach, along with a proposal for an UML notation of these concepts. Role types encapsulate common behavior that is required in certain situations into separate modules. In contrast to subclassing or delegation – as standard techniques in object-orientation for encapsulating and altering behavior – role types allow for flexible and dynamic change of behavior without reinstantiating the object.

The scope of this paper lies in formalizing the method dispatch in role-oriented software models. Method dispatch is a mechanism in object-oriented software models that determines and invokes the correct piece of code for a certain method call [4]. Subsidiary to method dispatch along the inheritance hierarchy in standard object-oriented models, role-oriented modeling adds another dimension of method dispatch along the role-play relation. It is our goal to

provide a sound, concept-based representation of this kind of method dispatch. However, our approach shall not be seen as a mechanism to implement method dispatch in role-oriented languages in order to allow for better performance. It shall serve as a design aid with whose help role modelers can check their models for correctness and may receive design advices to improve their models.

Formal Concept Analysis (FCA) establishes a connection between binary relations and complete lattices [2]. Its basic elements are **formal contexts**, i. e. triplets (G, M, I) where G is a set of **objects**, M is a set of **attributes** and $I \subseteq G \times M$ describes whether an object *has* an attribute. Introducing two derivation operators for $A \subseteq G$ resp. $B \subseteq M$

$$A^I := \{m \in M \mid \forall\ g \in A : gIm\} \subseteq M, \qquad B^I := \{g \in G \mid \forall\ m \in B : gIm\} \subseteq G$$

one can create **formal concepts** of a formal context (G, M, I) as pairs (A, B) with $A \subseteq G, B \subseteq M, A^I = B, B^I = A$. A is then called **extent**, B is called **intent**. With introducing an order relation on the set $\mathfrak{B}(G, M, I)$ of concepts via

$$(A_1, B_1) \leq (A_2, B_2) :\Leftrightarrow A_1 \subseteq A_2 \quad (\Leftrightarrow B_1 \subseteq B_2)$$

the basic theorem of FCA [2, p. 20] states that $\underline{\mathfrak{B}}(G, M, I) := (\mathfrak{B}(G, M, I), \leq)$ is indeed a complete lattice.

An interesting way to combine two contexts is done via so-called bonds. 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 $m \in M_t$. As stated in [1, p. 15] each bond J_{st} induces two morphisms

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

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

The rest of the paper is organized as follows: In Section 2 we show the essential contexts to represent a role model in order to describe the method dispatch. A more detailed explanation of the model construction, as well as some results that show the unambiguousness of our approach can be found in [6]. Section 3 presents our dispatch algorithm as well as an example for clarification. Concluding this paper, Section 4 summarizes our results and Section 5 gives an outlook towards future work.

2 Model Construction

Since we provided a concept-based approach for describing role models and role play [5] it appears consequent to construct a concept-based formalization of the role-oriented method dispatch in order to create a sound and extensive formalization apparatus which could support software designers in their work.

2.1 Combined Representation for the Static Model

To create a suitable formal context for our purpose, we keep close to the approach in [3] using type names as formal objects, type methods as formal attributes and

type-method-incidence as context incidence. Due to the restricted length of this paper we leave the explicit construction of both (role and base type) hierarchies to [6] and instantly present the combined context.

Definition 1. *Let $\mathbb{C} = (B, R, P)$ be a role model*[1]*, M_B, M_R sets of base resp. role type methods, I_B, I_R the respective incidence relations. Let V_B, V_R be sets of* ***virtual objects***[2]*. The formal context (G, M, J) with*

$$G := \hat{B} \dot\cup \hat{R}, \quad M := M_B \cup M_R, \quad J := \hat{I}_B \cup \hat{I}_R \cup J_{BR}$$

such that $J_{BR} \subseteq \hat{B} \times M_R$ is called ***binding context*** *of $\mathbb{C}$. We have*

$$V_B := \big\{(b, m) \mid \exists\ b \in B, m \in M_B : bI_Bm)\big\}, \quad N_B := \big\{(v, m) \mid \exists\ v = (b, m) \in V_B\big\}$$

V_R and N_R are defined analogously. If J_{BR} forms a bond between $(\hat{B}, M_B, \hat{I}_B)$ and $(\hat{R}, M_R, \hat{I}_R)$ and fulfills the following conditions

$$\forall\ b \in B, m \in M_R : (b, m) \in J_{BR} \Leftrightarrow \exists\ r \in m^{\hat{I}_R} : bPr \tag{1}$$

$$\forall\ v = (b, w) \in V_B : v^J \subseteq b^J \tag{2}$$

$$\forall\ v \in V_B : |v^{J_{BR}}| = 1 \tag{3}$$

$$\forall\ m \in M_R : m^{J_{BR}} \neq \varnothing \tag{4}$$

(G, M, J) is called ***proper binding context****.*

The virtual objects in V_B resp. V_R are necessary to explicitly distinguish between the methods. We thus are able to adress each method with a special concept and thus receive unique mappings under φ_{BR} resp. ψ_{BR}.

The first two conditions of Definition 1 describe that the bond combines only such base types (and their respective virtual objects) with role methods if the according base type can play the according role type. The third condition says that each virtual object needs to be bound to exactly one role method. This is a comprehensible claim, since virtual objects in a sense represent base type methods and we assumed that each base type method is bound to exactly one role type method [6]. And lastly, the fourth condition says that for each role method there needs to exist at least one virtual object (and thus at least one base method) that is bound to this role method.

2.2 Dynamic Model for the Base Type Hierarchy

Since we want to describe the method dispatch during runtime it is necessary to describe the runtime instances in a formal context as well. According to [8] role types do not provide their own instances but are played by base type instances.

Definition 2. *Let $\hat{B}, M_B, \hat{I}_B$ be as defined in Definition 1. Let $\mathcal{I}^t$ describe the set of all active instances at the point of runtime $t \in T$. We will then introduce*

[1] Cf. [5, p. 5]

[2] From now on let $\hat{B} := B \dot\cup V_B, \hat{R} := R \dot\cup V_R, \hat{I}_B := I_B \cup N_B, \hat{I}_R := I_R \cup N_R$.

the ***method call context*** $\hat{\mathbb{B}}^t := (\hat{B}^t, M_B, \hat{I}_B^t)$ *via*

$$\hat{B}^t := \hat{B} \cup \mathcal{I}^t, \quad \hat{I}_B^t := \hat{I}_B \cup C^t$$

where $C^t := \{(i,m) \mid \exists\ i \in \mathcal{I}^t, m \in M_B : i \text{ receives a call from } m\}$

It has to be said that we assume a sequential execution of our role model, i. e. each instance can only call one single method at a time. This is sufficient since one can easily map parallel activities to a sequential execution plan.

3 Performing the Method Dispatch

It is essential for role modeling to determine at runtime the method dispatch between base types and the according role types. At modeling time (i. e. when setting up the role model) each method $m_b \in b^{I_B}$ of a base type $b \in B$ is assigned to a method $m_r \in r^{I_R}$ of a respective role type $r \in R$ that can be played by b in order to alter the behavior of the base type when playing this role. At runtime it is necessary to correctly dispatch method calls from the base type method to the appropriate role type method to guarantee correct altering of the behavior.

For resolving the method dispatch, we will use the bond-induced morphisms φ_{BR} and ψ_{BR} of the proper binding context as recalled in Section 1.

Let $t \in T$ be a point of runtime and $i \in \mathcal{I}^t$ a certain instance receiving a method call from $m \in M_B$. Algorithm 1 shows how the role method $\tilde{m} \in M_R$ that is bound to m can be determined.

Algorithm 1 An algorithm for method dispatch

Require: method call context $\mathbb{B}^t = (\hat{B}^t, M_B, \hat{I}_B^t)$, proper binding context (G, M, J), instance $i \in \mathcal{I}^t$, method $m \in M_B$
Ensure: $m \in i^{\hat{I}_B^t}$
1: $c := (m^{\hat{I}_B}, m^{\hat{I}_B \hat{I}_B})$
2: $\tilde{c} := \varphi_{BR}(c)$
3: $\tilde{m} := \mathrm{int}(\tilde{c})$
4: **return** $\tilde{m}$

The algorithm requires the appropriate method call context as well as the proper binding context and gets an active instance $i \in \mathcal{I}^t$ as well as a base type method $m \in M_B$ as inputs. It needs to be ensured that i indeed receives a call from m. Applying the bond-induced morphism φ_{BR} from the base type context to the role type context to the attribute concept of m we receive a concept having only role methods in its intent. One can show that φ_{BR} maps concepts having only one base method in their intent towards concepts having only one role method in their intent which guarantees the determinacy of the result. [6]

Example 1. A proper binding context for the role model in Figure 1 is shown in Figure 2. Let us assume a set $\mathcal{I} := \{\texttt{Aßmann}, \texttt{Mühle}, \texttt{Wende}\}$ of runtime instances. If we further assume a point of runtime where Professor `Aßmann` holds a lecture and explains a situation, Student `Wende` writes down some notes and Student `Mühle` tries to chatter with his neighbor, we receive a method call context as depicted in

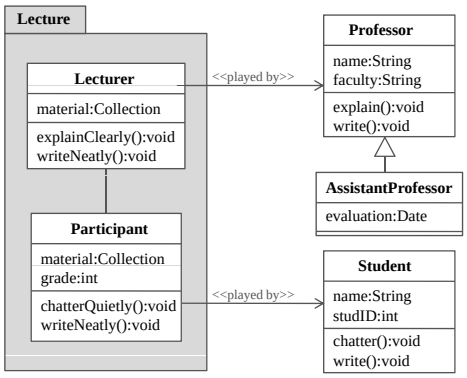

Fig. 1. An example role model *Lecture*

J	write()	explain()	chatter()	wrNeatly()	exClearly()	chQuietly()
Professor	×	×		×	×	
Pr1	×			×		
Pr2		×			×	
AssistantProfessor	×	×		×	×	
As1	×			×		
As2		×			×	
Student	×		×	×		×
St1	×			×		
St3			×			×
Lecturer				×	×	
Le1				×		
Le2					×	
Participant				×		×
Pa1				×		
Pa3						×

Fig. 2. A proper binding context of the Lecture from Figure 1. The attribute concept of the attribute *write()* is marked lightgray, while the mapping of this concept under φ_{BR} is marked in a darker gray.

$\hat{I}_B^t$	write()	explain()	chatter()
Professor	×	×	
Pr1	×		
Pr2		×	
`Aßmann`		×	
AssistantProfessor	×	×	
As1	×		
As2		×	
Student	×		×
St1	×		
St3			×
`Mühle`			×
`Wende`	×		

Fig. 3. The method call context of a specific situation during the Lecture from Figure 1

Figure 3. As we can see from the marked concepts in Figure 2, the base method `write()` is properly mapped towards the role method `writeNeatly()`.

4 Summary and Conclusion

Role-oriented software modeling is an approach towards object-oriented software engineering, gaining a higher encapsulation and modularization of software models by separating the behavior from the object. The behavior is encapsulated in special modules, *roles*, and is woven into the software model during runtime.

The crucial point in role modeling is the so-called method dispatch, which redirects method calls to base methods towards appropriate role methods and

thus enables the intended change of type behavior. Since the model designer determines at modeling time which base methods can be altered by which role methods when playing the respective role, it has to be guaranteed that performing the method dispatch during runtime strictly follows these assignments.

Our approach uses a context construction via so-called bonds to create special formal contexts which uniquely represent the assignment between base and role methods. We then presented an algorithm using these contexts to determine the role method that is assigned to a given base method. Our construction can thus be used to assist the process of role-oriented software modeling.

5 Outlook

Together with the results from [5], where we introduced a basic context representation for base and role type hierarchy, as well as for the role-play relation, this paper can be seen as a basic fundament for a formal, concept-based description language for role-oriented software modeling.

However, there are still a lot of open fields of research to completely cover role modeling with concept-based constructions. Among others, it is on the one hand necessary to describe composition and decomposition of large role models, e. g. to build kind of a design advisor for role models that helps to identify redundancy or inconsistencies of the model. On the other hand, it is necessary to provide an extensive framework to represent role-play constraints or special characteristics of role-oriented software design, like multiple role play. Since we have already applied FCA successfully towards the foundations of role modeling, a further research in this direction will be very promising.

References

1. Bernhard Ganter. Relational Galois Connections. In *ICFCA 2007 Proceedings*, pages 1–17. Springer, 2007.
2. Bernhard Ganter and Rudolf Wille. *Formale Begriffsanalyse: Mathematische Grundlagen*. Springer, Heidelberg, 1996.
3. Robert Godin and Petko Valtchev. Formal Concept Analysis-based Class Hierarchy Design in Object-Oriented Software Development. In *Formal Concept Analysis: Foundations and Applications*, pages 304–323. Springer, 2005.
4. Wade Holst and Duane Szafron. A General Framework for Inheritance Management and Method Dispatch in Object-Oriented Languages. In *ECOOP 1997 Proceedings*, pages 276–301, 1997.
5. Henri Mühle and Christian Wende. Describing Role Models in Terms of Formal Concept Analysis. In *ICFCA 2010 Proceedings*, pages 241–255. Springer, 2010.
6. Henri Mühle. Using Bonds for Describing Method Dispatch in Role-Oriented Software Models. Preprint MATH-AL-04-2010, TU Dresden, Institut für Algebra, 2010.
7. T. Reenskaug, P. Wold, and O. A. Lehne. *Working with Objects: The OOram Software Engineering Method*. Manning Publications, Greenwich, CT, 1996.
8. Friedrich Steimann. On the Representation of Roles in Object-Oriented and Conceptual Modelling. *Data Knowledge Engineering*, 35:83–106, 2000.