A Metamodel Family for Role-based Modeling and Programming Languages

Thomas Kühn, Max Leuthäuser, Sebastian Götz,
Christoph Seidl, Uwe Aßmann

Technische Universität Dresden
Software Technology Group
{thomas.kuehn3|max.leuthaeuser|christoph.seidl|uwe.assmann}@tu-dresden.de
sebastian.goetz@acm.org

Abstract. Role-based modeling has been proposed almost 40 years ago as a means to model complex and dynamic domains, because roles are able to capture both context-dependent and collaborative behavior of objects. Unfortunately, while several researchers have introduced the notion of roles to modeling and programming languages, only few have captured both the relational and the context-dependent nature of roles. In this work, we classify various proposals since 2000 and show the discontinuity and fragmentation of the whole research field. To overcome discontinuity, we propose a family of metamodels for role-based modeling languages. Each family member corresponds to a design decision captured in a feature model. In this way, it becomes feasible to generate a metamodel for each role-based approach. This allows for the combination and improvement of the different role-based modeling and programming languages and paves the way to reconcile the research field.

1 Introduction

Role-based modeling has been proposed in 1973 by Charles W. Bachman [2] as a means to model complex and dynamic domains, because roles are able to capture both context-dependent and collaborative behavior of objects. Consequently, they were introduced in various fields of research ranging from data modeling [2, 31] via conceptual modeling [56, 26] through to programming languages [3, 35, 8]. Unfortunately, each of these approaches focuses on a specific field without taking results of other fields into account.[1] As a result, the years of research on role-based modeling had almost no influence on software development practice. This is troubling, because current software systems are characterized by increasing complexity and context-dependence. Moreover, they are designed by means of objects and references, introduced in 1967 [45].

Despite the fact that relationships and roles are represented in various domain modeling languages, e.g., the Entity-Relationship Model [13] and the Unified Modeling Language (UML) [52], their implementation is buried in classes

[1] Please note that this work considers Role-Based Access Control (RBAC) [17] as a special application for roles with a rather narrow scope.

containing collections of references making their implementation overly complex and error prone [53, 6]. The situation becomes even worse, if the domain model requires that two unrelated objects take the same place in a relation, because the developer must introduce a counter-intuitive super type for both objects [56]. Unfortunately, while several researchers have introduced the notion of roles to programming languages [10, 8, 11, 37, 51], only few provide an underlying metamodel for their context-dependent or relational roles. To make things worse, only few approaches incorporate both the context-dependent and the relational nature of roles, e.g., [46, 34]. In summary, the various approaches cannot be combined easily because they lack a common metamodel.

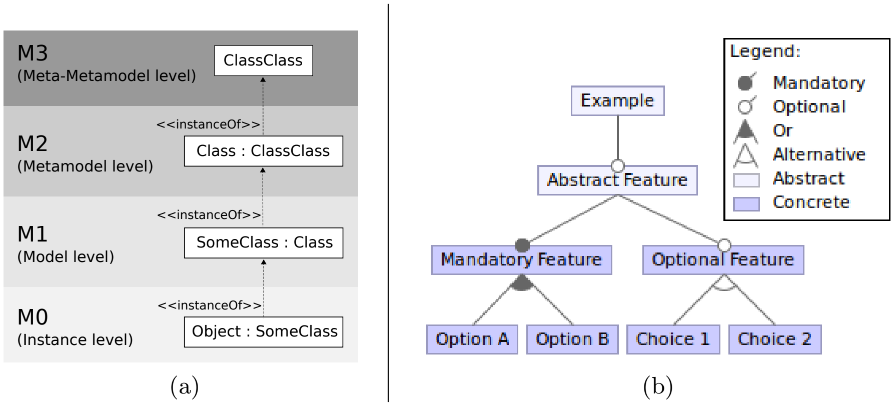
This work classifies the various notions proposed since 2000 along 26 features of roles and shows the apparent discontinuity and fragmentation of the whole research field. We attribute this situation to the missing compatibility of the various approaches. To cure this, we propose not only a single metamodel but a family of metamodels for role-based modeling languages (RML). Each member of the family corresponds to the 26 design decisions captured in a feature model. In this way, it becomes feasible to generate a metamodel for two approaches and then combine them by mapping their sibling metamodels to a merged metamodel. This allows for the combination and improvement of the different RMLs and paves the way to reconcile the research field.

This paper is structured as follows. Sect. 2 gives a general introduction to modeling languages, roles, and feature modeling. Afterwards, Sect. 3 introduces the classification scheme used in the next section (Sect. 4) to evaluate the various approaches published since the year 2000. As a result of this evaluation, Sect. 5 introduces a metamodel family for RMLs to overcome the discontinuity and fragmentation in the research field by providing a common source to model and compose existing approaches. The discussion is concluded by highlighting the related work (Sect. 6) and a summary and outlook to further research (Sect. 7).

2 Preliminaries

2.1 Elements of Modeling Languages

A modeling language has syntax (defining the notational aspects) and semantics (defining the meaning) [32]. A metamodel can be seen as an explicit model containing constructs and rules necessary to build more specific models within a domain of interest. Hence, they are models of modeling languages describing their abstract syntax. The *Meta Object Facility* (MOF) [49], for instance, assigns data to the following four meta levels (see Fig. 1a). The instance level (M0) encompasses concrete data. Models are located in the model level (M1). This level covers, e.g., concrete instantiations of UML-models, logical data models or process models that are defining data at M0. Metamodels like mentioned above are contained in the metamodel level (M2). The most abstract level, the meta-metamodel level (M3), is responsible for defining models at M2 and can be bootstrapped by itself to prevent an infinite number of meta-levels.



(a) (b)

Fig. 1: Visualization of the Meta-Object Facility (a) and feature models (b)

2.2 Ontological Foundations of the Role Concept

To classify the role concept, two meta-predicates are sufficient: rigidity and dependence [24]. *Rigidity*, as explained in [23], distinguishes properties that must hold for all instances of a concept in every possible world. As an example for the former, the concept *Human* is rigid, because each instance is necessarily a human until it ceases to exist. The latter case is further described as *anti-rigidity* by Guizzardi et al. [29], denoting properties that can cease to hold for all instances of such a concept. Thus, a *Customer* is *anti-rigid* because every instance may stop being a customer without ceasing to exist. *Dependence*, as explained in [24], characterizes properties as being meaningful only in combination with a counter-property. Thus, a dependent property always demands for another property against which it is semantically founded. For example, the concept *Customer* is founded against the counter-concept *Seller* in the context of a *Shop*.

Roles are defined as anti-rigid and dependent concept [24]. The anti-rigidity characterizes the ability of roles to be played. The dependence characterizes the need of roles to be defined as part of a context, which includes besides the role itself at least one corresponding counter-role. In contrast to roles, *naturals* are defined as rigid and independent concepts. Thus, naturals are not played, but can serve as players for roles. In addition, naturals do not have to be part of contexts as they do not require a counter-concept.

2.3 Feature Modeling

Modeling the variability of a certain domain has, among others, been studied in the context of software product line engineering [14]. Here, an often used technique to capture variability is feature modeling [44]. Therein, a feature model decomposes a domain's concerns into interconnected features, which form a decision tree. For this purpose, constraints can be defined between the features, e.g., whether the existence of a feature is mandatory or if a set of features mutually excludes each other. Fig. 1b depicts an exemplary feature model. It comprises a single top feature (*Example*), which is decomposed into a single optional feature (*AbstractFeature*) being subject to further decomposition. The types of constraints and features used in this paper are shown in the legend of this figure.

3 Classification of Role-based Modeling Languages

After introducing the basic elements of modeling languages and the role concept, this section presents a thorough analysis of the various *Role-based Modeling Languages* (RML) as well as *Role-based Programming Languages* (RPL) proposed since the year 2000. We choose that year, because Friedrich Steimann published a thorough analysis of RML in that year [56]. Additionally, he identified 15 features of roles useful to classify and compare all subsequent approaches. Since then, many modeling languages utilizing roles have been published. However, only two applied Steimann's classification scheme, namely [11] and [37]. Hence, it is time for another thorough analysis and classification of the various contemporary approaches.

3.1 Steimann's Features

Before we evaluate the literature body, we need to describe our classification scheme. Therefore, we briefly recapitulate *Steimann's classification* and introduce additional features of roles retrieved from the contemporary literature. The latter is crucial, because current approaches have shifted their focus to the context-dependent nature of roles not included in Steimann's classification.

In particular, it contains a list of 15 features attributed to roles by various researchers [56, pp. 86-87]. This list, enumerated in Fig. 2, captures different views on roles. Nevertheless, it has three major drawbacks, if it is used as a classification scheme. First, as Steimann already admitted in [56, pp. 86], some of the listed features are conflicting, e.g., Feature 14 and 15 querying whether a role has a shared or its own identity. Second, there are implicit dependencies among some of the features. Consider, for instance, an approach where roles have no properties and behavior (Feature 1). Such an approach can never satisfy Feature 10, 11 and 13 depending on the structure of roles. Last, several features concern different levels of the model hierarchy. Features 4, 5, 9, 10, 12, 14, and 15 only apply to roles at runtime. Thus, these features cannot be applied to approaches focusing solely on *the model level* (M1). Henceforth, we explicitly associate the features to the levels they affect and, later on, will organize the features in a feature model rather than a plain list. Nevertheless, this initial list of features still proves useful as a classification scheme. However, it does not encompass the view that roles can be used to model context-dependent properties [41] and behavior [37, 50, 43].

1.	Roles have properties and behaviors	*(M1, M0)*
2.	Roles depend on relationships	*(M1)*
3.	Objects may play different roles simultaneously	*(M1, M0)*
4.	Objects may play the same role (type) several times	*(M0)*
5.	Objects may acquire and abandon roles dynamically	*(M0)*
6.	The sequence of role acquisition and removal may be restricted	*(M1, M0)*
7.	Unrelated objects can play the same role	*(M1)*
8.	Roles can play roles	*(M1, M0)*
9.	Roles can be transferred between objects	*(M0)*
10.	The state of an object can be role-specific	*(M0)*
11.	Features of an object can be role-specific	*(M1)*
12.	Roles restrict access	*(M0)*
13.	Different roles may share structure and behavior	*(M1)*
14.	An object and its roles share identity	*(M0)*
15.	An object and its roles have different identities	*(M0)*

Fig. 2: Fiedrich Steimann's 15 classifying features, extracted from [56].

16.	Relationships between roles can be constrained	*(M1)*
17.	There may be constraints between relationships	*(M1)*
18.	Roles can be grouped and constrained together	*(M1)*
19.	Roles depend on compartments	*(M1, M0)*
20.	Compartments have properties and behaviors	*(M1, M0)*
21.	A role can be part of several compartments	*(M1, M0)*
22.	Compartments may play roles like objects	*(M1, M0)*
23.	Compartments may play roles which are part of themselves	*(M1, M0)*
24.	Compartments can contain other compartments	*(M1, M0)*
25.	Different compartments may share structure and behavior	*(M1)*
26.	Compartments have their own identity	*(M0)*

Fig. 3: Additional classifying features, derived from the literature.

3.2 Additional Features

To include the new perspective on roles, this section gives a list of the features attributed to roles that we have identified in the literature.

16. Relationships between roles can be constrained. If roles depend on relationships, then it might be possible to further constrain them by *intra-relationship constraints* [30, 8, 48], i.e., irreflectivity, total order or exclusive parthood.

17. There may be constraints between relationships. In contrast to feature 16, this property suggests the existence of *inter-relationship constraints*, like the subset constraint [31, 30, 10, 48].

18. Roles can be grouped and constrained together. Most approaches suggesting to constrain roles [15, 11, 9] do not permit to group them and apply constraints to a whole group of related roles as suggested in [60, 37].

Together, these three properties specify ways to constrain roles, but do not account for their context-dependence. However, in this case, the use of the term *context* leads to a dichotomy of its meaning. On the one hand, according to Anind Dey [16], "context [represents] any information that can be used to characterize the situation of an entity". Thus, everything that can be attributed to an object in a situation contributes to its context. On the other hand, within modeling languages, context represents a collaboration or container of a fixed, limited scope [19, 41, 50, 43]. To overcome this dichotomy, researchers avoided the term context by using other terms, i.e., *Environments* [60], *Institutions* [4], *Teams* [36] and *Ensembles* [34]. In turn, we use the term *Compartment* as a generalization of these terms to denote an objectified collaboration with a limited number of participating roles and a fixed scope[2].

19. Roles depend on compartments. Most of the recent approaches agree that roles are dependent on some sort of context. We call them compartments [36, 60, 4, 19, 41, 50, 43, 34]. A typical example of a compartment is a university, which contains the roles *Student* and *Teacher* collaborating in *Courses* [37, 7, 47].

20. Compartments have properties and behaviors like objects [19, 41, 50, 43].

21. A Role can be part of several compartments [4, 60, 19, 46]. This property suggests that a role (type) can be part of more than one compartment. Consider again the role type *Teacher*. It can be used in different compartments, i.e., *School* or *University*, where it might be implemented and constrained differently [4].

22. Compartments may play roles like objects. While most approaches use compartments as a grouping mechanism, compartments can be seen as entities similar to naturals being able to play roles, as well. This view is captured within the metamodel for roles [19] and implemented in ObjectTeams/Java [37].

23. Compartments may play roles which are part of themselves. Continuing the argument of feature 22, compartments might be allowed to play roles belonging to the same compartments, as possible in [19, 37].[3]

24. Compartments can contain other compartments. In addition to the previous properties, three approaches allow compartments to contain other compartments [37, 41, 42]. This *nesting* is proposed to further structure compartments into smaller sub-compartments [37, 42] and, thus, enable the representation of a university containing academic departments which in turn contain faculties.

25. Different compartments may share structure and behavior [40, 41]. This means that definitions of compartments may inherit properties, features, roles, and constraints from each other. However, to fully support inheritance and polymorphism of compartments, the rules of family polymorphism have to be applied [40].

26. Compartments have their own identity. This feature is acknowledged by all approaches who treat compartments as first-class entities of the instance level [60, 37, 46, 50, 42, 34]. Thus, this feature is a prerequisite for the existence of compartments at runtime. However, it is an open question whether this identity is unique or composed from the identities of the participating objects.

[2] Note that compartments are defined top-down in the conceptual model whereas contexts are built bottom-up from individual sensor readings.

[3] This feature is described in §2.1.2 (b) of ObjectTeams/Java's language definition [39].

From this list, condensed in Fig. 3, it becomes apparent that researchers have successfully applied the concept of roles to the domain of context-dependent (or context-aware) systems. This has led to a number of new features attributed to roles affecting both model and instance level. Surprisingly, the definitional dependence of roles [25] is still applicable to compartments representing the definitional boundary and execution scope for their enclosing roles. Hence, the first 18 features highlight the relational nature of roles whereas the last eight emphasize the context-dependent nature of roles. As a result, this list is suitable to further study and classify the various RMLs and RPLs.

4 Survey of Recent Approaches

After devising a proper classification scheme, this section applies it to survey the various contemporary RMLs and RPLs. For that reason, we checked all role-based modeling or programming language (excluding RBAC) published between the year 2000 and 2014 by either *IEEE*, *ACM*, *Springer* or *ScienceDirect*. Additionally, only those approaches were selected that provided enough information about their role model to actually apply our classification. In summary, we have selected nine modeling languages ranging from data modeling via conceptual modeling to architecture modeling and seven programming languages with either relational or context-dependent roles. However, due to the fact that some of the identified features of roles only affect the instance level (M0), such features are not applicable to modeling languages defining conceptual models. For programming languages, on the other hand, each feature is applicable because they incorporate both the model and the instance level.

4.1 Modeling Languages

In the following, we investigate the various RMLs proposed for conceptual modeling, data modeling, and generalization of the role concept.

For *conceptual modeling* **Lodwick** [56] is the first formal modeling language for relational roles. Its formal definition includes *natural types* filling *role types*, whereas the latter are placeholders in a binary *relationship*. However, roles are only represented on the conceptual model and do not carry on to instances of that model [56]. **Onto-UML** [28], on the other hand, is an ontologically founded conceptual modeling language developed by Guizzardi et al. [29, 28] to overcome the syntactical and semantical shortcomings of UML. The underlying *Universal Foundational Ontology* (UFO) [27] contains *RoleTypes*, *RoleMixins*, and *Relators*; and is used to annotate UML classes and relations with stereotypes. Role mixins, however, are only used to model role types playable by unrelated types [29]. In sum, both Lodwick and Onto-UML only deal with the creation of formal conceptual models, but are neither concerned with the representation of roles at runtime nor context-dependent entities. In contrast to them, the **Helena Approach** [34] incorporates both concerns. It was proposed by Hennicker et al. [34] to specify the collaborative behavior of *Ensembles* of distributed

Components and designed to cope with the high dynamics of collaborative executions. Here, *Ensemble Structures* are reification of collaborations containing *Role Types* defining the behavior of *Components* playing that role [34]. Additionally, the communication between two roles is restricted by *Role Connectors* [34], i.e. directed channels between two role types, similar to a relationship. Consequently, their model not only captures both natures of roles in a simple model but also the semantics by means of *Labeled Transition Systems*. Nevertheless, they missed to specify the actual interaction between the player and its assuming roles.

In the field of *data modeling*, however, interactions are of no concern. Herein, **Object-Role Modeling (ORM) 2** [30] is the most mature fact-oriented data modeling methodology. Roles, however, are only included as unnamed places in binary relationships. Despite that, ORM provides a large number of available constraints for these relationships including role constraints, inter- and intra-relationship constraints [30]. Nevertheless, it did not embrace the possible flexibility provided by the role concept. This is different for the **Information Networking Model** (INM) [46] designed to overcome the lack of classical RML models to capture context-dependent information. Hence, INM introduces the concept of *Contexts* to group *Roles*. While this approach allows to nest contexts with attributes containing roles, INM cannot constrain the various kinds of relations [46]. In sum, INM would be a good extension to ORM 2, if they were compatible.

In contrast to the previous approaches, the following ones *generalize the role concept*. The **Generic Role Model** [15], for instance, introduces a new inheritance relation, denoted *role relationship*, to permit dynamic changing classes, multiple instantiation of the same class, and context-dependent access [15]. While roles are mingled within the inheritance hierarchy on the model level (M1), they are represented as adjunct objects on the instance level (M0) leading to object schizophrenia [38]. On the down side, it does not account for the relational and context-dependent nature of roles. The **E-CARGO Model** [60], on the other hand, is a role-based model for *computer-supported cooperative work* (CSCW) encompassing *Agents* playing *Roles* collaborating in *Groups* working on *Objects* in defined *Environments* [60]. Groups are used to arrange and manage collaborating agents and their assumed roles [60], whereas environments, such as contexts, specify the workspace of several groups and limit the number of roles played simultaneously [60, Appendix]. Thus, the combination of groups as collaborations and environments as their instantiation resemble compartments. In sum, while the target domain is cooperative work, the underlying model is applicable to role-based software systems as well. Similar to E-CARGO, **Data Context Interaction** (DCI) [50] emphasizes context-dependent roles. Trygve Reenskaug et al. [50] proposed this paradigm to point out that *Data* plays a *Role* in *Interactions* encapsulated in *Contexts* [50]. In particular, objects serve as data containers whose behavior is defined in roles part of a certain context. The context manages the binding of role instances to data objects as well as their interaction [50]. Additionally, several implementations of this paradigm exist, e.g., in Scala [22]. One of the first **Metamodel for Roles** was proposed by Genovese [19] in an attempt to define the most general definition of roles. It incorporates both natures of roles and, thus, introduces *Players*, *Roles* and *Contexts* on both the model and instance level together with relations denoting which roles belong to which context and be filled by which player [19]. Furthermore, he introduced *Sessions* to specify the binding of attributes when roles collaborate with one another in a context. Besides that, the only possibility to adjust this general metamodel to a target language is to specify additional constraints to both model and instance level, which is only briefly discussed in [19].

4.2 Programming Languages

After focusing on RMLs, this section investigates contemporary RPLs. Notably, most approaches are extensions to Java, which are either compiled to Java source code [20, 33, 3, 9] or directly to bytecode [36]. Hence, we divide the discussion into RPLs that support *plain roles*, *relational roles*, as well as *contextual roles*.

The first class of RPLs focuses solely on implementing objects playing roles. Hence, **Chameleon** [20] features roles with so called *constituent methods* allowing to overwrite methods of their players, which work like advices in Aspect-oriented Programming (AOP). However, the major drawback of Chameleon is the fact that roles extend their player to gain access to the player, which is both conceptually wrong [56] and limits the flexibility of roles. **Rava** [33] overcomes these issues by employing the *Role-Object-Pattern* [12] extended with the *Mediator-Pattern* [18]. They use special keywords to steer the generation of the necessary management code. Due to the use of the Role-Object Pattern and generation to plain Java, this solution suffers from object schizophrenia [38]. **JavaStage** [9] eludes this problem, by only supporting static roles, i.e., the roles are directly compiled into the possible players as inner classes. To avoid name clashes, it employs a customizable method renaming strategy. Its main advantages is the capability to specify a list of required methods instead of a specific player class. Surprisingly, this approach limits itself to static roles unable to represent their relational and context-dependent nature.

Consequently, we proceed with **Rumer** [8], which contributes relationships as first class citizens and modular verification over shared state. Furthermore, Balzer et al. [8] provide several intra relationship constraints usable to further restrict these relationships. Roles, on the other hand, are the named places of a relationship with attributes and methods but without inheritance. Despite that, roles are only accessible within a relationship and not from their player. Consequently, it is promising to combine this approach with another one with context-dependent roles, described next.

The most sophisticated approach to context-dependent roles so far is **ObjectTeams/Java** (OT/J) [36]. Similar to Chameleon above, OT/J allows to override methods of their player by aspect weaving. Besides that, it introduces *Teams* to represent compartments whose inner classes automatically become roles. Notably, OT/J supports both the inheritance of roles and teams whereas the latter leads to family polymorphism [40]. On the downside, it does neither support multiple unrelated player types for a role type nor first class relationships and only a limited form of constraints. This is similar to **powerJava** [3], which also introduces compartments, denoted *Institutions* [3], whose inner classes represent roles. However, PowerJava features the distinction between role interface and role implementation where the former is callable from outside a specific institution and the latter is the institution-specific implementation of the same interface [1]. Both Rava and powerJava are the only research prototypes providing a working compiler. Nonetheless, the project has been abandoned [59]. A more recent approach towards context oriented programming is **NextEJ** [42] as the successor of EpsilonJ [54]. It provides *Contexts* as first class citizens which do not only group roles but also represent an activation scope at runtime. These *context activation scopes* can be nested and act as a barrier where all roles are instantiated and bound automatically. So far, they only published their type-system of the core calculus [43] and no compiler for NextEJ. Surprisingly, no approach published so far included both relational and context-dependent roles.

4.3 Summary

After describing the various approaches, Table 1 shows the classification of the previous RMLs and RPLs by investigating the number of fulfilled features. The table lists the modeling and programming language approach in chronological order. Each feature can be either fulfilled, not fulfilled, possible to fulfill or not applicable. In detail, we have classified features possible to fulfill if they can be fulfilled by reusing model elements but are not supported by the underlying model, and not applicable if they only affect the instance level not treated by the particular approach. In sum, it depicts the progress in the research field.

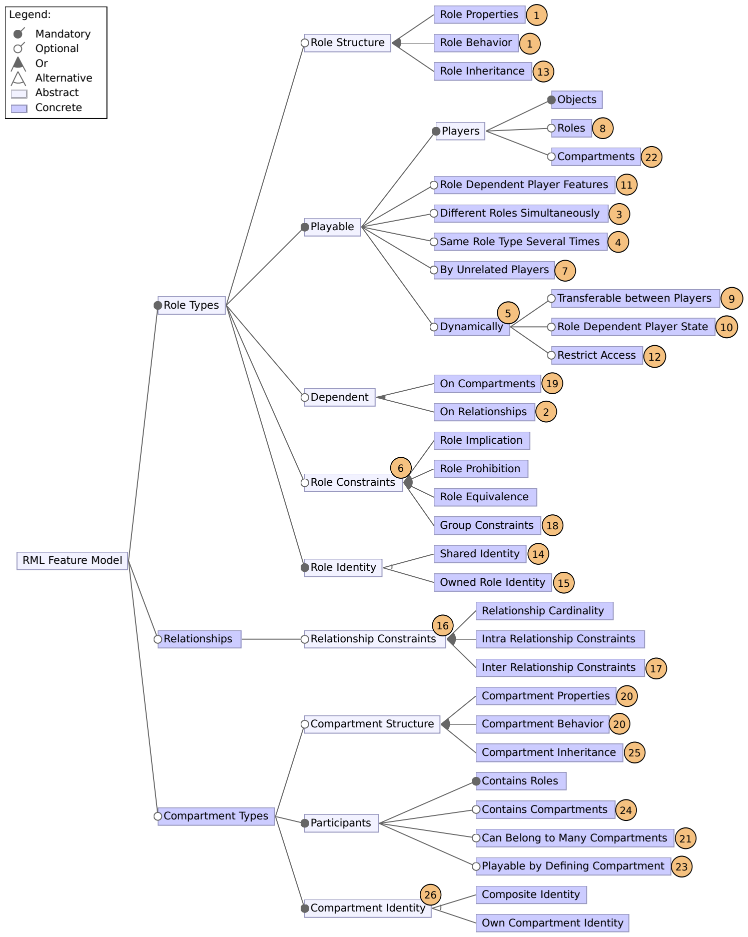
At a first glance the surveyed research field seams to advance over the years as each approach increased the number of supported features. However, at a closer look, the table indicates that the research field suffers from fragmentation and discontinuity. The former denotes that each approach focuses solely on a specific goal in a specific domain and do not take results of other related domains into account. To make things worse, our evaluation indicates that more than half of the approaches were unaware of the possible features of roles or other related approaches. Moreover, the features of roles implemented in the various programming languages were not transferred back to modeling languages and vice versa. This might be the result of diversity or negligence in the research community. Discontinuity, on the other hand, highlights the fact that each approach builds its role concept from scratch. None of the investigated approaches reused either formal models or metamodels as their basis for their approach. Moreover, solutions to the representation of roles were not applied to other works but just reinvented. As a result, the various approaches for relational or context-dependent roles cannot be combined easily because they neither share a common underlying model nor a common understanding of roles. Apparently, there is no continuous improvement or combination of previously proposed role-based languages.

In sum, these results are surprising, considering the foundations Steimann [56] provided. The next section proposes a solution to the incompatibility of the various approaches and thus tackles the discontinuity of the research field.

Table 1: Evaluation of role-based modeling and programming languages.

Feature	**Lodwick** 2000 [56]	**Generic Role Model** 2002 [15]	**ORM 2** 2005 [30]	**E-CARGO Model** 2006 [60]	**Metamodel for Roles** 2007 [19]	**INM** 2009 [46]	**DCI (in Scala)** 2009 [50]	**Onto-UML** 2012 [28]	**Helena Approach** 2014 [34]	**Chameleon** 2003 [20]	**OT/J** 2005 [36]	**Rava** 2006 [33]	**powerJava** 2006 [5]	**Rumer** 2007 [8]	**NextEJ** 2009 [42]	**JavaStage** 2012 [9]
1	■	■	□	⊞	■	■	■	■	■	■	■	■	■	■	■	■
2	■	□	■	□	□	⊞	⊞	⊞	⊞	□	⊞	□	⊞	■	⊞	□
3	■	■	■	■	⊞	■	■	⊞	■	■	■	■	■	■	■	■
4	■	■	■	■	■	■	□	∅	■	■	■	□	■	■	■	□
5	■	■	■	■	■	∅	⊞	∅	■	■	■	■	⊞	■	■	□
6	⊞	■	⊞	⊞	∅	□	□	□	□	□	■	□	■	■	□	■
7	■	□	□	⊞	■	□	■	⊞	■	■	□	■	■	⊞	■	■
8	□	■	□	□	■	■	□	■	□	□	■	□	■	□	■	■
9	⊞	□	∅	■	⊞	∅	□	∅	□	■	□	□	■	□	■	□
10	⊞	■	∅	⊞	⊞	■	■	■	■	■	■	■	■	■	■	■
11	⊞	■	□	□	⊞	■	■	■	■	■	■	■	■	■	■	■
12	∅	⊞	∅	■	∅	∅	□	∅	■	■	■	■	■	■	■	■
13	⊞	■	□	□	■	■	□	⊞	□	□	■	■	■	□	□	■
14	■	□	■	⊞	⊞	■	⊞	■	□	⊞	⊞	□	□	■	⊞	□
15	□	■	□	■	⊞	□	⊞	□	■	■	■	■	■	□	■	■
16	■	□	■	□	■	□	□	■	□	□	□	□	□	■	□	□
17	□	□	■	□	■	□	□	■	□	□	□	□	□	□	□	□
18	□	□	□	⊞	□	□	□	□	□	□	■	□	□	⊞	⊞	□
19	⊞	□	⊞	■	■	■	■	⊞	■	□	■	□	⊞	⊞	■	□
20	□	□	□	□	■	■	■	■	■	□	■	□	■	■	■	□
21	⊞	□	□	■	■	⊞	□	■	⊞	□	□	□	■	□	■	□
22	□	□	■	□	■	■	□	□	□	□	■	□	□	■	□	□
23	□	□	□	□	■	□	□	□	□	□	■	□	□	□	□	□
24	□	□	□	□	□	■	□	□	□	□	■	□	⊞	■	■	□
25	□	□	□	□	■	■	□	□	□	□	■	□	□	□	□	□
26	□	□	■	■	⊞	■	■	□	■	□	■	□	⊞	■	■	□
	Modeling Languages									Programming Languages						

■: yes, ⊞: possible, □: no, ∅: not applicable



(a)

$$RoleTypes.Dependent.OnRelationships \Leftrightarrow Relationships \tag{1}$$

$$RoleTypes.Dependent.OnCompartments \Leftrightarrow CompartmentTypes \tag{2}$$

$$RoleImplication \Rightarrow RoleEquivalence \tag{3}$$

$$RoleTypes.Playable.Players.Compartments \Rightarrow CompartmentTypes \tag{4}$$

(b)

Fig. 4: Feature model (a) and cross-tree constraints (b) for role-based modeling languages

5 A Metamodel Family for Role-Based Languages

One of the reasons for the discontinuity in the research field is the incompatibility of the various approaches. However, to be able to freely combine the various RMLs and RPLs, they must have compatible metamodels. Unfortunately, only few approaches actually defined and published their underlying metamodel, e.g., [37]. Consequently, it is infeasible to create or combine the metamodels of two approaches. Thus, we propose a method to generate metamodels for RML based on the features of a particular language. Henceforth, we employ feature modeling as the suitable development methodology together with the tool *FeatureIDE* [58] as development environment and *DeltaEcore* [55] to generate the language metamodels in Ecore [57].

5.1 Feature Model for Role-Based Languages

The first step of feature modeling is to generate a feature model as a hierarchical representation of the 26 identified features of roles. In this way, we elucidate the various implicit dependencies of the features presented previously. Fig. 4 (a) depicts the resulting feature model for RML. It encompasses three main feature arcs for *role types*, *relationships*, and *compartment types* to group features dependent on the existence of these entities. Please note that those features are marked mandatory, which is essential for the existence of that entity. Consider for instance that role types must at least be playable by objects. In addition to the dependencies of features evident in the feature model, we had to define four *cross-tree constraints* [58], depicted in Fig. 4 (b). These constraints ensure that a configuration contains all entities on which the *Role Type* depends (Eq. 1 and 2), that *Role Equivalence* is supported, if the *Role Implication* is (Eq. 3), and that compartment types are supported, if compartments can be players (Eq. 4). To further increase the traceability, we annotated each feature with the corresponding number of the feature list (Fig. 2 and 3). Thus, it becomes evident that the resulting feature model captures and elucidates the dependencies of the 26 features of roles. In summary, the feature model can now be used to define a configuration by selecting the various features. For reasons of simplicity, we focus on two particular configurations (of over 7200 possible ones), namely the *feature minimal configuration* and the *feature complete configuration*.

5.2 Feature Minimal Metamodel

In a feature minimal configuration, only mandatory features are selected. Thus, only natural types (with structures and inheritance), which can play role types, exist. Role types, however, are merely annotations, because they only have a name and lack structure, inheritance, and relationships. Fig. 5 shows this minimal configuration of the feature model. This metamodel exhibits, beside the general definition of types with attributes and operations, a specific *Role Model* class. This class represents the default container for all role types (and possibly relationships and *constraints*) generated whenever the configuration does not

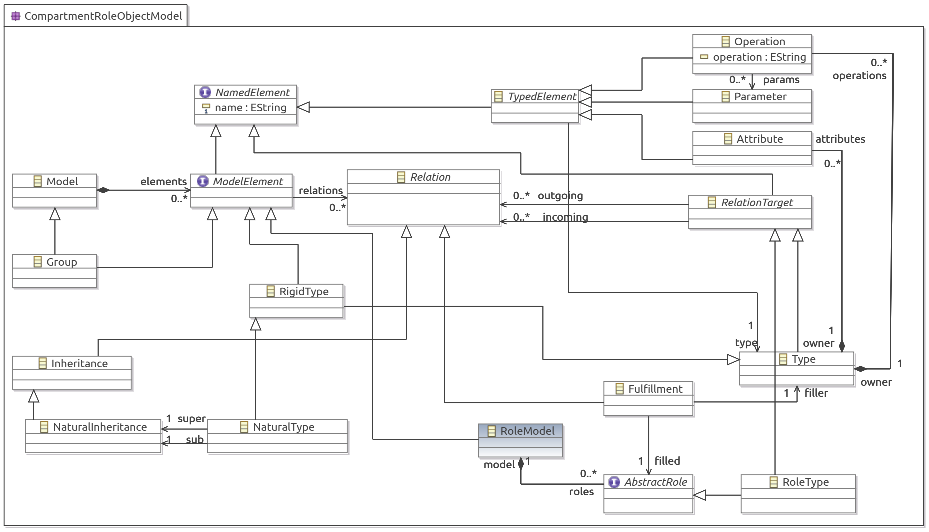


Fig. 5: Ecore metamodel of a feature minimal metamodel

include compartment types. In sum, this model is similar to a standard object-oriented metamodel with the additional ability to mark classes as role types.

5.3 Feature Complete Metamodel

In contrast to the minimal configuration, a feature complete configuration selects as many features as validly possible. However, due to the fact that a metamodel can only reflect features of the model level (M1), we omit all features solely affecting instance level (M0). As a result, the feature complete metamodel incorporates natural types, role types, relationships, and compartment types as classes. Roles can be played by naturals, other roles, and compartments.

Fig. 6 shows the corresponding metamodel highlighting all classes corresponding to the selected features. Thus, it encompasses the various relations between the various types, e.g., the fulfillment relation, the various inheritance relations as well as the different role and relationship constraints. Additionally, it includes a typical list of intra- and inter-relationship constraints, parthood constraints as well as the `RoleGroup` class for constraints on groups of roles. Thus, this metamodel represents the unification of the various features of roles proposed in the literature.

5.4 Mapping Features to Variation Points

After describing both the feature minimal (Fig. 5) and the feature complete metamodel (Fig. 6), this section describes how variants can be derived by adding, modifying and removing classes and references of the feature minimal metamodel.

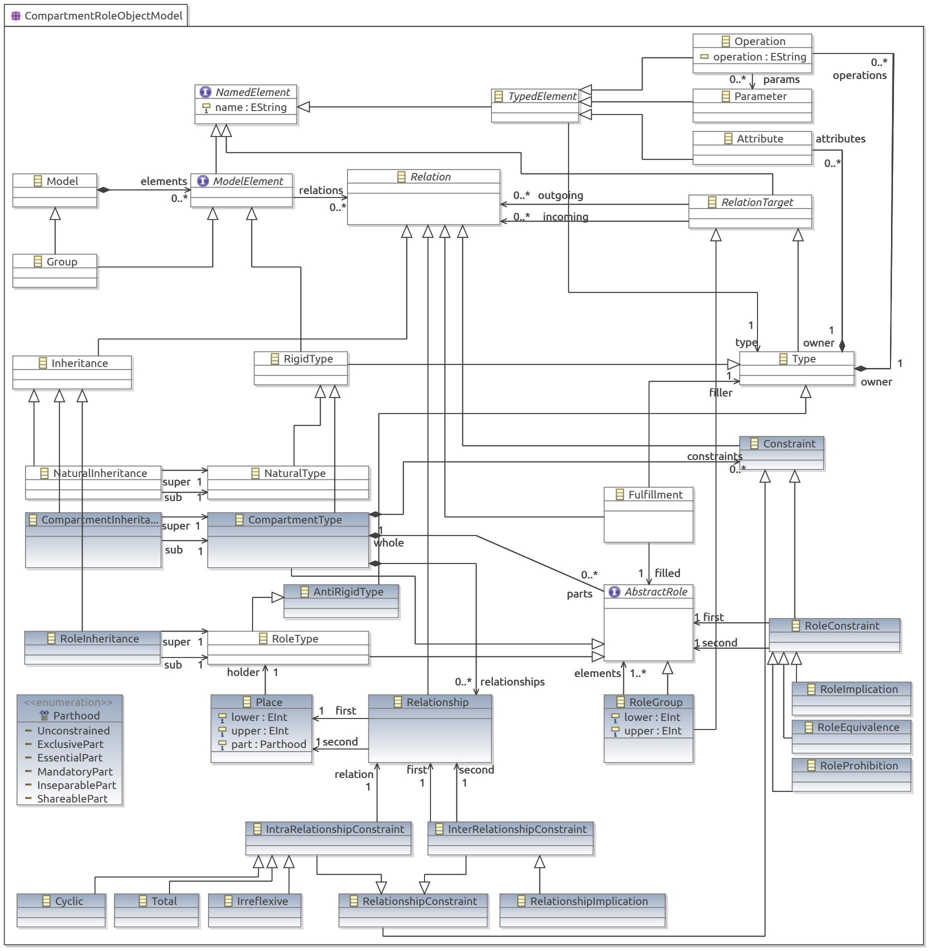


Fig. 6: Ecore metamodel of a feature complete metamodel

Henceforth, we distinguish four kinds of *variation points* of the metamodel family and specify their mapping to the corresponding features. The first kind of variation point directly corresponds to classes (highlighted in Fig. 6), i.e., their existence in the metamodel is directly linked to the selection of a specific feature. More precisely, the following classes directly correspond to features: `On Relationships` (Feature 2), `RoleConstraint` (Feature 6), `RoleInheritance` (Feature 13), `IntraRelationshipConstraint` (Feature 16), `InterRelationshipConstraint` (Feature 17), `RoleGroup` (Feature 18), and `CompartmentInheritance` (Feature 25). Thus, selecting such a feature entails adding the corresponding class together with the respective incoming and outgoing references. The only exception is the `CompartmentType` class corresponding to Feature 19, which is replaced by a `RoleModel` class if the feature is deselected. The second kind of variation points correspond to the targets of references in the metamodel. In particular, the *filler* reference of class `Fulfillment` either points to `NaturalType`, `RigidType`, `Type` or to a generic Player interface depending on the combined selection of Features 8 and 22, declaring whether compartments and/or roles can play roles as well. In contrast, the third kind of variation point changes the inheritance relation of specific classes to change their properties or implemented interfaces. Thus, `RoleType` and `CompartmentType` only inherit (indirectly) from *Type*, if Features 1 and 20 are selected, respectively, i.e., they have properties and behavior. Otherwise, they would inherit from `RelationTarget` and in the latter case also from `ModelElement`. A similar example is the inheritance from `AbstractRole`, which is only present if compartments can be nested (Feature 24). The last kind of variation points cannot be captured by standard Ecore models, because they correspond to invariants that must be satisfied by instances of that particular metamodel. This holds for the Features 3, 7, 11, and 23 which broaden the number of valid models. However, we must add one invariant for each of these features if they were not selected. Unfortunately, the specification of these invariants is beyond the scope of this paper. Altogether, these variation points are sufficient to generate each member of the metamodel family for a given configuration by iteratively transforming the feature minimal metamodel. We have developed a generator as a proof of concept[4] by employing the facilities provided by FeatureIDE and the DeltaEcore framework for delta modeling.

6 Related Work

Most of the related work on role-based modeling languages has been discussed previously. Henceforth, we focus on other surveys, metamodels, and feature models for role-based modeling languages.

To our knowledge, only one other *survey* on roles in information systems [61] has been published since the year 2000. This survey has a broader perspective and takes social-roles, modeling-roles, CSCW-roles, RBAC-roles, system-roles as well as agent-roles into account. Additionally, they classify all investigated approaches in their contexts. Nevertheless, our survey is more focused on a rigorous classification of role-based modeling and programming languages.

While several researchers tried to establish a *metamodel* for roles before, none of them was adopted by other approaches. Consider for instance, Lodwick, the formal definition of roles, proposed by Steimann [56] to consolidate the different notions of roles in conceptional modeling. This approach, despite of its generality, suffers from various limitations, e.g., the lack of representation of roles on the instance level (M0) or the consideration of context-dependent roles. Genovese [19] overcomes this by introducing contexts, roles, and players on both the model (M1) and instance level (M0). He tried to establish the most general definition of roles. In fact, his metamodel is too general to be readily applicable to

[4] See `http://st.inf.tu-dresden.de/RML` for the prototypical implementation.

any approach in the literature. This is the case, because one has to specify constraints on each level of the meta level hierarchy. Consequently, the metamodel might capture most of the features of roles. However, it is not easily adaptable to the other more limited approaches. The *Generic Role Model* [15], on the other hand, only defines the plays relation on both the model (M1) and the instance level (M0) together with formal semantics. Thus, this metamodel can be viewed as the least common denominator of the various notions of roles. In sum, the proposed metamodel family of RML ranges in between Dahchour's generic role model and Genovese's metamodel.

Last but not least, we are only aware of one other work on the nature of roles [21] providing a *feature model* for roles. This work, however, investigates roles as a language construct and collects and investigates the features of such a construct. Therefore, these features are mostly concerned with the instance level (M0) and are specific to execution environments. This feature model, however, did not consider the context-dependent nature of roles.

7 Conclusion

Roles are both relational and context-dependent by nature. However, most approaches to role-based modeling emphasize only one aspect. Unfortunately, the various approaches cannot be combined easily, due to the lack of compatible metamodels. In this paper, we have approached this problem by first conducting a literature survey and second proposing a family of metamodels for RML. For the former, we added 11 new features emphasizing the context-dependence to the preexisting features of roles [56]. This feature set has been used to classify the various RML and RPL proposed since 2000. This evaluation has shown that the research field suffers from fragmentation and discontinuity. To overcome the latter issue, we have created a feature model for RML from which a family of metamodels can be generated. In this way, researchers are now able to generate a metamodel for their specific approach and, more importantly, for other approaches they want to reuse or combine. In addition, both the listed features and the feature model can be reused to evaluate or develop subsequent role-based approaches.

Acknowledgements

This work is funded by the German Research Foundation (DFG) within the Research Training Group "Role-based Software Infrastructures for continuous-context-sensitive Systems" (GRK 1907), in the Collaborative Research Center 912 "Highly Adaptive Energy-Efficient Computing" and the European Social Fund (ESF) and the Federal State of Saxony within project "VICCI" #100098171. Special thanks go to Sebastian Richly, Tobias Jäkel, and Stephan Böhme for their suggestions and to Ulrike Schöbel for improving this paper.

References

[1] Arnaudo, E., Baldoni, M., Boella, G., Genovese, V., Grenna, R.: An implementation of roles as affordances: powerjava (Aug 31 2009)

[2] Bachman, C.W.: The programmer as navigator. Commun. ACM 16(11), 635–658 (1973)

[3] Baldoni, M., Boella, G., van der Torre, L.: powerjava: ontologically founded roles in object oriented programming languages. In: Haddad, H. (ed.) SAC. pp. 1414–1418. ACM (2006)

[4] Baldoni, M., Boella, G., Van Der Torre, L.: powerjava: ontologically founded roles in object oriented programming languages. In: Proceedings of the 2006 ACM symposium on Applied computing. pp. 1414–1418. ACM (2006)

[5] Baldoni, M., Boella, G., van der Torre, L.: Roles as a coordination construct: Introducing powerjava. Electr. Notes Theor. Comput. Sci 150(1), 9–29 (2006)

[6] Balzer, S., Burns, A., Gross, T.: Objects in context: An empirical study of object relationships. Tech. Rep. 594, ETH Zürich (May 2008)

[7] Balzer, S., Eugster, P., Gross, T.: Relations: Abstracting object collaborations (Feb 06 2008)

[8] Balzer, S., Gross, T., Eugster, P.: A relational model of object collaborations and its use in reasoning about relationships. In: Ernst, E. (ed.) ECOOP. Lecture Notes in Computer Science, vol. 4609, pp. 323–346. Springer (2007)

[9] Barbosa, F.S., Aguiar, A.: Modeling and programming with roles: introducing javastage. Tech. rep., Instituto Politécnico de Castelo Branco (2012)

[10] Bierman, G., Wren, A.: First-class relationships in an object-oriented language. In: ECOOP 2005 - Object-Oriented Programming. pp. 262–286. Springer-Verlag (2005)

[11] Boella, G., Van Der Torre, L.: The ontological properties of social roles in multi-agent systems: Definitional dependence, powers and roles playing roles. Artificial Intelligence and Law 15(3), 201–221 (2007)

[12] Bäumer, D., Riehle, D., Siberski, W., Wulf, M.: The role object pattern. In: Washington University Dept. of Computer Science (1998)

[13] Chen, P.: The entity-relationship model - toward a unified view of data. ACM Transactions on Database Systems 1(1), 9–36 (1976)

[14] Czarnecki, K., Østerbye, K., Völter, M.: Generative programming. In: Object-Oriented Technology ECOOP 2002 Workshop Reader. pp. 15–29. Springer (2002)

[15] Dahchour, M., Pirotte, A., Zimányi, E.: A generic role model for dynamic objects. In: Advanced Information Systems Engineering. pp. 643–658. Springer (2002)

[16] Dey, A.K.: Understanding and using context. Personal and ubiquitous computing 5(1), 4–7 (2001)

[17] Ferraiolo, D., Cugini, J., Kuhn, D.R.: Role-based access control (rbac): Features and motivations. In: Proceedings of 11th Annual Computer Security Application Conference. pp. 241–48 (1995)

[18] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Pearson Education (1994)

[19] Genovese, V.: A meta-model for roles: Introducing sessions. Roles' 07 p. 27 (2007)

[20] Graversen, K.B., Østerbye, K.: Implementation of a role language for object-specific dynamic separation of concerns. In: AOSD03 Workshop on Software-engineering Properties of Languages for Aspect Technologies (2003)

[21] Graversen, K.B.: The nature of roles. Ph.D. thesis, PhD thesis:/Kasper Bilsted Graversen.–Copenhagen, IT University of Copenhagen Copenhagen (2006)

[22] Grue, M.: ScalaDCI. https://github.com/DCI/scaladci (2014), [Online; accessed 24-May-2014]

[23] Guarino, N., Carrara, M., Giaretta, P.: An ontology of meta-level categories. In: KR. pp. 270–280 (1994)

[24] Guarino, N., Welty, C.: A formal ontology of properties. In: Knowledge Engineering and Knowledge Management Methods, Models, and Tools, pp. 97–112. Springer (2000)

[25] Guarino, N., Welty, C.A.: An overview of ontoclean. In: Handbook on ontologies, pp. 201–220. Springer (2009)

[26] Guizzardi, G.: Ontological foundations for structure conceptual models. Ph.D. thesis, Centre for Telematics and Information Technology, Enschede, Netherlands (2005)

[27] Guizzardi, G., Wagner, G.: Towards ontological foundations for agent modelling concepts using the unified fundational ontology (UFO). Springer (2005)

[28] Guizzardi, G., Wagner, G.: Conceptual simulation modeling with onto-uml. In: Proceedings of the Winter Simulation Conference. p. 5. Winter Simulation Conference (2012)

[29] Guizzardi, G., Wagner, G., Guarino, N., van Sinderen, M.: An ontologically well-founded profile for uml conceptual models. In: Advanced Information Systems Engineering. pp. 112–126. Springer (2004)

[30] Halpin, T.: ORM 2. In: On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops. pp. 676–687. Springer (2005)

[31] Halpin, T.A.: Object-role modeling (orm/niam). In: Handbook on Architectures of Information Systems. pp. 81–102. Springer (1998)

[32] Harel, D., Rumpe, B.: Modeling languages: Syntax, semantics and all that stuff. Tech. rep., Technische Universität Braunschweig (2004)

[33] He, C., Nie, Z., Li, B., Cao, L., He, K.: Rava: Designing a java extension with dynamic object roles. In: Engineering of Computer Based Systems, 2006. ECBS 2006. 13th Annual IEEE International Symposium and Workshop on. pp. 7–pp. IEEE (2006)

[34] Hennicker, R., Klarl, A.: Foundations for ensemble modeling–the helena approach. In: Specification, Algebra, and Software, pp. 359–381. Springer (2014)

[35] Herrmann, S.: Object teams: Improving modularity for crosscutting collaborations. In: Akşit, M., Mezini, M. (eds.) Net.Object Days 2002 (Oct 2002)

[36] Herrmann, S.: Programming with roles in ObjectTeams/Java. Tech. rep., AAAI Fall Symposium (2005)

[37] Herrmann, S.: A precise model for contextual roles: The programming language objectteams/java. Applied Ontology 2(2), 181–207 (2007)

[38] Herrmann, S.: Demystifying object schizophrenia. In: Proceedings of the 4th Workshop on MechAnisms for SPEcialization, Generalization and inHerItance. pp. 2:1–2:5. MASPEGHI '10, ACM, New York, NY, USA (2010)

[39] Herrmann, S., Hundt, C.: Objectteams/java language definition (otjld) version 1.3.1. http://www.objectteams.org/def/1.3.1 (may 2013), [Online; accessed 28-May-2014]

[40] Herrmann, S., Hundt, C., Mehner, K.: Translation polymorphism in object teams. Tech. rep., TU Berlin (2004)

[41] Hu, J., Liu, M.: Modeling context-dependent information. In: Proceedings of the 18th ACM conference on Information and knowledge management. pp. 1669–1672. ACM (2009)

[42] Kamina, T., Tamai, T.: Towards safe and flexible object adaptation. In: International Workshop on Context-Oriented Programming. p. 4. ACM (2009)

[43] Kamina, T., Tamai, T.: A smooth combination of role-based language and context activation. FOAL 2010 Proceedings p. 15 (2010)

[44] Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-oriented domain analysis (foda). Tech. rep., Software Engineering Institute, Carnegie Mellon University (1990)

[45] Kay, A.C.: The early history of smalltalk. In: HOPL Preprints. pp. 69–95 (1993)

[46] Liu, M., Hu, J.: Information networking model. In: Conceptual Modeling-ER 2009, pp. 131–144. Springer (2009)

[47] Liu, M., Hu, J.: Modeling complex relationships. In: Database and Expert Systems Applications. pp. 719–726. Springer (2009)

[48] Masolo, C., Guizzardi, G., Vieu, L., Bottazzi, E., Ferrario, R.: Relational roles and qua-individuals. In: AAAI Fall Symposium on Roles, an interdisciplinary perspective. pp. 103–112 (2005)

[49] OMG: OMG: Meta Object Facility (MOF) Core Specification. Object Managment Group, 2.4.1 edn. (June 2013), ptc/11-09-13

[50] Reenskaug, T., Coplien, J.O.: The dci architecture: A new vision of object-oriented programming. An article starting a new blog:(14pp) http://www. artima. com/articles/dci_vision. html (2009)

[51] Reenskaug, T., Coplien, J.O.: The DCI architecture: A new vision of object-oriented programming. artima developer (2011)

[52] Rumbaugh, J., Jacobson, R., Booch, G.: The Unified Modelling Language Reference Manual. Addison-Wesley, 1st edn. (Jan 1999)

[53] Rumbaugh, J.E.: Relations as semantic constructs in an object-oriented language. In: OOPSLA. pp. 466–481 (1987)

[54] S. Monpratarnchai, T.T.: The design and implementation of a role model based language, EpsilonJ. In: Proceedings of the 5th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON 2008) (2008)

[55] Seidl, C., Schaefer, I., Aßmann, U.: DeltaEcore–a model-based delta language generation framework. In: Modellierung. pp. 81–96 (2014)

[56] Steimann, F.: On the representation of roles in object-oriented and conceptual modelling. Data & Knowledge Engineering 35(1), 83–106 (2000)

[57] Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: eclipse modeling framework. Pearson Education (2008)

[58] Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: Featureide: An extensible framework for feature-oriented software development. Science of Computer Programming 79, 70–85 (2014)

[59] Wielenga, G.: On powerjava: "roles" instead of "objects". https://blogs.oracle.com/geertjan/entry/on_powerjava_roles_instead_of (jan 2013), [Online; accessed 28-May-2014]

[60] Zhu, H., Zhou, M.: Role-based collaboration and its kernel mechanisms. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 36(4), 578–589 (Jul 2006)

[61] Zhu, H., Zhou, M.: Roles in information systems: A survey. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 38(3), 377–396 (2008)