



**TECHNISCHE
UNIVERSITÄT
DRESDEN**

Faculty of Computer Science Institute of Software and Multimedia Technology

Software Technology Group

A FAMILY OF ROLE-BASED LANGUAGES

Thomas Kühn

Born on: 11.09.1985 in Karl-Marx-Stadt (now Chemnitz)

DISSERTATION

to achieve the academic degree

DOKTOR-INGENIEUR (DR.-ING.)

Referee

Prof. Dr. Colin Atkinson

Supervising professors

Prof. Dr. Uwe Aßmann

Prof. Dr.-Ing. Wolfgang Lehner

Submitted on: 3.3.2017

ABSTRACT

Role-based modeling has been proposed in 1977 by Charles W. Bachman [Bachman et al., 1977], 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 via conceptual modeling through to programming languages [Steimann, 2000a]. More importantly, because current software systems are characterized by increased complexity and context-dependence [Murer et al., 2008], there is a strong demand for new concepts beyond object-oriented design. Although mainstream modeling languages, i.e., Entity-Relationship Model, Unified Modeling Language, are good at capturing a system's structure, they lack ways to model the system's behavior, as it dynamically emerges through collaborating objects [Reenskaug and Coplien, 2009]. In turn, roles are a natural concept capturing the behavior of participants in a collaboration. Moreover, roles permit the specification of interactions independent from the interacting objects. Similarly, more recent approaches use roles to capture context-dependent properties of objects. The notion of roles can help to tame the increased complexity and context-dependence. Despite all that, these years of research had almost no influence on current software development practice. To make things worse, until now there is no common understanding of roles in the research community and no approach fully incorporates both the context-dependent and the relational nature of roles [Kühn et al., 2014]. In this thesis, I will devise a formal model for a *family of role-based modeling languages* to capture the various notions of roles [Kühn et al., 2015a]. Together with a software product line of *Role Modeling Editors*, this, in turn, enables the generation of a *role-based language family* for Role-based Software Infrastructures (RoSI).

1 INTRODUCTION

“Modeling is one of the most fundamental processes of the human mind.”

– [Rothenberg et al., 1989]

In other words, Jeff Rothenberg reminds us that, modeling is the basic ability of abstracting aspects of reality to better comprehend and reason about certain aspects of reality avoiding its *complexity*, *danger* and *irreversibility* [Rothenberg et al., 1989]. In detail, he characterized *modeling* as activity “to represent a particular referent cost-effectively for a particular cognitive purpose”. This is particularly true for *conceptual modeling*, which is “the activity of formally describing some aspects of the physical and social world around us for purposes of understanding and communication.” [Mylopoulos, 1992]. By extension, a conceptual model is a formal description of parts of a subject domain by means of concepts and their interrelations. In contrast to other models, such as street maps or floor plans, conceptual models are not only required to be understood by humans, but also by computers. Thus, this enables their use not only for communication and problem-solving, but also for formal validation and artifact generation. Although most would assume that classical conceptual modeling languages, such as the Entity-Relationship Model (ER) [Chen, 1976] or the Unified Modeling Language (UML) [Rumbaugh et al., 1999], are appropriate conceptual modeling languages, several researchers, e.g., [Steimann, 2000c, Atkinson and Kühne, 2002, Guizzardi et al., 2004, Liu and Hu, 2009], have pointed out their deficiencies when used to model more complex, context-dependent, and dynamic domains. This, in turn, makes these conceptual modeling languages inappropriate for such domains, and – in the words of Jeff Rothenberg – their use “can do considerable harm” [Rothenberg et al., 1989].

The PhD thesis – A Family of Role-Based Languages – contributes to the field of conceptual modeling by investigating role-based modeling languages (RMLs) as promising approach to more appropriately model nowadays complex, context-dependent and dynamic domains.

Both role-based modeling languages (RMLs) and role-based programming languages (RPLs) have been investigated for several decades.¹ The first account for the application of roles to modeling dates back to 1977, when Bachman and Daya proposed the *Role Data Model* [Bachman et al., 1977]. They facilitate roles as “a defined behavior pattern which may be assumed by entities of different kind” [Bachman et al., 1977, p.45] to handle different entity types playing the same role type coherently [Bachman et al., 1977]. Till the year 2000, the term “role” has occurred in multiple areas within computer science, e.g., access control,² knowledge representation, conceptual modeling, data modeling, as well as object-oriented design and implementation [Steimann, 2000a]. However, after Steimann surveyed the contemporary literature on roles [Steimann, 2000b], he correctly observes that by 2000 “the influence of the role data model on modelling has at best been modest” [Steimann, 2000b, p.85]. While Steimann is right when he claims that roles as behavioral pattern were not adopted in modeling, most modeling languages feature roles as named places at the end of relationships, e.g., ER [Chen, 1976] and UML [Rumbaugh et al., 1999]. Thus, while roles are well-established in modeling languages their semantics differ and their full potential is rarely utilized. Since Steimann’s survey, more elaborate applications of the role concept have been proposed ranging from knowledge representation [Loebe, 2005, Guarino and Welty, 2009], via data modeling [Halpin, 2005, Liu and Hu, 2009, Jäkel et al., 2015], and conceptual modeling [Guizzardi and Wagner, 2012, Hennicker and Klarl, 2014] through to object-oriented design and implementation [Baldoni et al., 2006, Herrmann, 2005, Balzer et al., 2007]. Although these approaches employ

¹By the 8th of October 2017, it will be exactly 40 years.

²Throughout this thesis we consider Role-Based Access Control (RBAC) [Ferraiolo et al., 1995] as a special application for roles with a rather narrow scope.

roles to model, reason, and implement context-dependent behavior of objects to cope with the requirements of mobile and pervasive applications [Herrmann, 2005, Liu and Hu, 2009], their proposals had almost no impact on mainstream modeling and programming languages. Reenskaug and Coplien perfectly put this into perspective, when they emphasize that “*Object-oriented programming languages traditionally afford no way to capture collaborations between objects*” [Reenskaug and Coplien, 2009] and that “*roles [could] capture collections of behaviors that are about what objects do*” [Reenskaug and Coplien, 2009]. In other words, the past years of research on roles had next to no influence on current software development practice, in spite of clear evidence that roles can tame the increased complexity and context-dependence of current context-adaptive, distributed software systems.

From the previous discussion, one could argue that the introduction of roles failed due to the insufficiency of the role concept. While it is true that no big case study has shown the sufficiency of role-based modeling and programming, it does not necessarily follow that the lack of adoption in practice is a result of its insufficiency. In my opinion, there are four basic reasons why roles have not been adopted by more researchers and practitioners. First, until now there is no common understanding of roles in the research community [Steimann, 2000b, Kühn et al., 2014]. Instead, each approach focuses on certain features attributed to roles. Second, the research field itself suffers from *discontinuity* and *fragmentation* of the various role definitions [Kühn et al., 2014]. In particular, most approaches *did not* take previous results into account and *did not* continuously improve role-based modeling. Third, there are only few approaches, e.g., [Genovese, 2007, Zhu and Zhou, 2006, Hennicker and Klarl, 2014], that provide a formal foundation for their RML incorporating most of the features of roles [Steimann, 2000b, Kühn et al., 2014]. Last but not least, most role-based modeling and programming languages are not readily applicable, because of their complexity, level of abstraction, and/or missing tool support. These issues not only hinder researchers improving previous approaches, but also software practitioners exploring new modeling and programming languages. The first two issues can be tackled by developing a family of role-based modeling languages by means of the features of roles. However, the third and fourth issue must be addressed by providing a comprehensive formal foundation for roles, a role-based modeling language incorporating most features of roles, and, finally, readily applicable tools that support its use for the design of future role-based software systems.

To achieve these goals, the PhD thesis makes the following contributions to the research areas on role-based modeling and programming languages:

1. It extends the initial list of features of roles by adding 12 new features retrieved from contemporary approaches, discussed in Chapter 2 of the thesis.
2. Based on these features a Systematic Literature Review (SLR) [Kitchenham, 2004] was conducted to survey the contemporary literature on RMLs (Chapter 4) and RPLs (Chapter 5).
3. Next, it establishes the foundations of the RMLs (Chapter 7) by introducing the *Compartment Role Object Model (CROM)*, a framework for conceptual modeling that incorporates most of the features of roles into a coherent, graphical modeling language and provides a set-based formalization of roles.
4. Additionally, a corresponding graphical modeling editor, called *Full-fledged Role Modeling Editor (FRaMED)*, is presented that allows the creation, manipulation and provisioning of *CROM* models.
5. To approach the *discontinuity* and *fragmentation* of the research area on roles, this PhD thesis proposes a metamodeling approach for the creation of a family of RMLs (Chapter 8).

Henceforth, this summary highlights key contributions of the thesis.

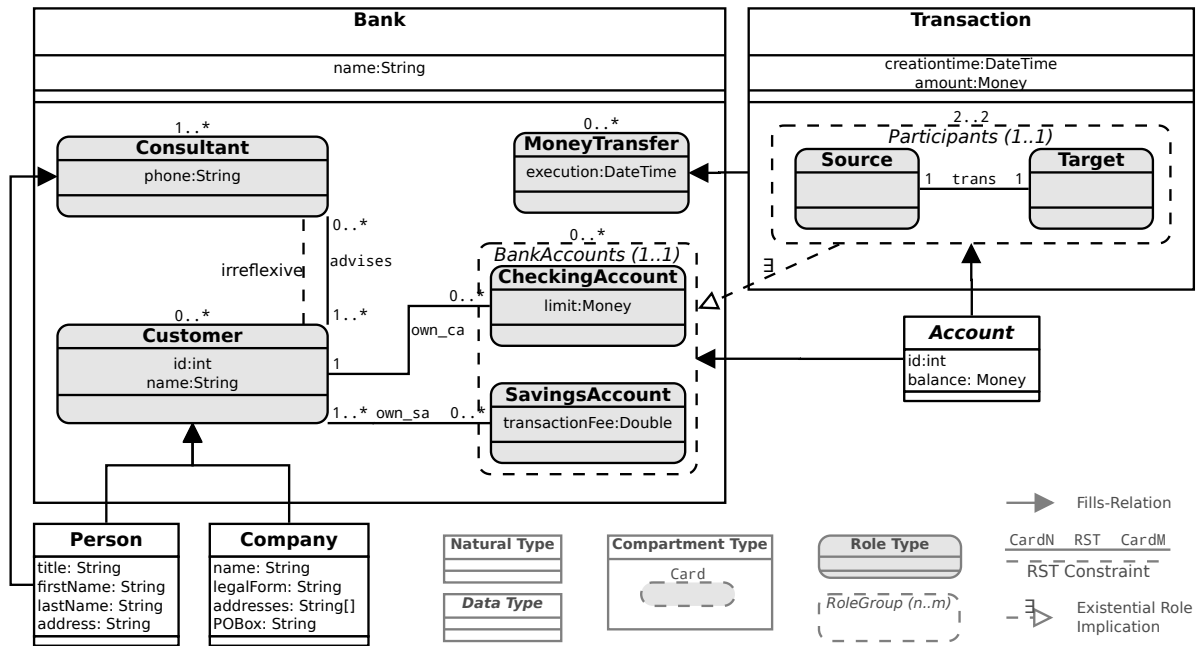


Figure 1: Role model of the banking application with additional constraints.

2 BACKGROUND

The notion of roles is very old. Yet, there is still no common understanding of roles in the literature [Steimann, 2000b, Kühn et al., 2014]. On the contrary, [Steimann, 2000b], [Kühn et al., 2014], and this PhD thesis identified 27 features attributed to roles that are grouped into the three natures of roles, e.g., the behavioral, the relational, and the context-dependent nature of roles.

2.1 RUNNING EXAMPLE

Before briefly introducing the natures of roles, Figure 1 introduces the example role model of a banking application that is used throughout the PhD thesis and this document. It describes a Bank as a compartment managing Customers, who own CheckingAccounts and SavingsAccounts. They can be advised by one or more Consultants. However, the advises relationship is constrained to be irreflexive, to prohibit self advising consultants. The Transaction compartment is specified to orchestrate the transfer of money between exactly two Accounts by means of the roles Source and Target. Moreover, a unique Target counterpart for each Source has to exist. This is ensured by the one-to-one cardinality of the trans relation. Additionally, the role group with 1..1 cardinality enforces that one account cannot be Source and Target in the same Transaction. Finally, Persons can play the roles Consultant and Customer; Companies only Customer; and Accounts the roles CheckingAccount, SavingsAccount, Source, and Target. Last but not least, each Account referenced in a transaction must be a valid account in a Bank. In sum, this scenario is small enough to serve as a comprehensive running example, however, it still imposes several design challenges for application designers. These challenges will be addressed in the following sections, to motivate the different natures of roles.

2.2 NATURE OF ROLES

The **behavioral nature** establishes that unrelated objects can play roles and roles adapt the behavior of playing objects [Steimann, 2000b, Kühn et al., 2014]. Additionally, objects can play roles of a different type multiple times. Consider, for instance, the role of a *customer* of a bank that can be played by either a *person* or a *company* and allows them to make transactions, deposits, and withdrawals. Obviously, one *person* can be a *customer* in several banks. This nature is usually captured by the *fills* relation between classes and role types denoting those classes whose objects can play roles of the given type. In contrast, the **relational nature** states that roles denote the binding ends of relationships. This nature is present in most modeling languages, e.g. ER [Chen, 1976] and UML [Rumbaugh et al., 1999]. Still, these languages do not foster the dynamism and flexibility of roles, as roles degenerate to named placeholders. Hence, [Bachman et al., 1977, Steimann, 2000b, Halpin, 2005, Balzer and Gross, 2011, Jäkel et al., 2015] introduced roles tied to relationships as first-class citizens permitting them to be played by unrelated objects and having relationship specific properties. To model that *consultants advise customers*, one can specify a relationship type *advises* between the *consultant* and *customer* role types and add relationship specific fields. However, all these modeling languages assume that relationships are context-independent and cannot play roles themselves. Thus, *transactions* between *accounts* managed by a *bank* cannot be modeled properly. To resolve this, RMLs have incorporated the **context-dependent nature** of roles, e.g., [Genovese, 2007, Reenskaug and Coplien, 2009, Hennicker and Klarl, 2014, Jäkel et al., 2016], that characterizes roles and relationships as context-dependent. Both are encapsulated in a *context* as definitional boundary. Yet, different approaches use different terms for this conceptual entity. Consequently, *compartments* were introduced in [Kühn et al., 2014] to generalize the different terms. Compartments can have properties and behavior, as well as play roles themselves. In accordance, a *transaction* for the transferal of money from a *source* to a *target* account would be modeled as compartment type and would play the role of a *money transfer* within a *bank* compartment. In contrast to context-dependent roles, only few approaches also include context-dependent relationships, e.g., [Hennicker and Klarl, 2014, Jäkel et al., 2016]. Including these natures into one RML already leads to a rich modeling framework, yet its expressiveness is determined by the available kinds of **constraints**. These range from classical *relationship cardinalities*, e.g., [Chen, 1976, Rumbaugh et al., 1999], over mathematical *relationship constraints*, e.g., [Balzer and Gross, 2011, Halpin, 2005], to *role constraints*, e.g., [Riehle and Gross, 1998; zhu2006; Hennicker and Klarl, 2014]. Within a *bank* compartment type, for instance, one could specify that each *consultant* advises at least one *customer* with relationship cardinalities, that no *consultant* advises himself as a *customer* with an intra-relationship constraint, and that each *bank* must have at least one *consultant* [Kühn et al., 2015a]. Although various RMLs have introduced different kinds of constraints, only CROM [Kühn et al., 2015a] includes most of them into a coherent model. Consequently, *FRaMED* is founded on its notation and definitions.

2.3 CLASSIFICATION OF ROLES

The classification proposed by Steimann [2000b] encompasses a list of 15 features attributed to roles. This list, enumerated in Table 1, mostly captures the behavioral and relational nature of roles. In fact, only Feature 2 denotes that “*roles depend on relationships*” [Steimann, 2000b, p.86]. Hence, all the other features contribute to the behavioral nature of roles. In general, Steimann’s classification has proven to be useful and has been employed to classify several contemporary approaches, e.g., [Herrmann, 2007, Boella and Van Der Torre, 2007]. Nevertheless, his classification scheme has two major shortcomings. First, several features concern either the type and/or the instance level.

Table 1: Steimann’s 15 classifying features, extracted from [Steimann, 2000b].

1. Roles have properties and behaviors	(M1, M0)
2. Roles depend on relationships	(M1, M0)
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)

For instance, Features 4, 5, 9, 10, 12, 14, and 15 only apply to role instances [Kühn et al., 2014]. Especially, Feature 5 and 12 are usually not applicable to modeling languages, as they do not provide an operational semantics. To indicate the affected level, Table 1 appends *M1* and *M0* to each feature denoting whether the type or the instance level is affected. Finally, it cannot fully distinguish contemporary RMLs and RPLs, as it does not capture the *context-dependent nature of roles* and the various *modeling constraints*. In conclusion, the following paragraph introduces the additional characteristics of roles found by investigating the contemporary literature.

In accordance to Steimann’s pragmatic approach, the investigation of contemporary RMLs and RPLs, published in [Kühn et al., 2014, Sec.3.2] has uncovered the following features of roles. As this thesis studies the representation of roles and role models in role-based languages, we focused on the type level representation of roles rather than their implementation. In detail, these additional features, summarized in Table 2, not only incorporate the *context-dependent nature of roles*, but also the various constraints found in contemporary RMLs and RPLs. Accordingly, Steimann’s initial list encompasses both the behavioral and relational nature of roles, the additional features extend the relational nature and add the context-dependent nature to roles. Consequently, while Steimann’s 15 features are still applicable, they need to be accompanied by 12 additional features to sufficiently assess the diversity of the contemporary role-based languages. As a result, this list of 27 characteristic features allows for a fine-grained distinction of the various definitions of roles presented in the contemporary literature. Henceforth, this classification scheme is employed to review, compare and classify the various contemporary RMLs and RPLs.

3 COMPARISON OF ROLE-BASED LANGUAGES

The PhD thesis conducted a SLR that uncovered, evaluated and compared 27 relevant contemporary role-based languages. This section only briefly elucidates the findings and results of the conducted literature review. In general, as shown in Table 3 and 4, the survey provides evidence that both the research field on role-based modeling and role-based programming languages suffer

Table 2: Additional classifying features, partially published in [Kühn et al., 2014].

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)
27. The number of roles occurring in a compartment can be constrained	(M1)

from fragmentation and discontinuity [Kühn et al., 2014]. Especially, as most approaches reinvent the notion of roles without taking previous definitions into account. Although this might be the result of negligence, I argue that this is due to a lack of a common understanding of roles among researchers. In fact, researchers attribute different features to roles that fit their particular use case or application domain without being aware of the relations to features of related works. This becomes evident when answering the research questions.

First, is there a common subset of features all contemporary approaches satisfy? Yes, but it only consists of Feature 1 stating that roles have properties and behaviors, as well as Feature 3 declaring that objects can play multiple roles simultaneously. Besides these two, the investigation of the rows of both tables indicate that neither the set of RMLs nor the set of RPLs share a significant amount of common features. This, in turn, is surprising due to the fact that Steimann [2000b] already established such a set by defining *Lodwick* as the lowest common denominator. *Second, how did Steimann's seminal work influenced the research field?* In fact, his work had only a limited influence on contemporary RMLs and RPLs. In particular, only few role-based languages actually applied his classification scheme [Kim et al., 2003, Herrmann, 2005, Zhu and Zhou, 2006, Boella and Van Der Torre, 2007, Pradel and Odersky, 2009]. Nonetheless, none of the role-based languages used or extended *Lodwick's* definition of roles. In sum, less than half (11 of 26) of the approaches referenced [Steimann, 2000b] as related work. Evidently, his work did not harmonize and foster the research on role-based languages as he intended [Steimann, 2000b]. *Finally, have advances in RMLs been adopted by later RPLs and vice versa?* In the same way as *Lodwick* was overlooked, most RMLs only considered other modeling languages as related work. Conversely, most RPLs relate themselves to other programming languages, but, at least *ObjectTeams/Java (OTJ)* [Herrmann, 2005], *powerJava* [Boella and Van Der Torre, 2007], and *Scala Roles* [Pradel and Odersky, 2009] founded their notion of roles on the conceptual framework provided by Steimann [2000b]. In consequence, the SLR uncovered the following problems in the research fields on RMLs and RPLs:

- There is neither a *common understanding* nor *common feature set* shared among the different contemporary role-based modeling and programming languages.
- The research fields on RMLs and RPLs are characterized by an ongoing *discontinuity* and *fragmentation*. Specifically, most approaches reinvent the role concept without taking the definitions of preceding related approaches into account.

Table 3: Comparison of role-based modeling languages, extended from [Kühn et al., 2014]

Features [Kühn et al., 2014]	Lodwick [Steimann, 2000b]	Generic Role Model [Dahchour et al., 2002]	TAO [Da Silva et al., 2003]	RBML [Kim et al., 2003]	Role-Based Patterns [Kim and Carrington, 2004]	ORM 2 [Halpin, 2005]	E-CARGO [Zhu and Zhou, 2006]	Metamodel for Roles [Genovese, 2007]	INM [Liu and Hu, 2009]	DCI [Reenskaug and Coplien, 2009]	OntoUML [Guizzardi and Wagner, 2012]	Helena Approach [Hennicker and Klarl, 2014]
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	□	□	■	□	□	■	■	⊞	■	■	□	■
27	□	□	□	■	■	□	■	⊞	□	□	⊞	■

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

Table 4: Comparison of role-based programming languages, extended from [Kühn et al., 2014]

Features [Kühn et al., 2014]	EpsilonJ [Ubayashi and Tamai, 2001]	Chameleon [Graversen and Østerbye, 2003]	RICA-J [Serrano and Ossowski, 2004]	JAWIRO [Selçuk and Erdoğan, 2004]	OT/J [Herrmann, 2005]	Rava [He et al., 2006]	powerJava [Baldoni et al., 2006]	Rumer [Balzer et al., 2007]	First-Class Relationships [Nelson et al., 2008]	Scala Roles [Pradel and Odersky, 2009]	NextEJ [Kamina and Tamai, 2009]	JavaStage [Barbosa and Aguiar, 2012]	Relations [Harkes and Visser, 2014]
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	■	□	■	□	■	□	■	■	■	■	■	□	■
27	⊞	□	□	□	⊞	□	□	□	□	□	⊞	□	■

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

Table 5: Ontological classification of concepts

Concept	Rigidity	Foundedness	Identity	Examples
Natural Types	yes	no	unique	<i>person, account</i>
Role Types	no	yes	derived	<i>customer, consultant</i>
Compartment Types	yes	yes	unique	<i>transaction, bank</i>
Relationship Types	yes	yes	composite	<i>advises, own_ca</i>

- Only four RMLs provide a sufficient *formal foundation* for roles able to incorporate *all natures of roles*, i.e. [Da Silva et al., 2003, Genovese, 2007, Liu and Hu, 2009, Hennicker and Klarl, 2014]. Regardless, none of them is able to *support all features of roles*.
- Last but not least, most role-based modeling and programming languages are *not readily applicable*, due to their complexity, ambiguous terminology, and/or missing tool support. Even though *OT/J* represents a feature rich, practically usable programming language, there is no corresponding *readily applicable modeling language*.

To approach these problems, the second part of the PhD thesis aims at harmonizing both research fields by providing the formal foundations of combined role-based modeling languages, as well as a family of RMLs supported by a flexible modeling editor.

4 FOUNDATIONS OF ROLE-BASED MODELING LANGUAGES

To provide a coherent understanding of roles, the PhD thesis provides both the ontological foundation and a comprehensive formal model for roles, denoted *Compartment Role Object Model (CROM)*, that combines all natures of roles as well as various modeling constraints [Kühn et al., 2015a,b]. Moreover, the lack of tools that support the design, validation, and generation of role-based software systems is addressed by developing a corresponding *Full-fledged Role Modeling Editor* [Kühn et al., 2016], a fully functional modeling editor that includes all natures and proposed modeling constraints. In conclusion, both *CROM* and *FRaMED* provide all means necessary to allow both researchers and practitioners to model, reason about, and implement role-based software systems.

Moreover, to provide a clear ontological distinction for the various concepts, the PhD thesis utilized three well-established ontological metaproperties: *rigidity*, *identity* and *foundedness* (dependence) [Guarino and Welty, 2000, 2009, Guizzardi, 2005, Mizoguchi et al., 2012]. Accordingly, these three metaproperties are sufficient to distinguish the concepts found in RMLs. Table 5 summarizes the ontological classification of *natural types*, *role types*, *compartment types*, and *relationship types* with respect to the introduced metaproperties.

Henceforth, the initial formalization of the *Compartment Role Object Model (CROM)* is highlighted that combines the behavioral, relational, and context-dependent nature of roles into a comprehensive and coherent formal framework [Kühn et al., 2015a]. Additionally, this formalization incorporates most of the modeling constraints found in the literature as well as *global role constraints* as additional class of role constraints. By extension, the framework is easy to comprehend and implement as it is only based on *set theory* and *first-order logic*. In sum, the formal framework is separated into three parts, as outlined in Figure 2. For simplicity, both *fields* and *methods* have been omitted from the following definitions, however, the necessary additions are presented in Section 7.4 of the PhD thesis. Henceforth, the discussion only highlights the core aspects of a modeling language that are captured on the model and constraint level.

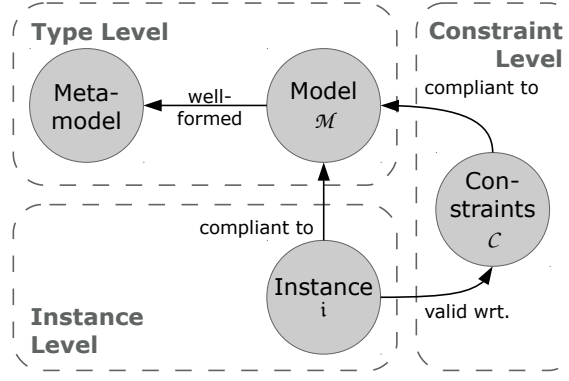


Figure 2: Overview of the presented formal model

Definition 4.1 (Compartment Role Object Model). *Let NT , RT , CT , and RST be mutual disjoint sets of natural types, role types, compartment types, and relationship types, respectively. Then a Compartment Role Object Model (CROM) is a tuple $\mathcal{M} = (NT, RT, CT, RST, fills, rel)$ where $fills \subseteq T \times CT \times RT$ is a relation and $rel: RST \times CT \rightarrow (RT \times RT)$ is a partial function. Here, $T := NT \cup CT$ denotes the set of all rigid types, i.e. all natural and compartment types.*

A CROM is denoted well-formed if the following axioms hold:

$$\forall rt \in RT \exists! ct \in CT \exists t \in T: (t, ct, rt) \in fills \quad (7.1)$$

$$\forall ct \in CT \exists (t, ct, rt) \in fills \quad (7.2)$$

$$\forall rst \in RST \exists ct \in CT: (rst, ct) \in \mathbf{domain}(rel) \quad (7.3)$$

$$\forall (rt_1, rt_2) \in \mathbf{codomain}(rel): rt_1 \neq rt_2 \quad (7.4)$$

$$\forall (rst, ct) \in \mathbf{domain}(rel): rel(rst, ct) = (rt_1, rt_2) \wedge (_, ct, rt_1), (_, ct, rt_2) \in fills \quad (7.5)$$

In this definition, $fills$ denotes that rigid types can play roles of a certain role type in a given compartment type and rel captures the two role types at the respective ends of a relationship type defined in a compartment type.³ Accordingly, the well-formedness rules restrict both the $fills$ relation and the rel function. On the one hand, the first two axioms ensure that each role type participates in exactly one compartment type and is filled by at least one rigid type (7.1) as well as that each compartment type defines at least one participating role type (7.2). On the other hand, the other axioms make sure that each relationship type is defined at least in on compartment type (7.3). Moreover, the rel function is restricted to an irreflexive codomain (7.3), such that the two related role types participate in the same compartment type the relationship is defined in (7.5). Notably though, although a relationship type can occur in multiple compartment types with different definitions, each role type belongs to exactly one compartment type. Using this definition, a formal model of our running example can be created, as follows:

³For a given function $f: A \rightarrow B$, $\mathbf{domain}(f) = A$ returns the domain and $\mathbf{codomain}(f) = B$ the range of f .

Example 4.1 (Compartment Role Object Model). Let $\mathcal{B} = (NT, RT, CT, RST, fills, rel)$ be the model of the bank (Figure 1), where the individual components are defined as follows:

$$\begin{aligned}
NT &:= \{Person, Company, Account\} \\
RT &:= \{Customer, Consultant, CA, SA, Source, Target, MoneyTransfer\} \\
CT &:= \{Bank, Transaction\} \\
RST &:= \{own_ca, own_sa, advises, trans\} \\
fills &:= \{(Person, Bank, Customer), (Company, Bank, Customer), (Bank, Bank, Customer), \\
&\quad (Person, Bank, Consultant), (Account, Bank, CA), (Account, Bank, SA), \\
&\quad (Transaction, Bank, MoneyTransfer), \\
&\quad (Account, Transaction, Source), (Account, Transaction, Target)\} \\
rel &:= \{(own_ca, Bank) \rightarrow (Customer, CA), (own_sa, Bank) \rightarrow (Customer, SA), \\
&\quad (advises, Bank) \rightarrow (Consultant, Customer), (trans, Transaction) \rightarrow (Source, Target)\}
\end{aligned}$$

The bank model \mathcal{B} is simply created from Figure 1 in three steps. First, all the natural types, compartment types, role types, and relationship types are collected into the corresponding set.⁴ Second, the set of role types contained in each compartment type and the corresponding player types are collected in the *fills* relation. Finally, the *rel* function is defined for the role types at the ends of each relationship type in each compartment type, accordingly. Thus, a CROM model can be retrieved from its graphical representation.

The constraint level, in turn, augments the formal model to represent the various constraints found in the literature review. In detail, the PhD thesis presents *Role Groups* [Kühn et al., 2015a] as a novel construct to specify local role constraints and *Quantified Role Groups* as a corresponding global role constraint. In sum, a constraint model defines *local role constraint*, *intra-* and *inter-relationship constraints* declared for a specific compartment type, as well as *global role constraints* declared for a whole CROM.

Definition 4.2 (Constraint Model). Let $\mathcal{M} = (NT, RT, CT, RST, fills, rel)$ be a well-formed CROM and $IRC := \{\triangleleft, \otimes\}$ the set of inter-relationship constraints. Then $\mathcal{C} = (rolec, card, intra, inter, grolec)$ is a Constraint Model over \mathcal{M} , where $rolec: CT \rightarrow 2^{Card \times RG}$, and $card: RST \times CT \rightarrow (Card \times Card)$ are partial functions, as well as $intra \subseteq RST \times CT \times \mathbb{E}$ and $inter \subseteq RST \times CT \times IRC \times RST$ are relations with \mathbb{E} as the set of functions $e: 2^O \times 2^O \times 2^{O \times O} \rightarrow \{0, 1\}$ ranging over the set of objects O . Additionally, $grolec \subseteq QRG$ is a finite set of quantified role groups. A Constraint Model is compliant to the CROM \mathcal{M} if the following axioms hold:

$$\forall ct \in \mathbf{domain}(rolec) \forall (_, a) \in rolec(ct): atoms(a) \subseteq parts(ct) \quad (7.10)$$

$$\mathbf{domain}(card) \subseteq \mathbf{domain}(rel) \quad (7.11)$$

$$\forall (rst, ct, _) \in intra: (rst, ct) \in \mathbf{domain}(rel) \quad (7.12)$$

$$\forall (rst_1, ct, _, rst_2) \in inter: (rst_1, ct), (rst_2, ct) \in \mathbf{domain}(rel) \wedge rst_1 \neq rst_2 \quad (7.13)$$

Here, $parts(ct) := \{rt \mid (t, ct, rt) \in fills\}$ collects all role types defined within a compartment type.

Specifically, *rolec* collects all local role constraints imposed on specific compartment types. Each local role constraint, in turn, defines both a cardinality and a role group, such that the cardinality specifies the occurrence of objects satisfying the given role group. In accordance to that, *card* assigns a cardinality to a relationship type defined in a compartment type. Additionally, *intra* defines

⁴Henceforth, SA and CA are abbreviations for *savings account* and *checking account*, respectively.

a set of *intra-relationship constraint* imposed on a relationship type in a compartment type, such that each constraint is given as an evaluation function. This function takes the domain A , range B , and the tuple set $\mathbb{R} \subseteq A \times B$ of a relationship and returns either zero or one. For instance, to define that the *advises* relationship type is *irreflexive*, a corresponding evaluation function returns one if $\forall x \in A \cup B: (x, x) \notin \mathbb{R}$ and zero otherwise. Of course, these evaluation functions directly correspond to mathematical properties of relations, e.g., reflexive, total, cyclic, and acyclic. In the same way, *inter* denotes a set of *relationship implications* and *exclusions* declared between two relationship types defined in the same compartment type. In particular, a *relationship implication* imposes a subset relation upon the two relationships, whereas a *relationship exclusion* enforces their disjointness. Notably, all these constraints are defined locally to a compartment type, i.e., no constraint crosses the boundary of a compartment type. Finally, *grolec* declares a set of quantified role groups that each object in a given Compartment Role Object Instance (CROI) must satisfy. In conclusion, the constraint model captures not only most modeling constraints introduced in RMLs, but also *global role constraints* as novel kind of role constraint. Naturally, a *constraint model* can be easily defined for the constraints of the banking application, depicted in Figure 1.

Example 4.2 (Constraint Model). *Let \mathcal{B} be the bank model from Example 4.1 and irreflexive an evaluation function for relationships. Then $\mathcal{C}_{\mathcal{B}} = (\text{rolec}, \text{card}, \text{intra}, \text{inter}, \text{grolec})$ is the constraint model, where the components are defined as:*

$$\begin{aligned} \text{rolec} &:= \{\text{Bank} \rightarrow \{(1..∞, \text{Consultant}), (0..∞, \text{bankaccounts})\}, \\ &\quad (\text{Transaction} \rightarrow \{(2..2, \text{participants})\})\} \\ \text{card} &:= \{(\text{own_ca}, \text{Bank}) \rightarrow (1..1, 0..∞), (\text{own_sa}, \text{Bank}) \rightarrow (1..∞, 0..∞), \\ &\quad (\text{advises}, \text{Bank}) \rightarrow (0..∞, 1..∞), (\text{trans}, \text{Transaction}) \rightarrow (1..1, 1..1)\} \\ \text{intra} &:= \{(\text{advises}, \text{Bank}, \text{irreflexive})\} \\ \text{inter} &:= \{(\text{own_ca}, \text{Bank}, \otimes, \text{own_sa})\} \\ \text{grolec} &:= \{\text{accountimpl}\} \end{aligned}$$

A constraint model can be obtained by basically mapping the graphical constraints to their formal counterparts. Within each compartment type, role groups with cardinalities are added to the *rolec* mapping, relationship cardinality to the *card* function, intra-relationship constraints to the *intra* relation, and relationship implications/exclusions to the *inter* relation. Afterwards, the global role constraints are translated to quantified role groups, e.g., the *accountimpl*, and added to *rolec*. For the sake of argument, the example additionally introduces a relationship exclusion between *own_ca* and *own_sa* to the constraint model. Using this definition, the notion of validity is introduced for CROIs with respect to a given *Constraint Model* to specifying when a given instance fulfills the imposed constraints.

4.1 FULL-FLEDGED ROLE MODELING EDITOR

Conversely, this section highlights the first *Full-fledged Role Modeling Editor (FRaMED)*, a graphical modeling editor for the design of *Compartment Role Object Models*. Introduced in [Kühn et al., 2016], the editor embraces all natures of roles as well as most of the presented modeling constraints. Besides providing a full-fledged modeling editor, FRaMED also features distinction code generators generating formal representations of CROM, data definitions for a role-based database or source code of role-based programming languages. Consequently, FRaMED is able to provide all means necessary to allow practitioners to model, reason about, and implement role-based systems.

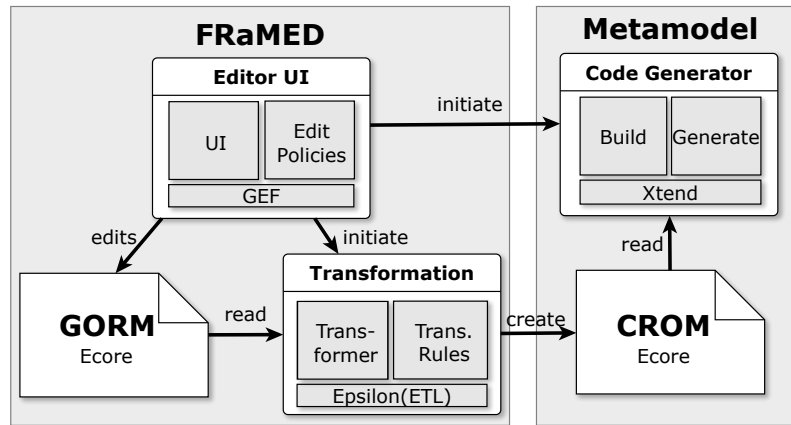


Figure 3: Architecture of FRaMED, extracted from tep@kuehn2016framed.

In general, *FRaMED* is built on the *Eclipse* platform⁵ and is available on *GitHub*.⁶ The editor was implemented using a model-driven approach and employs the *Eclipse Modeling Framework (EMF)*⁷ and most of the following discussion has been published in [Kühn et al., 2016]. Figure 3 provides an overview of its software architecture.

First, both the formalization of CROMs and constraint models have been encoded into a single corresponding *Ecore* metamodel. The resulting *CROM metamodel* is maintained in a separate plugin⁸ and represents the central artifact for the editor and adjunct tools. However, this metamodel only represents the structure of CROMs and is void of any layout information, e.g. positions, rectangles, and bend points. To further decouple *FRaMED* from this metamodel, the editor represents a separate plugin, which only emits instances of the CROM metamodel. Within the *FRaMED* plugin, the *Editor UI* handles all user interactions and is implemented employing the *Graphical Editing Framework (GEF)*.⁹ Internally, the plugin uses a custom *Ecore* metamodel for the graphical representation of CROM, denoted *Graphical Object Relation Model (GORM)*. This metamodel is tailored towards the graphical aspects of role models, such as *shapes*, *relations*, *segments*, and *bend points*. As a result, *FRaMED*'s implementation only depends on the GORM, simplifying the development and extension of the editor. Nonetheless, whenever a GORM instance is saved (as a `*.crom_dia` file), another plugin is tasked with its transformation to the corresponding CROM (saved as `*.crom` file). The *Transformation* plugin, in turn, utilizes the *Epsilon* framework,¹⁰ a rule-based model-to-model transformation engine, to declaratively specify the translation of GORM files to CROM files. Finally, once a CROM file is created, a user can trigger the *Code Generator* plugin to either generate a formal representation of the role model or generate corresponding source code. While the former can be used to validate the designed CROM, the latter can be completed to a working role-based application or a role-based database. Granted, this architecture is rather complex, however, it facilitates separation of concerns by establishing the CROM metamodel as central representation of the foundation for RMLs. This permits the separate evolution of the metamodel, editor, and code generators, as well as the development of additional tools, while assuring that all adhere to the structure and terminology defined in the CROM metamodel and, by extension, the presented formalization.

⁵<https://eclipse.org>

⁶<https://github.com/leondart/FRaMED/releases/tag/v2.0.3>

⁷<https://eclipse.org/modeling/emf>

⁸<https://github.com/Eden-06/CROM>

⁹<https://eclipse.org/gef>

¹⁰<http://www.eclipse.org/epsilon>

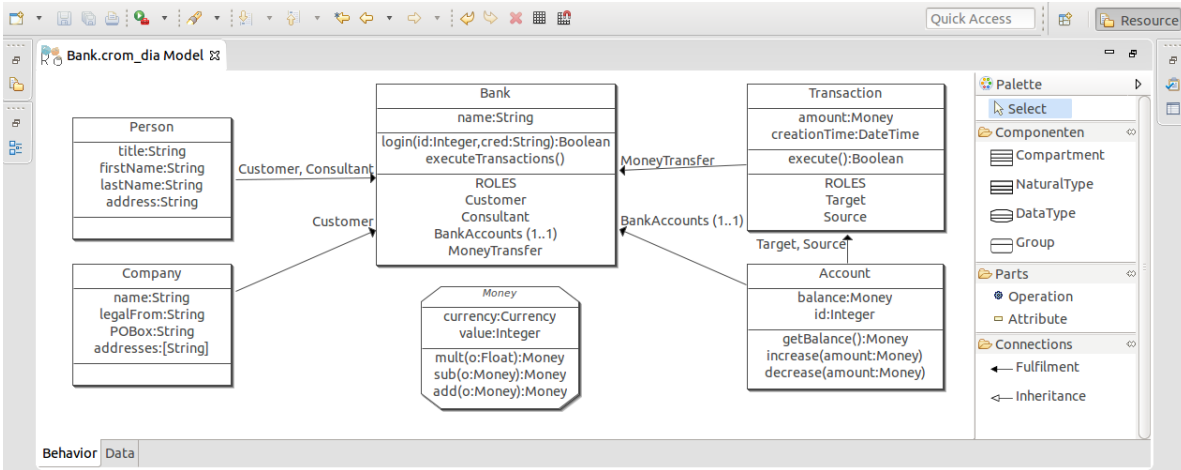


Figure 4: Banking application modeled in *FRaMED*, from [Kühn et al., 2016].

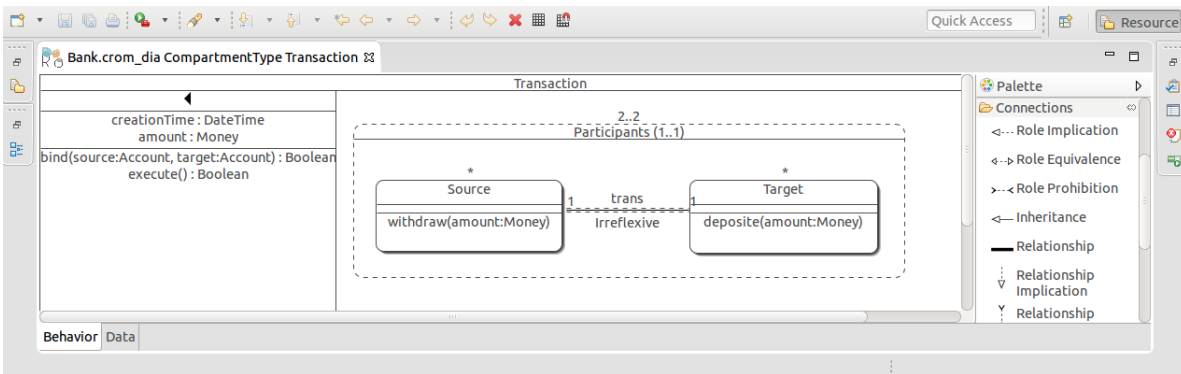


Figure 5: Focus view of the *Transaction* compartment type, from [Kühn et al., 2016].

The visual representation of CROMs is separated into two distinct levels: the *top-level view* and *focus view*. In the *top-level view* of FRaMED, shown in Figure 4, the user can create natural, data, and compartment types; specify their inheritance relation; as well as create and refine the *fills* relation [Kühn et al., 2016]. Model elements are added by selecting them in the palette, dragging them to the canvas, and naming them accordingly. In contrast to the common graphical notation, the *fills* relation is drawn from the player type to the compartment type, first. Afterwards, the role types filled by a *fills* relation can be selected using the “Fulfill Roles” dialog (accessible via its context menu) and are then listed adjacent to the *fills* relation. Finally, the user is able to *step into* a selected compartment type by clicking on the “Step In” context menu item. As a result, the *focus view* is opened showing the internals of the selected compartment type, as depicted in Figure 5. Here, one can create role types, role groups, and relationship types between two role types, as well as specify various role constraints, intra-relationship constraints, and inter-relationship constraints. While most of these model elements are selected and drawn from the palette, intra-relationship constraints are added by selecting the relationship type to constrain and opening the “Relationship Constraints ...” dialog via the context menu. Last but not least, to exit the focus view the user simply clicks on the “step out” context menu item. Besides all that, whenever the current role model is saved, the corresponding *crom* file is generated.

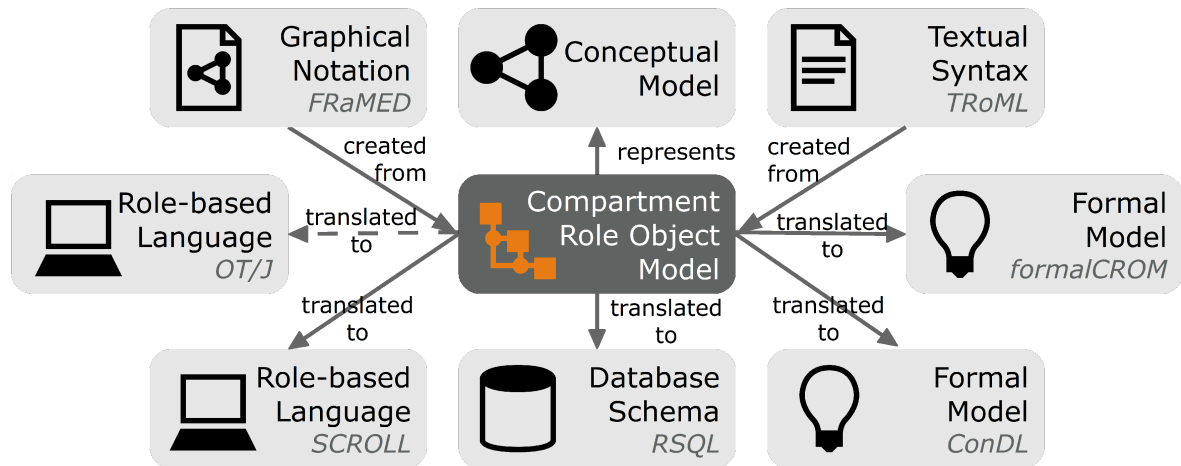


Figure 6: Overview of tool support for CROM.

While it is true that a modeling editor alone is useful for domain analysts and researchers, software practitioners strive for more tool supported integrated into the development environment. In particular, practitioners require code generators for various target languages to quickly validate or implement their domain model. To gratify their needs, FRaMED comes equipped with a set of code generators, as illustrated in Figure 6. These are available upon right-clicking on a CROM file (*.crom) and expanding the “Generate” context menu item to “RSQL Data Definition”, “SCROLL Code”, “OWL Ontology”, and “Formal CROM”, which triggers the corresponding code generator.

In conclusion, FRaMED is not just a graphical modeling editor for the CROM modeling language, but also the first role modeling editor that supports all natures of roles and the various constraints presented in the literature. Moreover, it enables the design of role-based software systems by providing additional means for validation and code generation. Furthermore, FRaMED is open source and freely available, in order to let both researchers and practitioners harness the power of role-based modeling. Additionally, FRaMED (v2.0.3) has been packaged as a virtual machine,¹¹ to guarantee simple installation for practitioners and reproducibility for researchers.

5 FAMILY OF ROLE-BASED MODELING LANGUAGES

Thus far, one would assume that the introduction of a foundational role-based modeling language able to fulfill most of the features of roles is sufficient, to convince other researchers to adopt and utilize the modeling language. However, when considering the history of RMLs and RPLs, it becomes evident that this is, generally, not the case. In fact, both *Taming Agents and Objects* [Da Silva et al., 2003] and *Scala Roles* [Pradel and Odersky, 2009] serve as counter example, i.e., role-based languages that have not been adopted, in spite of their superior feature set. To put it bluntly, after introducing yet another RML, there is still **no** common RML able to capture these divergent definitions of roles. In fact, this appears to be one of the main reasons for the apparent *fragmentation* and *discontinuity* within the research fields on RMLs and RPLs. Thus, instead of requiring all researcher to agree on a common definition of roles, the solution is to introduce a *family of role-based modeling languages* that harmonizes and reconciles their divergent views. In particular, the 27 classifying features of roles were utilized to identify the commonalities and differences of the contemporary

¹¹<http://st.inf.tu-dresden.de/intern/framed/framed-ubuntu.ova>

role-based languages. Moreover, to directly address the *discontinuity* among contemporary role-based approaches, the *family of metamodels for role-based languages* was introduced in [Kühn et al., 2014]. Specifically, it is able to generate compatible *metamodel variants* for arbitrary role-based languages. Thus, researchers can simply generate a metamodel for their specific approach and, more importantly, for previous approaches they intend to combine or reuse [Kühn et al., 2014]. Furthermore, to tackle the *fragmentation* of the various contemporary RMLs, FRaMED is upgraded to a fully dynamic *Software Product Line (SPL)* for the *family of RMLs*. As a result, researchers can tailor FRaMED to support their particular *language variant*. In conclusion, the *family of RMLs* addresses both the *fragmentation* and *discontinuity*, present in the research on roles.

5.1 FEATURE MODEL FOR ROLE-BASED LANGUAGES

Following a *feature-oriented software design (FOSD)* methodology, the first step is to generate a feature model as a hierarchical representation of the 27 identified features of roles (cf. Chapter 2.3). This, in turn, elucidates the various implicit dependencies of the classifying features of roles.

Figure 7 depicts the resulting feature model for RMLs, published in [Kühn et al., 2014]. To better trace, how the classifying features of roles have been mapped to the feature model, the feature nodes have been annotated with the corresponding number in the feature list (cf. Figure 1 and Figure 2). In detail, the feature model specifies three main feature arcs, e.g., `Role Types`, `Relationships`, and `Compartment Types`, to group all features dependent on the existence of these modeling concepts. Notably though, those features that are essential for the existence of a concept, are marked as mandatory. The mandatory feature `Naturals of Player`, for instance, denotes that role types can always be played by naturals. In conclusion, the feature model encompasses all 27 classifying features of roles. Besides that, the model additionally includes features for Riehle's *role constraints* [Riehle and Gross, 1998], i.e., `Role Implication`, `Role Exclusion`, and `Role Equivalence`, and two options for compartment identity, i.e., `Composite Identity`, and `Own Compartment Identity`. In conclusion, the feature model manages to arrange all 27 features of roles with respect to their dependencies to roles, relationships, and compartments. In addition to the dependencies of features visible in the feature model, four additional *cross-tree constraints* [Thüm et al., 2014] have been defined, as shown in Figure 8. Conversely, these constraints ensure that a configuration contains all entities on which the `Role Type` depends (8.1 and 8.2), that `Role Equivalence` is included, whenever the `Role Implication` is selected (8.3), and that `Compartment Types` are supported, if `Compartments` can be players (8.4).

It follows, then that the presented feature model faithfully captures and elucidates the dependencies of the 27 classifying features of roles. Henceforth, the feature model is used to define a configuration by selecting the various features.

5.2 FAMILY OF METAMODELS FOR ROLE-BASED MODELING LANGUAGES

To facilitate the generation of metamodel variants, a corresponding feature-oriented metamodel generator, denoted *RoSI CROM*, has been implemented and published in [Kühn et al., 2014]. In general, the generator was developed utilizing two *Eclipse* plugins designed to support FOSD: *FeatureIDE* and *DeltaEcore*. In particular, *FeatureIDE*¹² [Thüm et al., 2014] provided the foundation for the metamodel generator by offering dedicated editors for the specification of feature models and configurations. In fact, the feature model, depicted in Figure 7, was designed within the *FeatureIDE*. However, the *RoSI CROM* was implemented following a *delta modeling* approach using *DeltaEcore* [Seidl et al., 2014].

¹²http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide

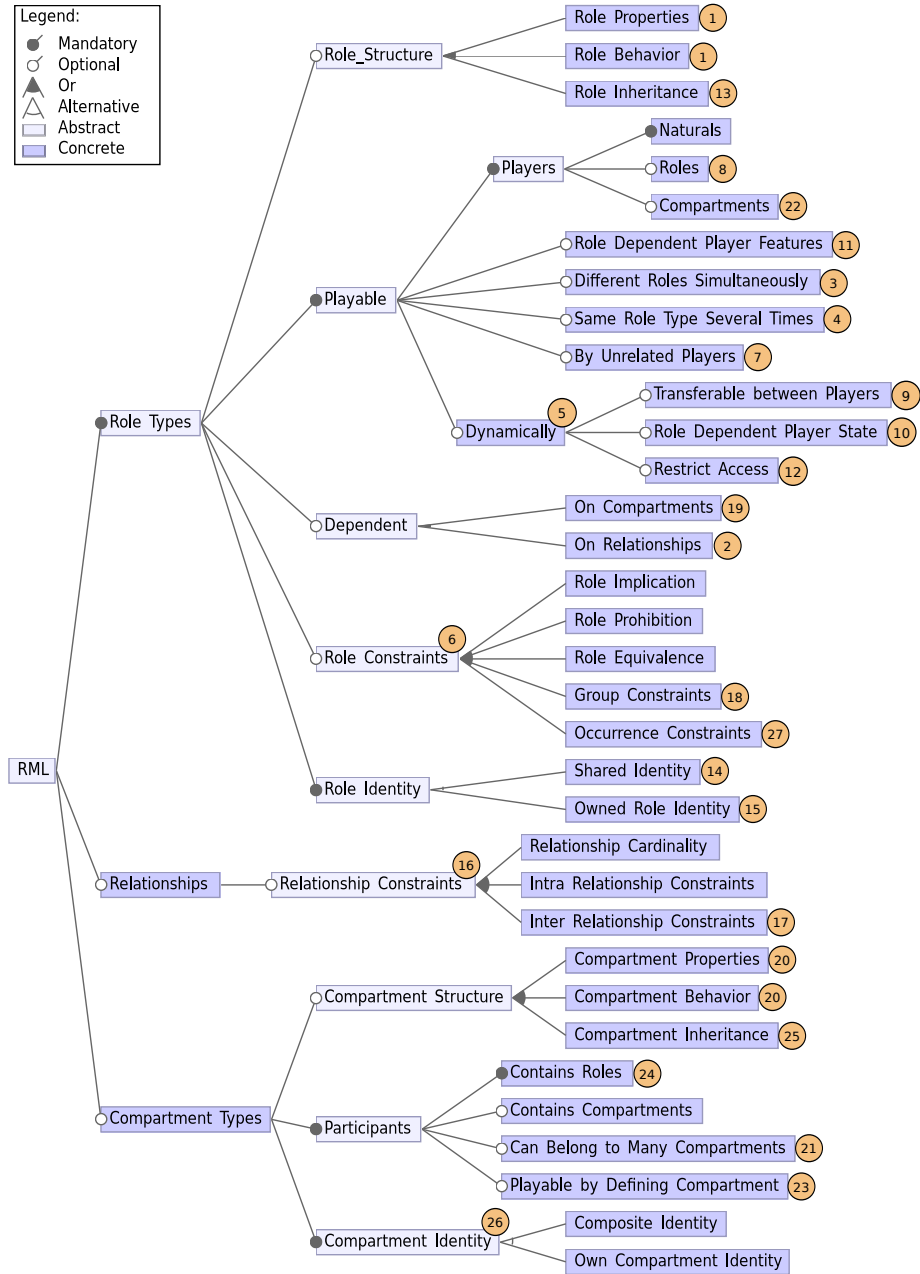


Figure 7: Feature model for role-based modeling languages, extended from [Kühn et al., 2014].

$$RoleTypes.Dependent.OnRelationships \Leftrightarrow Relationships \quad (8.1)$$

$$RoleTypes.Dependent.OnCompartments \Leftrightarrow CompartmentTypes \quad (8.2)$$

$$RoleImplication \Rightarrow RoleEquivalence \quad (8.3)$$

$$RoleTypes.Playable.Players.Compartments \Rightarrow CompartmentTypes \quad (8.4)$$

Figure 8: Cross-tree constraints of the feature model for RMLs.

Specifically, *DeltaEcore*¹³ allows for declaring the changes associated with selecting a feature within *delta modules*. These modules, in turn, manipulate a given base model by adding, modifying, or removing model elements. Additionally, the *feature mapping* connects features to delta modules by specifying their application conditions.¹⁴ Consequently, *RoSI CROM* employs a feature minimal metamodel as its base model; features 34 distinct *delta modules*; and implements the feature mapping, accordingly. Figure 9 establishes the general architecture of the metamodel generator.

In conclusion, *RoSI CROM* is a feature-oriented generator able to generate all variants of the family of metamodels. Moreover, due to the good integration of *DeltaEcore* into *FeatureIDE*, the metamodel generator is incredibly easy to use, once the *RoSI CROM* project is imported. Furthermore, the employed delta modeling approach ensures the scalability and evolvability of the metamodel family, as it permits researchers to easily add new features to the metamodel family by providing corresponding *delta modules* and modifying the *feature mapping*. Finally, *RoSI CROM* makes it viable for researchers to generate metamodels for arbitrary role-based modeling and programming languages simply by providing the corresponding configuration to the metamodel generator. Besides all that, *RoSI CROM* is open source and available on *GitHub*,¹⁵ as well.

5.3 FIRST FAMILY OF ROLE MODELING EDITORS

So far, there exists no graphical modeling editor able to support the *family of role-based modeling languages*. Thus, the PhD thesis proposes the development of the *first family of role modeling editors*. That is a feature-oriented, dynamic SPL of graphical modeling editors that enables the flexible configuration of RML variants, as well as the creation, manipulation, validation, and code generation of the corresponding CROM models. To achieve these goals, the Full-fledged Role Modeling Editor (*FRaMED*) is upgraded to a dynamic, feature-oriented SPL, accordingly. As a result of this extension, researchers will be able to tailor the *FRaMED SPL* to support their particular RML variant that corresponds to their understanding of roles.

Before discussing the underlying architecture of the *FRaMED SPL*, it is important to conceptualize the user's interaction with the modeling editor. In general, it should behave just like the original role modeling editor, *FRaMED*, however, it must permit its users to change the configuration of the underlying RML at any point in time. Additionally, it is conceivable that a user wants to edit multiple CROM models simultaneously, where each can belong to a different RMLs, i.e., with different feature configurations. It follows, then that each GORM model must additionally carry the configuration of the corresponding RML. By extensions, the modeling editor must change its behavior dynamically in accordance with the configuration of the currently opened GORM instance. As a result, the *FRaMED SPL* includes a "*Configuration*" tab for each opened editor canvas that allows for modifying the configuration of the underlying RML.

After describing the intended use of the *family of role modeling editors*, this section describes the necessary extensions to upgrade *FRaMED* to a dynamic SPL. As it turns out, the previously established architecture (cf. Figure 3) makes it a viable target for the creation of an SPL. In fact, most of the original implementation can be reused as is, whereas only key modules had to be replaced or extended. Accordingly, Figure 9 provides an overview on the modified architecture of *FRaMED*. Basically, the editor has been modified in four ways. First and foremost, the editor now loads the *effective feature model for RMLs* and the *edit policy mapping* upon startup. Secondly, the GORM model is extended to incorporate the current configuration of the language variant. Thus, when loading a

¹³<http://deltaecore.org>

¹⁴https://github.com/Eden-06/RoSI_CROM

¹⁵https://github.com/Eden-06/RoSI_CROM

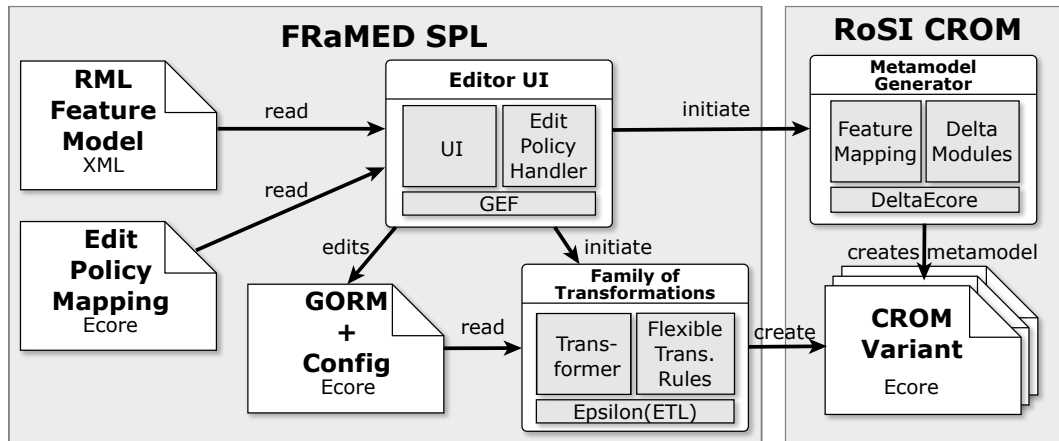


Figure 9: Architecture of the family of modeling editors.

graphical model (*.crom_dia), the editor also loads its feature configuration. Internally, each feature configuration is managed by a `Configuration` object (provided by the *FeatureIDE*) that verifies and ensures its validity. Third, the *transformation rules* have been adapted to take the feature configuration of the given GORM instance into account. Finally, *FRaMED SPL* includes the *RoSI CROM* generator as additional plugin to create the corresponding metamodel for a given feature configuration. Accordingly, whenever a GORM instance is saved, the metamodel generator is triggered first to create the corresponding metamodel variant, if not already present. Afterwards, the *transformation* plugin uses this metamodel variant as target to generate the corresponding model.

These modifications suffice to upgrade *FRaMED* to an SPL. However, to establish the *FRaMED SPL* as a dynamic product line, the editor must be able to dynamically adapt its palette and its edit policies in accordance with the current feature configuration. While it is feasible to implement dynamically changing *palette entries*, it is impractical to modify each and every previously implemented edit policy manually. Therefore, an *edit policy handler* is introduced that loads the *edit policy mapping* and adapts the static edit policies, accordingly. The *edit policy mapping*, in turn, maps features to specific *edit policies*, for instance, to prohibit *inheritance* relations between compartment types if compartment inheritance is not selected. In conclusion, the *family of RMLs* and its implementation within the *FRaMED SPL* permits researchers to design role models tailor to their use case. In fact, the *FRaMED SPL* is not only the *first family of role modeling editors*, but also the first role modeling editor able to embody all contemporary role-based modeling languages. Moreover, it tames *fragmentation* by enabling researchers to design individual role-based approaches with different features of roles, while maintaining that the models can be shared, combined, reused, and extended by others. Furthermore, the *FRaMED SPL* still provides the necessary tool support for validation and code generation. Although, the *FRaMED SPL* is a research prototype, it is made open source and freely available on *GitHub*.¹⁶

6 SUMMARY OF THE PHD THESIS

Throughout the course of the PhD thesis, it became evident that although roles have been used for conceptual modeling for almost 40 years, their underlying nature and formal foundations have not been fully uncovered. Besides Steimann's seminal paper *on the representation of roles* [Steimann,

¹⁶https://github.com/leondart/FRaMED/tree/develop_branch

2000b], this thesis finally establishes the underlying natures of roles, as well as their formal foundation. However, instead of providing yet another role-based modeling language (RML) for conceptual modeling, this PhD thesis established a complete and coherent *family of role-based modeling language*, as well as the corresponding *family of modeling editors*.

In particular, this thesis surveyed the contemporary literature on roles in the first part and presented the *foundations for RML*, as well as the *Family of RMLs* in the second part. In detail, Chapter 2 introduced the behavioral, relational, and context-dependent natures of roles; as well as extended Steimann's list of classifying features of roles by including 12 new features that have been retrieved from contemporary role-based languages. Based on these 27 features, a Systematic Literature Review (SLR) was designed and conducted to survey contemporary role-based languages (Chapter 2). In fact, this literature review identified 12 distinct RMLs and 14 RPL published between the year 2000 and 2016. Afterwards, Chapter 4 and Chapter 5 discussed and evaluated each of the contemporary RMLs and RPLs. Finally, Chapter 6 presented the results of the conducted SLR and the corresponding evaluation of contemporary role-based languages. This evaluation, in particular, identified four problems in the research fields on RMLs and RPLs. There is neither a *common understanding* nor *common feature set* shared among the different contemporary role-based modeling and programming languages. Moreover, the research fields on RMLs and RPLs are characterized by an ongoing *discontinuity* and *fragmentation*. Specifically, most approaches reinvent the role concept without taking the definitions of preceding related approaches into account. Furthermore, only four RMLs provide a sufficient *formal foundation* for roles able to incorporate *all natures of roles*, i.e. [Da Silva et al., 2003, Genovese, 2007, Liu and Hu, 2009, Hennicker and Klarl, 2014]. Regardless, none of them is able to *support all classifying features of roles*. Finally, most RMLs and RPLs are *not readily applicable*, due to their complexity, ambiguous terminology, and/or missing tool support. Especially, there exists no feature rich, practically usable graphical modeling editor for an RML. Consequently, the second part addressed these issues by first providing the foundations for RMLs and then introducing a the *family of RMLs*. Specifically, Chapter 7 established the foundations of RMLs by providing both a comprehensive ontological foundation for roles (Chapter 7.1) and a common graphical notation for RMLs. Above all, this chapter introduced and extended the Compartment Role Object Model (CROM) (Chapter 7.3 and 7.4), a formal framework for conceptual modeling that incorporated the three natures of roles and the modeling constraints. Besides that, Chapter 7.5 additionally presented *Full-fledged Role Modeling Editor (FRaMED)* as a readily applicable graphical modeling editor for CROM. In contrast, Chapter 8 addressed both the apparent *discontinuity* and *fragmentation* in the research fields on RMLs and RPLs. On the one hand, Chapter 8.1 established the *family of metamodels for role-based languages* that permits researchers to easily generate metamodels for arbitrary role-based languages they intend to combine and/or reuse. On the other hand, Chapter 8.2 finally introduced the *family of RMLs* and upgraded *FRaMED* to a fully dynamic *SPL* to support the language family. Ultimately, both the *metamodel family* and *family of RMLs* have been introduced to tackle the apparent *fragmentation* and *discontinuity* within the research community.

In sum, the PhD thesis presented the following contributions to the field of conceptual modeling, in general, and the field of role-based modeling and programming languages, in particular:

- A thorough literature survey on contemporary RMLs and RPLs published since the year 2000.
- The extension of the list of classifying features of roles [Steimann, 2000b] introducing 12 additional features of roles [Kühn et al., 2014].
- The introduction of concise ontological foundation for RMLs [Kühn et al., 2015a].
- The formalization of both *CROM* [Kühn et al., 2015a,b] and *CROM_I* as a comprehensive *formal foundation* for RMLs.

- The development of an *award winning* Full-fledged Role Modeling Editor (FRaMED) [Kühn et al., 2016].
- Introduction of the *family of RMLs* based on the *feature model for RMLs*.
- The implementation of the *RoSI CROM* metamodel generator to facilitate the *family of meta-models for role-based languages* [Kühn et al., 2014].
- The extension of *FRaMED* to a dynamic SPL supporting the creation of language variants of the *family of RML*, and thus introducing the *first family of role modeling editors*.

REFERENCES

- Atkinson, C. and Kühne, T. (2002). Rearchitecting the UML Infrastructure. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 12(4):290–321.
- Bachman, C. W., Daya, M., Bachman, C. W., and Daya, M. (1977). The Role Concept in Data Models. In *Proceedings of the Third International Conference on Very Large Data Bases*, volume 3, pages 464–476.
- Baldoni, M., Boella, G., and Van Der Torre, L. (2006). Roles as a Coordination Construct: Introducing powerJava. *Electronic Notes in Theoretical Computer Science*, 150(1):9–29.
- Balzer, S. and Gross, T. (2011). Verifying Multi-Object Invariants with Relationships. In Mezini, M., editor, *Lecture Notes in Computer Science*, volume 6813 of *25th European Conference on Object-Oriented Programming*, pages 359–383. Springer.
- Balzer, S., Gross, T., and Eugster, P. (2007). A Relational Model of Object Collaborations and Its Use in Reasoning About Relationships. In Ernst, E., editor, *ECOOP*, volume 4609 of *Lecture Notes in Computer Science*, pages 323–346. Springer.
- Barbosa, F. S. and Aguiar, A. (2012). Modeling and Programming with Roles: Introducing JavaStage. *Frontiers in Artificial Intelligence and Applications*, 246:124–145.
- Boella, G. and Van Der Torre, L. (2007). 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.
- Chen, P. (1976). The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36.
- Da Silva, V., Garcia, A., Brandão, A., Chavez, C., Lucena, C., and Alencar, P. (2003). Taming Agents and Objects in Software Engineering. In *International Workshop on Software Engineering for Large-Scale Multi-agent Systems*, pages 1–26. Springer.
- Dahchour, M., Pirotte, A., and Zimányi, E. (2002). A Generic Role Model for Dynamic Objects. In *Advanced Information Systems Engineering*, pages 643–658. Springer.
- Ferraiolo, D., Cugini, J., and Kuhn, D. R. (1995). Role-Based Access Control (RBAC): Features and Motivations. In *Proceedings of 11th Annual Computer Security Application Conference*, pages 241–48.
- Genovese, V. (2007). A Meta-Model for Roles: Introducing Sessions. In *Proceedings of the 2nd Workshop on Roles and Relationships in Object Oriented Programming, Multiagent Systems, and Ontologies*, pages 27–38. Technische Universität Berlin.
- Graversen, K. B. and Østerbye, K. (2003). Implementation of a Role Language for Object-Specific Dynamic Separation of Concerns. In *AOSD03 Workshop on Software-engineering Properties of Languages for Aspect Technologies*.
- Guarino, N. and Welty, C. (2000). A Formal Ontology of Properties. In *Knowledge Engineering and Knowledge Management Methods, Models, and Tools*, pages 97–112. Springer.

- Guarino, N. and Welty, C. A. (2009). An Overview of OntoClean. In *Handbook on Ontologies*, pages 201–220. Springer.
- Guizzardi, G. (2005). *Ontological Foundations for Structure Conceptual Models*. PhD thesis, Centre for Telematics and Information Technology, Enschede, Netherlands.
- Guizzardi, G. and Wagner, G. (2012). Conceptual Simulation Modeling with Onto-UML. In *Proceedings of the Winter Simulation Conference*, page 5. Winter Simulation Conference.
- Guizzardi, G., Wagner, G., Guarino, N., and van Sinderen, M. (2004). An Ontologically Well-Founded Profile for UML Conceptual Models. In *Advanced Information Systems Engineering*, pages 112–126. Springer.
- Halpin, T. (2005). ORM 2. In *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, pages 676–687. Springer.
- Harkes, D. and Visser, E. (2014). Unifying and Generalizing Relations in Role-Based Data Modeling and Navigation. In *International Conference on Software Language Engineering*, pages 241–260. Springer.
- He, C., Nie, Z., Li, B., Cao, L., and He, K. (2006). 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*, pages 7–pp. IEEE.
- Hennicker, R. and Klarl, A. (2014). Foundations for Ensemble Modeling – The Helena Approach. In *Specification, Algebra, and Software*, pages 359–381. Springer.
- Herrmann, S. (2005). Programming with Roles in ObjectTeams/Java. *AAAI Fall Symposium Roles, an interdisciplinary perspective*, (FS-05-08):73–80.
- Herrmann, S. (2007). A Precise Model for Contextual Roles: The Programming Language ObjectTeams/Java. *Applied Ontology*, 2(2):181–207.
- Jäkel, T., Kühn, T., Hinkel, S., Voigt, H., and Lehner, W. (2015). Relationships for Dynamic Data Types in RSQL. In *Datenbanksysteme für Business, Technologie und Web (BTW)*.
- Jäkel, T., Kühn, T., Voigt, H., and Lehner, W. (2016). Towards a Contextual Database. In *20th East-European Conference on Advances in Databases and Information Systems*.
- Kamina, T. and Tamai, T. (2009). Towards Safe and Flexible Object Adaptation. In *International Workshop on Context-Oriented Programming*, page 4. ACM.
- Kim, D.-K., France, R., Ghosh, S., and Song, E. (2003). A Role-Based Metamodeling Approach to Specifying Design Patterns. In *Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual International*, pages 452–457. IEEE.
- Kim, S.-K. and Carrington, D. (2004). Using Integrated Metamodeling to Define OO Design Patterns with Object-Z and UML. In *Software Engineering Conference, 2004. 11th Asia-Pacific*, pages 257–264. IEEE.
- Kitchenham, B. (2004). Procedures for Performing Systematic Reviews. *Keele, UK, Keele University*, 33(2004):1–26.
- Kühn, T., Bierzynski, K., Richly, S., and Aßmann, U. (2016). FRaMED: Full-Fledge Role Modeling Editor (Tool Demo). In *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering, SLE 2016*, pages 132–136, New York, NY, USA. ACM.
- Kühn, T., Böhme, S., Götz, S., and Aßmann, U. (2015a). A Combined Formal Model for Relational Context-Dependent Roles. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering*, pages 113–124. ACM.
- Kühn, T., Böhme, S., Götz, S., and Aßmann, U. (2015b). A Combined Formal Model for Relational Context-Dependent Roles (Extended). Technical Report TUD-FI15-04-Sept-2015, Technische Universität Dresden.
- Kühn, T., Leuthäuser, M., Götz, S., Seidl, C., and Aßmann, U. (2014). A Metamodel Family for Role-Based Modeling and Programming Languages. In *Software Language Engineering*, volume 8706

- of *Lecture Notes in Computer Science*, pages 141–160. Springer.
- Liu, M. and Hu, J. (2009). Information Networking Model. In *Conceptual Modeling-ER 2009*, pages 131–144. Springer.
- Loebe, F. (2005). Abstract vs. Social Roles – A Refined Top-Level Ontological Analysis. In *In Procs. of AAAI Fall Symposium Roles, an interdisciplinary perspective*. Citeseer.
- Mizoguchi, R., Kozaki, K., and Kitamura, Y. (2012). Ontological Analyses of Roles. In *Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on*, pages 489–496. IEEE.
- Murer, S., Worms, C., and Furrer, F. J. (2008). Managed Evolution. *Informatik-Spektrum*, 31(6):537–547.
- Mylopoulos, J. (1992). Conceptual Modelling and Telos. In Loucopoulos, P. and Zicari, R., editors, *Conceptual Modeling, Databases, and Case: An Integrated View of Information Systems Development*, pages 49–68. John Wiley & Sons, Inc.
- Nelson, S., Pearce, D. J., and Noble, J. (2008). First Class Relationships for OO Languages. In *Proceedings of the 6th International Workshop on Multiparadigm Programming with Object-Oriented Languages (MPOOL 2008)*.
- Pradel, M. and Odersky, M. (2009). Scala Roles: Reusable Object Collaborations in a Library. In *Software and Data Technologies*, pages 23–36. Springer.
- Reenskaug, T. and Coplien, J. O. (2009). 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*.
- Riehle, D. and Gross, T. (1998). Role Model Based Framework Design and Integration. In *Proceedings OOPSLA '98, ACM SIGPLAN Notices*, pages 117–133.
- Rothenberg, J., Widman, L. E., Loparo, K. A., and Nielsen, N. R. (1989). *The Nature of Modeling*, volume 3027. Rand.
- Rumbaugh, J., Jacobson, R., and Booch, G. (1999). *The Unified Modelling Language Reference Manual*. Addison-Wesley, 1st edition.
- Seidl, C., Schaefer, I., and Altmann, U. (2014). DeltaEcore—A Model-Based Delta Language Generation Framework. In *Modellierung*, pages 81–96.
- Selçuk, Y. E. and Erdoğan, N. (2004). JAWIRO: Enhancing Java with Roles. In *International Symposium on Computer and Information Sciences*, pages 927–934. Springer.
- Serrano, J. M. and Ossowski, S. (2004). On the Impact of Agent Communication Languages on the Implementation of Agent Systems. In *International Workshop on Cooperative Information Agents*, pages 92–106. Springer.
- Steimann, F. (2000a). *Formale Modellierung mit Rollen*. PhD thesis, Universität Hannover. Habilitation thesis.
- Steimann, F. (2000b). On the Representation of Roles in Object-Oriented and Conceptual Modelling. *Data & Knowledge Engineering*, 35(1):83–106.
- Steimann, F. (2000c). A Radical Revision of UML’s Role Concept. In *UML 2000 - The Unified Modeling Language*, pages 194–209. Springer.
- Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., and Leich, T. (2014). FeatureIDE: An Extensible Framework for Feature-Oriented Software Development. *Science of Computer Programming*, 79:70–85.
- Ubayashi, N. and Tamai, T. (2001). Separation of Concerns in Mobile Agent Applications. In *International Conference on Metalevel Architectures and Reflection*, pages 89–109. Springer.
- Zhu, H. and Zhou, M. (2006). Role-Based Collaboration and Its Kernel Mechanisms. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 36(4):578–589.