TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik

TECHNISCHE BERICHTE
TECHNICAL REPORTS

ISSN 1430-211X

TUD-FI15-04-September 2015

T. Kühn[1], S. Böhme[2], S. Götz[1], U. Aßmann[1]
[1]) Software Technology Group
[2]) Theoretical Computer Science

A Combined Formal Model for Relational Context-Dependent Roles (Extended)

TechnischeUniversität Dresden
Nutzerberatung CD
01062 Dresden
Germany
http://tu-dresden.de/cd

A Combined Formal Model for Relational Context-Dependent Roles (Extended)

Thomas Kühn
Software Technology Group
TU Dresden, Germany
thomas.kuehn3@tu-dresden.de

Stephan Böhme
Theoretical Computer Science
TU Dresden, Germany
stephan.boehme@tu-dresden.de

Sebastian Götz
Uwe Aßmann
Software Technology Group
TU Dresden, Germany
sebastian.goetz@acm.org
uwe.assmann@tu-dresden.de

Abstract

Role-based modeling has been investigated for over 35 years as a promising paradigm to model complex, dynamic systems. Although current software systems are characterized by increasing complexity and context-dependence, all this research had almost no influence on current software development practice, still being discussed in recent literature. One reason for this is the lack of a coherent, comprehensive, readily applicable notion of roles. Researchers focused either on relational roles or context-dependent roles rather then combining both natures. Currently, there is no role-based modeling language sufficiently incorporating both the relational and context-dependent nature of roles together with the various proposed constraints. Hence, this paper formalizes a full-fledged role-based modeling language supporting both natures. To show its sufficiency and adequacy, a real world example is employed.

Categories and Subject Descriptors I.6.4. [*Simulation and Modeling*]: Model Validation and Analysis—Role-based Modeling; I.6.5. [*Simulation and Modeling*]: Model Development—Formal Modeling

1. Introduction

Charles W. Bachmann was the first researcher to investigate roles back in the year 1977. He proposed role-based modeling [1] to capture both context-dependent and collaborative behavior of objects. Since then, a large variety of approaches in different research areas, ranging from data modeling [1, 16, 23] via conceptual modeling [15, 30] through to programming languages [3, 5, 19, 26] emerged. More importantly, because current software systems are characterized by increased complexity and context-dependence [25], there is a strong demand for new concepts beyond object-oriented design. Although mainstream object-oriented modeling languages are good at capturing a system's structure, they lack ways to model the systems behavior, as it dynamically emerges through collaborating objects [27]. 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 [19, 23]. 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 enable the practical application of roles, two major requirements have to be fulfilled. First, besides the intuitive semantics underlying the role concept, its notions must be formally specified to create a coherent understanding of relational and context-dependent roles. Second, to permit scalability of role-based modeling languages automatic mechanisms to validate the well-formedness and consistency are required. Therefore, a formal model is indispensable. In sum, a major blocking factor for the practical application of roles is their lack of a consistent formal model. Thus, this paper provides a formal model for a role-based modeling language supporting both natures of roles. This allows for the formal and automatic verification of well-formedness, compliance, and validity of models at design time and their instances at runtime.

*This technical report has been published both within the Saxon State and University Library Dresden (SLUB) and the Quality Content of Saxony (Qucosa) under the ISSN: **1430-211X** and the technical report identifier: **TUD-FI15-04-September 2015***

The paper is structured as follows: Section 2 introduces a small running example used for the remainder of the paper. Afterwards, Section 3 elaborates on the nature of roles and surveys contemporary role-based modeling languages. The main contribution of this paper, a formal modeling language for context-dependent, relational roles, is presented in Section 4 and employed in Section 5. Section 6 discusses our reference implementations, classifies our formal model, and its limitations. Section 7 compares the presented formal model to related approaches. A brief overview on completed and future research efforts concludes the paper.

2. Running Example

Before diving into role-based modeling, we first employ the real world scenario of a small banking application, extracted from [27]. A bank is a financial institution that employs consultants and provides banking services to their customers, who are either persons or companies. Customers can be advised by consultants, own several savings and checking accounts, and perform transactions. Transactions encapsulate the process of transferring money from exactly one source account to one target account. They are initiated by a customer, however, managed and executed by the bank. Additionally, financial regulations require that consultants should not advise themselves as a customer and checking accounts must have exactly one owner whereas savings accounts can have multiple owners. Furthermore, no account can be both a checking and a savings account as well as both the source and target of the same transaction.

3. Nature of Roles

Roles are not a new concept in modeling languages, however, the only generally accepted feature of roles is that they can be played by unrelated objects [22, 30]. The role of a *customer* of a bank, for instance, can be played by either a *person* or a *company*, regardless of them being related or not. Besides that, [30] and [22] have shown that there is no common understanding of roles in the literature. They identified 26 classifying features attributed to roles, shown in Figure 1. For now, two of these features are important, because they help to classify approaches to support either the relational nature or context-dependent nature of roles, namely Feature 2 and 19. Additionally, several features indicate that roles and role models are subject to constraints, e.g., Features 6, 16, 17, and 18. Accordingly, the following discussion is trisected.

3.1 Relational Nature

Modeling languages usually feature some notion of role. Consider, ER [7] and UML [29], where roles denote the named ends of relationships or associations.

Either way, they neither have properties (Feature 1) nor the ability to be played by unrelated objects (Feature 7). In the case of UML, one could argue that this can be resolved by modeling roles as classes and use inheritance to relate them to their players. This, however, fails to capture the intended dynamism of roles and results in exponentially many classes [30]. Several role-based modeling languages [1, 4, 16, 21, 30] introduced roles tied to relationships as first-class citizens. Hence, these approaches can represent the *advises* relationship between *consultants* and *customers* and the latter be played by either *persons* or *companies*. These languages assume that all roles and relationships are equally relevant to an object's properties. Thus, there is no notion of context, scope, or institution on which roles and relationships depend on. In most approaches (except [4, 16]) relationships cannot play roles themselves. In the banking application, the transaction must be modeled as a relationship, but cannot be tied to the specific bank owning this transaction. This, in turn, prevents reusing the notion of a financial transaction, as it is now tied to the modeled bank. To resolve this dilemma it must be understood that relationships themselves are context-dependent.

3.2 Context-Dependent Nature

To incorporate the missing contextual-dependency of roles, recent role-based modeling languages [11, 18, 27] focused on the context-dependent nature. These approaches introduce some sort of *context* to encapsulate the roles relevant to a certain situation or interaction. Due to the fact that the term itself is massively overloaded, several researchers introduced other terms to denote the context of a role, e.g., *environment* [31], *institution* [2], *ensemble* [18], and *compartment* [22]. Henceforth, the term *compartment* is used because its definition, as *"objectified collaboration with a limited number of participating roles and a fixed scope"* [22], encompasses all of the other notions. In conclusion, these approaches capture the contextual dependence of roles by making the compartments their definitional boundary. For the banking example, the *transaction* can be considered a compartment, as it captures the transfer of money from a *source* to a *target* account. While most of these approaches assume that roles depend on (some kind of) compartment (Feature 19) and have properties (Feature 20), only few consider compartments as objects able to play roles themselves (Feature 22). The latter is crucial to ensure that the *transaction* compartment (instance) can be owned by a *bank* compartment (instance) managing its creation and execution. Nevertheless, most approaches relied solely on compartments [2, 11, 27, 31] and did not include context-dependent relationships between roles.

1. Roles have properties and behaviors
2. Roles depend on relationships
3. Objects may play different roles simultaneously
4. Objects may play the same role (type) several times
5. Objects may acquire and abandon roles dynamically
6. The sequence of role acquisition and removal may be restricted
7. Unrelated objects can play the same role
8. Roles can play roles
9. Roles can be transferred between objects
10. The state of an object can be role-specific
11. Features of an object can be role-specific
12. Roles restrict access
13. Different roles may share structure and behavior
14. An object and its roles share identity
15. An object and its roles have different identities
16. Relationships between roles can be constrained
17. There may be constraints between relationships
18. Roles can be grouped and constrained together
19. Roles depend on compartments
20. Compartments have properties and behaviors
21. A role can be part of several compartments
22. Compartments may play roles like objects
23. Compartments may play roles which are part of themselves
24. Compartments can contain other compartments
25. Different compartments may share structure and behavior
26. Compartments have their own identity

Figure 1: Classifying features of roles, extracted from [22, 30]

3.3 Constraining Role Models

So far the discussion revolved around the nature of roles in role-based modeling languages and did not cover their capabilities to specify particular constraints for roles and relationships. Typically, modeling languages support various constraints on relationships (Feature 16). *Cardinality constraints*, for instance, are featured in most modeling languages, e.g., ER [7] and UML [29]. They limit the number of entities related by a relationship. In our example, the *owns checking account* relationship must be constrained with cardinality *one* on the *customer* side and *zero-to-many* on the *checking account* to ensure that checking accounts are owned by exactly one customer. *Intra-relationship constraints* [4] represent mathematical constraints for relations that additionally constrain relationships. The *advises* relationship, for instance, can be constrained to be *irreflexive* ensuring that persons playing the *consultant* role cannot advise themselves as a *customer*. *Inter-relationship constraints* [16] are constraints between individual relationships (Feature 17). They can be used to define subsets or disjunctions between relationships. An example for the former is a *father son* relationship required to be a subset of the *parent child* relationship. While these constraints limit relationships, there are two kinds of constraints for roles. *Role constraints* limit the types of roles that can be played simultaneously by one object. They include notions to prohibit or require another role to be played if the other role is played [28]. In our example, the statement that an *account* cannot be a *checking account* and a *savings account* at the same time can be expressed with a *role-prohibition* between these roles. Moreover, [22, 31] suggested to employ notions to group and constrain roles together. In contrast, *occurrence constraints* [18, 31] limit the number of roles present in a compartment, i.e., the number of role instances of a particular type in a compartment instance. Consider the *transaction* compartment that requires the presence of exactly one *source* and one *target* role. Although various role-based modeling languages have introduced different kinds of constraints, no approach has included all of them into one coherent model.

4. A Formal Role-Based Modeling Language

This section introduces our formal model for *Compartment Role Object Models* (CROM) [22], *Compartment Role Object Instances* (CROI) and *Constraint Models* by first defining the underlying ontological foundations; then introducing their graphical notation; and finally providing their formal definitions.

4.1 Ontological Foundation

Before providing any formal definition, it is crucial to classify the different kinds of concepts employed by our modeling language. Without this distinction, designers of role-based systems cannot decide whether a concept should be modeled as either *Natural Type*, *Role Type*, *Compartment Type*, or *Relationship Type*. To provide a clear ontological distinction, three well-established ontological properties are used: *Rigidity*, *Foundedness* and *Identity* [12–14, 24]. The first denotes that instances of a rigid type belong to that type until they cease to exist [13, 14]. A *person*, for instance, can be considered a rigid type, because you can only stop being a person if you die. The second describes that instances of a founded type can only exist if another instance exists at the same time [12, 13, 24, 30]. The *customer* of our bank application is such a founded type, because a *customer* can only exist if the *bank* exists, as well. The last property

distinguishes whether instances of a certain type have a unique, derived, or composed identity [12]. A *person*, for instance, has a unique identity throughout its live time, whereas a *customer* derives its identity from the person in that role. The *trans* relationship, in turn, is identified by the combined identities of the *source* and *target* accounts.

These three ontological properties are sufficient to distinguish four kinds of concepts. *Natural Types* are rigid, not founded, and their instances carry their own unique identity. Thus, instances of natural types have an immutable, independent type and identity. The entities *person*, *company*, and *account* are natural types in our banking application. *Role Types*, in contrast, are not rigid [15], founded and their instances only derive their identity from their players. As such, role instances depend on both the identity of their player and a foundational relation to their context [24] (i.e., compartment). Thus, instances of a rigid type can dynamically adopt role types by playing its instances. As a result, most entities in our banking application become role types, e.g.: *consultants*, *customer*, *checking account*, *savings account*. *Compartment Types* are rigid, founded, and their instances have a unique identity, hence, their instances are founded on the existence of participating roles. For example, both the *bank* and *transaction* are considered compartment types. *Relationship Types* are rigid, founded and have a composed identity. They represent binary relationships between two distinct role types.[1] The identity of *links* (relationship instances) is composed from the identities of the players of the participating role. In sum, these concepts form the foundations for our modeling language.

4.2 Graphical Notation

This section facilitates the graphical notation for CROM and CROI by illustrating a role model and a possible instance for the banking application. Figure 2a depicts the example role model. 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*. Figure 2b, in turn, shows one possible instance of this model. It comprises two **Persons** *Peter* and *Klaus*, as well as a **Company** *Google* that play roles in the *bank* compartment instance. Each role is placed at the border of its respective player. For brevity, we omitted their individual attributes. *Klaus* and *Google* play the **Customer** role. The former owns a *CheckingAccount* and the latter owns a *SavingsAccount*. Besides that, *Google* is advised by *Peter* playing the **Consultant** role. Additionally, the model contains one **Transaction** compartment t where *Account1* and *Account2* play the **Source** and **Target** role, respectively. Thus, it represents a transaction from *Google's* savings account to *Klaus's* checking account. This transaction itself plays the role of a *MoneyTransfer* within the *bank* compartment. Intuitively, it is possible to check that the instance adheres to the intuitive semantics of the role model, however, to formally validate these models they have to be formalized.

4.3 Type Level

After introducing the ontological foundations and the graphical notation, we can introduce our formal model, starting on the type level. For brevity, we omitted the notion of *attributes* from these definitions. Nevertheless, the necessary additions are presented in the Appendix.

Definition 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 $\mathcal{M} = (NT, RT, CT, RST, fills, parts, rel)$ where $fills \subseteq (NT \cup CT) \times RT$ is a relation, $parts : CT \to 2^{RT}$ and $rel : RST \to (RT \times RT)$ are total functions. A CROM is denoted* well-formed *if the following axioms hold:*

$$\forall rt \in RT\ \exists t \in (NT \cup CT) : (t, rt) \in fills \quad (1)$$

$$\forall ct \in CT : parts(ct) \neq \emptyset \quad (2)$$

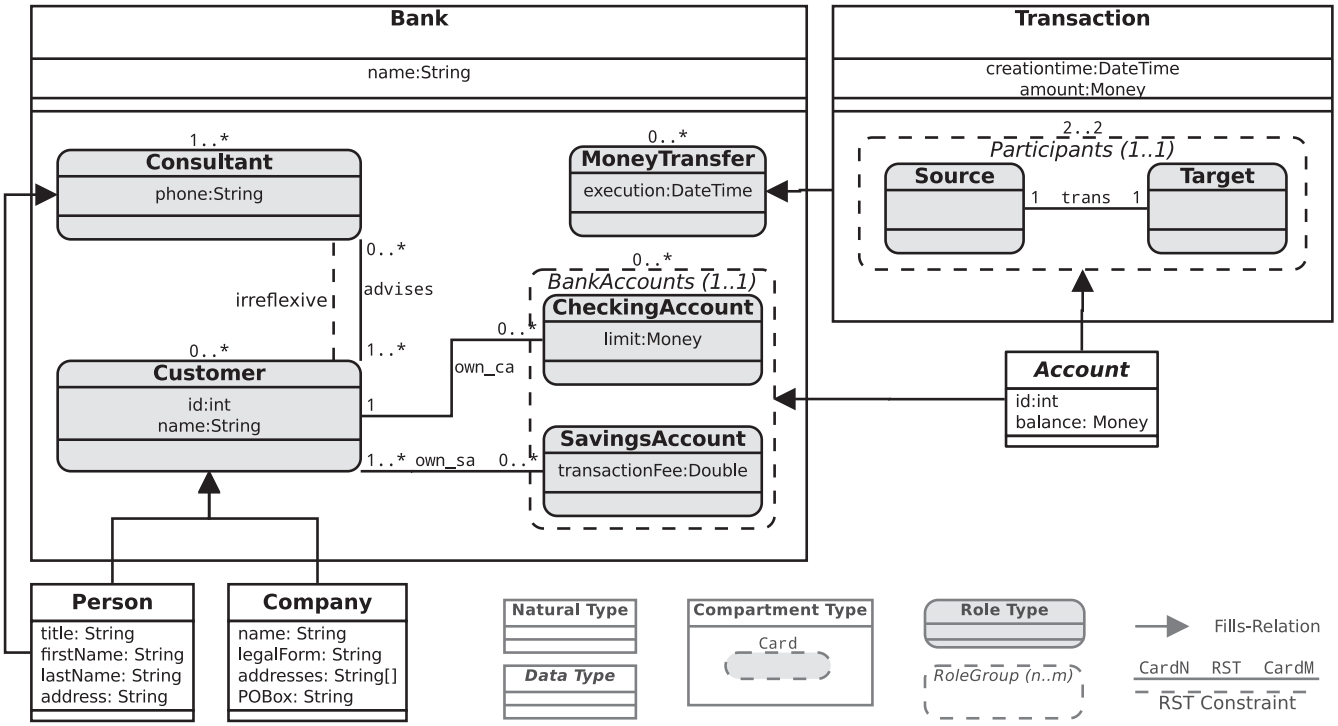
$$\forall rt \in RT\ \exists! ct \in CT : rt \in parts(ct) \quad (3)$$

$$\forall rst \in RST : rel(rst) = (rt_1, rt_2) \wedge rt_1 \neq rt_2 \quad (4)$$

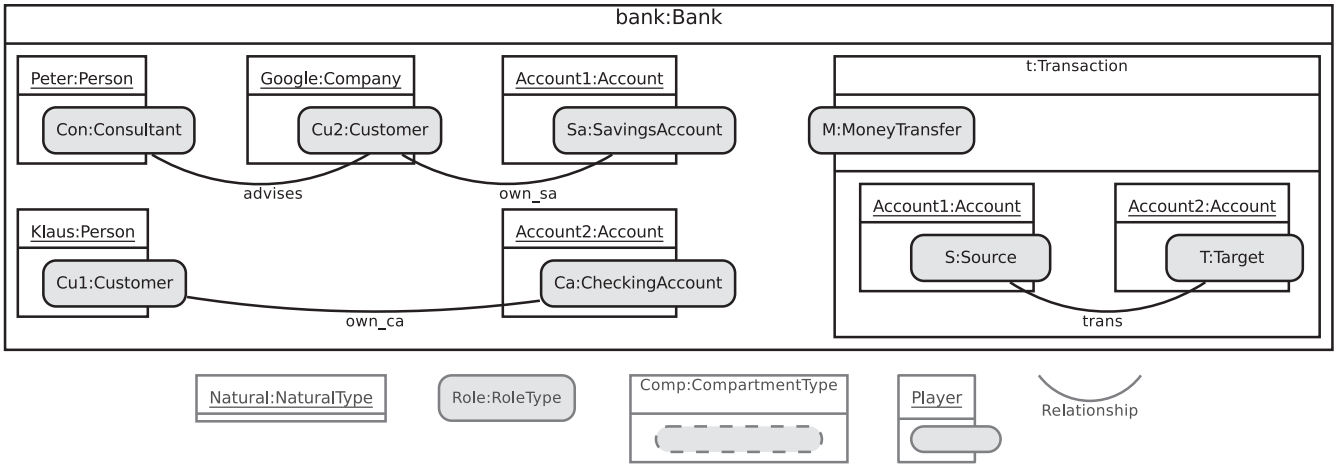
$$\forall rst \in RST\ \exists ct \in CT : rel(rst) = (rt_1, rt_2) \wedge rt_1, rt_2 \in parts(ct) \quad (5)$$

In detail, *fills* denotes that rigid types can play roles of a certain role type, *parts* maps compartment types to their contained role types, and *rel* captures the two role types at the respective ends of each relationship type. The well-formedness rules ensure that the *fills*-relation is surjective (1); each compartment type has a nonempty, disjoint set of role types as its parts (2, 3); and *rel* maps each relationship type to exactly two distinct role types of the same compartment type (4, 5).

[1] Note that each n-ary relationship can be represented with n binary relationships.



(a) Model



(b) Instance

Figure 2: Compartment Role Object Model and Instance of the banking application

Example 1 (Compartment Role Object Model)**.** *Let* $\mathcal{B} = (NT, RT, CT, RST, fills, parts, rel)$ *be the model of the bank (Figure 2a), where the idividual components are defined as follows:*

$$NT \coloneqq \{Person, Company, Account\}$$

$$RT \coloneqq \{Customer, Consultant, CA, SA, Source, Target, MoneyTransfer\}$$

$$CT = \{Bank, Transaction\}$$

$$RST = \{own\_ca, own\_sa, advises, trans\}$$

$$fills \coloneqq \{(Person, Consultant), (Person, Customer), (Company, Customer), (Account, Source), (Account, Target), (Account, CA), (Account, SA), (Transaction, MoneyTransfer)\}$$

$$parts \coloneqq \{Bank \to \{Consultant, Customer, CA, SA, MoneyTransfer\}, Transaction \to \{Source, Target\}\}$$

$$rel \coloneqq \{own\_ca \to (Customer, CA), own\_sa \to (Customer, SA), advises \to (Consultant, Customer), trans \to (Source, Target)\}$$

The bank model $\mathcal{B}$ is simply created from Figure 2a in four steps. First, all the natural types, compartment types, role types, and relationship types are collected into the corresponding set.[1] Second, the set of role types contained in each compartment type is assigned to the *parts*-function. Third, it is specified which natural type can *fill* which role type, and finally the *rel*-function is defined for the role types at the ends of each relationship type. Thus, CROMs can be retrieved from their graphical representation. The presented bank model $\mathcal{B}$ is well-formed, because each defined role type is filled by at least one natural type or compartment type (1), each compartment type consists of a non-empty (2) and disjoint (3) set of role types, and each relationship type is established between two distinct role types (4) of the same compartment type (5).

4.4 Instance Level

On this level, we distinguish naturals, roles, compartments and links as instances of their respective types.

Definition 2 (Compartment Role Object Instance)**.** *Let $\mathcal{M} = (NT, RT, CT, RST, fills, parts, rel)$ be a well-formed CROM and N, R, and C be mutual disjoint sets of Naturals, Roles and Compartments, respectively. Then a* Compartment Role Object Instance (CROI) *of $\mathcal{M}$ is a tuple $\mathfrak{i} = (N, R, C, type, plays, links)$, where $type : (N \to NT) \cup (R \to RT) \cup (C \to CT)$ is a labeling function, $plays \subseteq (N \cup C) \times C \times R$ a relation, and $links : RST \times C \to 2^{R^\varepsilon \times R^\varepsilon}$ is a total function (such that $R^\varepsilon := R \cup \{\varepsilon\}$ with $\varepsilon \notin R \cup N \cup C$). Moreover, $O := N \cup C$ denotes the set of all objects in $\mathfrak{i}$ and $O^c := \{o \in O \mid \exists r \in R : (o, c, r) \in plays\}$ the set of objects played in a compartment $c \in C$. To be* compliant *to the model $\mathcal{M}$ the instance $\mathfrak{i}$ must satisfy the following conditions:*

$$\forall (o,c,r) \in plays : (type(o), type(r)) \in fills \wedge type(r) \in parts(type(c)) \tag{6}$$

$$\forall (o,c,r),(o,c,r') \in plays : r \neq r' \Rightarrow type(r) \neq type(r') \tag{7}$$

$$\forall r \in R\ \exists! o \in O\ \exists! c \in C :\ (o,c,r) \in plays \tag{8}$$

$$\forall rst \in RST\ \forall c \in C :\ (\varepsilon,\varepsilon) \notin links(rst,c) \tag{9}$$

$$\begin{aligned} \forall rst \in RST\ \forall c \in C\ \forall r \in R\ \forall o \in O\ \exists \hat{r} \in R^\varepsilon :\ & rel(rst) = (rt_1, rt_2) \wedge \\ & \big(\big((o,c,r) \in plays \wedge type(r) = rt_1\big) \\ & \quad \Leftrightarrow \big((r,\hat{r}) \in links(rst,c)\big)\big) \wedge \\ & \big(\big((o,c,r) \in plays \wedge type(r) = rt_2\big) \\ & \quad \Leftrightarrow \big((\hat{r},r) \in links(rst,c)\big)\big) \end{aligned} \tag{10}$$

$$\forall rst \in RST\ \forall c \in C\ \forall (r_1,r_2) \in links(rst,c) \cap R \times R : (r_1,\varepsilon),(\varepsilon,r_2) \notin links(rst,c) \tag{11}$$

The *type* function assigns a distinct type to each instance, *plays* identifies the objects (either natural or compartment) playing a certain role in a specific compartment, and *links* captures the roles currently linked by a relationship type in a certain compartment. A compliant CROI has to satisfy the given six axioms that guarantee consistency of both the *plays*-relation and the *links*-function with the model $\mathcal{M}$. Axioms (7) and (8) restrict the *plays*-relation, such that an object is prohibited to play instances of the same role type multiple times in one compartment and each role has one distinct player in one distinct compartment. Axiom (10) ensures that if and only if a role participates in a compartment and its role type is linked by a relationship type, then a corresponding tuple in the *links*-function for that compartment exists. To reflect that a role is not related to any counter role (11), a role can also be linked to the empty role ε. This ensures that each role played in a compartment c is presented in the corresponding $links(rst, c)$-function without forcing this role to be linked to a counter role. However, because an object can play only one role of a certain type in one compartment (7), the traditional semantics of cardinality constraints of relationships is retained in this compartment. This ensures the cardinality constraints only locally to compartments, i.e., a natural can play a role of a certain type multiple times if each of them is played in a different compartment. Hence, ε is introduced to capture zero-to-one and zero-to-many relationships without leaving out roles participating in that relationship while allowing to play roles of the same type multiple times.

Besides, these definitions entail that $links(rst, c) = \emptyset$ for each compartment c with a type not containing the relationship type rst.[2]

Example 2 (Compartment Role Object Instance)**.** *Let $\mathcal{B} = (NT, RT, CT, RST, fills, parts, rel)$ be the well-formed CROM defined in Example 1; then $\mathfrak{b} = (N, R, C, type, plays, links)$ is an instance of that model (Figure 2b), where the components are defined as follows:*

$$\begin{aligned} N &:= \{Peter, Klaus, Google, Account_1, Account_2\} \\ R &:= \{Cu_1, Cu_2, Con, Ca, Sa, S, T, M\} \\ C &:= \{bank, t\} \\ type &:= \{(Cu_1 \to Customer), (Cu_2 \to Customer), \\ &\quad (Con \to Consultant), (Ca \to CA), (Sa \to SA), \\ &\quad (S \to Source), (T \to Target), \\ &\quad (M \to MoneyTransfer), (bank \to Bank), \\ &\quad (t \to Transaction), \dots\} \end{aligned}$$

[1] Henceforth, *SA* and *CA* are abbreviations for *SavingsAccount* and *CheckingAccount*, respectively.

[2] A sufficient proof can be found in the Appendix.

$$
\begin{aligned}
\mathit{plays} \coloneqq \{&(\mathit{Klaus}, \mathit{bank}, \mathit{Cu}_1), (\mathit{Google}, \mathit{bank}, \mathit{Cu}_2),\\
&(\mathit{Peter}, \mathit{bank}, \mathit{Con}), (\mathit{Account}_1, \mathit{bank}, \mathit{Sa}),\\
&(\mathit{Account}_2, \mathit{bank}, \mathit{Ca}), (t, \mathit{bank}, M),\\
&(\mathit{Account}_1, t, S), (\mathit{Account}_2, t, T)\}\\
\mathit{links} \coloneqq \{&(\mathit{own\_ca}, \mathit{bank}) \to \{(\mathit{Cu}_1, \mathit{Ca}), (\mathit{Cu}_2, \varepsilon)\},\\
&(\mathit{own\_sa}, \mathit{bank}) \to \{(\mathit{Cu}_1, \varepsilon), (\mathit{Cu}_2, \mathit{Sa})\},\\
&(\mathit{advises}, \mathit{bank}) \to \{(\varepsilon, \mathit{Cu}_1), (\mathit{Con}, \mathit{Cu}_2)\},\\
&(\mathit{trans}, t) \to \{(S, T)\}\}
\end{aligned}
$$

The CROI $\mathfrak{b}$ is created, from Figure 2b, by collecting all the naturals, compartments, and roles accordingly; mapping their respective types; linking the roles to their players; and assigning a tuple for each depicted relationship.[3] Notably, $\mathfrak{b}$ must contain a tuple for the roles Cu_1 and Cu_2 in the *own_ca*, *own_sa* and *advises* relationships regardless of their actual relation to a counter role. These tuples link those roles to the empty counter role ε instead. It can be shown that the CROI $\mathfrak{b}$ is compliant to the CROM $\mathcal{B}$. Due to space limitations, this proof had to be omitted. Next, we introduce three auxiliary functions used to validate both the cardinality and the intra-relationship constraints.

Definition 3 (Auxiliary Functions)**.** *Let RST be the set of relationship types of a well-formed CROM $\mathcal{M}$, and $\mathfrak{i} = (N, R, C, \mathit{type}, \mathit{plays}, \mathit{links})$ a CROI compliant to that model $\mathcal{M}$. Then the auxiliary functions* pred *and* succ*, as well as the* inverse *of the plays-relation for roles $\overline{\cdot} : R^\varepsilon \to O^\varepsilon$ and its extension to the links-function are defined for $r \in R$, $\mathit{rst} \in \mathit{RST}$, and $c \in C$ (with $O^\varepsilon \coloneqq O \cup \{\varepsilon\}$):*

$$
\begin{aligned}
\mathit{pred}(\mathit{rst}, c, r) &\coloneqq \{r' \mid (r', r) \in \mathit{links}(\mathit{rst}, c) \land r' \neq \varepsilon\}\\
\mathit{succ}(\mathit{rst}, c, r) &\coloneqq \{r' \mid (r, r') \in \mathit{links}(\mathit{rst}, c) \land r' \neq \varepsilon\}\\
\overline{r} &\coloneqq \begin{cases} \varepsilon & \textbf{\textit{if}}\ r = \varepsilon \\ o & \textbf{\textit{if}}\ \exists(o, \_, r) \in \mathit{plays} \end{cases}\\
\overline{\mathit{links}(\mathit{rst}, c)} &\coloneqq \{(\overline{r_1}, \overline{r_2}) \mid (r_1, r_2) \in \mathit{links}(\mathit{rst}, c)\}
\end{aligned}
$$

The first two functions collect all the predecessors or successors of a given role in a relationship within a specific compartment instance. For the CROI $\mathfrak{b}$ (Example 2) $\mathit{pred}(\mathit{own\_ca}, \mathit{bank}, \mathit{Ca})$ would return the set containing Cu_1. The existence of the next two functions, i.e., the inverse *plays* and inverse *links*-function, is assured by (8) requiring a unique player and compartment for each role instance. For the bank instance, $\overline{\mathit{links}(\mathit{trans}, t)}$ would return a singleton set with $(\mathit{Account}_1, \mathit{Account}_2)$ This function is used later on to evaluate whether a relationship is irreflexive, surjective, acyclic, and so forth [4, 16].

[3] For brevity, the types of the naturals were omitted from the *type*-function.

4.5 Constraint Level

This section extends the formal model to represent the various constraints by first introducing *Role Groups* as a new construct to specify role constraints; then defining *Constraint Models*; and finally specifying when a given CROI fulfills the imposed constraints.

Definition 4 (Syntax of Role Groups)**.** *Let RT be the set of role types; then the* set of Role Groups *RG is defined inductively:*

- *If $rt \in RT$, then $rt \in RG$, and*
- *If $B \subseteq RG$ and $n, m \in \mathbb{N} \cup \{\infty\}$ with $n \leq m$, then $(B, n, m) \in RG$.*

Definition 5 (Atoms Function)**.** *Let $\mathcal{M} = (NT, RT, CT, RST, \mathit{fills}, \mathit{parts}, \mathit{rel})$ be a well-formed CROM; then $\mathit{atoms} : RG \to 2^{RT}$ is a function, defined as:*

$$
\mathit{atoms}(a) \coloneqq \begin{cases} \{a\} & \textbf{\textit{if}}\ a \in RT \\ \bigcup_{b \in B} \mathit{atoms}(b) & \textbf{\textit{if}}\ a \equiv (B, n, m) \end{cases}
$$

Definition 6 (Semantics of Role Groups)**.** *Let RT be the set of role types of a well-formed CROM $\mathcal{M}$, $\mathfrak{i} = (N, R, C, \mathit{type}, \mathit{plays}, \mathit{links})$ a CROI compliant to $\mathcal{M}$, $c \in C$ a compartment, and $o \in O^c$ an object playing a role in c. Then the* semantics of Role Groups *is defined by the evaluation function $(\cdot)^{\mathcal{I}_o^c} : RG \to \{0, 1\}$:*

$$
a^{\mathcal{I}_o^c} \coloneqq \begin{cases} 1 & \textbf{\textit{if}}\ a \in RT \land \exists(o, c, r) \in \mathit{plays} : \mathit{type}(r) = a \\ & \textbf{\textit{or}}\quad a \equiv (B, n, m) \land n \leq \sum_{b \in B} b^{\mathcal{I}_o^c} \leq m \\ 0 & \textbf{\textit{otherwise}} \end{cases}
$$

Role groups constrain the set of roles an object o is allowed to play simultaneously in a certain compartment c. In case a is a role type, $rt^{\mathcal{I}_o^c}$ checks whether o plays a role of type rt in c. If a is a role group (B, n, m), it checks whether the sum of the evaluations for all $b \in B$ is between n and m.

Example 3 (Role Groups)**.** *The following role groups can be extracted from Figure 2a:*

$$
\begin{aligned}
\mathit{bankaccounts} &\coloneqq (\{\mathit{CA}, \mathit{SA}\}, 1, 1)\\
\mathit{participants} &\coloneqq (\{\mathit{Source}, \mathit{Target}\}, 1, 1)
\end{aligned}
$$

The formal representation of role groups directly correspond to their graphical representation. In general, it can be shown that both Riehle's role constraints [28] and any propositional formula are representable with role groups.[4] As such, both role groups represent *role-prohibitions*, as they model an *exclusive-or*. Likewise, a *role-implication* from consultant to customer would be modeled as: $(\{(\{\mathit{Consultant}\}, 0, 0), \mathit{Customer}\}, 1, 2)$.

[4] Our results indicate that the constraint logic covered by Riehle's constraints is a proper subset of propositional logic.

This, in turn, is equivalent to the formula $\neg Consultant \vee Customer$ and thus to the intended semantics of the *role-implication*.

Definition 7 (Constraint Model)**.** *Let* $\mathcal{M} = (NT, RT, CT, RST, fills, parts, rel)$ *be a well-formed CROM and* $Card \subset \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ *the set of cardinalities represented as* $i..j$ *with* $i \leq j$*. Then* $\mathcal{C} = (rolec, card, intra)$ *is a* Constraint Model over $\mathcal{M}$*, where* $rolec : CT \to 2^{Card \times RG}$ *and* $card : RST \to (Card \times Card)$ *are total functions,* $intra \subseteq RST \times \mathbb{E}$ *is a relation with* $\mathbb{E}$ *as the set of functions* $e : 2^{D \times D} \to \{0, 1\}$*. A Constraint Model is* compliant *to the CROM* $\mathcal{M}$ *if the following axiom holds:*

$$\forall ct \in CT \, \forall (c, a) \in rolec(ct) \colon atoms(a) \subseteq parts(ct) \quad (12)$$

The $atoms : RG \to 2^{RT}$ *function recursively computes all role types within a given role group.*

In detail, *rolec* collects the set of root role groups for each compartment type combined with a cardinality limiting the occurrence of role groups in each compartment, *card* assigns a cardinality to each relationship type, and *intra* is a relation between relationship types and evaluation functions mapping a given set of tuples over $D \times D$ to either zero or one. Moreover, (12) ensures that each role group can only encompass role types that are part of the same compartment type. Notably, all these constraints are defined locally to a compartment type, i.e., no constraint crosses the boundary of a compartment type.

Example 4 (Constraint Model)**.** *Let* $\mathcal{B}$ *be the bank model from Example 1. Then* $\mathcal{C}_\mathcal{B} = (rolec, card, intra)$ *is the constraint model, derived from Figure 2a, where the components are defined as:*

$$\begin{aligned}
rolec \coloneqq \{&Bank \to \{(1..\infty, Consultant), \\
&\quad (0..\infty, bankaccounts)\}, \\
&Transaction \to \{(2..2, participants)\}\} \\
card \coloneqq \{&own_ca \to (1..1, 0..\infty), \\
&own_sa \to (1..\infty, 0..\infty), \\
&advises \to (0..\infty, 1..\infty), \\
&trans \to (1..1, 1..1)\} \\
intra \coloneqq \{&(advises, irreflexive)\}
\end{aligned}$$

Here, $irreflexive(\mathbb{R})$ *returns* 0 *if there is a tuple* $(a, a) \in \mathbb{R}$ *and otherwise* 1*.*

A constraint model can be obtained by basically mapping the graphical constraints to their formal counterparts: role groups with cardinalities to the *rolec*-mapping, relationship cardinality to the *card*-function, and intra relationship constraints to the *intra*-relation. Because each role group contains only role types of the same compartment type (12), $\mathcal{C}_\mathcal{B}$ is compliant to the CROM $\mathcal{B}$. The last step is to define when a given CROI can be considered valid wrt. a constraint model.

Definition 8 (Validity)**.** *Let* $\mathcal{M} = (NT, RT, CT, RST, fills, parts, rel)$ *be a well-formed CROM,* $\mathcal{C} = (rolec, card, intra)$ *a constraint model compliant to* $\mathcal{M}$*, and* $\mathfrak{i} = (N, R, C, type, plays, links)$ *a CROI compliant to* $\mathcal{M}$*. Then* $\mathfrak{i}$ is valid with respect to $\mathcal{C}$ *iff the following conditions hold:*

$$\forall ct \in CT \; \forall (i..j, a) \in rolec(ct) \; \forall c \in C_{ct} : \\ i \leq \left(\sum\nolimits_{o \in O^c} a^{\mathcal{I}^c_o} \right) \leq j \quad (13)$$

$$\forall (o, c, r) \in plays \; \forall (crd, a) \in rolec(type(c)) : \\ type(r) \in atoms(a) \Rightarrow a^{\mathcal{I}^c_o} = 1 \quad (14)$$

$$\forall rst \in RST \; \forall c \in C \; \forall (r_1, r_2) \in links(rst, c) : \\ card(rst) = (i..j, k..l) \wedge \\ \left(r_2 \neq \varepsilon \Rightarrow i \leq \left|pred(rst, c, r_2)\right| \leq j\right) \wedge \\ \left(r_1 \neq \varepsilon \Rightarrow k \leq \left|succ(rst, c, r_1)\right| \leq l\right) \quad (15)$$

$$\forall c \in C \; \forall (rst, f) \in intra : links(rst, c) = \emptyset \vee \\ f(\overline{links(rst, c)}) = 1 \quad (16)$$

Here, $C_{ct} \coloneqq \{c \in C \mid type(c) = ct\}$ *denotes the subset of* C *containing only instances of type* $ct \in CT$*.*

Each axiom verifies a particular set of constraints. The first two validate the occurrence and fulfillment of role groups, such that only those objects (naturals or compartments) are checked that play a corresponding role in the constrained compartment (14), and such that there are enough of such objects in that compartment (13). In contrast to them, (15) checks whether relationships respect the imposed cardinality constraints. Last, (16) applies the evaluation function to the set of players in a relationship by instantiating the domain $D \times D$ of this function to $O^\varepsilon \times O^\varepsilon$. Finally, the formal model is not only able to capture the relational and context-dependent nature of roles, but also allows for the validation of various constraints imposed on these models.

Example 5 (Validity)**.** *To prove that the instance* $\mathfrak{b}$ *(Example 2) of the bank model* $\mathcal{B}$ *is* ***valid*** *wrt. the constraint model* $\mathcal{C}_\mathcal{B}$*, each of these axioms must hold.*

Proof. For the first axiom (13), we have to investigate the occurrence of *Consultant*:

$$\begin{aligned}
&\big((Peter, bank, Con) \in plays \wedge \\
&\quad type(Con) = Consultant\big) \Rightarrow Consultant^{\mathcal{I}^{bank}_{Peter}} = 1 \\
\Longrightarrow &\forall c \in \{bank\} : 1 \leq \left(\sum\nolimits_{o \in O^{bank}} Consultant^{\mathcal{I}^c_o} \right)
\end{aligned}$$

and that exactly 2 objects fulfill the *participants* role group, firstly $Account_1$:

$$\begin{aligned}&\big((\text{Account}_1, t, r) \in plays \wedge type(S) = Source \wedge\\ &\quad \nexists(Account_1, t, r) \in plays : type(r) = Target\big)\\ \Longrightarrow &Source^{\mathcal{I}^{bank}_{Account_1}} = 1 \wedge Target^{\mathcal{I}^{bank}_{Account_1}} = 0\\ \Longrightarrow &\text{participants}^{\mathcal{I}^{bank}_{Account_1}} = 1\end{aligned}$$

secondly $Account_2$:

$$\begin{aligned}&\big((Account_2, t, T) \in plays \wedge type(T) = Target \wedge\\ &\quad \nexists(Account_2, t, r) \in plays : type(r) = Source\big)\\ \Longrightarrow &Source^{\mathcal{I}^{bank}_{Account_1}} = 1 \wedge Target^{\mathcal{I}^{bank}_{Account_1}} = 0\\ \Longrightarrow &\text{participants}^{\mathcal{I}^{bank}_{Account_1}} = 1\\ \Longrightarrow &\forall c \in \{t\} : (\textstyle\sum_{o \in O^c} \text{participants}^{\mathcal{I}^c_o}) = 2\big)\end{aligned}$$

Because there exist three naturals *Peter*, $Account_1$, and $Account_2$ satisfying the two role constraints *Consultant* and *participants*, respectively, axiom (13) holds for $\mathfrak{b}$.

Next, we validate axiom (14). From the previous proof we have seen that $Account_1$ already fulfills the *participants* role group. Thus, we only check the *bankaccounts* role group for its fulfillment:

$$\begin{aligned}&\big((Account_1, bank, Sa) \in plays \wedge type(Sa) = SA \wedge\\ &\quad \nexists(Account_1, bank, r) \in plays : type(r) = CA\big)\\ \Longrightarrow &\text{bankaccounts}^{\mathcal{I}^{bank}_{Account_1}} = 1\end{aligned}$$

Likewise, $Account_2$ is checked for the fulfillment of this role group:

$$\begin{aligned}&\big((Account_2, bank, Ca) \in plays \wedge type(Ca) = CA \wedge\\ &\quad \nexists(Account_2, bank, r) \in plays : type(r) = SA\big)\\ \Longrightarrow &\text{bankaccounts}^{\mathcal{I}^{bank}_{Account_2}} = 1\end{aligned}$$

As a consequence, axiom (14) holds because *Peter* plays the *Consultant* role and each account fulfills both the *bankaccounts* and *participants* role group.

For the next axiom, it suffices to evaluate the following expression to validate axiom (15) for the bank model $\mathcal{B}$ with the two compartments *bank* and t of $\mathfrak{b}$.

$$\begin{aligned}&\Big(\forall(r_1, r_2) \in links(own\_ca, bank) :\\ &\quad r_2 \neq \varepsilon \Rightarrow \big|pred(own\_ca, bank, r_2)\big| = 1\Big)\\ \wedge&\Big(\forall(r_1, r_2) \in links(own\_sa, bank) :\\ &\quad r_2 \neq \varepsilon \Rightarrow 1 \leq \big|pred(own\_sa, bank, r_2)\big|\Big)\\ \wedge&\Big(\forall(r_1, r_2) \in links(advises, bank) :\\ &\quad r_1 \neq \varepsilon \Rightarrow 1 \leq \big|succ(advises, bank, r_1)\big|\Big)\\ \wedge&\Big(\forall(r_1, r_2) \in links(trans, t) :\\ &\quad r_2 \neq \varepsilon \Rightarrow \big|pred(trans, t, r_2)\big| = 1 \wedge\\ &\quad r_1 \neq \varepsilon \Rightarrow \big|succ(trans, t, r_1)\big| = 1\Big)\end{aligned}$$

Next, the *links*-function is replaced by its definition in Example 3:

$$\begin{aligned}&\Big(\forall(r_1, r_2) \in \{(Cu_1, Ca), (Cu_2, \varepsilon)\} :\\ &\quad r_2 \neq \varepsilon \Rightarrow \big|pred(own\_ca, bank, r_2)\big| = 1\Big)\\ \wedge&\Big(\forall(r_1, r_2) \in \{(Cu_1, \varepsilon), (Cu_2, Sa)\} :\\ &\quad r_2 \neq \varepsilon \Rightarrow 1 \leq \big|pred(own\_sa, bank, r_2)\big|\Big)\\ \wedge&\Big(\forall(r_1, r_2) \in \{(\varepsilon, Cu_1), (Con, Cu_2)\} :\\ &\quad r_1 \neq \varepsilon \Rightarrow 1 \leq \big|succ(advises, bank, r_1)\big|\Big)\\ \wedge&\Big(\forall(r_1, r_2) \in \{(S, T)\} :\\ &\quad r_2 \neq \varepsilon \Rightarrow \big|pred(trans, t, r_2)\big| = 1 \wedge\\ &\quad r_1 \neq \varepsilon \Rightarrow \big|succ(trans, t, r_1)\big| = 1\Big)\end{aligned}$$

It is easy to see that the above conjunction is satisfied for these tuple sets.

To verify the last axiom (16), it suffices to validate the inverse of the links function for the *advises* relationship type of the *bank* compartment:

$$\begin{aligned}&irreflexive(\overline{links(advises, bank)}) = 1\\ \Longrightarrow &irreflexive(\{(\varepsilon, Klaus), (Peter, Google)\}) = 1\end{aligned}$$

Trivially, the tuple set is irreflexive, and thus axiom (16) is satisfied, as well.

The CROI $\mathfrak{b}$ is a **valid** instance of the model $\mathcal{B}$ wrt. the constraint model $\mathcal{C}_{\mathcal{B}}$, because it satisfies all axioms. □

Despite of this informal validation, the idea of our formal model is to support both formal and automated validation of well-formedness, compliance, and validity.

5. Complete Walkthrough

After introducing the complete formal model this section illustrates the process of formally evaluating a given model instance. To do this formalization of another example, prove its compliance, and check its validity.

5.1 Graphical Model

Figure 3 shows another possible instance of the banking model (Figure 2a). It features the **Persons** *Peter* and *Klaus* playing a **Customer** role in the *bank* compartment both owning the checking account *Ca*. Moreover, the **Company** *Google* is another customer of the bank owning the savings account *Sa*. In addition to that, the bank manages the money transfer from $Account_2$ to $Account_2$ within the **Transaction** *t*.

In comparison to the first instance (Example 2), this instance does not encompass a person playing the **Consultant** role. Hence, this instance apears be not a valid instance of the bank model $\mathcal{B}$. However, to proof this, we need to formalize this instance and check if it is compliant to the bank $\mathcal{B}$ and valid wrt. the constraint model $\mathcal{C}_\mathcal{B}$.

5.2 Formalization

Example 6 (Second Compartment Role Object Instance)**.** *Let $\mathcal{B} = (NT, RT, CT, RST, fills, parts, rel)$ be the well-formed CROM defined in Example 1; then $\mathfrak{a} = (N, R, C, type, plays, links)$ is the second instance of that model (Figure 3), where the set of naturals is defined as $N = \{Peter, Klaus, Google, Account_1, Account_2\}$, the set of roles as $R = \{Cu_1, Cu_2, Cu_3, Ca, Sa, S, T, M\}$, and the set of compartments as $C = \{bank, t\}$. The other components are defined, as:*

$$\begin{aligned} type \coloneqq \{&(Peter \to Person), (Klaus \to Person),\\ &(Google \to Company), (Account_1 \to Account),\\ &(Account_2 \to Account), (Cu_1 \to Customer),\\ &(Cu_2 \to Customer), (Cu_3 \to Customer),\\ &(Ca \to CA), (Sa \to SA), (S \to Source),\\ &(T \to Target), (M \to MoneyTransfer),\\ &(t \to Transaction), (bank \to Bank)\} \end{aligned}$$

After defining the type function, the plays-mapping and links-function are defined as:

$$\begin{aligned} plays \coloneqq \{&(Klaus, bank, Cu_1), (Google, bank, Cu_2),\\ &(Peter, bank, Cu_3), (Account_1, bank, Sa),\\ &(Account_2, bank, Ca), (t, bank, M),\\ &(Account_2, t, S), (Account_2, t, T)\} \end{aligned}$$

$$\begin{aligned} links(own\_ca, bank) &\coloneqq \{(Cu_1, Ca), (Cu_2, \varepsilon), (Cu_3, Ca)\}\\ links(own\_sa, bank) &\coloneqq \{(Cu_1, \varepsilon), (Cu_2, Sa), (Cu_3, \varepsilon)\}\\ links(advises, bank) &\coloneqq \{(\varepsilon, Cu_1), (\varepsilon, Cu_2), (\varepsilon, Cu_3)\}\\ links(trans, t) &\coloneqq \{(S, T)\} \end{aligned}$$

Example 7 (Compliance of the Second Instance)**.** *The instance $\mathfrak{a} = (N, R, C, type, plays, links)$ is **compliant** to the bank model $\mathcal{B}$.*

Proof. To show that $\mathfrak{a}$ is *compliant* to the bank model $\mathcal{B}$, we need to proof that the axioms from (6) to (11) hold for $\mathfrak{a}$. For (6), the type conformance of the *plays*-relation is checked.

$$\begin{aligned} (6) \Rightarrow &\big((type(Klaus), type(Cu_1)) \in fills \wedge\\ &\quad type(Cu_1) \in parts(type(bank))\big)\\ &\wedge\big((type(Google), type(Cu_2)) \in fills \wedge\\ &\quad type(Cu_2) \in parts(type(bank))\big)\\ &\wedge\big((type(Peter), type(Con)) \in fills \wedge\\ &\quad type(Con)) \in parts(type(bank))\big)\\ &\wedge\big((type(Account_1), type(Sa)) \in fills \wedge\\ &\quad type(Sa) \in parts(type(bank))\big)\\ &\wedge\big((type(Account_2), type(Ca)) \in fills \wedge\\ &\quad type(Ca) \in parts(type(bank))\big)\\ &\wedge\big((type(t), type(M)) \in fills \wedge\\ &\quad type(M) \in parts(type(bank))\big)\\ &\wedge\big((type(Account_1), type(S)) \in fills \wedge\\ &\quad type(S) \in parts(type(t))\big)\\ &\wedge\big((type(Account_2), type(T)) \in fills \wedge\\ &\quad type(T) \in parts(type(t))\big)\\ \Rightarrow &\big((Person, Customer) \in fills \wedge\\ &\quad Customer \in parts(Bank)\big)\\ &\wedge\big((Company, Customer) \in fills \wedge\\ &\quad Customer \in parts(Bank)\big)\\ &\wedge\big((Person, Consultant) \in fills \wedge\\ &\quad Consultant \in parts(Bank)\big)\\ &\wedge\big((Account, SA) \in fills \ \wedge \ SA \in parts(Bank)\big)\\ &\wedge\big((Account, CA) \in fills \ \wedge \ CA \in parts(Bank)\big)\\ &\wedge\big((Transaction, MoneyTransfer) \in fills \wedge\\ &\quad MoneyTransfer \in parts(Bank)\big)\\ &\wedge\big((Account, Source) \in fills \wedge\\ &\quad Source \in parts(Transaction)\big)\\ &\wedge\big((Account, Target) \in fills \wedge\\ &\quad Target \in parts(Transaction)\big) \end{aligned}$$

As the *plays*-relation is type conform to the *fills*-relation and the *parts*-function, $\mathfrak{a}$ fulfills (6).

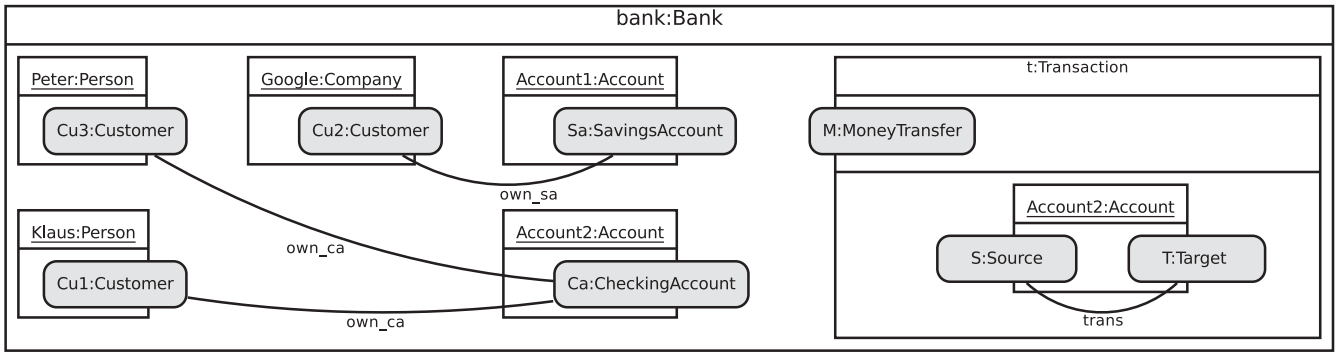


Figure 3: Another Compartment Role Object Instance of the banking application

For (7) each object playing multiple roles must be checked. Thus, only $Account_2$ needs to be checked:

$$(7) \Rightarrow \big(S \neq T \Rightarrow type(S) \neq type(T)\big) \Rightarrow \big(Source \neq Target\big)$$

Consequently, $\mathfrak{a}$ also satisfies (7).

For (8), we have to ensure that for each role there exists a unique player and compartment.

$$\begin{aligned}(8) \Rightarrow & \forall r \in \{Cu_1, Cu_2, Cu_3, Ca, Sa, S, T, M\} \\ & \exists! o \in O\ \exists! c \in C:\ (o,c,r) \in plays \\ & \Rightarrow \big((Klaus, bank, Cu_1) \in plays \\ & \wedge\ (Google, bank, Cu_2) \in plays \\ & \wedge\ (Peter, bank, Cu_3) \in plays \\ & \wedge\ (Account_1, bank, Sa) \in plays \\ & \wedge\ (Account_2, bank, Ca) \in plays \\ & \wedge\ (t, bank, M) \in plays \\ & \wedge\ (Account_2, t, S) \in plays \\ & \wedge\ (Account_2, t, T) \in plays\big)\end{aligned}$$

As for every role there is exactly one tuple in the *plays*-relation, axiom (8) is satisfied by $\mathfrak{a}$.

For (9), the *links*-function should never return $(\varepsilon, \varepsilon)$.

$$\begin{aligned}(9) \equiv & \forall rst \in RST\ \forall c \in C:\ (\varepsilon,\varepsilon) \notin links(rst, c) \\ & \Rightarrow \big((\varepsilon,\varepsilon) \notin links(own\_ca, bank) \wedge \\ & \quad (\varepsilon,\varepsilon) \notin links(own\_sa, bank) \wedge \\ & \quad (\varepsilon,\varepsilon) \notin links(advises, bank) \wedge \\ & \quad (\varepsilon,\varepsilon) \notin links(trans, t)\big) \\ & \Rightarrow \big((\varepsilon,\varepsilon) \notin \{(Cu_1, Ca), (Cu_2, \varepsilon), (Cu_3, Ca)\} \wedge \\ & \quad (\varepsilon,\varepsilon) \notin \{(Cu_1, \varepsilon), (Cu_2, Sa), (Cu_3, \varepsilon)\} \wedge \\ & \quad (\varepsilon,\varepsilon) \notin \{(\varepsilon, Cu_1), (\varepsilon, Cu_2), (\varepsilon, Cu_3)\} \wedge \\ & \quad (\varepsilon,\varepsilon) \notin \{(S, T)\}\big)\end{aligned}$$

Hence, (9) is obviously satisfied.

For (10), the correspondence between the *plays*-relation and *links*-function must be ensured.

$$\begin{aligned}(10) \Rightarrow & \forall rst \in \{own\_ca, own\_sa, advises, trans\} \\ & \forall c \in \{bank, t\}\ \forall r \in R\ \forall o \in O\ \exists \hat{r} \in R^{\varepsilon}: \\ & rel(rst) = (rt_1, rt_2) \wedge \\ & \big(\big((o,c,r) \in plays \wedge type(r) = rt_1\big) \\ & \quad \Leftrightarrow \big((r,\hat{r}) \in links(rst, c)\big)\big) \wedge \\ & \big(\big((o,c,r) \in plays \wedge type(r) = rt_2\big) \\ & \quad \Leftrightarrow \big((\hat{r}, r) \in links(rst, c)\big)\big) \\ \Rightarrow & \forall r \in R\ \forall o \in O\ \exists \hat{r} \in R^{\varepsilon}: \\ & \big(\big((o, bank, r) \in plays \wedge type(r) = Customer\big) \\ & \quad \Leftrightarrow \big((r,\hat{r}) \in links(own\_ca, bank)\big)\big) \wedge \\ & \big(\big((o, bank, r) \in plays \wedge type(r) = CA\big) \\ & \quad \Leftrightarrow \big((\hat{r}, r) \in links(own\_ca, bank)\big)\big) \\ \wedge & \forall r \in R\ \forall o \in O\ \exists \hat{r} \in R^{\varepsilon}: \\ & \big(\big((o, bank, r) \in plays \wedge type(r) = Customer\big) \\ & \quad \Leftrightarrow \big((r,\hat{r}) \in links(own\_sa, bank)\big)\big) \wedge \\ & \big(\big((o, bank, r) \in plays \wedge type(r) = SA\big) \\ & \quad \Leftrightarrow \big((\hat{r}, r) \in links(own\_sa, bank)\big)\big) \\ \wedge & \forall r \in R\ \forall o \in O\ \exists \hat{r} \in R^{\varepsilon}: \\ & \big(\big((o, bank, r) \in plays \wedge type(r) = Consultant\big) \\ & \quad \Leftrightarrow \big((r,\hat{r}) \in links(advises, bank)\big)\big) \wedge \\ & \big(\big((o, bank, r) \in plays \wedge type(r) = Customer\big) \\ & \quad \Leftrightarrow \big((\hat{r}, r) \in links(advises, bank)\big)\big) \\ \wedge & \forall r \in R\ \forall o \in O\ \exists \hat{r} \in R^{\varepsilon}: \\ & \big(\big((o, t, r) \in plays \wedge type(r) = Source\big) \\ & \quad \Leftrightarrow \big((r,\hat{r}) \in links(trans, t)\big)\big) \wedge \\ & \big(\big((o, t, r) \in plays \wedge type(r) = Target\big) \\ & \quad \Leftrightarrow \big((\hat{r}, r) \in links(trans, t)\big)\big)\end{aligned}$$

Table 1 shows the variable assignments that satisfy the previous formula (excluding all trivial assignments). Hence, $\mathfrak{a}$ fulfills Axiom (10).

Table 1: Fulfilling variable assignments for Axiom (10)

rst	c	r	o	$\hat{r}$
own_ca	$bank$	Cu_1	$Klaus$	Ca
own_ca	$bank$	Cu_2	$Google$	ε
own_ca	$bank$	Cu_3	$Peter$	Ca
own_ca	$bank$	Ca	$Account_2$	Cu_1
own_sa	$bank$	Cu_1	$Klaus$	ε
own_sa	$bank$	Cu_2	$Google$	Sa
own_sa	$bank$	Cu_3	$Peter$	ε
own_sa	$bank$	Sa	$Account_1$	Cu_2
$advises$	$bank$	Cu_1	$Klaus$	ε
$advises$	$bank$	Cu_2	$Google$	ε
$advises$	$bank$	Cu_3	$Peter$	ε
$trans$	t	S	$Account_2$	T
$trans$	t	T	$Account_2$	S

For (11), we have to check all links between two role instances.

$$
\begin{aligned}
(11) \Rightarrow & \forall rst \in \{own\_ca, own\_sa, advises, trans\} \\
& \forall c \in \{bank, t\}\ \forall (r_1, r_2) \in links(rst, c) \cap R \times R : \\
& (r_1, \varepsilon), (\varepsilon, r_2) \notin links(rst, c) \\
\Rightarrow & \big(\forall (r_1, r_2) \in \{(Cu_1, Ca), (Cu_3, Ca)\} : \\
& (r_1, \varepsilon), (\varepsilon, r_2) \notin \{(Cu_1, Ca), (Cu_2, \varepsilon), (Cu_3, Ca)\}\big) \\
\wedge & \big(\forall (r_1, r_2) \in \{(Cu_2, Sa)\} \\
& (r_1, \varepsilon), (\varepsilon, r_2) \notin \{(Cu_1, \varepsilon), (Cu_2, Sa), (Cu_3, \varepsilon)\}\big) \\
\wedge & \big(\forall (r_1, r_2) \in \{(S, T)\} \\
& (r_1, \varepsilon), (\varepsilon, r_2) \notin \{(S, T)\}\big)
\end{aligned}
$$

As each conjunction is satisfied, (11) is fulfilled.

Because each of the six axioms is fulfilled, $\mathfrak{a}$ is proven to be compliant to the bank model $\mathcal{B}$. □

The formalization of the second instance and the verification of its compliance are only the initial steps in validating its conformance to the modeled domain.

5.3 Validation

The final step in evaluating a CROI is to check its validity wrt. to a given constraint model. While the initial example was a valid instance wrt. $\mathcal{C}_\mathcal{B}$, the second instance will be our negative example.

Example 8 (Validity of the Second Instance)**.** *The instance $\mathfrak{a}$ of the bank model $\mathcal{B}$ is **not valid** wrt. the constraint model $\mathcal{C}_\mathcal{B}$.*

Proof. To show that $\mathfrak{a}$ is *not valid* wrt. the constraint model $\mathcal{C}_\mathcal{B}$, it suffices that one of the axioms from (13) to (16) is violated. Nevertheless, we inspect all violated axioms to give a detailed explanation of the various constraints. Initially, we assume that axiom (13) holds for a and deduce a contradiction:

$$
\begin{aligned}
(13) \Longrightarrow & \forall (i..j, a) \in rolec(Bank)\ \forall c \in C_{Bank} : \\
& i \leq \big(\sum\nolimits_{o \in O^c} a^{\mathcal{I}_o^c}\big) \leq j \\
\Longrightarrow & \forall (i..j, a) \in \{(1..\infty, Consultant),\ \dots\} \\
& \forall c \in C_{Bank} : i \leq \big(\sum\nolimits_{o \in O^c} a^{\mathcal{I}_o^c}\big) \leq j \\
\Longrightarrow & \forall c \in \{bank\} : 1 \leq (\sum\nolimits_{o \in O^c} Consultant^{\mathcal{I}_o^c}) \\
\Longrightarrow & 1 \leq (\sum\nolimits_{o \in O^{bank}} Consultant^{\mathcal{I}_o^{bank}}) \\
\Longrightarrow & (\exists o \in O^{bank} : Consultant^{\mathcal{I}_o^{bank}} = 1) \\
\Longrightarrow & \exists (o, bank, r) \in plays : type(r) = Consultant
\end{aligned}
$$

This is violated by the fact that the *plays*-relation does not contain any role of type *Consultant* currently played in the *bank* compartment.

Similarly, we assume that Axiom (14) holds for the CROI $\mathfrak{a}$ and deduce a contradiction.

$$
\begin{aligned}
(14) \Longrightarrow & \forall (o, t, r) \in plays \\
& \forall (i..j, a) \in rolec(Transaction) : \\
& type(r) \in atoms(a) \Rightarrow a^{\mathcal{I}_o^c} = 1 \\
\Longrightarrow & \forall (o, t, r) \in plays \\
& \forall (i..j, a) \in \{(2..2, \text{participants})\} : \\
& type(r) \in atoms(a) \Rightarrow a^{\mathcal{I}_o^t} = 1 \\
\Longrightarrow & \forall (o, t, r) \in plays : type(r) \in \{Source, Target\} \\
& \Rightarrow \text{participants}^{\mathcal{I}_o^t} = 1
\end{aligned}
$$

To resolve this implication, we select $(Account_2, t, S) \in plays$ as a candidate, as it fulfills the precondition $type(S) \in \{Source, Target\}$.

$$
\begin{aligned}
& \text{participants}^{\mathcal{I}^t_{Account_2}} = 1 \\
\Longrightarrow & (\{Source, Target\}, 1, 1)^{\mathcal{I}^t_{Account_2}} = 1 \\
\Longrightarrow & 1 \leq \big(\sum\nolimits_{b \in \{Source, Target\}} b^{\mathcal{I}^t_{Account_2}}\big) \leq 1 \\
\Longrightarrow & \big(Source^{\mathcal{I}^t_{Account_2}} + Target^{\mathcal{I}^t_{Account_2}}\big) = 1
\end{aligned}
$$

This is a contradiction, because $Account_2$ plays both the roles S of type *Source* and T of type *Target* in the compartment t. This, in turn, means that the result of the last addition is two instead of one. Again, this axiom is violated by the CROI $\mathfrak{a}$.

Finally, let us suppose that axiom (15), ensuring the cardinality of relationship types, is satisfied by $\mathfrak{a}$, then unrolling the axiom for $own\_ca \in RST$ in $\mathcal{B}$ results in

the following formula:

$$
\begin{aligned}
(15) \Longrightarrow & \forall c \in C\ \forall (r_1, r_2) \in \mathit{links}(\mathit{own\_ca}, c) : \\
& \mathit{card}(\mathit{own\_ca}) = (1..1, 0..\infty) \wedge \\
& \big(r_2 \neq \varepsilon \Rightarrow 1 \leq \big|\mathit{pred}(\mathit{own\_ca}, c, r_2)\big| \leq 1\big) \wedge \\
& \big(r_1 \neq \varepsilon \Rightarrow 0 \leq |\mathit{succ}(\mathit{own\_ca}, c, r_1)| \leq \infty\big) \\
\Longrightarrow & \forall c \in C\ \forall (r_1, r_2) \in \mathit{links}(\mathit{own\_ca}, c) : \\
& r_2 \neq \varepsilon \Rightarrow \big|\mathit{pred}(\mathit{own\_ca}, c, r_2)\big| = 1
\end{aligned}
$$

This entails that this also holds for the *bank* compartment and the link $(Cu_3, Ca) \in \mathit{links}(\mathit{own\_ca}, \mathit{bank})$.

$$
\begin{aligned}
& \Longrightarrow \big|\mathit{pred}(\mathit{own\_ca}, \mathit{bank}, Ca)\big| = 1 \\
& \Longrightarrow \big|\{Cu_2, Cu_3\}\big| = 1
\end{aligned}
$$

This is an obvious contradiction, and thus, a violation of axiom (15).

Notably, Axiom (16) is the only one satisfied by a, because the inverse of $\mathit{links}(\mathit{advises}, \mathit{bank})$ is obviously irreflexive, as there are no *Consultant* roles in $\mathfrak{a}$.

In conclusion, the CROI $\mathfrak{a}$ is an **invalid** instance of the CROM $\mathcal{B}$ wrt. the constraint model $\mathcal{C}_\mathcal{B}$, because at least one axiom is violated. □

Finally, this example shows the simplicity of the formal model and formal proofs. Despite of that, the proofs are merely technical and can be automated easily.

6. Discussion

This section indicates the adequacy and sufficiency of the presented formal model by discussing reference implementations developed as a proof of concept; classifying our approach; and pointing out limitations.

6.1 Reference Implementation

The formal model solely relies on set semantics and first-order logic. Hence, it is readily applicable for implementation and thus automation. To prove this, two reference implementations were developed based on *Python*[5] and *Scala*[6], respectively. These implementations can be used to create CROMs, CROIs and constraint models, as well as automatically check their well-formedness, compliance, and validity. The provided implementations directly correspond to the formal definitions. In particular, each individual axiom is implemented by means of *all* and *any* functions and generator expressions that directly correspond to *universal* and *existential* quantification in first-order logic. Besides implementing the banking application, a series of tests have been specified to check the implemented axioms not only testing positive and negative cases but also the various combinations

[5] https://github.com/Eden-06/formalCROM

[6] https://github.com/max-leuthaeuser/ScalaFormalCROM

$$\forall rt \in RT\ \exists t \in (NT \cup CT) : (t, rt) \in \text{fills}$$

(a) Formal

```
1  def axiom1(crom):
2    return all( any( (t,rt) in crom.fills
         for t in (crom.nt | crom.ct) )
       for rt in crom.rt)
```

(b) Python

Figure 4: Representation of Axiom (1)

for the 16 axioms. This indicates the orthogonality of the axioms, i.e. that no axiom entails another. In sum, the reference implementations can be used to develop and test other implementations of our formal model, as well as to investigate specializations and extensions to our formal model.

As pointed out previously, we translated each axiom to a function returning a boolean. Figure 4 shows an example of such a translation, for the first axiom. Basically, the *universal quantification* $\forall rt \in RT$ is written as `all( ... for rt in crom.rt )` and the *existential quantification* $\exists t \in (NT \cup CT)$ as `any( ... for t in (crom.nt | crom.ct) )`, where `crom.nt | crom.ct` computes the union of NT and CT. The final test `(t,rt) in crom.fills` is then embedded into these *generator* expressions resulting in Figure 4b. An excerpt of the *Python* implementation is presented in the Appendix.

6.2 Classification

To confirm the adequacy of our formal model to capture both natures of roles, we apply the 26 features of roles [22, 30] to our model. Table 2 summarizes the classification of our approach and compares it to other related approaches. In fact, only 23 features apply to modeling languages without operational semantics [22]. As a result, our formal model fully supports 17 features of roles, whereas only one feature is *possible* to represent, namely Feature 24 stating that compartments can contain compartments. In our model this can be simulated by having the contained compartment play a role in the container compartment. For instance, the *transaction* compartment is contained inside the *bank* compartment, because it is playing the *MoneyTransfer* role in the *bank* (Figure 2 b). In turn, only five features are not supported by our model. Feature 8, for instance, stating that roles can play roles could have been easily modeled within our formalization. However, we argue that there is no difference between a football player playing the role of a striker or a person playing both roles at the same time. As it turns out, the underlying rationale that only football players can be strikers would simply be modeled as a role constraint. For similar reasons, we

disregard Feature 21, stating that a role can be part of several compartments. Although a role is existentially dependent on a compartment, it would not make sense to define a role within two compartments. Arguably, while it makes sense to have a customer role in both a shop and a bank, they do not have the same properties and thus can not have the same type. In sum, the presented formal model supports not only most features of roles but also both natures of roles.

6.2.1 Limitations

Despite that, the formal model has three known shortcomings. First, the current constraint model does not encompass inter-relationship constraints (Feature 17). Consequently, neither subset nor mutual exclusion constraints between relationships can be expressed. Nevertheless, these constraints can be added easily by augmenting the constraint model adding a relation as well as compliance and validation rules. Second, the current constraint model does not permit constraints between two compartments. This restriction hinders the specification of role constraints between compartments, e.g., that each account participating in a *Transaction* compartment must also play the role of either a savings or checking account in a *Bank* compartment. To integrate global constraints like these, the notion of role groups, the constraint model, and their validation must be extended. This is not trivial, because it includes quantifying over multiple compartment instances within one constraint. The next shortcoming is the lack of inheritance among natural types, role types, as well as compartment types (Features 13, 25). It was left out, because it entails several semantical issues and unresolved questions regarding the interaction of natural inheritance and role inheritance together with family polymorphism [10] via compartment inheritance. Thus, adding inheritance must be postponed until these semantical issues can be resolved. At large, while the last two limitations are incurred by major semantical issues, all of them can be resolved by augmenting the presented formal definitions.

7. Related Work

This section compares related role-based modeling languages by means of the 26 features of roles [22, 30], introduced in Section 3. A general comparison of these related role-based modeling languages, is extracted from [22] and depicted in Table 2. Henceforth, we distinguish four classes of related approaches wrt. the two natures of roles they support, namely *plain*, *relational*, *context-dependent*, and *hybrid*.

The *Generic Role Model for Dynamic Objects* [9] belongs to the first class, because its roles neither depend on a relationship nor on any kind of compartment. Hence, the presented formalism focuses solely on the dynamics of the *plays*-relation between roles and their players. Besides that, it provided a semi-formal model for the type and the instance level including an operational semantics based on guarded role transitions. On the down side, the presented semantics is rather limited.

Next, *relational approaches*, have already been surveyed in the year 2000 by Steimann [30]. In an effort to unify and formalize preceding role-based modeling languages, like [6, 8, 17], he introduced *LODWICK* as a unified role-based modeling language [30]. Like our ap-

Feature	**Lodwick** [30]	**Generic Role Model** [9]	**ORM 2** [16]	**E-CARGO Model** [31]	**Metamodel for Roles** [11]	**INM** [23]	**DCI** [27]	**Onto-UML** [15]	**Helena Approach** [18]	**RSQL** [21]	**formal CROM**
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.	□	□	■	■	⊞	■	■	□	■	□	■

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

Table 2: Comparison of role-based modeling languages, extracted from [22].

proach, its formal model focused on the structure of role models. Additionally, it supports the definition of n-ary relations between naturals and includes two disjoint inheritance relations. Despite all that its instance model does not include role instances, and thus cannot capture the structure and features of roles at runtime. Henceforth, we investigate more recent approaches featuring relational roles. *Onto-UML* [15] is a top level ontology including roles. It uses similar ontological predicates to distinguish *Naturals*, *Roles* and *Relators* for both universals (types) and individuals (instances). However, the model lacks a notion of context-dependence and the ability to let unrelated objects play the same role. *Object-Role Modeling* (ORM) 2 [16] is a well-established, fact-oriented data modeling language. However, it only includes roles as unnamed places at the ends of relationships connecting *entity types* [16]. Nevertheless, ORM supports a large number of constraints for these relationships including role constraints, inter- and intra-relationship constraints [16]. *RSQL* [20, 21] is a role-based query language for a role-based database system. It provides a formal data model featuring *Naturals*, *Roles*, and *Relationship Types* on both the type and instance level. This model has many similarities to our formalization of CROMs and CROIs, e.g., the notion of ε roles or cardinality constraints for relationships. Nonetheless, it features neither context-dependent roles nor other kinds of role or relationship constraints.

In contrast, the next approaches have introduced *context-dependent* roles to modeling languages. The *E-CARGO* model [31], introduced for computer-supported cooperative work, distinguishes between several entities ranging from *Agents* playing *Roles* defined in either *Environments* or *Groups*. Still, only agents can play roles and the model only includes occurrence constraints for roles. The *Metamodel for Roles* [11] tries to be the most general formalization of context-dependent roles. Similar to our model, it distinguishes between *Players*, *Roles*, and *Context* on the type and the instance level. Moreover, it introduces properties and inheritance for each of these kinds [11]. Yet, the metamodel is too general to be useful, because the sets of entities are not required to be disjoint (on both the type and instance level). Thus, every definition might effect the same entity, rendering the three distinct inheritance relations useless. Moreover, each definition is accompanied by a set of unspecified constraints to capture the desired structure of the metamodel; things a metamodel should capture at least. The *Information Networking Model* (INM) [23] is a data modeling approach [23] designed to overcome the inability of data models to capture context-dependent information. While this approach allows to model nested *Contexts* with attributes containing *Roles*, the various kinds of relations cannot be constrained [23]. *Data Context Interaction* (DCI) [27] is a new paradigm beyond object-oriented design that revolves around the notions of *Data* playing *Roles* in interactions encapsulated in a *Context*. Although the paradigm is described both abstractly and by example, its semantics is not formally specified.

The only hybrid model, to our best knowledge, is presented in the *HELENA* approach [18]. It features *Ensembles* as compartments to capture a collaborative task by means of roles that are played by *Components*. An *Ensemble (Structure)* contains a set of *Role Connectors* that act as directed communication channels between roles. In particular, *HELENA* provides formal definitions for both type and instance level, as well as an operational semantics based on sets and labeled transition systems [18]. In contrast to our model, role connectors are not bidirectional like our relationships. Furthermore, *HELENA* only supports occurrence constraints on roles, and none of the other kinds of constraints.

8. Conclusion

This work is based on the classification of roles by [22, 30] and the family of role-based modeling languages, proposed in [22]. However, as our goal was to provide a comprehensive definition for role-based modeling, the presented model only encompasses simple, formal definitions for the type and instance level. Additionally, it comprises definitions for cardinality and intra-relationship constraints, as well as the newly introduced role groups. Moreover, we have shown that our formal model is suitable for both manual and automatic evaluation of well-formedness, compliance, and validity. Finally, the provided reference implementations can be used to apply, further explore, and extend our role-based modeling language.

In future, we will augment the formal model to include inheritance for naturals and compartments, as well as global role constraints. Moreover, our goal is to use the formal model as a reference to develop a customizable family of formal role-based modeling languages with full tool support, e.g.: graphical editor, schema, and code generators. In fact, our goal is to include these rules into an integrated development environment for role-based systems to check the well-formedness of role models as well as the compliance and validity of their instances.

Acknowledgments

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) and in the Collaborative Research Center 912 "Highly Adaptive Energy-Efficient Computing". Special thanks go to Sebastian Richly.

References

[1] C. W. Bachman, M. Daya, C. W. Bachman, and M. Daya. The role concept in data models. In *Proceedings of the third international conference on Very Large Data Bases*, volume 3, pages 464–476, 1977.

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

[3] M. Baldoni, G. Boella, and L. van der Torre. powerJava: ontologically founded roles in object oriented programming languages. In H. Haddad, editor, *SAC*, pages 1414–1418. ACM, 2006.

[4] S. Balzer and T. R. Gross. Verifying multi-object invariants with relationships. In *ECOOP 2011–Object-Oriented Programming*, pages 358–382. Springer, 2011.

[5] S. Balzer, T. Gross, and P. Eugster. A relational model of object collaborations and its use in reasoning about relationships. In E. Ernst, editor, *ECOOP*, volume 4609 of *Lecture Notes in Computer Science*, pages 323–346. Springer, 2007. ISBN 978-3-540-73588-5.

[6] C. Bock and J. Odell. A more complete model of relations and their implementation: Roles. *Journal of Object-Oriented Programming*, 11(2):51–+, 1998.

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

[8] W. W. Chu and G. Zhang. Associations and roles in object-oriented modeling. In *Conceptual Modeling—ER'97*, pages 257–270. Springer, 1997.

[9] M. Dahchour, A. Pirotte, and E. Zimányi. A generic role model for dynamic objects. In *Advanced Information Systems Engineering*, pages 643–658. Springer, 2002.

[10] E. Ernst. Family polymorphism. In *ECOOP 2001—Object-Oriented Programming*, pages 303–326. Springer, 2001.

[11] V. Genovese. 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, 2007.

[12] N. Guarino and C. A. Welty. An overview of ontoclean. In *Handbook on ontologies*, pages 201–220. Springer, 2009.

[13] N. Guarino, M. Carrara, and P. Giaretta. An ontology of meta-level categories. In *KR*, pages 270–280, 1994.

[14] G. Guizzardi. *Ontological foundations for structure conceptual models*. PhD thesis, Centre for Telematics and Information Technology, Enschede, Netherlands, 2005.

[15] G. Guizzardi and G. Wagner. Conceptual simulation modeling with onto-uml. In *Proceedings of the Winter Simulation Conference*, page 5. Winter Simulation Conference, 2012.

[16] T. Halpin. ORM 2. In *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, pages 676–687. Springer, 2005.

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

[18] R. Hennicker and A. Klarl. Foundations for ensemble modeling–the helena approach. In *Specification, Algebra, and Software*, pages 359–381. Springer, 2014.

[19] S. Herrmann. Programming with roles in ObjectTeams/-Java. *AAAI Fall Symposium Roles, an interdisciplinary perspective*, 2005.

[20] T. Jäkel, T. Kühn, H. Voigt, and W. Lehner. RSQL - a query language for dynamic data types. In *Proceedings of the 18th International Database Engineering & Applications Symposium*, IDEAS '14, pages 185–194, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2627-8.

[21] T. Jäkel, T. Kühn, S. Hinkel, H. Voigt, and W. Lehner. Relationships for dynamic data types in RSQL. In *Datenbanksysteme für Business, Technologie und Web (BTW)*, 2015.

[22] T. Kühn, M. Leuthäuser, S. Götz, C. Seidl, and U. Aßmann. 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, 2014.

[23] M. Liu and J. Hu. Information networking model. In *Conceptual Modeling-ER 2009*, pages 131–144. Springer, 2009.

[24] R. Mizoguchi, K. Kozaki, and Y. Kitamura. Ontological analyses of roles. In *Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on*, pages 489–496. IEEE, 2012.

[25] S. Murer, C. Worms, and F. J. Furrer. Managed evolution. *Informatik-Spektrum*, 31(6):537–547, 2008.

[26] M. Pradel and M. Odersky. Scala roles: Reusable object collaborations in a library. In *Software and Data Technologies*, pages 23–36. Springer, 2009.

[27] T. Reenskaug and J. O. Coplien. 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.

[28] D. Riehle and T. Gross. Role model based framework design and integration. In *Proceedings OOPSLA '98, ACM SIGPLAN Notices*, pages 117–133, Oct. 1998.

[29] J. Rumbaugh, R. Jacobson, and G. Booch. *The Unified Modelling Language Reference Manual*. Addison-Wesley, 1st edition, 1999.

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

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

Appendix

Definition 9 (Additions to Compartment Role Object Model)**.** *Let* $\mathcal{M} = (NT, RT, CT, RST, fills, parts, rel)$ *be a CROM, and Attr the set of attribute names. Then* $\mathcal{N} = (NT, RT, CT, RST, Attr, fills, parts, rel, fields)$ *denotes the augmented CROM, where* $fields : (NT \cup RT \cup CT) \times Attr \rightarrow (NT \cup CT)$ *is a partial function assigning a distinct rigid type to the attribute of an entity type. It is assumed that* $fields(t, a) = \bot$ *for all undefined attributes* $a \in Attr$ *of* $t \in (NT \cup RT \cup CT)$.

Definition 10 (Addition to Compartment Role Object Instance)**.** *Let* $\mathcal{N} = (NT, RT, CT, RST, Attr, fills, parts, rel, fields)$ *be a well-formed, augmented CROM. Then an augmented* Compartment Role Object Instance (CROI) *of* $\mathcal{N}$ *is a tuple* $j = (N, R, C, type, plays, links, attr)$ *where* $attr : (N \cup R \cup C) \times Attr \rightarrow (N \cup C)$ *is a partial function assigning the objects to the respective attributes of the entities. Notably, an augmented CROI is well-formed wrt. to the CROM* $\mathcal{N}$ *iff all axioms of Definition 2 as well as the following axiom hold:*

$$\begin{aligned} &\forall o \in (N \cup R \cup C)\ \forall a \in Attr : fields(type(o), a) \neq \bot \\ &\quad \Rightarrow type(attr(o, a)) = fields(type(o), a) \end{aligned} \tag{17}$$

Theorem 1 (Emptyness)**.** *Let* $\mathcal{M} = (NT, RT, CT, RST, fills, parts, rel)$ *be a well-formed CROM,* $i = (N, R, C, type, plays, links)$ *a CROI compliant to* $\mathcal{M}$*,* $rst \in RST$ *an arbitrary relationship type,* $ct \in CT$ *an arbitrary compartment type, and* $c \in C$ *an arbitrary compartment; then the following implication holds:*

$$\begin{aligned} &rel(rst) = (rt_1, rt_2) \wedge rt_1, rt_2 \in parts(ct) \wedge type(c) \neq ct \\ &\Rightarrow links(rst, c) = \emptyset \end{aligned}$$

Proof. Assume $links(rst, c) \neq \emptyset$ and let $(r_1, r_2) \in links(rst, c)$. From (9) we know that either $r_1 \neq \varepsilon$ or $r_2 \neq \varepsilon$. If $r_1 \neq \varepsilon$, we get by (10) that $(o, c, r_1) \in plays$ for some $o \in N \cup C$ and $type(r_1) = rt_1$. Due to (6) this implies $rt_1 \in parts(type(c))$ which contradicts $rt_1 \in parts(ct) \wedge type(c) \neq ct$. The same arguments hold for $r_2 \neq \varepsilon$. Hence, it follows that $links(rst, c) = \emptyset$. □

```
class CROM:

  def __init__(self,nt,rt,ct,rst,fills,parts,rel):
    self.nt=frozenset(nt)
    self.rt=frozenset(rt)
    self.ct=frozenset(ct)
    self.rst=frozenset(rst)
    self.fills=frozenset(fills)
    self.parts=dict(parts)
    self.rel=dict(rel)
    assert mutual_disjoint([self.nt,self.rt,self.ct,self.rst])
    assert total_function(self.ct,self.parts)
    assert total_function(self.rst,self.rel)

  def wellformed(self):
    return self.axiom1() and self.axiom2() and self.axiom3() and \
    self.axiom4() and self.axiom5()

  def axiom1(crom):
    return all( any( (t,rt) in crom.fills for t in (crom.nt | crom.ct) ) for rt in
        crom.rt)

  def axiom2(crom):
    return all( len(crom.parts[ct])>0 for ct in crom.ct )

  def axiom3(crom):
    return all( len( [True for ct in crom.ct if rt in crom.parts[ct]] )==1 for rt in
        crom.rt )

  def axiom4(crom):
    return all( crom.rel[rst][0]!=crom.rel[rst][1] for rst in crom.rst )

  def axiom5(crom):
    return all( any(set(crom.rel[rst]) <= set(crom.parts[ct]) for ct in crom.ct) for
        rst in crom.rst)
```

Figure 5: Implementation of CROM in Python

```
class CROI:

  def __init__(self,n,r,c,type1,plays,links):
    self.n=set(n)
    self.r=set(r)
    self.c=set(c)
    self.type1=dict(type1)
    self.plays=set(plays)
    self.links=dict(links)
    assert mutual_disjoint([self.n,self.r,self.c,set([None])])
    assert total_function(self.n | self.r | self.c,self.type1)

  def compliant(self,crom):
    return crom.wellformed() and self.axiom6(crom) and self.axiom7(crom) and \
    self.axiom8(crom) and self.axiom9(crom) and self.axiom10(crom) and self.axiom11(
        crom)

  def axiom6(croi,crom):
    return all(((croi.type1[o],croi.type1[r]) in crom.fills) and (croi.type1[r] in
        crom.parts[croi.type1[c]]) for o,c,r in croi.plays)

  def axiom7(croi,crom):
    return all(croi.type1[r_1]!=croi.type1[r] for o_1,c_1,r_1 in croi.plays for o,c,r
        in croi.plays if o_1==o and c_1==c and r_1!=r)

  def axiom8(croi,crom):
    return all( len(set([(o,c) for o,c,r_1 in croi.plays if r_1==r]))==1 for r in croi
        .r )

  def axiom9(croi,crom):
    return all( (None,None) not in croi.links[(rst,c)] for rst in crom.rst for c in
        croi.c if (rst,c) in croi.links)

  def axiom10(croi,crom):
    return all( any((((o,c,r) in croi.plays and croi.type1[r]==crom.rel[rst][0]) ==
        bool((r,r_1) in croi.links[(rst,c)])) and (((o,c,r) in croi.plays and croi.
        type1[r]==crom.rel[rst][1]) == bool((r_1,r) in croi.links[(rst,c)]) ) for r_1
        in croi.repsilon() ) for rst in crom.rst for c in croi.c if (rst,c) in croi.
        links for r in croi.r for o in croi.o())

  def axiom11(croi,crom):
    return all(((r_1,None) not in croi.links[(rst,c)]) and ((None,r_2) not in croi.
        links[(rst,c)]) for rst in crom.rst for c in croi.c if (rst,c) in croi.links
        for r_1,r_2 in croi.links[(rst,c)] if r_1 != None and r_2 != None)

  def o(self):
    return self.n | self.c

  def o_c(self,c):
    return [ o for o,c_1,r in self.plays if c_1==c ]

  def repsilon(self):
    return self.r | set([None])
```

Figure 6: Implementation of CROI in Python

```
class ConstraintModel:

  def __init__(self,rolec,card,intra):
    self.rolec=dict(rolec)
    self.card=dict(card)
    self.intra=frozenset(intra)

  def compliant(self,crom):
    return crom.wellformed() and self.axiom12(crom)

  def axiom12(cm,crom):
    return all( atoms(a) <= set(crom.parts[ct]) for ct in crom.ct if ct in cm.rolec
        for crd,a in cm.rolec[ct] )

  def validity(self,crom,croi):
    return self.compliant(crom) and croi.compliant(crom) and self.axiom13(crom,croi)
        and \
    self.axiom14(crom,croi) and self.axiom15(crom,croi) and self.axiom16(crom,croi)

  def axiom13(cm,crom,croi):
    return all( \
    crd[0] <= sum( [evaluate(a,croi,o,c) for o in croi.o_c(c)] ) <= crd[1] \
    for ct in crom.ct if ct in cm.rolec for crd,a in cm.rolec[ct] for c in croi.c if
        croi.type1[c]==ct )

  def axiom14(cm,crom,croi):
    return all( evaluate(a,croi,o,c)==1 for o,c,r in croi.plays if croi.type1[c] in cm
        .rolec for crd,a in cm.rolec[croi.type1[c]] if croi.type1[r] in atoms(a) )

  def axiom15(cm,crom,croi):
    return all( \
    ( cm.card[rst][0][0] <= len( croi.pred(rst,c,r_2) ) <= cm.card[rst][0][1] ) and \
    ( cm.card[rst][1][0] <= len( croi.succ(rst,c,r_1) ) <= cm.card[rst][1][1] ) \
    for rst in crom.rst if rst in cm.card for c in croi.c if (rst,c) in croi.links for
        r_1,r_2 in croi.links[(rst,c)] )

  def axiom16(cm,crom,croi):
    return all( f( croi.overline_links(rst,c) )==1 for c in croi.c for rst,f in cm.
        intra if (rst,c) in croi.links)
```

Figure 7: Implementation of the Constraint Model in Python