TECHNISCHE UNIVERSITÄT DRESDEN

Role-based Data Management

Kurzfassung der Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
Dipl.-Wirt.-Inf. Tobias Jäkel
geboren am 02. August 1985 in Altdöbern

Betreuender Hochschullehrer:
Prof. Dr.-Ing. Wolfgang Lehner

Dresden, Januar 2017

Role-based Data Management (Extended Abstract)

Tobias Jäkel

Database systems build an integral component of today's software systems and as such they are the central point for storing and sharing a software system's data while ensuring global data consistency at the same time. Introducing the primitives of roles and their accompanied metatype distinction in modeling and programming languages, results in a novel paradigm of designing, extending, and programming modern software systems. Unfortunately, database systems, as important component and global consistency provider of such systems, do not keep pace with this trend. The absence of a metatype distinction, in terms of an entity's separation of concerns, in the database system results in various problems for the software system in general, for the application developers, and finally for the database system itself. In case of relational database systems, these problems are concentrated under the term role-relational impedance mismatch. The main problem is that persistent data management activities have to be performed by the applications and the database system loses its characteristic as single point of truth in a software system. To overcome the role-relational impedance mismatch and bring the database system back in its rightful position as single point in a software system, we argue for an introduction of role-based semantics in a database system. In detail, we aim at an adaptation of both, a database system's data system and set-oriented interface, to represent role-based semantics as first class citizen. Precisely, we propose a novel database model in combination with database operators. On this basis, we introduce a novel query language syntax and query processing model. As third component, we present a role-based result representation that preserve the role-based semantics in a query results and provides clients functionalities to access this information. These three components are combined as RSQL approach. Finally, we compare these components against a relational representation. This evaluation clearly shows the advantages of the RSQL approach in a role-based software system.

1 Introduction

The digital revolution has been shaping a world in which software dominates our day-to-day life, from tracking our exercise results on a wearable, over smart homes in which the coffee machine is customizable by a smartphone application, to self-driving cars. Hence, nowadays software is ubiquitous [22], but in different kinds of ubiquity. At first, software is physically ubiquitous in space, by running on mobile devices that constantly move between different locations and different users. Secondly, software runs based on the Internet which enables a logical ubiquity in space. As the Internet-based software is available all over the globe, it exposes its end points in many countries with different jurisdictions, or different cultures. Finally, today's software is ubiquitous in time, in terms of longevity. Applications are running for decades and face a constant change.

Traditionally, software is well structured in types by embedding its behavior in methods and functions. Once the software is compiled, its behavior is fixed for its whole lifetime. Hence, changing its behavior results in a recompilation and application restart. This characteristic is inherited from the types to the runtime objects, once instantiated it is bound to a certain type and the sets of attributes and methods are fixed and only the values can be changed. However, such software has no explicit notion of a context, hence, runtime objects act identically static, independent of the context [11]. That does not mean software cannot be written to be context-sensitive, rather this means that context adaptation mechanisms have to be implemented along with the regular functional behavior. In the ubiquitous scenario in which changes occur very frequently, it is infeasible to manually revise the context checks as well as recompile and restart the whole application.

To cope with the challenges of ubiquitous software, researchers have proposed several approaches, including the concept of roles that has been initially introduced by Bachman and Daya in computer science [2]. The key idea of roles, as design principle, is to split an entity into several metatypes, especially the entity core and the role types an entity may play over its lifetime. This split from a single and static metatype to several metatypes with special semantics, enables a separation of concerns on several levels. At first, the functional behavior is separated from the context-adaptation mechanisms. Secondly, the use of roles relaxes the situation of the statically described behavior and structure in the types. Thirdly, the actual behavior and structure of an entity can be varied after instantiation by start or stop playing roles in certain contexts. Moreover, this enables different lifetimes of various parts of an entity.

Database systems are an integral component of today's software systems, because they provide standard data management functionalities that are used by the applications. Thus, database systems provide several guarantees on which the applications can rely without taking care of these functionalities by themselves [17]. First, it encapsulates the persistent data management from the applications. Moreover, as central data storage facility, it ensures global data consistency for the whole application landscape, which gives it the characteristic as single point of truth in a software system. Even in a ubiquitous software world these guarantees have to be ensured by a database system. In detail, we assume such software world to be modeled and implemented by using roles as extension points to entities. Unfortunately, there is no database system available that is able to explicitly model roles as extension points of entities in certain contexts, so far. Generally, there are two ways of coping with the absence of role-based semantics in the database system: (i) hiding this absence as much as possible by implementing mapping engines or (ii) implement a database system that is aware of the role-based semantics.

In this thesis, we argue for an integration of roles and their related semantics in a database system, to overcome the semantic gap between role-based software and traditional database systems. Thus, we propose an adaptation of a database system's *data system layer* to a novel database model and an adjusted processing model in combination with a redesigned *set-oriented interface*.

In detail, this thesis provides the following contributions. At first, we investigate the problems of traditional, especially relational, database systems in combination with role-based software systems. These problems are concentrated under the term **role-relational impedance mismatch**, which describes problems from a database system's, application's and application developers', and software system's perspective. On this basis, several **requirements to overcome this role-relational impedance mismatch** are defined and evaluated against four possible architecture layout solutions. As only a full database system integration is able to overcome this mismatch, we propose a novel database model. This logical database model is called **RSQL database model** and features roles as first class citizen that are embedded in a certain context. To adapt the processing model to this database model, we overview several **logical database operators**. The set-oriented interface is redesigned by defining the syntax **RSQL query language** to create a role-based database schema and populate the corresponding database. Additionally, the query language statements are connected to the logical database operators in order to generate a query execution plan. Moreover, the **RSQL Result Net** is specified as an adapted query result representation. This represents the set-oriented interface redefinition from an output perspective. This includes the preservation of metamodel semantics within the result as well as a definition to navigate through the result by leveraging the special metatype interrelations. Finally, we **demonstrate the benefits** of the RSQL database model, the query language, and the result representation in comparison to their relational mapping.

2 Modeling With Roles

Roles as modeling primitives are nothing novel, in fact this concept has been introduced by Bachman and Daya in 1977 [2] as extension to the network data model. However, roles provide advantages compared to traditional modeling languages, like the Unified Modeling Language (UML), and programming languages, like Java; structure and behavior can be added to an existing object during runtime. Over the recent decades, a lot of role-based approaches have been proposed, all having a different notion of roles. This zoo of role notions has been surveyed by Kühn et al. as well as Steimann. In this section we classify the different role notions and introduce. Moreover, we present metatype distinction in combination with an ontological foundation.

2.1 Classification of Role Approaches

The term role and the underlying concepts are omnipresent in our every day's life, in our communication and speech as well as in the way we think. However, the origins of this term may go back to the ancient world, especially to the ancient theater. In theater, a role refers to a character or figure an actor or an actress plays in a certain play [2]. This captures the core notion of what a role is very well. Firstly, a role is something abstract someone or something can act like. Secondly, the role itself is defined independently of its player, because the role does not define who or what is going to play that corresponding role. Finally, the role requires a player to be filled with life, because there must be someone who adopts the role's behavior and structure. This idea has been adopted in computer science as well. In particular, we utilize roles to enable entities managed in a software system to dynamically gain and lose behavior as well as structure during runtime. This particular role perception has been utilized a lot, for instance in modeling [8, 2, 25, 21, 24, 9, 5] or programming languages [10, 3, 20, 7, 23, 16]. Based on the surveys of Steimann [24] and Kühn et al. [19], role approaches can be classified by three aspects: the relational aspect, the behavioral and structural aspect, and the context-dependent aspect. Based on these three perspectives, seven classes of approaches are possible, which are indicated and referenced by the corresponding number in Figure 1. In this thesis we aim for a holistic role-based data model that provides support for all aspects and is classified in the seventh category.

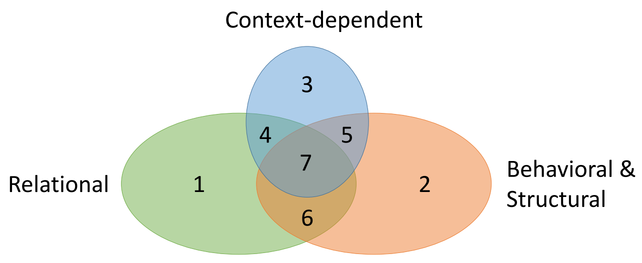

Figure 1: Classification of Existing Role Notions Based on the Three Aspects: Relational, Behavioral and Structural, and Context-dependent.

To clearly understand the context-dependency of roles, we distinguish two notions of context. Firstly, context as environmental information and secondly, as objectified collaboration containing other entities. We rely on the objectified notion of a context as introduced and named compartment by Kühn et al. [19, p. 146].

2.2 Metatype Distinction

We distinguish four metatypes that are separated by the meta properties semantic *rigidity*, *foundedness*, and *identity*. These types are based on the Compartment Role Object Model specification given in [18] and summarized in Table 1. Firstly, *Natural Types* (NT) are considered to be rigid, non-founded, and have a unique identity. Secondly, *Compartment Types* (CT) are semantically rigid, but found and have a unique identity. Thirdly, *Role Types* (RT) are the opposite of Natural Types; they are anti-rigid and founded while their identity is derived from their player types.

Finally, *Relationship Types* (RST) are rigid, founded and have a composed identity. To distinguish these metatypes from the general terms, we refer to these metatypes by using an initial capital letter.

Concept / Property	*Rigidity*	*Foundedness*	*Identity*	*Example*
Natural Types	rigid	non-founded	unique	Person
Compartment Types	rigid	founded	unique	University
Role Types	anti-rigid	founded	derived	Student
Relationship Types	rigid	founded	composed	takes class

Table 1: Ontological Foundation of the Distinguished Metatypes

3 Need For a Role-based Database System

The concept of roles is established and implemented in modeling as well as programming languages, but not in database systems. This results in a role-relation impedance mismatch that occurs in case role-based software is used in combination with relational database systems. In this section we investigate this mismatch and argue for a database implementation of role-based semantics.

3.1 Role-Relational Impedance Mismatch

Database systems are usually embedded in a software system and do not occur in isolation. In detail, we assume a traditional database application setup, in which several applications access the database to persistently store and share data. Thus, the database system builds the central point for persistent data storage for a lot of applications and possibly application servers. Moreover, we assume that these applications implement roles and their accompanied metatype distinction as first class citizen. As there is no database system featuring roles as first class citizen available, we additionally assume a relational database system.

Such a setup results in the role-relational impedance mismatch that describes issues for the database system itself, the applications and application developers, and the software system in general, caused by the absence of role-based semantics in the database system. The main problem of such a software system is the database system's inability to ensure global consistency, because the metatype distinction and its accompanied constraints cannot be directly represented in the database system. Hence, the database system loses its characteristic as single point of truth in a role-based environment. As a consequence, invalid schemata may be implemented in the system, which are valid locally seen from an application perspective, but invalid from a global database perspective.

However, this situation results in problems for the applications and their developers, as they have to take over some data management tasks. In particular, application developers have to manually program the unavailable standard functionalities for a role-based data management by themselves, for each application. Moreover, there is no standard mapping from role-based semantics onto relational tables. Consequently, multiple different mappings may be applied within the same database. Additionally, applications and especially their mappings are required to be aligned and synchronized for shared data objects. Finally, application developers have to think in two different worlds, because they are not only concerned with the application's data model, but the database model as well.

From a database system perspective, it loses it single point of truth characteristic within the software system. Moreover, semantics that could be used to optimize the query plan are not part of the database model. Furthermore, relational DBS produces a relational result and role-based semantics have to be reconstructed.

In addition, the software system is not layered well, because persistent data management tasks are distributed over several layers. Next, there are no continuous and crosscutting role semantics throughout the entire software system, which results in a huge semantic gap between the transient runtime objects and the persistent ones. Finally, the software system suffers from a mapping overhead between the applications and the database system.

3.2 Related Approaches

The related approaches can be categorized into four classes: (i) traditional techniques that stick to a relational system, (ii) mapping engines, (iii) persistent programming languages, and (iv) database system implementations.

External views are a representative of the first category. They provide an application-specific perspective on the stored data. Unfortunately, they do not address any of the problems directly, rather external views lower the effort in query writing.

Mapping engines are very popular to bridge the object-oriented and relational worlds. For role-based software to relational database system there exist DAMPF [6] as client-side mapping engine and ConQuer [4] as conceptual query language. These approaches solve the problems from the application perspective, but keep the database system without the role-based semantics. Moreover, the persistent database management tasks remain distributed over several layers.

Persistent programming language, like DOOR [26] or Fibonacci [1], combine the persistent data storage with programming languages. However, they are focused on a single application and not on a software system. Hence, there is no central data storage, in fact each application maintains its individual storage.

Finally, the Information Networking Model is a representative for a database system integration [21, 12]. This approach suffers from a weak role notion and an implementation using classes and objects. Thus, the metatype distinction is not supported as first class citizen. However, a database system integration addresses all problems by bringing the database system back in its single point of truth position. In this thesis we aim at such a solution by introducing a novel data system layer and a redesigned set-oriented interface.

4 RSQL Database Model

To integrate the concept of roles as first class citizen in a database system, a sophisticated database model, supporting the notion of roles and a metatype distinction within an entity, is required as a first step. Thus, we introduce the RSQL database model consisting of Dynamic Data Types on the schema level and Dynamic Tuples on the instance level. Parts of these database model definitions have been published in [14, 13, 15]. Formal operator definitions on the basis of this database model complete this section.

4.1 Schema Level

The schema distinguishes the four aforementioned metatypes and their interrelations as base elements. In detail, a RSQL schema is formally defined as following.

Definition 1 (Schema)**.** Let NT, RT, CT, and RST be mutual disjoint sets of Natural Types, Role Types, Compartment Types, and Relationship Types, respectively. Moreover, let Card $\subset \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ be the set of cardinalities represented as $i..j$ with $i \leq j$. Then, a *Schema* is a tuple $\mathcal{S} = (NT, RT, CT, RST, \mathit{fills}, \mathit{parts}, \mathit{rel}, \mathit{card})$.

The *fills* relation stores information on which core type can fill which Role Types. The *parts* function holds a set of Role Types that are contained in each Compartment Type. Additionally, the *rel* function connects two Role Types and stores information which Role Type is connected by a certain Relationship Type with which counter Role Type. Finally, *card* assigns cardinality constraints to Relationship Types. In addition, we require several constraints to hold, for instance, a Role Type must be included in the *fills* relation at least once, and must be included in exactly on Compartment Type.

On top of these elements, we define a notion of **Dynamic Data Types** that integrates the various base types in an integrated logical data structure. In particular, a Dynamic Data Type consists of a core type and Role Types in two dimensions. One dimension for Role Types that can be filled, and one for those that can be contained in it. As only Compartment Types can contain Role Types, this dimension is populated for Dynamic Data Types having a Compartment Type as its core type. Moreover, several Dynamic Data Types may overlap in their Role Types, because a Role Type must be included at least once in the filling dimension and once in participating dimension. A graphical example of the Dynamic Data Types *SportsTeam* is visualized in Figure 2.

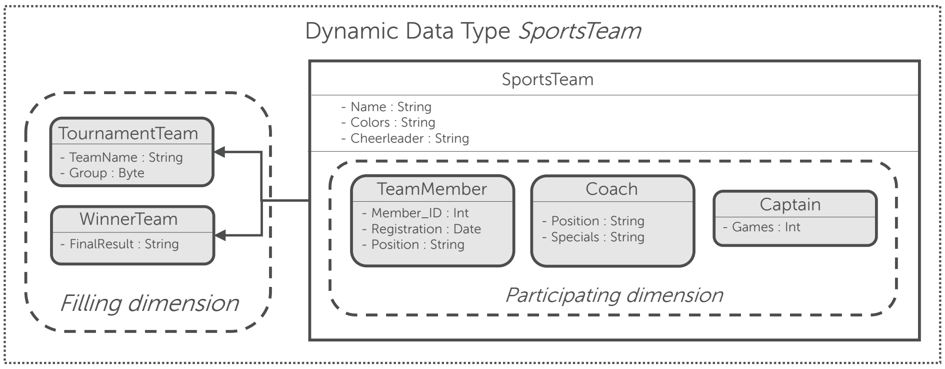

Figure 2: Visualization of the Dynamic Data Type *SportsTeam*

In detail, the Compartment Type *SportsTeam* forms an individual Dynamic Data Type. Additionally, it is able to fill the Role Types *TournamentTeam* and *WinnerTeam*. This represents the filling dimension. Moreover, it contains the Role Types *TeamMember*, *Coach*, and *Captain*, which represents the participating dimension of this Dynamic Data Type.

4.2 Instance Level

The instance level definitions comprise instances of the different types. Thus, Natural Types, Compartment Types, and Role Types are instantiated to Natural, Compartments, and Roles, respectively. Relationship Type instances are represented in a separate instance model function. In detail, the RSQL instance level is defined as follows.

Definition 2 (Instance)**.** Let $\mathcal{S} = (NT, RT, CT, RST, \mathit{fills}, \mathit{parts}, \mathit{rel}, \mathit{card})$ be a schema and N, R, and C be mutual disjoint sets of Naturals, Roles and Compartments, respectively. Then, an *Instance* of $\mathcal{S}$ is a tuple $\mathfrak{i} = (N, R, C, \mathit{type}, \mathit{plays}, \mathit{links})$.

The *type* function is a labeling function for the instances to their corresponding type. The *plays* relation stores information about which core plays which Roles in which Compartment. Finally, the *links* holds the information which Roles participate in which Relationships. Moreover, we require certain instance constraints, like a Role has exactly one player, to hold.

On top of these instance definitions, we specify the notion of **Dynamic Tuples** as integrated data structure to encapsulate the role-based semantics. This comprises a core and Roles separated into two dimensions, the playing and the featuring dimension. The first dimension holds Roles that are currently played and the featuring dimensions those Roles that are actively contained within this Dynamic Tuple. Furthermore, the Roles are grouped in sets by their respective Role Type. As Dynamic Data Types on the schema level, Dynamic Tuple overlap as well, but by their Roles. This means, a Role in the playing dimension of a certain Dynamic Tuple has to be part of another Dynamic Tuple in the featuring dimension. A graphical example of a Dynamic Tuple is illustrated in Figure 3.

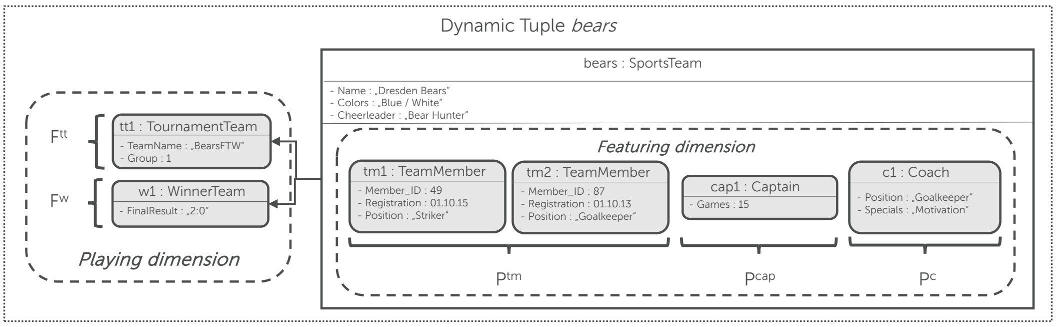

Figure 3: Dynamic Tuple *bears*

In detail, this example shows the Dynamic Tuple *bears* that consists of the core Compartment *bears*, which is an instance of the Compartment Type *SportsTeam*.

Moreover, it holds Roles in both dimensions. The playing dimension is populated with the Roles tt_1 and w_1 grouped in the sets F^{tt} and F^{wt}, respectively. In the featuring dimension four Roles are included in three sets. In particular, the Team-Member set P^{tm} consists of two Roles.

4.3 RSQL Operators

The database model features ten operators to process and manipulate Dynamic Tuples. In fact, the notion of Dynamic Tuples as role-based data structure is preserved throughout the entire processing. Table 2 provides an overview of the operators.

Operator	Functionality
Σ_{cex}	Schema-based filter
Π_α	Filters for Roles of queried Role Types
κ_α	Filters for overlapping Roles
Ω_{rst}	Filters for Roles participating in a Relationship
τ	Core union over Dynamic Data Types
$\setminus_{R-}$	Difference of Dynamic Tuples
$\setminus_R$	Difference of Dynamic Tuples on the Role dimensions level
$\cap_\circ^{RT_a,RT_b}$	Intersection of Dynamic Tuples
$\cup_\circ^{RT_a,RT_b}$	Union of Dynamic Tuples
$\sigma_{preicate}^{t,RT_{overlap}}$	Attribute-based filter

Table 2: An Overview of the Operators and Their Functionality

The operators are categorized into three classes: (i) operators on an entity's structure level, (ii) operators to enable usual set operations on the level of Dynamic Tuples, and (iii) operators on the attribute level. Operators of the first class filter Dynamic Tuples based on their structure, for instance, a Role of a certain Role Type has to be played or featured by a Dynamic Tuple, or Dynamic Tuples have to overlap by a particular Role. The second category consists of operators that enable set operations like a difference of two Dynamic Tuple sets or an intersection of such. Finally, operators of third class filter Dynamic Tuples as whole data structure or Roles as parts of this structure, based on their attribute values.

For instance, the operator κ_α consumes two sets of Dynamic Tuples and for each Dynamic Tuple pair it checks for overlapping Roles. Only if a Dynamic Tuple finds a partner that shares a Role in the opposite dimension, it passes this operator. Moreover, all Roles that do not find a partner are excluded from the returned Dynamic Tuple. In this sense, this operator filters and manipulates Dynamic Tuples by their overlapping points.

In sum, the schema and instance level definitions provide data structures that comprise the role-based semantics within an integrated type and instance specification. Additionally, the operators provide functions that preserves these structures throughout the query processing.

5 Query Language And Result Representation

To adapt the set-oriented database system interface, we introduce a novel query language and connect its elements with the operators. Additionally, we present the RSQL Result Net as role-based result representation. Parts of the query language and the result representation have been published in [14, 13, 15].

5.1 RSQL Data Query Language

RSQL's external interface description features syntax definitions to create a role-based database schema by using the RSQL data definition language. Moreover, the RSQL data manipulation language is used to populate the corresponding database and the actual query language to retrieve data from the database. The schema creates Dynamic Data Types indirect by creating Natural Types, Compartment Types, and Role Types, respectively. The same holds for the database population. Dynamic Tuples are created by inserting Natural and Compartments. Additionally, their extensions are achieved by inserting new Roles.

In the following, we will detail the RSQL data query language part. Querying role-based data with RSQL focuses on describing the schema instances have to meet and at which points they have to overlap. For instance, assume the following query.

```
SELECT * FROM Person p PLAYING Student s AND StudentAssistant sa,
  Uni u FEATURING s AND sa;
```

This query asks for **Persons** that play Roles of the Role Type **Student** and **StudentAssistant**. Moreover, a **University** is required that features these Roles. The key language element is the *config-expression*, which describes a Dynamic Data Type and the desired Role Types in the corresponding dimensions. In this example, two of these expressions are used. `Person p PLAYING Student s AND StudentAssistant sa` represents the first one and `Uni u FEATURING s AND sa` the second one. This can be seen as schema-based filter for Dynamic Tuples, because various Dynamic Tuples of the same Dynamic Data Type may have different schemata. Moreover, these expressions overlap at certain points, as Dynamic Data Type and Dynamic Tuples also do. Hence, this overlap specification represents an additional filter.

However, these syntax elements are directly connected to the operators of the database model. For instance, each overlap between two *config-expressions* results in a κ operator. The parameter α is set according to the overlapping Role Types. In this example, α is populated with $Student \wedge StudentAssistant$. Moreover, each *config-expression* results in a Σ and Π operator. The former filters Dynamic Tuples on the basis of their actual schema, and the latter reduces the Dynamic Tuples to the queried Role Types only. This means, there might exists Dynamic Tuples that play Roles of Role Types that are not queried. All these not queried Role Types will not be part of the resulting Dynamic Tuple.

5.2 RSQL Result Net

The result representation completes the set-oriented interface redesign. The RSQL Result Net (RuN) provides Dynamic Data Type and Dynamic Tuple notions that are shipped to the clients in an adapted interface. Moreover, it provides functionalities to browse through the net of interconnected Dynamic Tuples. The illustration in Figure 4 summarizes the architecture of RuN.

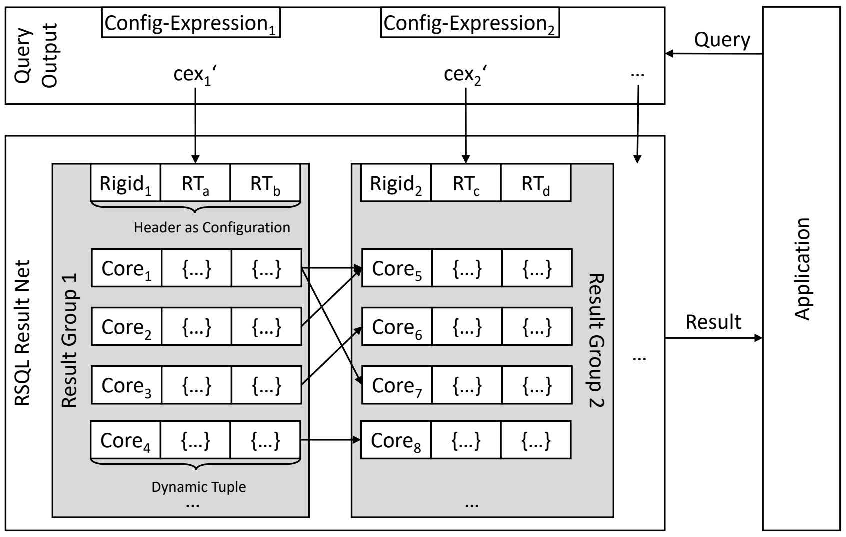

Figure 4: RSQL Result Net Architecture

In detail, RSQL queries return multiple sets of Dynamic Tuple, one set for each *config-expression*. Each set is represented in a result group, whereas all result groups form the RSQL Result Net. To access the information of Dynamic Tuple and navigate through various Dynamic Tuples, different functionalities are provided by RuN. First, each RuN has a main cursor that points to a Dynamic Tuple in a certain result group. This cursor can be iterated and moves from one Dynamic Tuple to another. Moreover, there exist endogenous and exogenous navigation paths. The former ones navigate from a Dynamic Tuple to the Roles in both dimensions. The latter ones navigate from a Role to the Dynamic Tuple it is played or featured in. Additionally, the relationship information can be used to navigate from one Role to its counter part by a certain Relationship Type.

6 Proof of Concept

To demonstrate the benefits of the RSQL approach, we compare it to a relational mapping of its metamodel. In detail, we compare the database models by the number of statements required to set up the corresponding database schema that simultaneously ensures consistency on the instance level. Moreover, the query languages are compared to each by their length. Finally, the client-side effort to process the query results is contrasted, to evaluate the result representation.

6.1 Evaluating the Database Model

In a first step, we map the RSQL database model onto a relational representation. To ensure the instance consistency constraints, we implement several triggers. For instance, to check that a Role is played by only one player and that there are no isolated Roles in the system. The employed university domain scenario features 19 types. To set up the corresponding schema, RSQL requires one statement per type, in sum 19. In contrast, the relational SQL version requires 83 statements. 30 to create the tables and 53 triggers that ensure consistency. Consequently, SQL requires 4.3 times more statements than RSQL in this scenario. In various other test, this ratio varied between 3 and 15. This clearly shows, that RSQL represents a role-based database scenario more efficiently than a relational mapping. Especially, the large amount of triggers increases the number of required SQL statements dramatically.

6.2 Evaluating the Query Language

To evaluate both query languages, we compared them by their length and number of used characters. These metrics provide indicators for a language's suitability. To have a fair comparison, we omitted unnecessary characters and labeling in the relational mapping. We compare two example queries, a simple one that consists of a single *config-expression* and a more complex one that uses four, partially overlapping, *config-expressions*. In the simple scenario, RSQL queries used approximately 3 times fewer words and characters than the SQL version. In contrast, the more complex scenario reveals a 2.3 times higher amount of words and characters in the SQL version. The relative difference shrinks from one to another example, but the absolute amount grows. Hence, we conclude that SQL queries on role-based data are generally longer and thus, harder to write and maintain and RSQL represents a suitable language interface.

6.3 Evaluating the Result Representation

For the comparison of the result representations, we measure the effort of their processing on the client side. In detail, we compare a RuN implementation with two possible relational processing models. The first implements an all in one scenario in a way that the whole data is queried within a single statement. The second option queries for the main data and executes additional queries to retrieve the related data. This simulates a join processing in the client.

The comparison is performed by several indicator metrics, like the number of queries executed, the size of the result set, or the lines of code used. To have a fair comparison, we implemented all processing models in the same code style. The evaluation comprises four scenarios that vary the number of implemented Role Types and the number of Roles per Role Type. As the evaluation clearly shows, the RSQL Result Net combines the positive aspects of both relational processing models. It uses fewer queries to retrieve the data by having a minimal set of instances in the result. Moreover, fewer lines of code are used to process it and extract the required information. Thus, we conclude that RSQL is a convenient result representation for role-based data.

7 Conclusions

As software has become ubiquitous in several ways, the paradigm to model and implement such software has changed. Roles, as modeling primitive for a separation of concerns within an entity, are established in modeling and programming languages, but not in database systems. This results in several problems that are combined under the term role-relational impedance mismatch, in case relation database systems are employed with role-based software in a software system. Most important, this mismatch takes the database system its single point of truth characteristic. To overcome this mismatch and bring the database system back in its rightful position, we argued for an implementation of role-based semantics in a database system. To achieve this, we presented a database model consisting of Dynamic Data Types on the schema level and Dynamic Tuples on the instance level. These definitions integrate the notion of roles and its accompanied metatypes distinction with a certain data structure. As adaption for a database system's external interface, we proposed a query language and result representation. In a final proof of concept, we demonstrated the benefits of our approach in comparison to a relation representation.

References

[1] Antonio Albano, Giorgio Ghelli, and Renzo Orsini. Fibonacci: A Programming Language for Object Databases. *The VLDB Journal*, 4(3):403–444, 1995.

[2] Charles Bachman and Manilal Daya. The Role Concept in Data Models. In *VLDB*, pages 464–476. VLDB Endowment, 1977.

[3] Stephanie Balzer, Thomas R Gross, and Patrick Eugster. A relational model of object collaborations and its use in reasoning about relationships. In *European Conference on Object-Oriented Programming*, pages 323–346. Springer, 2007.

[4] Anthony Bloesch and Terry Halpin. ConQuer: A Conceptual Query Language. In *International Conference on Conceptual Modeling*, pages 121–133. Springer, 1996.

[5] Mohamed Dahchour, Alain Pirotte, and Esteban Zimányi. A generic role model for dynamic objects. In *International Conference on Advanced Information Systems Engineering*, pages 643–658. Springer, 2002.

[6] Sebastian Götz. Dampf - Dresden Auto-Managed Persistence Framework. Diploma thesis, Technische Universität Dresden, 2010.

[7] Kasper Graversen and Kasper Østerbye. Implementation of a Role Language for Object-specific Dynamic Separation of Concerns. In *Workshop on Software-engineering Properties of Languages for Aspect Technologies*, 2003.

[8] Terry Halpin. Object-Role Modeling (ORM/NIAM). In *Handbook on Architectures of Information Systems*, pages 81–103. Springer, 1998.

[9] Rolf Hennicker and Annabelle Klarl. Foundations for Ensemble Modeling–The Helena Approach. In *Specification, Algebra, and Software*, pages 359–381. Springer, 2014.

[10] Stephan Herrmann. A Precise Model for Contextual Roles: The Programming Language ObjectTeams/Java. *Applied Ontology*, 2(2):181–207, 2007.

[11] Robert Hirschfeld, Pascal Costanza, and Oscar Nierstrasz. Context-oriented Programming. *Journal of Object Technology*, 7(3), 2008.

[12] Jie Hu, Qingchuan Fu, and Mengchi Liu. Query Processing in INM Database System. In *Web-Age Information Management*, pages 525–536. Springer, 2010.

[13] Tobias Jäkel, Thomas Kühn, Stefan Hinkel, Hannes Voigt, and Wolfgang Lehner. Relationships for Dynamic Data Types in RSQL. In *Datenbanksysteme für Business, Technologie und Web (BTW)*, 2015.

[14] Tobias Jäkel, Thomas Kühn, Hannes Voigt, and Wolfgang Lehner. RSQL - A Query Language for Dynamic Data Types. In *Proceedings of the 18th International Database Engineering & Applications Symposium*, pages 185–194, 2014.

[15] Tobias Jäkel, Thomas Kühn, Hannes Voigt, and Wolfgang Lehner. Towards a Role-Based Contextual Database. In *20th East European Conference on Advances in Databases and Information Systems*, pages 89–103. Springer International Publishing, 2016.

[16] Tetsuo Kamina and Tetsuo Tamai. Towards Safe and Flexible Object Adaptation. In *International Workshop on Context-Oriented Programming*, page 4. ACM, 2009.

[17] Alfons Kemper and Andre Eickler. *Datenbanksysteme: Eine Einführung.* Oldenbourg, 2006.

[18] Thomas Kühn, Stephan Böhme, Sebastian Götz, Christoph Seidl, and Uwe Aßmann. A Combined Formal Model for Relational Context-Dependent Roles. In *International Conference on Software Language Engineering*, pages 113–124. ACM, 2015.

[19] Thomas Kühn, Max Leuthäuser, Sebastian Götz, Christoph Seidl, and Uwe Aßmann. A Metamodel Family for Role-based Modeling and Programming Languages. In *7th International Conference on Software Language Engineering*, pages 141–160. Springer, 2014.

[20] Max Leuthäuser and Uwe Aßmann. Enabling View-based Programming with SCROLL: Using Roles and Dynamic Dispatch for Etablishing View-based Programming. In *Proceedings of the 2015 Joint MORSE/VAO Workshop on Model-Driven Robot Software Engineering and View-based Software-Engineering*, MORSE/VAO '15, pages 25–33. ACM, 2015.

[21] Mengchi Liu and Jie Hu. Information Networking Model. In *International Conference on Conceptual Modeling*, pages 131–144. Springer, 2009.

[22] Eila Niemelä and Juhani Latvakoski. Survey of Requirements and Solutions for Ubiquitous Software. In *Proceedings of the 3rd international Conference on Mobile and Ubiquitous Multimedia*, pages 71–78. ACM, 2004.

[23] Michael Pradel and Martin Odersky. Scala Roles: Reusable Object Collaborations in a Library. In *Software and Data Technologies*, pages 23–36. Springer, 2009.

[24] Friedrich Steimann. On the Representation of Roles in Object-oriented and Conceptual Modelling. *Data & Knowledge Engineering*, 35(1):83–106, October 2000.

[25] Roel Wieringa, Wiebren de Jonge, and Paul Spruit. *Roles and Dynamic Subclasses: A Modal Logic Approach*, pages 32–59. Springer, 1994.

[26] Raymond Wong, Lewis Chau, and Frederick Lochovsky. A Data Model and Semantics of Objects with Dynamic Roles. In *International Conference on Data Engineering*, pages 402–411. IEEE, Apr 1997.