Relations

A First Class Relationship and First Class Derivations Programming Language

Daco Harkes (5453086)
Delft University of Technology
d.c.harkes@student.tudelft.nl

Categories and Subject Descriptors D.3.2 [**Programming Languages**]: Language Classifications – Very high-level languages

Keywords declarative; model based; relations; relationships; derivations; derived values; reactive expressions

1. Introduction

Two useful features for data models are often not present in programming languages: relationships and derivations. Relationships between entities can be artificially encoded in pointers, nestings, foreign keys and tuples, but these all have their drawbacks. Derived values (Figure 2, line 9) can be realized with functions, but one has to cache these manually, or in materialized views (in databases), but the latter does not support all forms of recursion.

Different meta models provide different features for specifying data models in applications. To illustrate the need for a language with relationships and derivations as first class citizens the problems of each meta model are listed:

- OO-model: traversing relationships in both directions requires keeping pointers both ways consistent; relationships cannot be ternary or have attributes without lifting them to objects; and derived values have to be cached manually.
- Relational model [4]: recursive relations like trees can only be saved and queried in normalized form; views have restrictions on recursion, recursive aggregations are not supported; subtyping is not supported; and one cannot build an application solely with a relational database and has to deal with the OR impedance mismatch when using it with an OO language.
- Nested relational model [9]: This model does not other views on the data than the hierarchy it is saved in; and also has the impedance mismatch problem.
- Logic model (Datalog and Prolog): ordering, duplicates and grouping by is nontrivial; and relationships, as tuples, cannot be entities themselves.

These meta models support the features the others lack, but none of them has both first class relationships and derivations. Without language support relationships and derivations have to be encoded in artificial constructs. These add code complexity, hide design intent and are error prone. The goal of this project is to create a language that has these as first class citizens.

MODULARITY'14, April 22–26, 2014, Lugano, Switzerland.
ACM 978-1-4503-2773-2/14/04.
http://dx.doi.org/10.1145/2584469.2584473

2. Related work

Current approaches to add relationships to language do not solve all of the above listed problems. In 1987 Rumbaugh was the first to add relationships to a language [10]. His approach is preprocessor based and dynamic. It does not have relations as first class citizens and does not support symmetries and derived relationships. Our approach is static and does support these.

Noble and Pearce extended Java with first class relationships using aspects [8]. They argue that objects should be agnostic to relationships. In our approach entities know what relations they participate in. This allows using relations inside the derivations.

They also created the Java Query Language [12]. The query language uses value-based joins, like SQL. LINQ also uses value-based joins [7]. Our language does queries based on the relationships themselves, essentially navigating along the relationships.

Bierman and Wren also added first class relationships to Java named RelJ [2]. In their approach they support relationships as first class citizens. The relations are tuples, having unnamed ordered roles. In our approach the roles are named and unordered, allowing symmetry and querying based on roles. Also our approach is not a language extension.

Closely related work is the Rumer language by Balzer [1]. It features first class relations with roles and queries. Rumer provides reactive queries as well as imperative code. It has cardinalities specified in constraints and supports just binary relationships. Our approach differs in the fact that everything is a derivation and thus reactive, multiplicities are part of the type system instead of constraints and we support relations of all degrees.

Work related to derivations is firstly the field of (materialized) views in databases, for example [5]. There are grossly two ways to maintain derived data: deriving maintenance queries (algebra based) and reactive programming [11], using a dependency model. The language abstracts over this, so the language itself can stay the same while its compiler can use the above techniques.

3. The language

The starting point of designing the language is the Entity Relationship model [3]. The data model is specified in terms of entities, relationships and attributes. The data model also specifies all derivations. Derivations work like spreadsheets, but then on entities, attributes and relationships instead of cells.

The language does not have statements or functions, since everything can be expressed in derivation expressions (reactive expressions). There is no passing around of parameters for functions, these should be fetched in derivation expressions by navigating along relationships. Only CRUD-operations change state.

Only primitive types are allowed as attributes of entities or relationships. The way to relate two entities is by a relationship, not

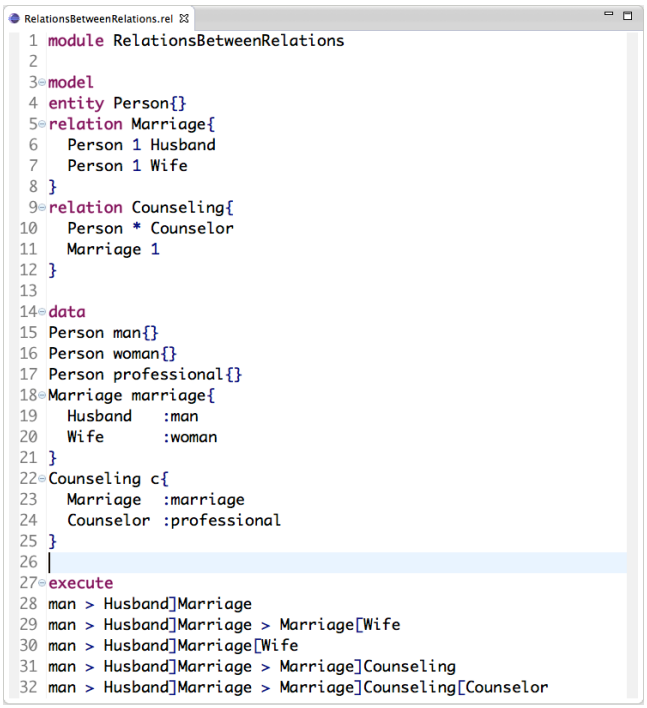

Figure 1. Relationships between relationships and navigators

by putting one in an attribute of the other. The reason for this is simply that relationships should all be first class relationships.

Another remark about the types is that there are no collection types. Instead of collections explicit cardinalities (or multiplicities) are stated for all relationships. These are orthogonal to types and interact when needed. For example when navigating along relationships both types and multiplicities are checked.

The current state of the prototype supports defining entities, relationships, attributes, derived attributes and navigating along relationships. The prototype is built with the Spoofax Language Workbench [6]. It has an editor with syntax highlighting, name resolution, type checking and generates Java code.

The language has three main parts: model, data and execute. The model describes an ER-model, the data instances of that model and lastly the execute part contains queries over the model. Model and Data are quite straight forward, except for the roles in the relations. These are expressed as type, multiplicity and role-name, where the multiplicity is from the participating entities point of view and role-names are optional.

Figure 1 shows a complete program where navigating along relationships is illustrated in the execute part. To navigate one starts with an entity, and specifies the relation and role to navigate along. The navigation syntax has two components; navigating from an entity into relation or navigating out of a relation to entities again. Also the role names are explicit in navigating.

On line 28 we navigate from a person to a marriage where the person has the role husband. On line 29 we subsequently navigate out of the marriage relation by the wife-role and line 30 is the shorthand for both. Because relationships are first class citizens we can navigate from the marriage relation to the counselling relation (line 31). Marriage has the marriage-role in counselling (notice the same name, the role name is not specified on line 11).

Figure 2 shows only the model, but with the derivations. The average grade can be calculated based on the averages of the children. To do this the navigation along relationships is used.

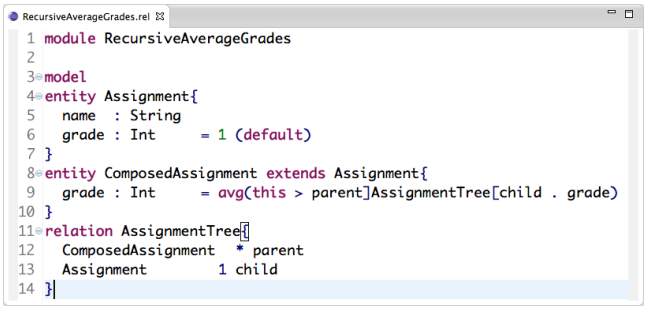

Figure 2. Recursive aggregation derivations in model

Future work includes refining the syntax, introducing shorthand notations and more powerful querying techniques, as well as invariants and transactions.

References

1 Balzer, Stephanie. *Rumer: a Programming Language and Modular Verification Technique Based on Relationships*. 2011.

2 Bierman, Gavin M. and Wren, Alisdair. First-Class Relationships in an Object-Oriented Language. In *ECOOP 2005 - Object-Oriented Programming, 19th European Conference, Glasgow, UK, July 25-29, 2005, Proceedings* (2005), Springer, 262-286.

3 Chen, Peter P. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Trans. Database Syst.*, 1, 1 (1976), 9-36.

4 Codd, E. F. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13, 6 (1970), 377-387.

5 Gupta, Ashish and Mumick, Inderpal Singh. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Eng. Bull.*, 18, 2 (1995), 3-18.

6 Kats, Lennart C. L. and Visser, Eelco. The Spoofax language workbench: rules for declarative specification of languages and IDEs. In *Proceedings of the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2010* (Reno/Tahoe, Nevada 2010), ACM, 444-463.

7 Meijer, Erik, Beckman, Brian, and Bierman, Gavin M. LINQ: reconciling object, relations and XML in the.NET framework. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006* (2006), ACM, 706.

8 Pearce, David J. and Noble, James. Relationship aspects. In *Proceedings of the 5th International Conference on Aspect-Oriented Software Development, AOSD 2006, Bonn, Germany, March 20-24, 2006* (2006), ACM, 75-86.

9 Roth, Mark A., Korth, Henry F., and Silberschatz, Abraham. Extended Algebra and Calculus for Nested Relational Databases. *ACM Trans. Database Syst.*, 13, 4 (1988), 389-417.

10 Rumbaugh, James E. Relations as Semantic Constructs in an Object-Oriented Language. In *OOPSLA* (1987), 466-481.

11 Salvaneschi, Guido and Mezini, Mira. Reactive behavior in object-oriented applications: an analysis and a research roadmap. In *Proceedings of the 12th annual international conference on Aspect-oriented software development* (2013), 37-48.

12 Willis, Darren, Pearce, David J., and Noble, James. Efficient Object Querying for Java. In *ECOOP 2006 - Object-Oriented Programming, 20th European Conference, Nantes, France, July 3-7, 2006, Proceedings* (2006), Springer, 28-49.