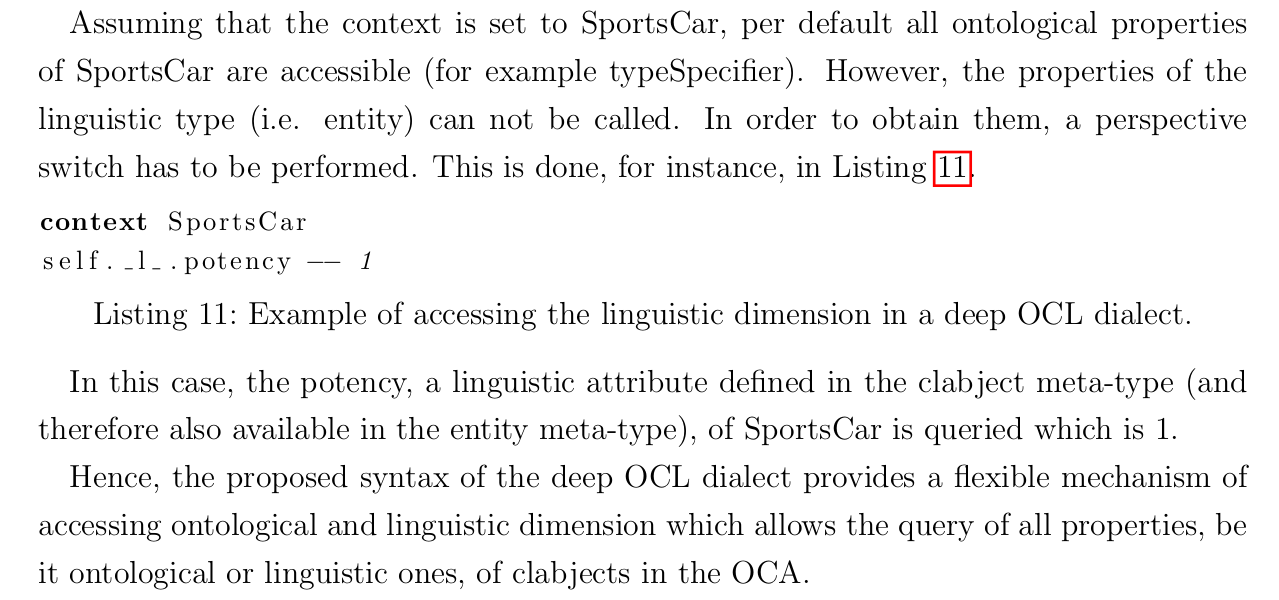
**OCL.**

*Research papers summary.*

1. **Dominik Kantner Specification and Implementation of a Deep OCL Dialect (thesis)**  
   The current implementation of Melanee lets to perform deep modeling across multiple levels. However, the current version of the framework at that moment was not able to apply object constraints across multiple modeling levels. In other words, it was not aware of deep modeling. Neither it supported modeling for both linguistic and ontological dimensions. This thesis aims to overcome those limitations by extending the existing OCL implementation to make it level agnostic.  
   **Goal:** Extend the existing Melanee framework with level agnostic object constraint language support.  
   **Techniques:**
   1. To enable deep modeling in Melanee, a new dialect called ‘LML’ was developed that takes into account two dimensions that are orthogonal to each other. The first one is called linguistic which serves to form a language structure of how the models should look. On the other hand, there is the so-called ontological dimension that is level agnostic and is needed to build domain-specific hierarchies. Moreover, ‘LML’ also makes use of the concept ‘potency’ that allows to instantiate attributes/associations across more than one level, thus enabling deep modeling.
   2. To implement the Deep OCL dialect, the Eclipse OCL project was used which allows developers to extend the API of the module to express their custom behavior when defining OCL expressions.
   3. How to take into account the notion of clabjects while defining an OCL constraint (clabject has both type and instance facets)? Implemented a so-called ‘context-switch’. An example is taken directly from the paper:  
      
   4. Since a clabject has a dual facet – linguistic and ontological, it is important to differentiate them. When one is being in the linguistic dimension, a standard OCL can be applied naturally, since it’s an operation only between two level. This allows querying entity-related attributes and methods like ‘potency’, ‘all deep instances’, etc. However, when one is using an ontological dimension, a custom navigation through a deep model was adopted and embedded into the deep OCL dialect.
   5. Since deep modeling crosses multiple meta-levels, they defined an integer property to define across how many instantiation levels a constraint should be validated. If this property is not given then the constraint is validated on the instantiation type where this constraint is defined as well as on every subtype of the type.
   6. Constraints can be defined on every clabject since every clabject has a type facet. And in two-level modeling constraints defined on a type level.
   7. What was not implemented:
      1. Potencies and potency ranges for constraints
      2. Constraints are implemented within the ‘invariant’ context on clabject and not on actions. Derived rules, attributes are also not considered.

**Results:** A deep OCL dialect was implemented, which extends the API of the Eclipse OCL module. This dialect allows defining invariant constraints on both ontological and linguistic levels.