IceDust 2: Composition of Calculation Strategies
and Multiplicity Bounds on Derived Bidirectional Relations

Daco C. Harkes and Eelco Visser
Delft University of Technology, The Netherlands
{d.c.harkes, e.visser}@tudelft.nl

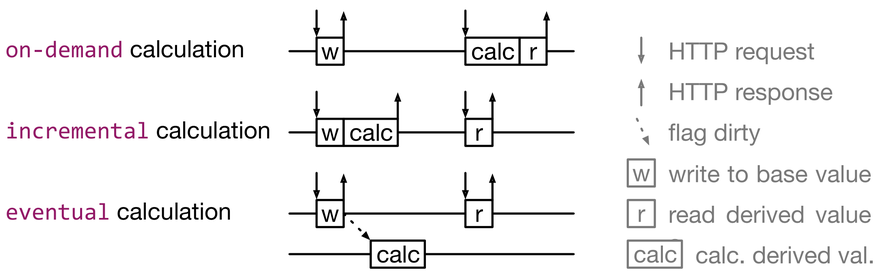
Declarative Data Modeling

Derived values are values calculated from base values. A key concern in *implementing* systems with derived values is minimizing the *computational* effort that is spent to re-compute derived values after updates to base values. A key concern in *modeling* systems with derived values is minimizing the *programming* effort to realize such minimal computations.

IceDust 2 is a data modeling language that supports **derived values**. A specification consists of **entities**, (derived) **attributes**, and (derived) **bidirectional relations** between entities. To minimize programming effort, calculation strategies can be selected per field (attribute or relation). The type system checks the soundness of the composition of these strategies.

Calculation Strategies

IceDust 2 provides three strategies for calculating the derived values: **on-demand**, **incremental** and **eventual**. The distinction between these strategies is when derived values are calculated:



Derived Relations

Derived values can be expressed with views in *relational databases*, but they do not provide **multiplicity bounds**. Derived values can also be expressed with expressions in *incremental* or *reactive programming*, but require significant boilerplate to encode **bidirectional derived values**.

Derived relations are specified as expressions in IceDust 2, this provides multiplicity bounds. Bidirectionality is provided by maintaining inverses.

Example IceDust 2 Specification

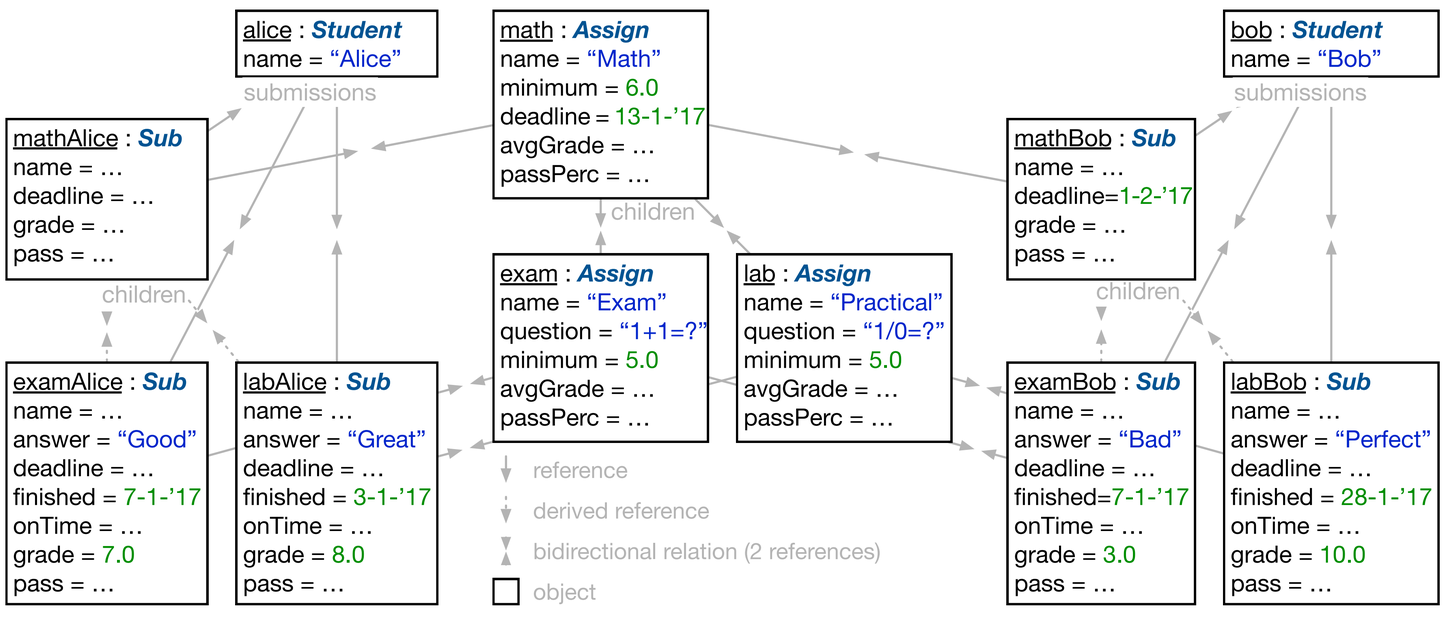
```
entity Assignment (eventual) {
  name     : String                                  Base Value Attribute
  question : String?
  deadline : Datetime?
  minimum  : Float
  avgGrade : Float?    = avg(submissions.grade)      Derived Value Attribute
  passPerc : Float?    = count(submissions.filter(x=>x.pass) / count(submissions)
}

entity Student {
  name     : String
}

entity Submission (incremental) {                    Calculation Strategy Selection
  name     : String    = assignment.name + " " + student.name        (on-demand)
  answer   : String?
  deadline : Datetime? = assignment.deadline <+ parent.deadline      (default)
  finished : Datetime?
  onTime   : Boolean   = finished <= deadline <+ true
  grade    : Float?    = if(conj(children.pass)) avg(children.grade) (default)
  pass     : Boolean   = grade >= assignment.minimum && onTime <+ false
}

relation Submission.student    1 <-> * Student.submissions    Bidirectional Relation
relation Submission.assignment 1 <-> * Assignment.submissions
relation Assignment.parent     ? <-> * Assignment.children

relation Submission.parent     ? =                            Derived Value
  assignment.parent.submissions.find(x => x.student == student) Bidirectional Relation
                                 <-> * Submission.children
```

Example Data

The math course consists of a lab and an exam. The minimum to pass the course is a 6, but for the lab and exam a 5 suffices. The deadline for the course is January 13th, 2017.

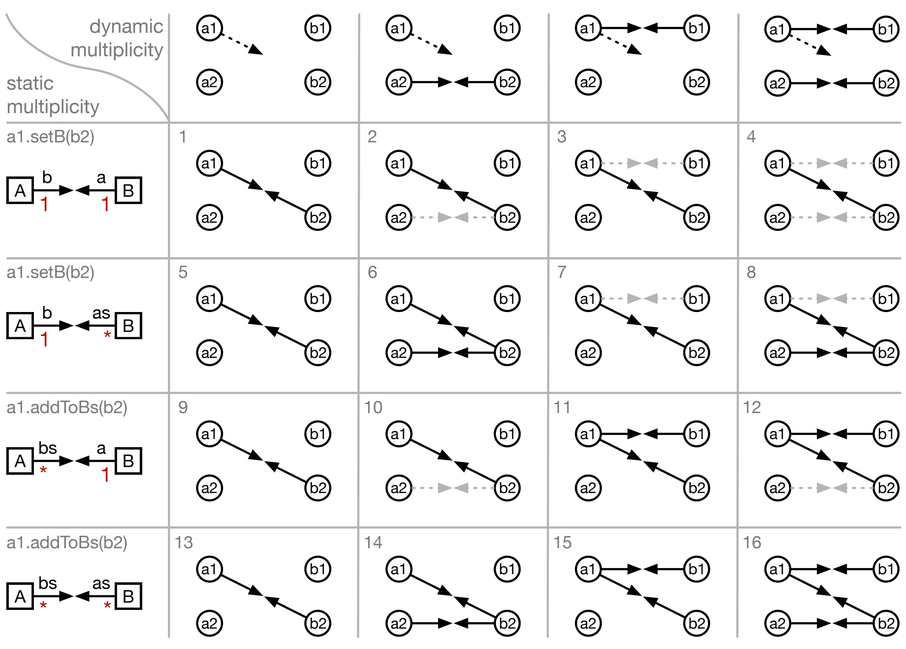
Alice passes the course, her grades are sufficient, and the lab is handed in on time. Bob's exam grade is insufficient. Bob's lab is late, but he received a personal deadline for the course.

Note that the parent-children relation for submissions is derived. And that deadlines recursively flow down the submission-tree while grades get recursively averaged up the tree.



Bidirectional Relations

When updating bidirectional relations, both multiplicity and bidirectionality have to be preserved. Multiplicities guide bidirectional updates in IceDust 2. For example executing lab.addToChildren(exam) implicitly removes math as parent from exam 10, as exam can at most have one parent. It is identical to executing exam.setParent(lab) 7.

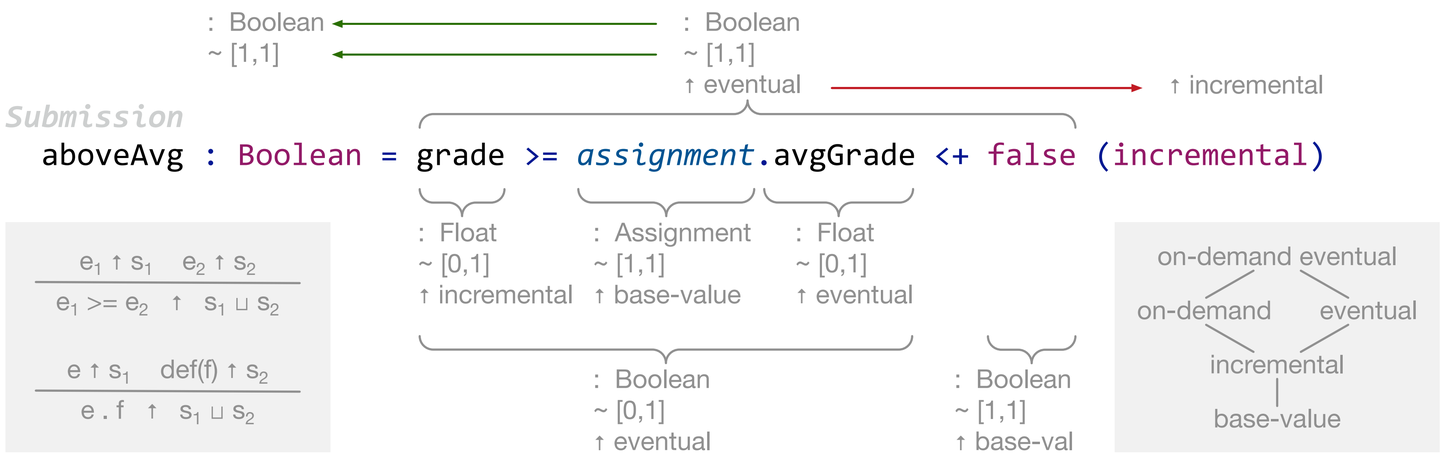


Sound Composition of Calculation Strategies

Calculation strategies should retain correctness and time complexity under composition. For example a read of an incremental value is O(1), as such it cannot reference an on-demand value, as it would have to check whether it changed.

For sound composition of calculation strategies, fields with a specified calculation strategy may only depend on fields with the same or a stronger calculation strategy. The grey box below to the right shows the lattice of calculation strategies, lower is stronger.

Checking composition of calculation strategies is orthogonal to checking types and multiplicities in IceDust 2. The left grey box shows two of the rules for composition checking. The example below will give an error as aboveAvg references avgGrade.



Derived Bidirectional Relations

Updating derived values might lead to bidirectionality maintenance, which in turn can lead again to updating derived values. For example when executing exam.setParent(lab), examAlice.parent and examBob.parent get updated. This causes the deadlines of those objects to be updated. Moreover, mathAlice.children, labAlice.children, mathBob.children, and labBob.children are also updated, which triggers re-execution of the grade for these objects.

```
Assignment.parent    -> submissions.parent   } also updates old and new
Submission.parent    -> deadline               submissions.parent.children
Submission.children  -> grade
Submission.grade     -> pass                 }
Submission.grade     -> parent.grade           fixpoint
Submission.pass      -> parent.grade         }
Submission.pass      -> assignment.passPerc
Submission.grade     -> assignment.avgGrade
```

Updating derived values and maintaining bidirectionality is recursive, and is executed until a fixpoint is reached. Updates for derived values are triggered via data-flow paths. A subset of the data-flow paths of the example specification is shown above.

Harkes, D. C., Visser, E.: *IceDust 2: Derived Bidirectional Relations and Calculation Strategy Composition.* ECOOP (2017)
Harkes, D. C., Groenewegen, D. M., Visser, E.: *IceDust: Incremental and Eventual Computation of Derived Values in Persistent Object Graphs.* ECOOP (2016)
Harkes, D. C., Visser, E.: *Unifying and Generalizing Relations in Role-Based Data Modeling and Navigation.* SLE (2014)

TU Delft