IceDust: Incremental and Eventual Computation of Derived Values in Persistent Object Graphs

Daco C. Harkes, Danny M. Groenewegen, and Elco Visser
Delft University of Technology, The Netherlands
d.harkes@tudelft.nl, d.m.groenewegen@tudelft.nl, e.visser@tudelft.nl

Problem
Object-oriented programming languages allow specification of derived values through getters that contain the code that calculates the derived value. However, this implies calculate on (each) read. Changing to a cached implementation requires code changes.

Relational Databases provide views, materialized and non-materialized, for calculating derived values. However, views limit expressiveness by limiting recursive aggregation.

Solution
IceDust is a language which allows data modeling with derived value attributes, and provides multiple calculation strategies as compiler options. This provides separation of the functional specification from the calculation strategy.

An IceDust specification consists of entities, attributes (base values and derived values) of entities, and bidirectional relations between entities.

Calculation Strategies
IceDust provides three calculation strategies for calculating the values of attributes: Calculate-on-Read, Calculate-on-Write and Calculate-Eventually. The high level difference between these strategies is the moment that derived values are calculated.

Benchmarks
We benchmarked derived values that depend on up to 100000 base values in varying workloads. We calculate a recursive aggregate in a tree. (See benchmark specification below.)

Dependency Analysis
IceDust specifications define the values of attributes in terms of other attributes. The Calculate-on-Write and Eventually-Consistent strategies require dependency and data flow information.

Calculation Strategy Implementations
The different calculation strategies require different code patterns to be generated by the compiler. Below is a snippet of meta-code that generates part of the Calculate-on-Read and Calculate-on-Write implementation.

// Calculate-on-Read
function calculate_a() : T { return e1; }

// Calculate-on-Write
function get_a() : T { return calculate_a(); }

for a : T in e2.ATTRIBUTES
    function calculate_a() : T { return e1; }
    // Calculate-on-Read
    function get_a() : T { return calculate_a(); }
    // Calculate-on-Write
    a : T;
    static a_dirty = Set<>;
    function get_a() : T { return this.a; }
    function update_a() { a = calculate_a(); }

    for e3.a : T where a2.entityE2.a = e3
        // Calculate-on-Write
        function get_a() : T { return this.a; }
        function update_a() { a = calculate_a(); }

    for e3.a : T where a2.entityE2.a2.dirty.addAll(path); }

    // Calculate-on-Write
    static function update_derived_values() {
        // go through all dirty and update until all empty
    }

    // Eventually-Consistent
    // Same as calc-on-write, but dirty flag to separate thread

Example IceDust Specification

Example Data

Benchmarks