Migrating to an Incremental Computing DSL
A case study of migrating WebLab’s business logic to IceDust

Daco C. Harkes, Elmer van Chastelet, and Eelco Visser
Delft University of Technology, The Netherlands
{t.c.harkes, e.vanchastelet, e.visser}@tudelft.nl

Problem
Information systems filter and process data to create new data: derived data. Derived data should be updated as base data is updated, and this should happen fast.

However, realizing a high performance implementation typically requires invasive changes to the business logic in the form of cache and cache invalidation code. Unfortunately, this obfuscates the original intent of the business logic in an abundance of caching patterns. These patterns make it less straightforward to validate that a program ‘does the right thing’.

Approach
Incremental computing DSLs aim to address this tension between performance and verifiability by automatically incrementally updating non-incremental specifications.

To provide empirical evidence on how extent migration of business logic to an ICL is useful, we report on a case study on a learning management information system: we migrated WebLab to IceDust.

Results

Validatability: improved
- Derived values, as a single source of computation, give developers confidence that they understand what the business logic means.
- During performance engineering developers can reason about the system based on calculation strategies, without worrying about inconsistencies.

Performance: drastically improved / slightly regressed
- The WebLab implementation in IceDust enables live statistics, which was infeasible manually.
- WebLab-IceDust performs similar or better compared to WebLab-vanilla, except for object creation.

Effort: no effort reduction
- The effort for additional business logic is significantly lower in the ICL, but the total effort is not reduced.
- While IceDust does not lead to an overall effort reduction or increase, it does increase separation of concerns.

Calculation Strategies

- on-demand calculation
- incremental calculation
- eventual calculation

Simplified WebLab Data Model

- **entity Assignment**
  - name : String (Base Value Attribute)
  - question : String
  - deadline: Datetime
  - min : Float
  - avgGrade : Float
  - pass : Boolean

- **entity Student**
  - name : String

- **relation Submission**
  - student : * core n m
  - submission : * n m

- **relation Assignment**
  - parent : * core n m
  - child : * n m

Simplified WebLab Data

- **Alice**
  - student : "Alice"
  - math : Assign
    - name : "Math"
    - question : "1+1=3?"
    - deadline : 13-1-17
    - avgGrade : 0.0
    - pass : false

- **Bob**
  - student : "Bob"
  - exam : Assign
    - name : "Exam"
    - question : "1+1=3?"
    - deadline : 1-2-17
    - avgGrade : 0.0
    - pass : false

Benchmarks

- Our first benchmarks are maximum system throughput under concurrent student actions. We show the average requests per second over 30 second runs, higher is better. More IceDust threads decrease performance, as less processing power is available for requests. Vanilla calculation cannot run concurrently with load, hence no measurements for one thread.

- The next two benchmarks are the system throughput under which live statistics can be maintained by IceDust. More IceDust threads increase performance, as derived values are calculated faster.

- The last three benchmarks are heavy weight teacher and administrative actions. For these we measure time to completion in seconds, lower is better. More IceDust threads increase performance, as derived values are calculated faster.