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
{d.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 is less straightforward to **validate** that a program *'does the right thing'*.

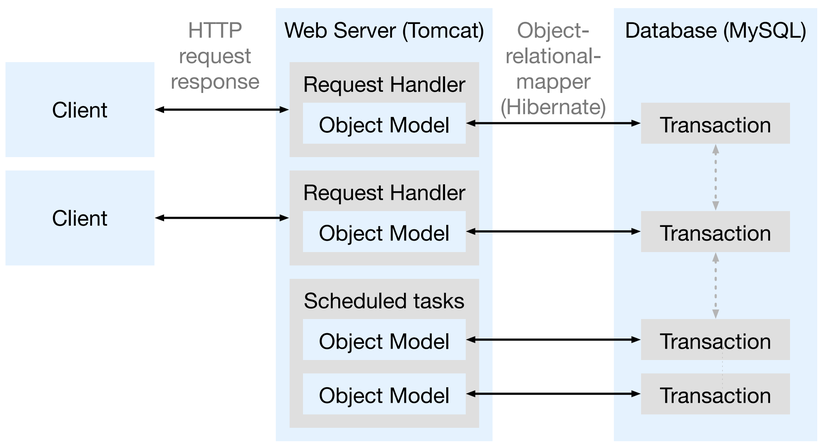
Approach

Incremental computing DSLs aim to address this tension between performance and validatability by automatically incrementalizing non-incremental specifications.

To provide **empirical evidence** to what 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.

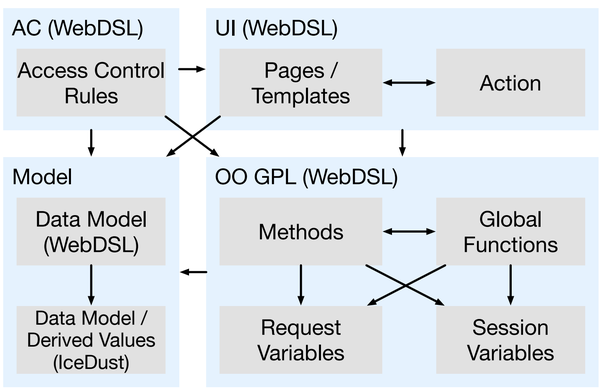
WebLab Server Architecture

WebLab uses a standard stateless architecture for web servers. HTTP requests are serviced in isolation by a request handler. A request handler loads (saves) the data from (to) the database by means of an object-relational mapper. Request handlers interact concurrently with the database through transactions. The scheduled task executor handles periodic or asynchronous tasks.



WebLab Code Architecture

In WebLab-IceDust we migrated the data model partially from WebDSL to IceDust. Moreover, we migrated the calculation of derived values from object-oriented GPL code (methods and global functions) to IceDust derived value expressions. Finally, we refactored the rest of the code to use these derived values rather than the GPL methods.



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

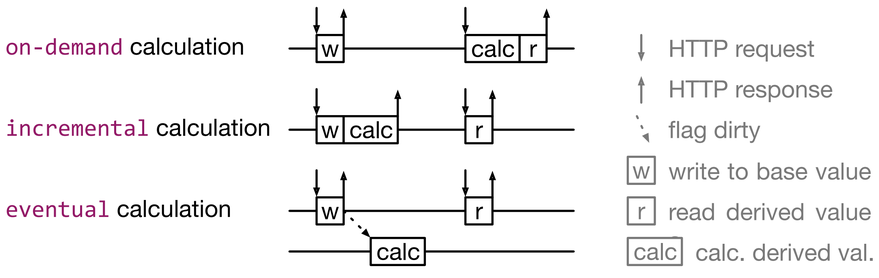
- The WebLab implementation in IceDust enables live statistics, which was infeasible manually.
- WebLab-IceDust performs similar or better compared to WebLab-vanilla, except for object creation.

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

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:



Simplified WebLab Data Model

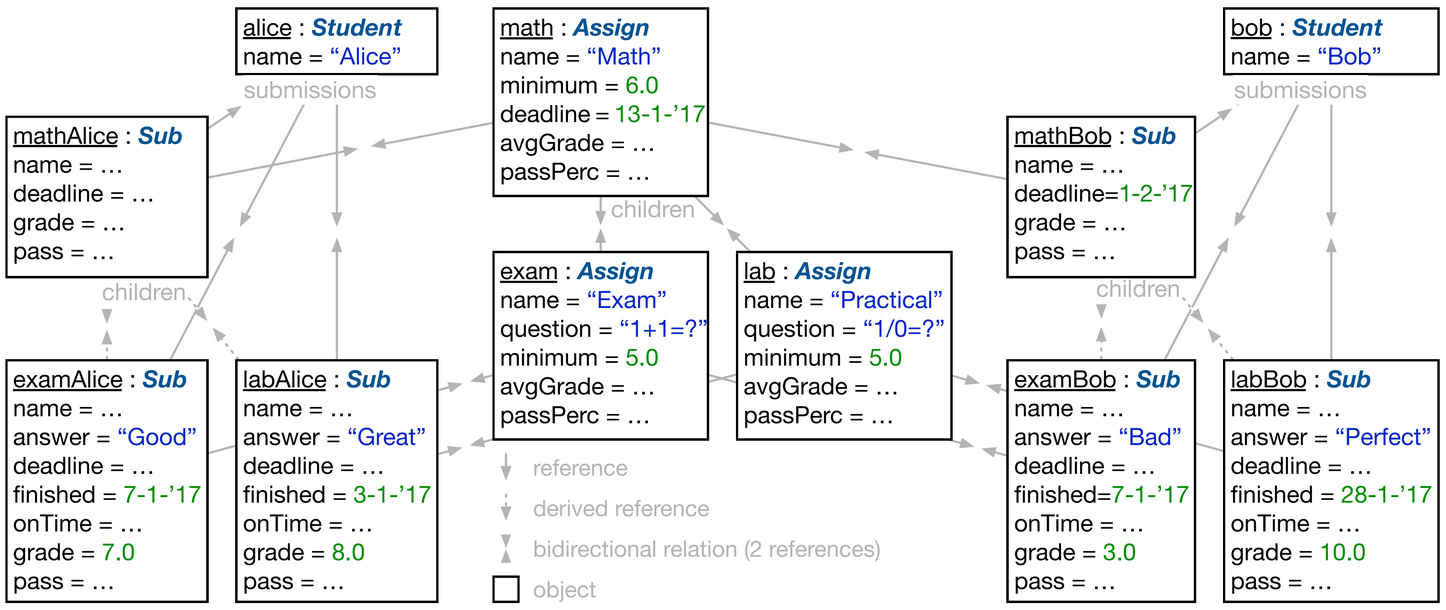
```
entity Assignment {
  name     : String   Base Value Attribute
  question : String?
  deadline : Datetime?
  minimum  : Float
  avgGrade : Float?     = avg(submissions.grade)   Derived Value Attribute
  passPerc : Float?     = count(submissions.filter(x=>x.pass) / count(submissions)
}

entity Student {
  name     : String
}

entity Submission {
  name     : String     = assignment.name + " " + student.name
  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
```

Simplified WebLab 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.



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 heavyweight 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.

		Impl.	Vanilla	IceDust							
		Threads	0	1	0	2	4	6	8	10	12	14	16
Action	**Unit**	**Course**											
read submission	req/sec	Small	136.04	-	129.65	112.53	109.58	103.11	92.68	84.39	74.66	67.29	61.18
		Medium	152.95	-	145.04	125.37	119.33	112.02	104.82	93.88	88.26	80.14	71.66
		Large	132.45	-	119.62	141.99	130.21	115.44	105.22	97.49	96.07	95.98	96.19
edit submission	req/sec	Small	31.06	-	120.49	115.52	115.42	114.88	114.48	114.18	114.18	113.55	113.25
		Medium	111.91	-	96.13	90.92	87.08	82.02	69.74	64.73	62.30	63.18	58.96
		Large	104.17	-	107.84	103.64	96.30	89.56	80.08	75.68	71.30	66.63	62.63
create submission	req/sec	Small	100.72	-	67.70	64.39	61.95	60.58	57.31	54.49	51.91	48.94	46.85
		Medium	137.07	-	67.91	64.45	65.55	62.12	58.37	58.17	55.65	55.80	49.03
		Large	139.34	-	35.93	35.73	34.02	33.58	32.52	31.73	31.40	29.47	28.79
edit submission (live stats)	req/sec	Small	-	-	-	117.45	114.38	113.85	113.31	112.85	112.41	111.61	113.68
		Medium	-	-	-	6.59	11.29	16.49	20.93	23.95	26.75	29.15	30.72
		Large	-	-	-	6.12	12.47	16.62	19.69	20.86	21.68	25.3	24.66
create submission (live stats)	req/sec	Small	-	-	-	9.02	15.73	20.56	22.86	24.92	26.85	27.83	29.30
		Medium	-	-	-	4.49	8.31	11.26	13.40	14.82	15.77	16.45	17.30
		Large	-	-	-	3.60	6.26	8.29	9.54	10.54	11.81	12.35	12.68
change deadline	sec/ 10 reqs	Small	-	303	-	195	110	78	64	56	51	48	47
		Medium	-	571	-	252	140	106	79	68	62	59	57
		Large	-	2324	-	408	232	172	143	128	120	117	102
change checklist w.	sec/ 100 reqs	Small	-	1220	-	21	15	14	12	12	11	11	12
		Large	-	19130	-	123	71	55	46	41	39	41	45
recalculate course	sec/req	Small	-	49	-	147	76	53	42	36	32	30	28
		Medium	-	1718	-	1557	795	549	434	373	333	318	302
		Large	-	18494	-	12624	6833	5078	4365	4049	3821	3797	3748

Harkes, D. C., van Chastelet, E., Visser, E. ***Migrating Business Logic to an Incremental Computing DSL: A Case Study.*** **SLE (2018)**
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