Delta Modeling in Practice

A Fredhopper Case Study

Michiel Helvensteijn [1 2] Radu Muschevici [3] Peter Y. H. Wong [4]

[1]CWI, Amsterdam, The Netherlands

[2]Leiden University, Leiden, The Netherlands

[3]Katholieke Universiteit Leuven, Leuven, Belgium

[4]Fredhopper, Amsterdam, The Netherlands

VaMoS 2012

Fredhopper®

http://www.hats-project.eu

Introduction

Delta Modeling Workflow (DMW)

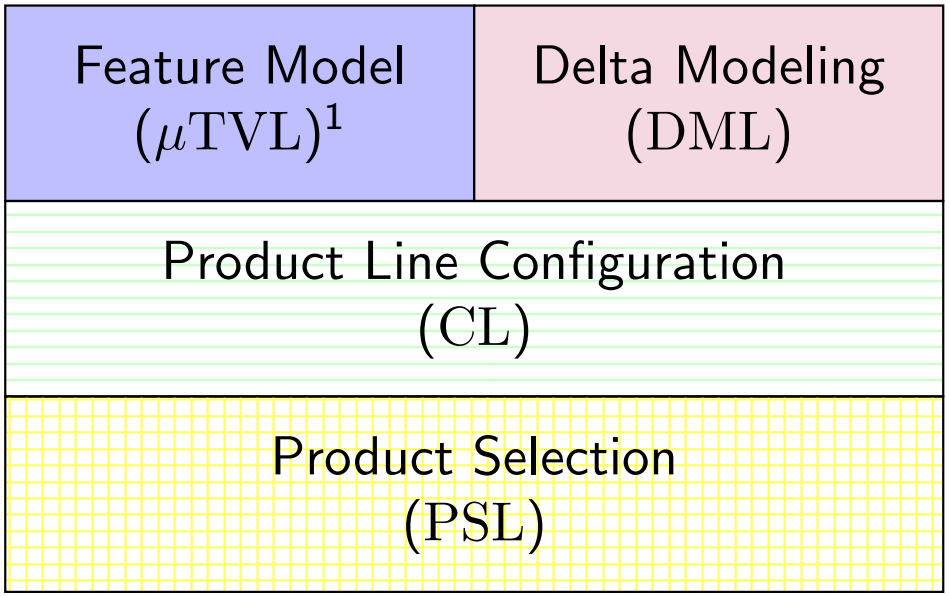
- DMW is a method for modeling software product lines (SPL)
- Abstract Behavioral Specification (ABS) is an executable modeling language that implements delta modeling as its core paradigm for modeling adaptable systems
- We report on the development of an industrial scale SPL in ABS following DMW to guide the application of delta modeling in practice

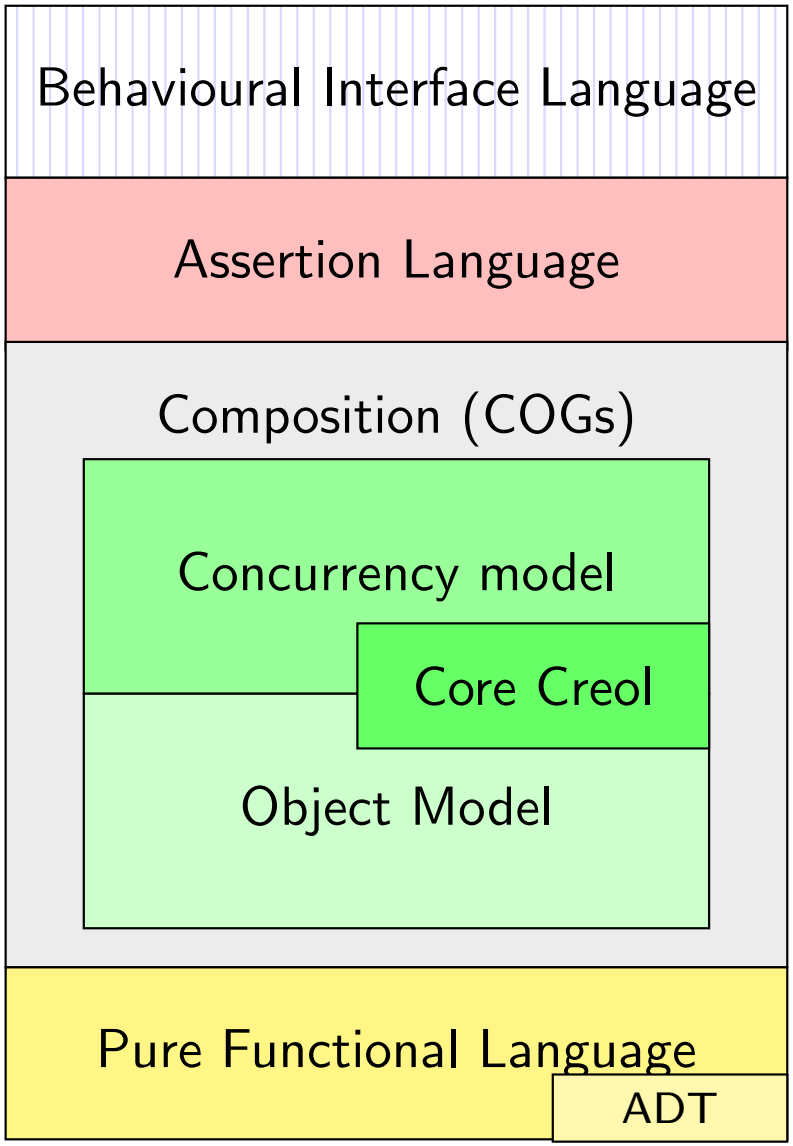
HATS

HATS: Highly Adaptable & Trustworthy Software Using Formal Models

- FP7 project
- Develop a tool-supported formal software engineering methodology for the design, analysis and implementation of highly adaptable and trustworthy software systems
- Started 1 March 2009, 48 months runtime
- Integrated Project, academically driven
- 9 academic partners, 2 industrial research, 1 SME
- 8 countries

ABS Language Layers





[1] Based on: A. Classen, Q. Boucher, P. Heymans. A Text-based Approach to Feature Modelling: Syntax and Semantics of TVL. SCP 2010.

Fredhopper Access Server (FAS)

Fredhopper

- eCommerce company based in Amsterdam,
- Flagship product – Fredhopper Access Server (FAS)
- FAS provides search and merchandising IT services for e-Commerce companies.
- FAS is a concurrent distributed object based system
- Clients include: Toys "R" Us, Clarks, ASOS, Debenhams, Philips

Fredhopper Access Server (FAS)

FAS Concepts

Data a set of product items to search from

Data Manager transforms customer's data into FAS input data

Query Engine receives query (e.g. "adidas shoes") and returns a subset of items satisfying that query.

Configuration how to present that subset of items (with add-ons like upsell promotions etc.)

Business Manager provides an interface for changing configuration

Fredhopper Access Server (FAS)

FAS Concepts

Data a set of product items to search from

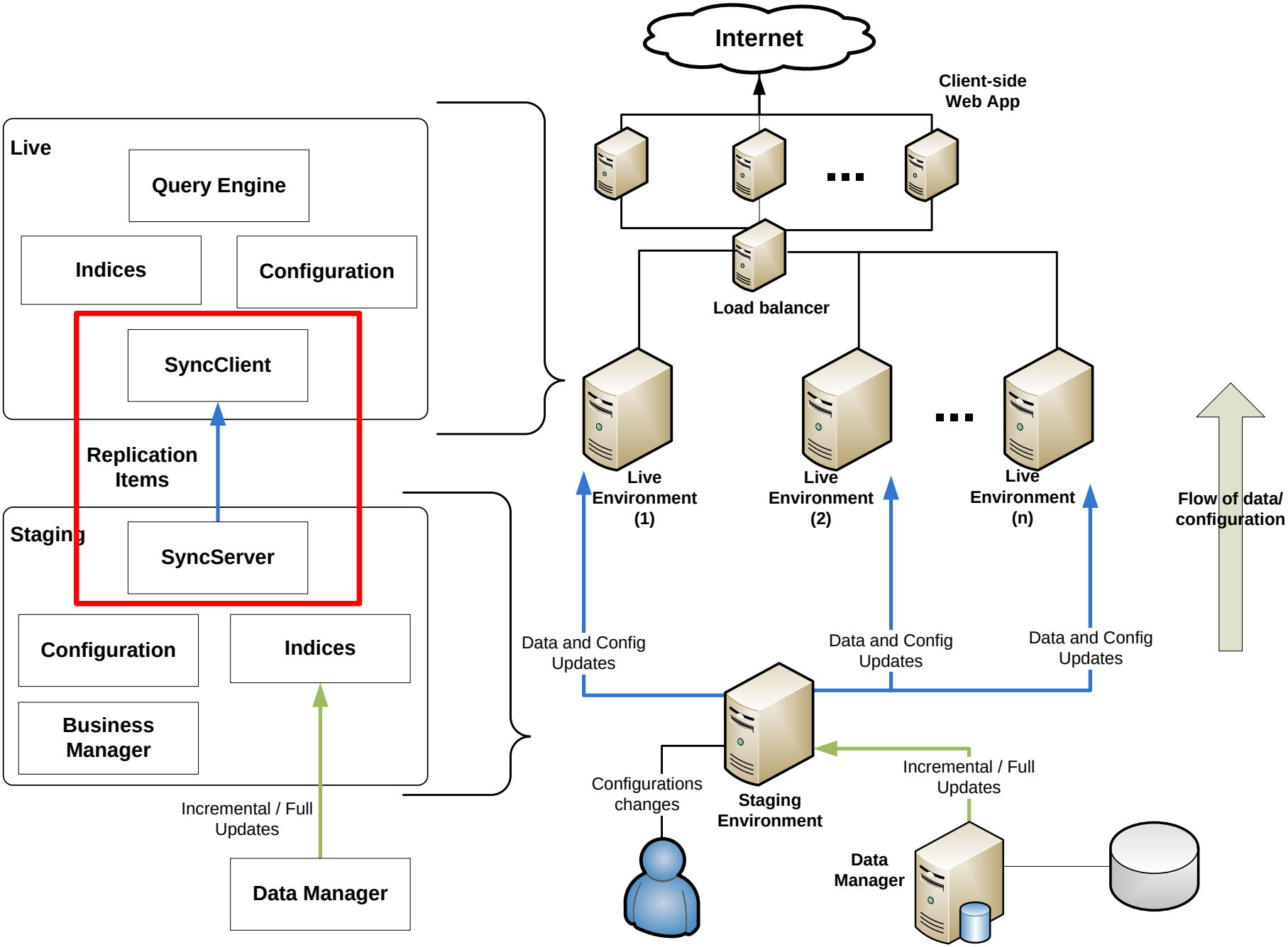
Data Manager transforms customer's data into FAS input data

Query Engine receives query (e.g. "adidas shoes") and returns a subset of items satisfying that query.

Configuration how to present that subset of items (with add-ons like upsell promotions etc.)

Business Manager provides an interface for changing configuration

Replication System updates data and configuration, and maintains their consistency throughout a FAS deployment (multi-threaded)

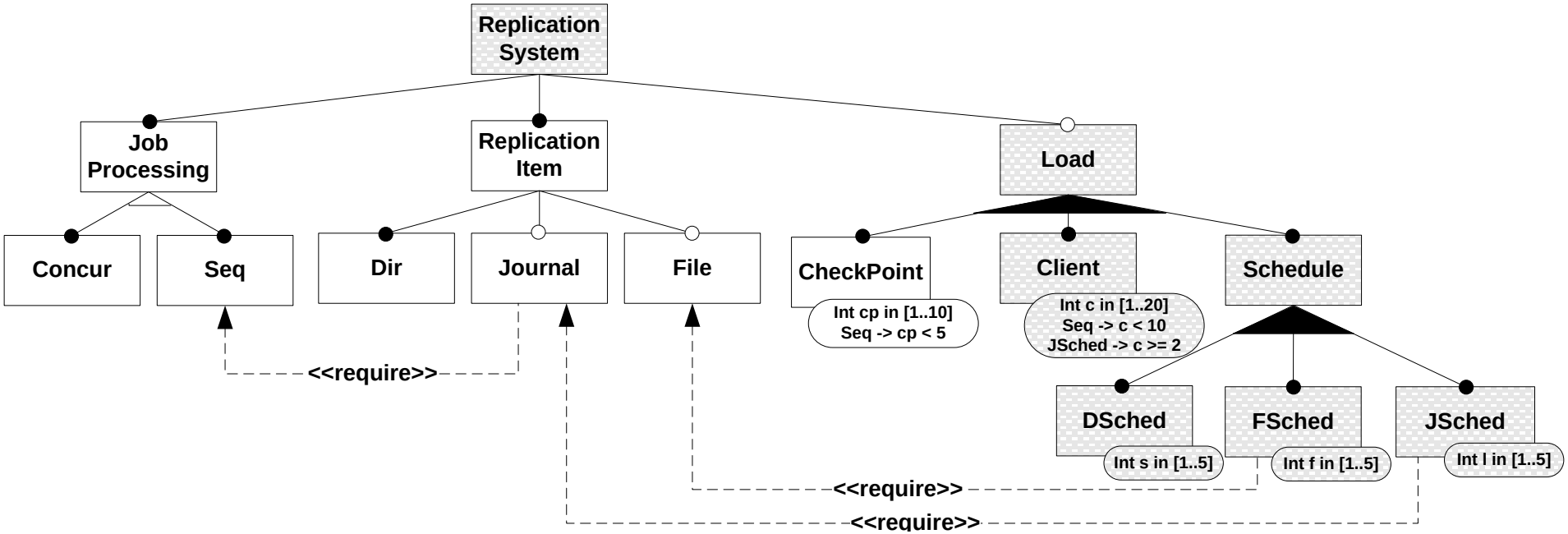

Internet
Client-side Web App
Load balancer
Live
Query Engine
Indices
Configuration
SyncClient
Replication Items
Staging
SyncServer
Configuration
Indices
Business Manager
Data Manager
Incremental / Full Updates
Live Environment (1)
Live Environment (2)
Live Environment (n)
Flow of data/ configuration
Data and Config Updates
Data and Config Updates
Data and Config Updates
Configurations changes
Staging Environment
Incremental / Full Updates
Data Manager

Replication System Product Line

Replication System Product Line

- synchronises configurations and data from the staging environment to all live environments
- consists of a server (SyncServer) and one or more clients (SyncClient)
- desired properties: deadlock and livelock freedom, replication consistency

Feature diagram of the Replication System



Variability (Features)

Replication Item data dependent variation

Job Processing resource dependent variation

Load resource dependent variation

μTVL model of the Replication System

```
root RS {
  group allof {
    JobProcessing { ... },
    ReplicationItem { ... },
    Load {
      group [1..3] {
        CheckPoint { ... },
        Client { Int c in [1 .. 20]; Seq -> c < 10; JSched -> c >= 2; },
        Schedule {
          group [1..3] {
            DSched { Int s in [1..5]; },
            FSched { Int f in [1..5]; require: File; },
            JSched { Int l in [1..5]; require: Journal; }
          }
        }}}}}
```

◂ Configuration ◂ Feature Interaction ◂ Implement Feature Interaction

Delta Modeling

Feature Load

Client Number of SyncClients

Schedule Number of locations in the file system at which changes to various replication item types are monitored: *DSched* for directories, *FSched* for file sets, and *JSched* for transaction journals.

Feature Interaction

Client and *JSched* require some extra implementation effort to make them interact properly:

$$\exists p: \; p \models \{Client\} \wedge p \models \{JSched\} \wedge p \not\models \{Client, JSched\}$$

▸ μTVL model

Structural Categorization

Load and *Schedule* are subcategorizations with no semantics:

$$\forall p: \; p \models F \iff p \models F \cup \{Load, Schedule\}$$

Delta Modeling

Structural Categorization

Load and *Schedule* are subcategorizations with no semantics:

$$\forall p: \; p \models F \iff p \models F \cup \{Load, Schedule\}$$

SPL Specification

Consider only features *RS*, *Load*, *Client*, *Schedule*, *DSched*, *FSched* and *JSched*: SPL specification $(\Psi', \models')$ where $\models' \subset \models$ and,

$$\begin{aligned}\Psi' = \quad & (\{RS\}, \varnothing, \{(RS, Load), (Load, Client), (Load, Schedule),\\ & (Schedule, DSched), (Schedule, FSched), (Schedule, JSched)\},\\ & \varnothing, \varnothing)\end{aligned}$$

Delta Modeling

Implement features in topological order of the feature diagram:

$$\langle RS, Load, Client, Schedule, DSched, FSched, JSched \rangle$$

Given SPL: $RS = (c, \varnothing, \varnothing, \varnothing)$, where c is a minimum core ABS model

```
class RSMain { } { new RSMain(); }
```

Delta Modeling

1. Implement feature *RS* with delta RD

```
delta RD {
  adds data SchedType = Dir | File | Journal; adds type CId = Int; ...
  adds class System(...) { ... }
  modifies class RSMain {
    adds Map<CP,Map<FId,Content>> datas = map[...];
    adds Map<SchedType,List<Schedule>> ss = map[...];
    adds Set<CId> cids = set[...];
    adds Unit run() {
      Map<CP,Map<FId,Content>> is = this.getDatas();
      List<Schedule> ss = this.getSchedules(); Set<CId> cs = this.getCids();
      new System(is,ss,cs); }
    adds List<Schedule> getSchedules() { return lookup(ss,Dir); }
    adds Map<CP,Map<FId,Content>> getDatas() { return datas; }
    adds Set<CId> getCids() { return cids; } } }
```

Step 1 results in SPL: $RS = (c, \{\text{RD}\}, \varnothing, \{(\text{RD}, \Phi')\})$:

Delta Modeling

2. No need to implement feature *Load*
3. Implement feature *Client* with delta CD

```
delta CD(Int c) {
  modifies class RSMain {
    modifies Set<CId> getCids() { return takeSet(RD.original(),c); }
  }}
```

Delta Modeling

4. Implement feature *DSched* with delta DD

```
delta DD(Int s) {
  modifies class RSMain {
    modifies List<Schedule> getSchedules() {
      return take(lookup(RD.original(),Dir),s);
    }}}
```

Delta Modeling

5. Implement feature *FSched* with delta FD

```
delta FD(Int f) {
  modifies class RSMain {
    modifies List<Schedule> getSchedules() {
      return take(lookup(RD.original(),File),f);
    }}}
```

Conflict between FD and delta DD: resolution with delta DFD

```
delta DFD {
  modifies class RSMain {
    modifies List<Schedule> getSchedules() {
      return concatenate(DD.original(),FD.original());
    }}}
```

Delta Modeling

6. Implement feature *JSched* with delta JD

```
delta JD(Int l) {
  modifies class RSMain {
    modifies List<Schedule> getSchedules() {
      return take(lookup(RD.original(),Journal),l);
    }}}
```

Implement feature interaction between *Client* and *JSched* with delta JCD

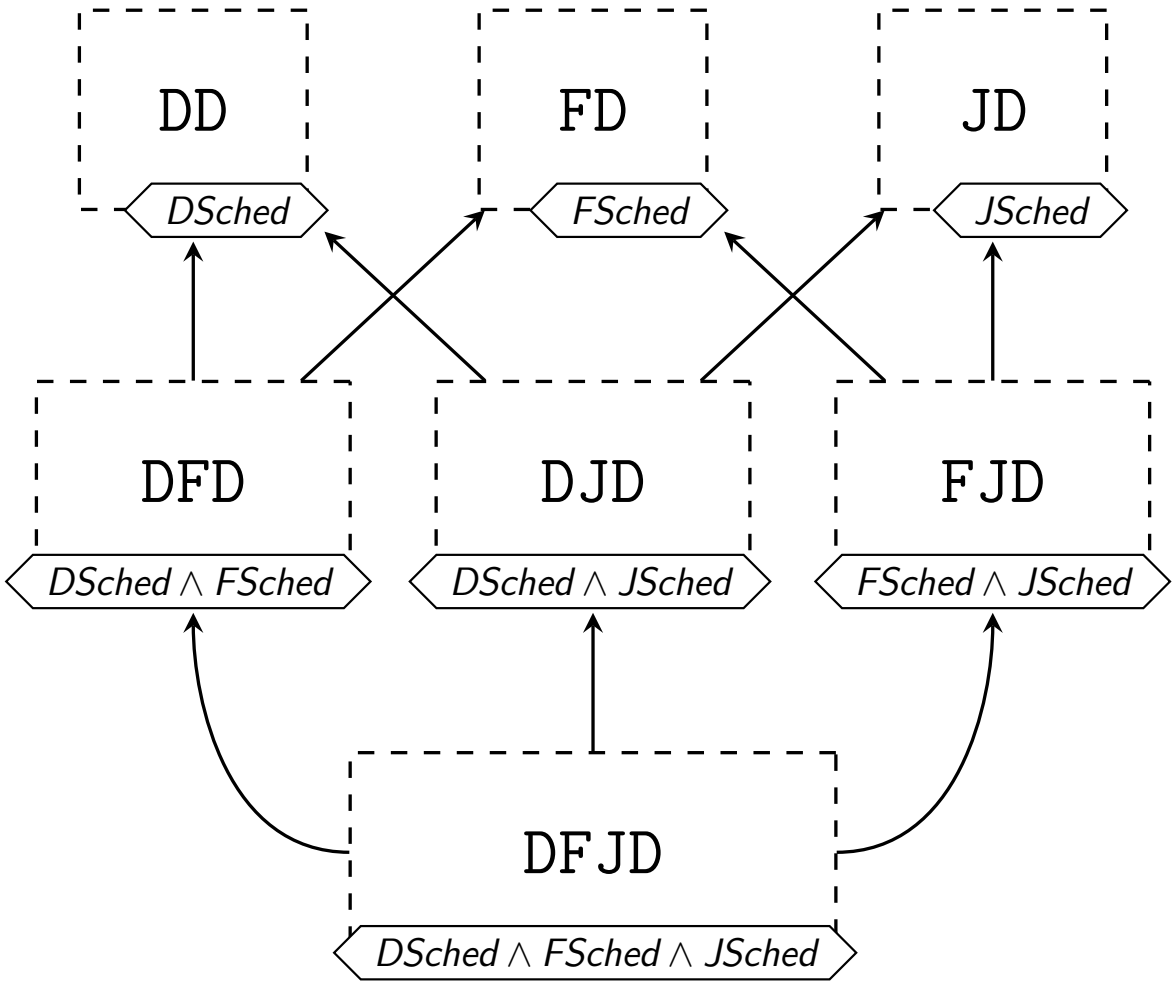
```
delta JCD {
  modifies class RSMain {
    modifies Set<CId> getCids() {
      Set<CId> cs = CD.original();
      if (size(cs) == 1) {
        cs = Insert(failSafe(),c);
      }
      return cs;
    }}}
```

▸ μTVL model

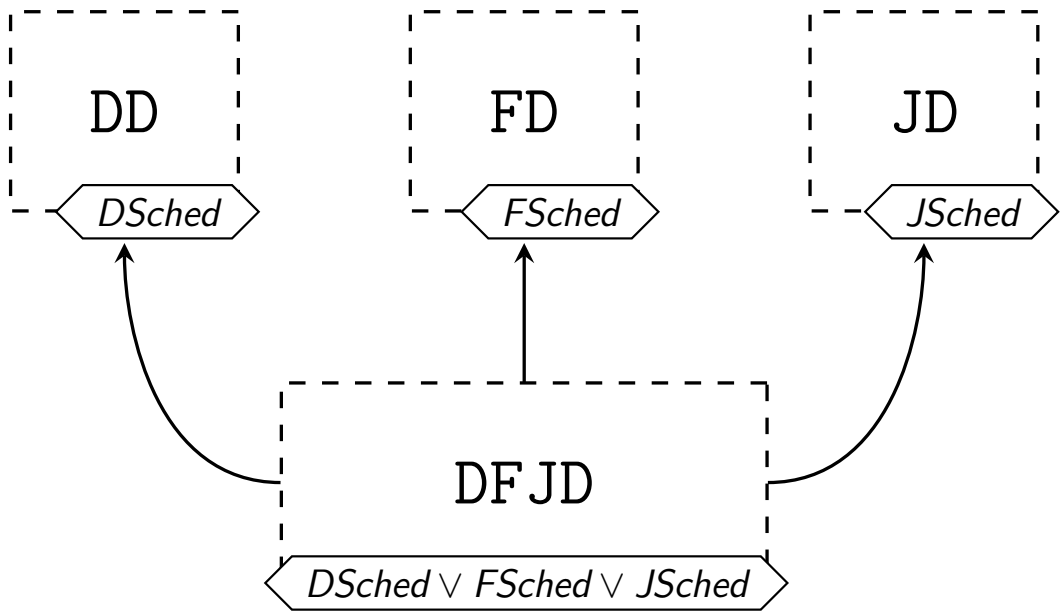
Delta Modeling

7. Resolve conflict between *DSched*, *FSched* and *JSched* ...

Delta Modeling



Delta Modeling



Delta Modeling

7. Resolve conflict between *DSched*, *FSched* and *JSched* with delta DFJD

```
delta DFJD(Bool ds, Bool fs, Bool js) {
  modifies class RSMain {
    modifies List<Schedule> getSchedules() {
      List<Schedule> ss = Nil;
      if (ds) { ss = DD.original(); }
      if (fs) { ss = concatenate(ss,FD.original()); }
      if (js) { ss = concatenate(ss,JD.original()); }
      return ss;
    }}}
```

Product Line Configuration & Selection

```
productline RS {
  features RS, Client, DSched, FSched, JSched;
  delta RD when RS;
  delta CD (Client.c) when Client after RD;
  delta DD(DSched.s) when DSched after RD;
  delta FD(FSched.f) when FSched after RD;
  delta JD(JSched.l) when JSched after RD;
  delta JCD when Client and JSched after CD, JD;
  delta DFJD(DSched, FSched, JSched)
            when DSched or FSched or JSched
            after DD, FD, JD;
}

product SingleClient (RS, Client{c=1}, DSched{s=3});
product TwoClients (RS, Client{c=2}, DSched{s=3});
```

▸ μTVL model

Statistics

Complete Case Study

- $\approx$ 5000 lines of codes
- 40 Classes
- 43 Interfaces
- 89 user-defined functions
- 17 user-defined data types
- 15 Features
- 10 (7) Deltas
- 96 (12024) Products

Feedback for ...

DMW

Completeness unambiguity, feature interaction, conflict resolution, error reduction

Flexibility three-way conflict resolution

Evolution relaxed empty initial product

Collaboration atomicity, noninterference

Tool support convert μTVL to Ψ

ABS

- Targeted **original**
- Feature Boolean
- Application condition*

Summary

So far...

- First account of using delta modeling to implement a system of industrial scale and of practical use
- Served as a test bed for DMW, providing feedback that was used to refine the workflow
- Evaluation of the practical applicability of ABS modeling language

Summary

So far...

- First account of using delta modeling to implement a system of industrial scale and of practical use
- Served as a test bed for DMW, providing feedback that was used to refine the workflow
- Evaluation of the practical applicability of ABS modeling language

Upcoming...

Timing constraint modelling timed behavior

Resource constraint modelling underlying resources

Verification specifying and verifying replication consistency