

Specification-based CSV Support in **VDM**

The 21st Overture Workshop

10th March 2023

Leo Freitas¹

Aaron John Buhagiar²

1. School of Computing, Newcastle University

2. Translational and Clinical Research Institute, Newcastle University

```
most file = path and file <> nil and
parser = parserChoice and strict = s and pos = {} and
data = empty_csv'(settings, headersToInstall);

--@doc loads CSV matrix through CSV parser IO, returning only if
launch | Debug
loadData() ==
(let mk_(ok, short_row_errors, data') = csv_read_data(file, pars
if ok then
(
if short_row_errors <> {} then
(
printf("CSV (IO) error: ignoring %1s short rows from %2s",
printErrors(short_row_errors);
if strict then
(
atomic (
data := empty_csv'(data'.settings, data'.headers);
pos := {};
);
--@doc if strict, clear data + pos; otherwise, carry on with i
return;
)
);
--@doc check invariants using the installed headers against read rows
let lcs: Errors = csv_invariants_failed(data') in
if (lcs = {}) then
(
--@doc if no file invariant errors
if file_invariant_check(data') = nil then
--@doc update the headers with names as well, if successful.
atomic (
data := data';
pos := lcs;
)
--@doc there file invariant errors
else
(
printf("CSV overall file invariant failed for %1s", [file]);
atomic (
data := if strict then empty_csv'(data'.settings, data'.headers) else da
pos := lcs;
```

Introduction

- Standard (RFC4180) data exchange format
- Ubiquitous use in a myriad of domains
 - Data science applications
 - Medical data and embedded
 - Payment systems
- Many variations and versions are used
- VDM standard CSV has limitations
- We created an improved CSV library with:
 - Data validation formal specification
 - Ease-of-use
 - High performance
- Distributed with the VDM Toolkit [1]



VDM standard CSV Library

- Slow and error-prone (e.g. low-level IO)
- Line-by-line parsing in VDM
- Limited support for various CSV formats
- Only hard-coded IO native calls available
 - Limits CSV performance and formats variety
- Imported data is of a wildcard (?) type
 - Users have to further introspect meaning
- Different from CSVReader
 - CSV as user data not structural information

1	2	3	4
1	,email	,email	,email
2	,Jacobah	,Myrtice.Jacobah@yopmail.com	,Myrtice.Jacobah@gmail.com
3	,Merell	,Ilse.Merell@yopmail.com	,Ilse.Merell@gmail.com
4	,Arley	,Karina.Arley@yopmail.com	,Karina.Arley@gmail.com
5	,Lunsford	,Audrie.Lunsford@yopmail.com	,Audrie.Lunsford@gmail.com
6	,Old	,Evaleen.Old@yopmail.com	,Evaleen.Old@gmail.com
7	,Riordan	,Lynde.Riordan@yopmail.com	,Lynde.Riordan@gmail.com
8	,Xerxes	,Kara-Lynn.Xerxes@yopmail.com	,Kara-Lynn.Xerxes@gmail.com
9	,Bendick	,Tonia.Bendick@yopmail.com	,Tonia.Bendick@gmail.com
10	,Drisko	,Joy.Drisko@yopmail.com	,Joy.Drisko@gmail.com
11	,Cecile	,Lanae.Cecile@yopmail.com	,Lanae.Cecile@gmail.com
12	,O'Rourke	,Ardeen.O'Rourke@yopmail.com	,Ardeen.O'Rourke@gmail.com
13	,Donoghue	,Cherrita.Donoghue@yopmail.com	,Cherrita.Donoghue@gmail.com
14	,Federica	,Gwenneth.Federica@yopmail.com	,Gwenneth.Federica@gmail.com
15	,Cottle	,Tera.Cottle@yopmail.com	,Tera.Cottle@gmail.com
16	,Peg	,Juliane.Peg@yopmail.com	,Juliane.Peg@gmail.com
17	,Milde	,Raf.Milde@yopmail.com	,Raf.Milde@gmail.com
18	,Izaak	,Belva.Izaak@yopmail.com	,Belva.Izaak@gmail.com
19	,Jillane	,Correy.Jillane@yopmail.com	,Correy.Jillane@gmail.com
20	,Mathilde	,Christian.Mathilde@yopmail.com	,Christian.Mathilde@gmail.com
21	,Talia	,Katuscha.Talia@yopmail.com	,Katuscha.Talia@gmail.com
22	,Ax	,Helena.Ax@yopmail.com	,Helena.Ax@gmail.com
23	,Dash	,Nanete.Dash@yopmail.com	,Nanete.Dash@gmail.com
24	,Joeann	,Alameda.Joeann@yopmail.com	,Alameda.Joeann@gmail.com
25	,Zamora	,Margarette.Zamora@yopmail.com	,Margarette.Zamora@gmail.com
26	,Madelene	,Arlena.Madelene@yopmail.com	,Arlena.Madelene@gmail.com
27	,Astra	,Jennica.Astra@yopmail.com	,Jennica.Astra@gmail.com
28	,Rozanna	,Bernardine.Rozanna@yopmail.com	,Bernardine.Rozanna@gmail.com
29	,Kolnick	,Nariko.Kolnick@yopmail.com	,Nariko.Kolnick@gmail.com
30	,Kaete	,Sean.Kaete@yopmail.com	,Sean.Kaete@gmail.com
31	,Auberbach	,Ayn.Auberbach@yopmail.com	,Ayn.Auberbach@gmail.com
32	,Rillings	,Ardys.Rillings@yopmail.com	,Ardys.Rillings@gmail.com
33	,Ruvolo	,Neila.Ruvolo@yopmail.com	,Neila.Ruvolo@gmail.com
34	,Barrus	,Rani.Barrus@yopmail.com	,Rani.Barrus@gmail.com
35	,Prouty	,Millie.Prouty@yopmail.com	,Millie.Prouty@gmail.com
36	,Tannie	,Suzette.Tannie@yopmail.com	,Suzette.Tannie@gmail.com
37	,Lay	,Steffane.Lay@yopmail.com	,Steffane.Lay@gmail.com
38	,Thornburg	,Carly.Thornburg@yopmail.com	,Carly.Thornburg@gmail.com
39	,Meli	,Clary.Meli@yopmail.com	,Clary.Meli@gmail.com
40	,Vary	,Dorthy.Vary@yopmail.com	,Dorthy.Vary@gmail.com
41	,Munn	,Ketti.Munn@yopmail.com	,Ketti.Munn@gmail.com
42	,Syd	,Meg.Syd@yopmail.com	,Meg.Syd@gmail.com
43	,Ivens	,Viki.Ivens@yopmail.com	,Viki.Ivens@gmail.com
44	,Brandice	,Dorice.Brandice@yopmail.com	,Dorice.Brandice@gmail.com
45	,Randene	,Delilah.Randene@yopmail.com	,Delilah.Randene@gmail.com
46	,Urania	,Deirdre.Urania@yopmail.com	,Deirdre.Urania@gmail.com
47	,Tufts	,Cissiee.Tufts@yopmail.com	,Cissiee.Tufts@gmail.com
48	,Imelida	,Ebonee.Imelida@yopmail.com	,Ebonee.Imelida@gmail.com
49	,Novo	,Annona.Novo@yopmail.com	,Annona.Novo@gmail.com
50	,Novo	,Annona.Novo@yopmail.com	,Annona.Novo@gmail.com

Design
Principles

Simple

Accurate

Fast

Effective

Types and Parsers

- CSV Lib offers basic types for column typing

```
CSVType = <Integer> | <Float> | <String> | <Boolean>;
```

```
CSVValue = int | real | String | bool;
```

- Multiple parsers are available

```
CSVParser = <Native> | <Univocity> | <Apache> | <OpenCSV> | <QuirkCSV>;
```

Simple: Ease of Use

- Accessible entry points that abstract from IO native calls
 - 1) Out-of-the-box setup
 - 2) Configurable (e.g. CSV settings)
- Allows direct native calls for better extensibility and control (3)
- Data validation can be formally specified

1

```
--@doc load file path with given headers and no invariants: insists on knowing header types.
Launch | Debug
loadSimpleHeadersCSV(path: Path, parserChoice: CSVParser, headerTypes: seq1 of CSVType, s: bool) ==
  (
    let headers: Headers0 =
      [ mk Header0(
        --@doc top-level call: given a file and typed headers (possibly with invariants)
        -- loads the CSV as part of the state

```

2

```
Launch | Debug
loadCSV(path: Path, parserChoice: CSVParser, settings: CSVSettings, headersToInstall: Headers0, s: bool) ==
  (
    printf("Loading CSV file %1s\n", [path]);
    if file_status(path) <> <Valid> then

```

3

```
Launch | Debug
csv_read_data: Path * CSVParser * CSVSettings * Headers0 -> bool * Errors * Data0
csv_read_data(path, parser, settings, headers) == is not yet specified
pre file_status(path) = <Valid>
post
  (let mk (ok, -, mk Data0(settings', headers', matrix', finv')) = RESULT in
```

Simple: Configurable Setup

- CSV entries strong typing defined through semantic headers as:
 - Column Name
 - Datatype
 - Default Value
 - Cell invariant
 - Column Invariant
- CSV settings define expected file properties as:
 - Presence of blank lines
 - Existence of a header row
 - Comment string

```
EXAMPLE_DATA_HEADERS: Headers =  
    [mk_Header0("Name"      , <String> ,      "Joe", nil, COL_INVARIANT_UNIQUE_NAME),  
     mk_Header0("Age"      , <Integer>,      MIN_AGE, CELL_INVARIANT_AGE, nil),  
     mk_Header0("Weight(Kg)", <Float> , MIN_WEIGHT_KG, CELL_INVARIANT_WEIGHT, nil),  
     mk_Header0("Height(cm)", <Float> , MIN_HEIGHT_CM, CELL_INVARIANT_HEIGHT, nil),  
     mk_Header0("BMI"      , <Float> ,      MIN_BMI, CELL_INVARIANT_BMI, nil)  
    ];
```

Simple: Reporting

- Simple and descriptive reporting
 - Short rows (i.e. not enough columns)
 - Declared type violations (e.g. string for nat)
 - User defined invariant violations
- Provides cell locations for correction
- Striving to have a strongly-typed CSV

```
successful for "CSVShortRowExample.csv.out"  
CSV file "CSVInvFailedExample.csv"  
Up with <Univocity> parser and 5 headers for "CSVInvFailedExample.csv"  
error: ignoring 1 short rows from "CSVInvFailedExample.csv"  
(1, 5): "CSV row 1 is too short for header: expected 5 columns"  
13 invariant failures for "CSVInvFailedExample.csv": 13 CSV invariant failures  
(1, 1): "Invalid col invariant: repeated names"  
(2, 1): "Invalid col invariant: repeated names"  
(2, 2): "Invalid cell invariant: below minimal age"  
(3, 1): "Invalid col invariant: repeated names"  
(3, 2): "Invalid cell invariant: above maximal age"  
(3, 3): "Invalid cell invariant: above maximal weight"  
(3, 5): "Invalid cell invariant: above maximal BMI"  
(4, 1): "Invalid col invariant: repeated names"  
(4, 4): "Invalid cell invariant: above maximal height"  
(4, 5): "Invalid cell invariant: below minimal BMI"  
(5, 1): "Invalid col invariant: repeated names"  
(6, 1): "Invalid col invariant: repeated names"  
(7, 1): "Invalid col invariant: repeated names"  
/ print successful for "CSVInvFailedExample.csv.out"  
/ invariant failure at 13 cells:  
(1, 1): "Invalid col invariant: repeated names"  
(2, 1): "Invalid col invariant: repeated names"  
(2, 2): "Invalid cell invariant: below minimal age"  
(3, 1): "Invalid col invariant: repeated names"  
(3, 2): "Invalid cell invariant: above maximal age"  
(3, 3): "Invalid cell invariant: above maximal weight"  
(3, 5): "Invalid cell invariant: above maximal BMI"  
(4, 1): "Invalid col invariant: repeated names"
```


Accurate

- Checks on imported data
 - Short rows
 - Declared header type validation
- User defined invariants
 1. Cell invariants
 2. Column invariants
 3. Row invariants
 4. File Invariants
- Lambda abstractions capture invariants as record fields
- Invariants return a **Reason**

1

```
CSVCellInv = (CSVType * CSVValue -> Reason);
```

2

```
CSVColInv = (Header0 * TransposedRow -> Reason);
```

3

```
CSVRowInv = (Headers0 * Row -> Reason);
```

4

```
CSVFileInv = (Headers0 * Matrix -> Reason);
```

Accurate: Cell Invariant

- Invariants that act upon cells directly
- Allows for cell validation
- Examples
 - Upper/lower bounds
 - Cell text validation
 - Specific value enforcement
 - etc...

```
CELL_INVARIANT_AGE: CSVCellInv =  
  (lambda -: CSVType, v: CSVValue &  
    if v < MIN_AGE then  
      "below minimal age"  
    else if v > MAX_AGE then  
      "above maximal age"  
    else  
      nil);  
  
CELL_INVARIANT_WEIGHT: CSVCellInv =  
  (lambda -: CSVType, v: CSVValue &  
    if v < MIN_WEIGHT_KG then  
      "below minimal weight"  
    else if v > MAX_WEIGHT_KG then  
      "above maximal weight"  
    else  
      nil);
```

Accurate: Column Invariant

- Column-wide invariant
- Allows for validation per header across rows
- Example
 - Uniqueness
 - Dependence

```
COL_INVARIANT_ORDERED: CSVColInv =  
  (lambda h: Header0, c: TransposedRow &  
    if not h.type in set {<Integer>, <Float>} then  
      "not number typed"  
    else if --not (forall i, j in set inds c & i < j => c(i) <= c(j))  
              (exists i, j in set inds c & i < j and c(i) > c(j)) then  
      "column is not in ascending order"  
    else  
      nil  
  );
```

Accurate: Row Invariant

- Row-wide invariant
- Allows for validation per row across all headers
- Examples
 - Consistency
 - Dependence
 - Redundance

```
ROW_INVARIANT_BMI_CONSISTENCY: CSVRowInv =
  (lambda h: Headers0, r: Row &
    if BMI_ROW_INDEX > len h then
      "invalid BMI header"
    else
      (let bmi: real = calculated_bmi(r) in
        if floor(bmi) = floor(r(BMI_ROW_INDEX)) then
          nil
        else
          "invalid BMI for given CSV weight and height: expected " ^
            val2seq_of_char[real](bmi) ^"; found " ^ val2seq_of_char[real](r(BMI_ROW_INDEX))
      )
  );
```

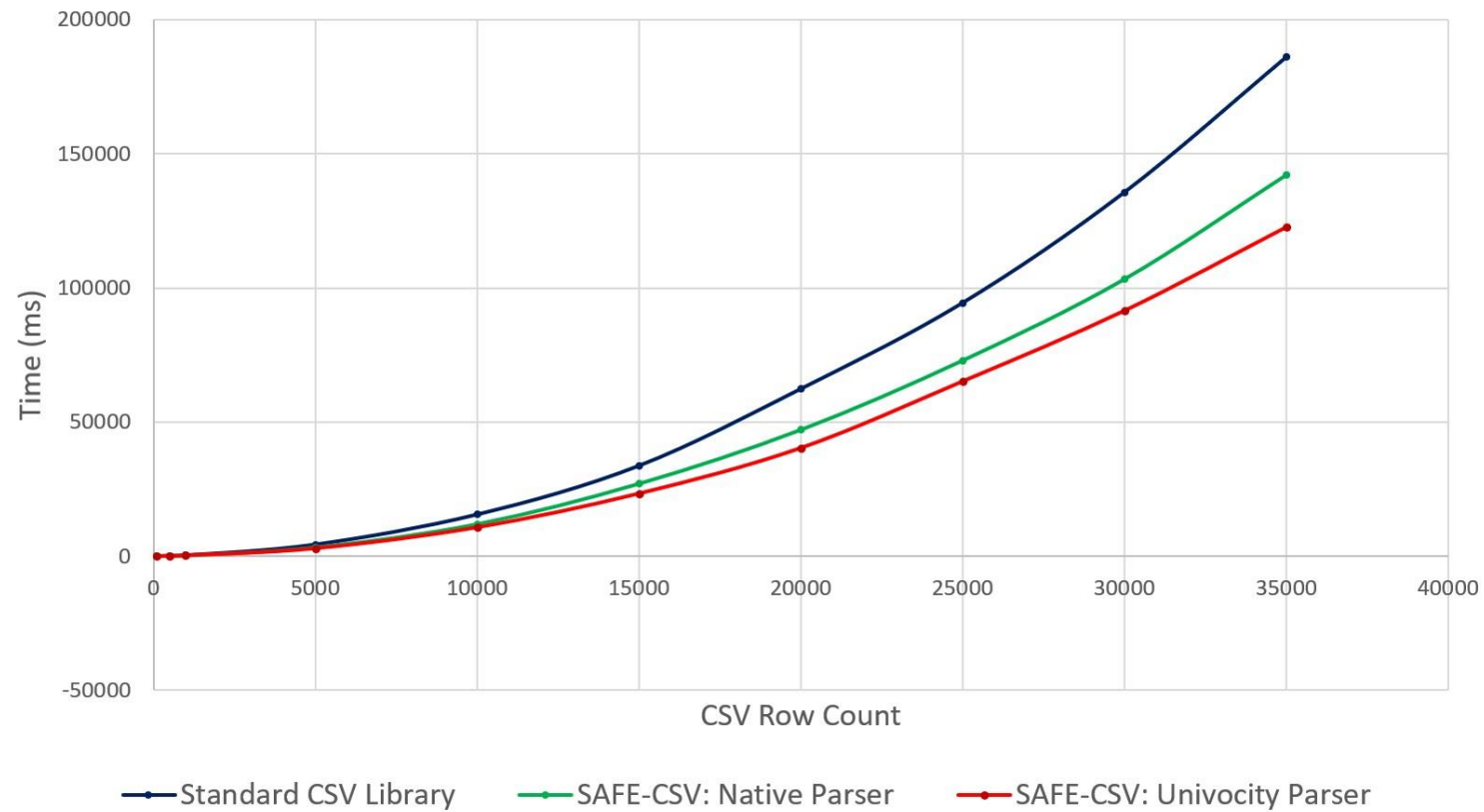
Accurate: File Invariant

- Invariant across all CSV cells
- Example
 - Dependence
 - Redundancy

```
FILE_INVARIANT_CONSISTENCY: CSVFileInv =
  (lambda h: Headers0, m: Matrix &
    if NAME_ROW_INDEX > len h then
      "invalid name header"
    else if AGE_ROW_INDEX > len h then
      "invalid age header"
    else
      (
        let
          dups : set of nat = { j | i, j in set inds m.cells & i <> j
                                and
                                check_name_age_uniqueness(m.cells(i),
                                                            m.cells(j))
          in
            if (dups <> {}) then
              "duplicate persons found in row(s) " ^ val2seq_of_char[set of nat](dups)
            else
              nil
```

Fast

Standard CSV Library, SAFE-CSV: Native Parser and SAFE-CSV: Univocity Parser



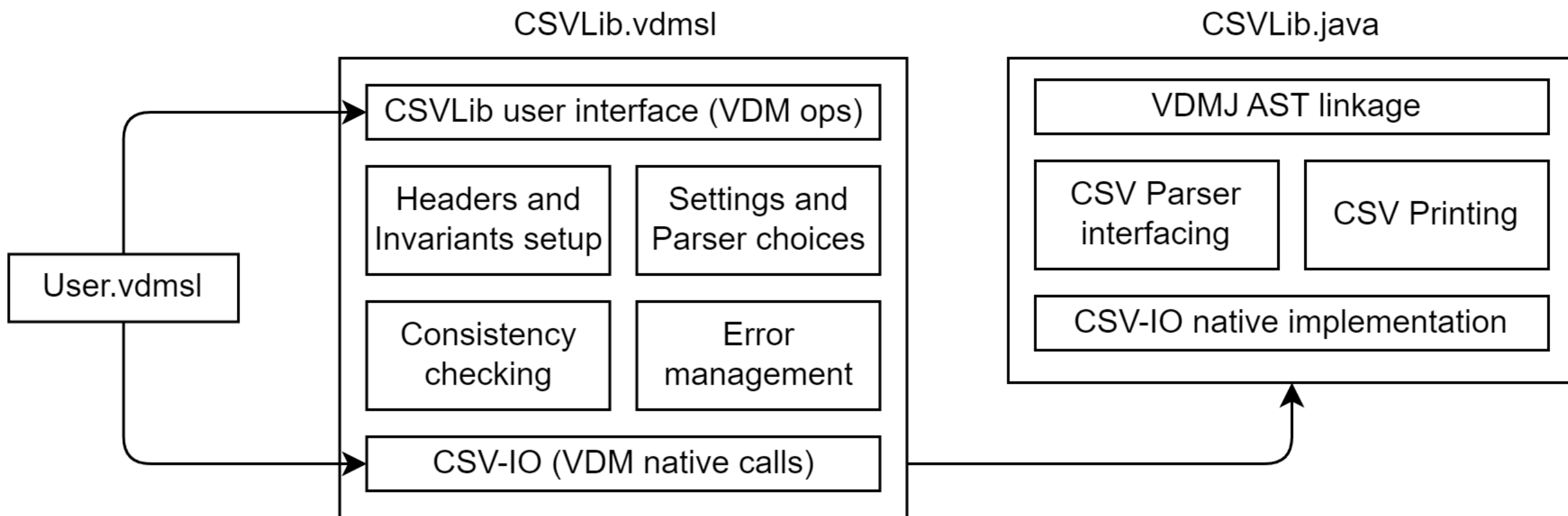
Effective

- Multiple CSV format variants
 - Better tolerance to CSV format variability (e.g. CSV UTF8, MS-DOS, Mac, etc.)
 - Delegate CSV format type to parsers (e.g. formalisation of CSV format itself)
- Ease of use: use of VDM state and operations as entry points
- Improved validation, error handling and reporting
- Faster performance through multiple CSV parsers

Effective

- Tested on multiple CSV format variants
 - CSV UTF8, MS-DOS and Mac
- Applied to multiple domains
 - ScubaTx organ preservation device medical device logs
 - UNOS (United Network for Organ Sharing) lung transplant history logs
 - EMV payment system transaction logs
 - Personalised medicine DSL “certificate of treatment”
 - Neonatal dialyser finite state machine control system definitions
 - Etc.

Library Architecture



Future Work

- Implementation of debugging environment
- Improved variety of CSV formats
 - Nested CSVs
 - CSV Settings
 - More CSV data types
 - Multiple CSV headers per file

Thanks for Listening