Data-transformer library

Michael Raskin

Moscow Center for Continuous Mathematical Education

April 2013

User-visible problem:

- Input data is garbage
- Input data requirements are constantly changing

Programmer's problems:

- How to define data format?
- How to keep all the code that touches a single piece of data always in sync?

User-visible problem:

- Input data is garbage
- Input data requirements are constantly changing

Programmer's problems:

- How to define data format?
- How to keep all the code that touches a single piece of data always in sync?

Spreadsheet: “555-01-55 uses 2-digit year format, corrected to 555-01-1955”
User: “Let’s just sort the first column without sorting the second one then”

Spreadsheet: “555-01-55 uses 2-digit year format, corrected to 555-01-1955”
User: “Let’s just sort the first column without sorting the second one then”

Initial scope:

- Validating CSV data
- Storing the data into a database
- Presenting data in a PDF

Later added to scope:

- Web forms

Currently web forms, csv files and SQL DB is used both for input and for output in production; PDF export is also used all the time
And validating logic keeps getting more complex. . .

Initial scope:

- Validating CSV data
- Storing the data into a database
- Presenting data in a PDF

Later added to scope:

- Web forms

Currently web forms, csv files and SQL DB is used both for input and for output in production; PDF export is also used all the time
And validating logic keeps getting more complex. . .

Initial scope:

- Validating CSV data
- Storing the data into a database
- Presenting data in a PDF

Later added to scope:

- Web forms

Currently web forms, csv files and SQL DB is used both for input and for output in production; PDF export is also used all the time
And validating logic keeps getting more complex. . .

Initial scope:

- Validating CSV data
- Storing the data into a database
- Presenting data in a PDF

Later added to scope:

- Web forms

Currently web forms, csv files and SQL DB is used both for input and for output in production; PDF export is also used all the time
And validating logic keeps getting more complex. . .

Initial scope:

- Validating CSV data
- Storing the data into a database
- Presenting data in a PDF

Later added to scope:

- Web forms

Currently web forms, csv files and SQL DB is used both for input and for output in production; PDF export is also used all the time

And validating logic keeps getting more complex. . .

Initial scope:

- Validating CSV data
- Storing the data into a database
- Presenting data in a PDF

Later added to scope:

- Web forms

Currently web forms, csv files and SQL DB is used both for input and for output in production; PDF export is also used all the time
And validating logic keeps getting more complex. . .

Portable schemas

- Portable
- Declarative
- Clear
- Restricted functionality

Would my application survive porting anyway?
(Spoiler: not likely)
Make simple things declarative and complex things possible!

Portable schemas

- Portable
- Declarative
- Clear
- Restricted functionality

Would my application survive porting anyway?
(Spoiler: not likely)
Make simple things declarative and complex things possible!

Portable schemas

- Portable
- Declarative
- Clear
- Restricted functionality

Would my application survive porting anyway?
(Spoiler: not likely)
Make simple things declarative and complex things possible!

Portable schemas

- Portable
- Declarative
- Clear
- Restricted functionality

Would my application survive porting anyway?
(Spoiler: not likely)
Make simple things declarative and complex things possible!

Classical procedural programming: there is code, and you pass it configuration and data
Classical OOP: there is an object, which contains configuration, and data, and is associated with code
Hybrid model: create an object keeping configuration and code, and briefly pass it actual data (usually used for complex processing of data streams)

Process one record at a time

Schema: an s-expression, written by hand (can contain literal function values and what not)

data-transformer instance:

- Stores the schema in a better format with some caches, hashes, etc.
- Briefly holds the data during processing

Data is not stored, it gets loaded, processed and exported in a quick succession

Process one record at a time

Schema: an s-expression, written by hand (can contain literal function values and what not)

data-transformer instance:

- Stores the schema in a better format with some caches, hashes, etc.
- Briefly holds the data during processing

Data is not stored, it gets loaded, processed and exported in a quick succession

Process one record at a time

Schema: an s-expression, written by hand (can contain literal function values and what not)

data-transformer instance:

- Stores the schema in a better format with some caches, hashes, etc.
- Briefly holds the data during processing

Data is not stored, it gets loaded, processed and exported in a quick succession

Process one record at a time
Schema: an s-expression, written by hand (can contain literal function values and what not)
data-transformer instance:

- Stores the schema in a better format with some caches, hashes, etc.
- Briefly holds the data during processing

Data is not stored, it gets loaded, processed and exported in a quick succession

Process one record at a time
Schema: an s-expression, written by hand (can contain literal function values and what not)
data-transformer instance:

- Stores the schema in a better format with some caches, hashes, etc.
- Briefly holds the data during processing

Data is not stored, it gets loaded, processed and exported in a quick succession

Record is an array of fields

Fields have parameters

Specify them or use default values; default values can depend on the values of other fields

Data is an array of values stored in the same order as field definitions

Record is an array of fields

Fields have parameters

Specify them or use default values; default values can depend on the values of other fields

Data is an array of values stored in the same order as field definitions

Input verification

- Day, month and year should be numbers

  input string verification

- Year should be in the 20th or the 21th century

  field content verification

- Date should be possible, 31-02-2013 is a bad idea

  global (cross-field) verification

Input verification

- Day, month and year should be numbers

  input string verification
- Year should be in the 20th or the 21th century

  field content verification
- Date should be possible, 31-02-2013 is a bad idea

  global (cross-field) verification

Input verification

- Day, month and year should be numbers

  input string verification
- Year should be in the 20th or the 21th century

  field content verification
- Date should be possible, 31-02-2013 is a bad idea

  global (cross-field) verification

Input verification

- Day, month and year should be numbers
  input string verification
- Year should be in the 20th or the 21th century
  field content verification
- Date should be possible, 31-02-2013 is a bad idea
  global (cross-field) verification

```
(defparameter *basic-schema*
  `((((:code-name :captcha-answer)
     (:display-name "Task answer")
     (:type :int)
     (:string-verification-error
     "Please enter a number")
     (:data-verification-error "Wrong answer")
     (:string-export ,(constantly "")))
    ((:code-name :email)
     (:display-name "Email")
     (:type :string)
     , (matcher "^(.+@.+[.].+|)\$")
     (:string-verification-error
     "Email is specified but it doesn't
       look like a valid email address"))))
```

```
(let
  ((schema (transformer-schema-edit-field
              *basic-schema* :captcha-answer
      (lambda (x)
        (set-> x :data-verification
  (lambda (y)
    (and y (= y captcha-answer))))))))
  ; some code using the schema
  )
```

Typical attributes:

- code name (for HTML form, SQL schemas, etc.)

```
    (:code-name :captcha-answer)
```

- readable name

```
    (:display-name "Task answer")
```

- field type (mainly for SQL schemas; also sets reasonable defaults for validation and parsing)

```
    (:type :int)
```

- validation procedures and error messages

```
    (:string-verification-error
        "Please enter a number")
```

- data formatting

```
    (:string-export ,(constantly ""))
```

Typical attributes:

- code name (for HTML form, SQL schemas, etc.)

```
    (:code-name :captcha-answer)
```

- readable name

```
    (:display-name "Task answer")
```

- field type (mainly for SQL schemas; also sets reasonable defaults for validation and parsing)

```
    (:type :int)
```

- validation procedures and error messages

```
    (:string-verification-error
        "Please enter a number")
```

- data formatting

```
    (:string-export ,(constantly ""))
```

Typical attributes:

- code name (for HTML form, SQL schemas, etc.)
```
    (:code-name :captcha-answer)
```
- readable name
```
    (:display-name "Task answer")
```
- field type (mainly for SQL schemas; also sets reasonable defaults for validation and parsing)
```
    (:type :int)
```
- validation procedures and error messages
```
    (:string-verification-error
        "Please enter a number")
```
- data formatting
```
    (:string-export ,(constantly ""))
```

Typical attributes:

- code name (for HTML form, SQL schemas, etc.)

```
    (:code-name :captcha-answer)
```

- readable name

```
    (:display-name "Task answer")
```

- field type (mainly for SQL schemas; also sets reasonable defaults for validation and parsing)

```
    (:type :int)
```

- validation procedures and error messages

```
    (:string-verification-error
        "Please enter a number")
```

- data formatting

```
    (:string-export ,(constantly ""))
```

Typical attributes:

- code name (for HTML form, SQL schemas, etc.)

```
    (:code-name :captcha-answer)
```

- readable name

```
    (:display-name "Task answer")
```

- field type (mainly for SQL schemas; also sets reasonable defaults for validation and parsing)

```
    (:type :int)
```

- validation procedures and error messages

```
    (:string-verification-error
        "Please enter a number")
```

- data formatting

```
    (:string-export ,(constantly ""))
```

CAPTCHA verification is injected into the schema right before use

```
(transformer-schema-edit-field
   *basic-schema* :captcha-answer
   (lambda (x)
     (set-> x :data-verification
       (lambda (y)
         (and y (= y captcha-answer))))))
```

Channel-specific features

- CSV: convert date fields into triples of fields for data components (also for web forms)
- SQL: specifying foreign keys; WHERE-conditions and source tables for generating queries
- Web forms: HTTP POST requests; file upload handler
- Web forms and PDF: preparation for CL-Emb templates

Was it a good idea?

- Reinventing the wheel? No previous wheel found
- Very few cases of save and load (or similar) code being mismatched. Nice
- Complex checks are still simple to integrate
- The more advanced, the less portable
- Some schema field parameters are coupled not just to Common Lisp, but to CLSQL, CL-Emb, etc.
- Feature-poor portable declarative schemas are generated automatically and correctly when needed
- Schema-using code mostly untouched; individual schemas relatively short and simple. Helps near deadlines in understaffed projects
- Small API quirks accumulate. Not specific to our library
- Wasteful implementations of some functionality

Was it a good idea?

- Reinventing the wheel? No previous wheel found
- Very few cases of save and load (or similar) code being mismatched. Nice
- Complex checks are still simple to integrate
- The more advanced, the less portable
- Some schema field parameters are coupled not just to Common Lisp, but to CLSQL, CL-Emb, etc.
- Feature-poor portable declarative schemas are generated automatically and correctly when needed
- Schema-using code mostly untouched; individual schemas relatively short and simple. Helps near deadlines in understaffed projects
- Small API quirks accumulate. Not specific to our library
- Wasteful implementations of some functionality

Was it a good idea?

- Reinventing the wheel? No previous wheel found
- Very few cases of save and load (or similar) code being mismatched. Nice
- Complex checks are still simple to integrate
- The more advanced, the less portable
- Some schema field parameters are coupled not just to Common Lisp, but to CLSQL, CL-Emb, etc.
- Feature-poor portable declarative schemas are generated automatically and correctly when needed
- Schema-using code mostly untouched; individual schemas relatively short and simple. Helps near deadlines in understaffed projects
- Small API quirks accumulate. Not specific to our library
- Wasteful implementations of some functionality

Was it a good idea?

- Reinventing the wheel? No previous wheel found
- Very few cases of save and load (or similar) code being mismatched. Nice
- Complex checks are still simple to integrate
- The more advanced, the less portable
- Some schema field parameters are coupled not just to Common Lisp, but to CLSQL, CL-Emb, etc.
- Feature-poor portable declarative schemas are generated automatically and correctly when needed
- Schema-using code mostly untouched; individual schemas relatively short and simple. Helps near deadlines in understaffed projects
- Small API quirks accumulate. Not specific to our library
- Wasteful implementations of some functionality

Was it a good idea?

- Reinventing the wheel? No previous wheel found
- Very few cases of save and load (or similar) code being mismatched. Nice
- Complex checks are still simple to integrate
- The more advanced, the less portable
- Some schema field parameters are coupled not just to Common Lisp, but to CLSQL, CL-Emb, etc.
- Feature-poor portable declarative schemas are generated automatically and correctly when needed
- Schema-using code mostly untouched; individual schemas relatively short and simple. Helps near deadlines in understaffed projects
- Small API quirks accumulate. Not specific to our library
- Wasteful implementations of some functionality

Was it a good idea?

- Reinventing the wheel? No previous wheel found
- Very few cases of save and load (or similar) code being mismatched. Nice
- Complex checks are still simple to integrate
- The more advanced, the less portable
- Some schema field parameters are coupled not just to Common Lisp, but to CLSQL, CL-Emb, etc.
- Feature-poor portable declarative schemas are generated automatically and correctly when needed
- Schema-using code mostly untouched; individual schemas relatively short and simple. Helps near deadlines in understaffed projects
- Small API quirks accumulate. Not specific to our library
- Wasteful implementations of some functionality

Was it a good idea?

- Reinventing the wheel? No previous wheel found
- Very few cases of save and load (or similar) code being mismatched. Nice
- Complex checks are still simple to integrate
- The more advanced, the less portable
- Some schema field parameters are coupled not just to Common Lisp, but to CLSQL, CL-Emb, etc.
- Feature-poor portable declarative schemas are generated automatically and correctly when needed
- Schema-using code mostly untouched; individual schemas relatively short and simple. Helps near deadlines in understaffed projects
- Small API quirks accumulate. Not specific to our library
- Wasteful implementations of some functionality

Was it a good idea?

- Reinventing the wheel? No previous wheel found
- Very few cases of save and load (or similar) code being mismatched. Nice
- Complex checks are still simple to integrate
- The more advanced, the less portable
- Some schema field parameters are coupled not just to Common Lisp, but to CLSQL, CL-Emb, etc.
- Feature-poor portable declarative schemas are generated automatically and correctly when needed
- Schema-using code mostly untouched; individual schemas relatively short and simple. Helps near deadlines in understaffed projects
- Small API quirks accumulate. Not specific to our library
- Wasteful implementations of some functionality

Was it a good idea?

- Reinventing the wheel? No previous wheel found
- Very few cases of save and load (or similar) code being mismatched. Nice
- Complex checks are still simple to integrate
- The more advanced, the less portable
- Some schema field parameters are coupled not just to Common Lisp, but to CLSQL, CL-Emb, etc.
- Feature-poor portable declarative schemas are generated automatically and correctly when needed
- Schema-using code mostly untouched; individual schemas relatively short and simple. Helps near deadlines in understaffed projects
- Small API quirks accumulate. Not specific to our library
- Wasteful implementations of some functionality

Thanks for your attention!