Comparison of document-verification languages

Margus Freudenthal

Theory days

Kääriku

30.01.2009

Background

- Cybernetica is working on Customs engine
- A Customs information system
- Processes customs documents
  - Import and export declarations
  - TIR carnets
  - Export reports
  - Manifests
  - Warehousing notices
  - etc.

Background

- Frequently recurring problem is verifying the correctness of the documents submitted to the Customs
- There are quite a lot of verification rules
- The rules are frequently changing
- The rules must be readable to domain specialists
- We used a **domain-specific language**

The Question

- What is the ideal language for this purpose?
- Which language features or constructs affect usability?
- What are the trade-offs between power and simplicity?
- Our method: iteratively design languages and experiment
- This presentation compares three quite different languages

The Requirements

- The input is XML document
- Output is list of errors
  - Errors must contain pointers to invalid elements
- Typical operations include
  - Presence and contents of dependent fields
    - If field X contais „foo”, field Y must contain „bar”
  - Comparing document with external data
    - Previous versions of the same document
    - Other documents, licences, etc. referenced by the document

The Requirements

- Readable by non-technical persons
- Expresses the intent, not the implementation
- Concise, presents the overview of the verification logic
- Free of technical details
- Runs on JVM

Language 1

- We reused language from another Customs project
- Simple, procedural, somewhat verbose
- Estonian keywords
- Looked like BASIC/COBOL derivative
- Compiled language with static type checking
- Extensible – functions and procedures written in Java

Language 1: Implementation

- Parsed with JavaCC
- Generate Java code
- Compiled class files are stored in database and loaded on demand
- Helper functions written in Java

Language 1: Example

```
KUI $SAD.Yldosa.Deklaratsioon.Olemus = "I" SIIS
    KUI $SAD.Yldosa.LahteRiik = "EE" SIIS
        veateade("Impordil ei tohi lähteriigiks olla Eesti.",
                "INVALID_VALUE", 15, $SAD.Yldosa.LahteRiik)
    IUK
MUIDU
    KUI $SAD.Yldosa.SihtRiik = "EE" SIIS
        veateade("Ekspordil ei tohi sihtriigiks olla Eesti.",
                "INVALID_VALUE", 17, $SAD.Yldosa.SihtRiik)
    IUK
IUK
```

Language 1: Another Example

```
UUS MUUTUJA pakendOlemas := EI
IGA $SAD.Kaup.Kirjeldus.Pakend JAOKS
    pakendOlemas := JAH
    KUI kuulubHulka(Pakend.Tyyp, "VQ, VG, VL, VY, VR, VO") SIIS
        KUI MITTE(onTaitmata(Pakend.Kogus)) SIIS
            veateade("Pakendi liigi "+Pakend.Tyyp+" korral peab pakendi kogus "+
                "olema täitmata.", "BOX_FILLED", 31, Pakend.Kogus)
        IUK
    MUIDU
        KUI MITTE(kuulubHulka(Pakend.Tyyp, "NE, NF, NG")) SIIS
            KUI onTaitmata(Pakend.Markeering) SIIS
                veateade("Pakendi liigi "+Pakend.Tyyp+" korral peab pakendi "+
                    "markeering olema täidetud.", "BOX_UNFILLED", 31, "")
            IUK
        IUK
        KUI onTaitmata(Pakend.Kogus) SIIS
            veateade("Pakendi liigi "+Pakend.Tyyp+" korral peab pakendi "+
                "kogus olema täidetud.", "BOX_UNFILLED", 31, "")
        IUK
    IUK
AGI
KUI $SAD.Yldosa.Deklaratsioon.Olemus = "E" SIIS
    KUI MITTE(pakendOlemas) SIIS
        veateade("Ekspordideklaratsioonil peavad vähemalt ühe pakendi andmed "
                olema sisestatud.", "PACKAGE_MISSING", 31, "")
    IUK
IUK
```

Language 1: Results

- Client analysts had trouble writing the rules
  - In fact, they had trouble writing the rules in Estonian also
- No means for abstraction
  - The only abstraction device is global variable
- Only primitive datatypes are supported
  - Cannot refer to lists or structured elements
- Result: lot of tedious copy-paste-modify
- Some systems had up to 12 000 lines of rules

Language 2

- Since customer was not interested in creating rules, we had more freedom in designing the language
- New top requirements were
  - User-defined procedures
  - Lists and structures can be passed to functions
  - List-processing functions, such as find, contains, ...
- Decision: write rules in **Scheme**
  - Ability to perform magic with macros

Language 2: Implementation

- We embedded our “DSL” in Scheme
  - ... using functions and macros
- We used **kawa** – quite feature rich Scheme implementation for JVM
  - Supports compiling and static type checking
  - Good Java integration
  - Nice syntax for property access

Language 2: Example

```
(xforeach data:goodsItem item GoodsItem
          (xforeach item:packages p Package
                    (cond
                      ((str-in-set? p:packageType '(VQ VG VL VY VR VO))
                       (prohibit p:numberOfPackages
                                 'tir.business.rule.prohibit.event-86
                                 "C060: Pakendi arv ei tohi täidetud olla.")
                       (prohibit p:numberOfPieces
                                 'tir.business.rule.prohibit.event-87
                                 "C060: Pakendi tükiarv ei tohi täidetud olla."))
                      ((str-in-set? p:packageType '(NE NF NG))
                       (prohibit p:numberOfPackages
                                 'tir.business.rule.prohibit.event-88
                                 "C060: Pakendi arv ei tohi täidetud olla.")
                       (expect p:numberOfPieces
                               'tir.business.rule.expect.event-89
                               "C060: Pakendi tükiarv peab olema täietud."))
                      (else (expect p:marksNumbersOfPackages
                                    'tir.business.rule.expect.event-90
                                    "C060: Pakendi markeering peab olemas olema.")
                            (expect p:numberOfPackages
                                    'tir.business.rule.expect.event-91
                                    "C060: Pakendite arv peab olemas olema.")
                            (prohibit p:numberOfPieces
                                      'tir.business.rule.prohibit.event-92
                                      "C060: Pakendite tükiarv ei tohi olemas olla."))))))
```

Language 2: Results

- Some analysts had previous knowledge of FP
  - FP course in university
- First-order procedures were quite handy
- Plenty of technical problems
  - Scheme-Java mismatch
  - Problems with static typing
- Resulting rules were often complex
  - Unnamed lambda expressions
  - Unnamed magic values
  - Hard to determine the original intent of the rules

Language 3

- Next language (Burula) was carefully designed, based on experience with previous languages
  - Specifically targeted for the most common types of checks
- Strong static type-checking
- Built-in support for common iteration patterns
- Enforce simple, uniform structure for rules
- Enforce explicit naming of stuff

Language 3: Implementation

- Custom parser and typechecker
- Compiles to JVM bytecode
- Can call helper functions written in Java

Language 3: Example

```
predicate bulk
    packages.kindOfPackages is ( 'VQ', 'VG', 'VL', 'VY', 'VR', 'VO' )

predicate unpacked
    packages.kindOfPackages is ( 'NE', 'NF', 'NG' )

group gi-packages
    packages cannot have numberOfPackages when bulk or unpacked
        error "C060: Pakendi arv ei tohi täidetud olla."
    packages cannot have numberOfPieces when bulk
        error "C060: Pakendi tükiarv ei tohi täidetud olla."
    packages must have numberOfPieces when unpacked
        error "C060: Pakendi tükiarv peab olema täidetud."
    packages must have marksNumbersOfPackages unless bulk or unpacked
        error "C060: Pakendi markeering peab olemas olema."
    packages must have numberOfPackages unless bulk or unpacked
        error "C060: Pakendite arv peab olemas olema."
    packages cannot have numberOfPieces unless bulk or unpacked
        error "C060: Pakendite tükiarv ei tohi olemas olla."
```

Language 3: Results

- No need to worry about technical details
  - Iteration, whether field is null etc.
- Rules look like human-language sentences
  - Especially, simple data dependency rules look very nice
- In general, people like it
  - However, one developer said that she found no differences between languages 1 and 3

Comparison

- Language 1
  - Attempt at „friendly” syntax
  - Lot of restrictions
  - Low level of abstraction
- Scheme
  - S-expression syntax
  - Very powerful, no restrictions
- Burula
  - Optimized for the most common case
  - Artificial restrictions to ensure good style
  - Tries to limit complexity of the rules

However...

- Actual usability information is very hard to get
- Right now we have similar rules written in three languages with some subjective opinions about their readability and ease of use
- Need a way to get some objective or quantifyable measures

Thanks!