Using Language-Oriented Programming – a Case Study

Margus Freudenthal
Cybernetica / University of Tartu

Our Situation

- We are working on Customs Engine
- Customs information system
- Processes customs documents
  - Import and export declarations
  - TIR carnets
  - Export reports
  - Manifests
  - Warehousing notices
  - etc.

Customs Engine

- Each document typically represents some kind of movement of goods
- Modular architecture: each module processes one type of document
- Modules communicate with each other and their EU counterparts
- Modules are based on common platform
  - Reusable components
  - Framework and reference architecture

Architectural Requirements

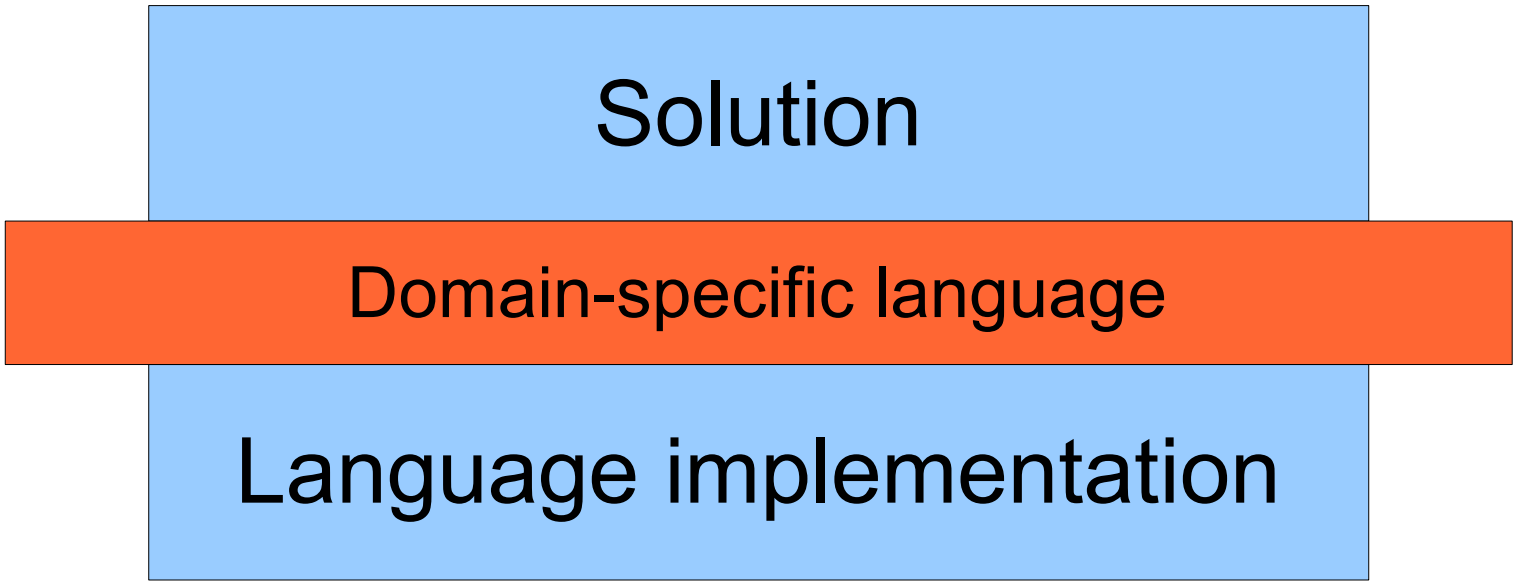
- Set of similar modules sharing common platform
  - Platform development costs divided among modules
  - Need for customizable components
- Complex business logic
  - Rules for verifying documents
  - Document state machines
  - Rules for processing and sending messages
  - Having clear overview of code is important

Architectural Requirements

- Changing business logic
  - Iterative development
  - Changing regulations
- Everything must run in JVM and J2EE

What to Do?

- We used language-oriented programming
- The general idea is to create a domain-specific language and write a program in that language



Language-Oriented Programming

- Separation of concerns
  - Technical decisions in language implementation
  - Functional decisions in solution
- High productivity
  - DSL has high level of abstraction and fits the problem domain
- Good maintainability
  - Solution is written in high-level language
  - Solution and language implementation can be evolved separately

Our Approach

- Platform components can be configured using DSLs
- Big, heavyweight DSLs created for important parts
- Templating used for smaller, less important languages

Heavyweight DSL: Burula

- Short for **bu**siness **ru**le **la**nguage
- Used to specify document verification rules

```
predicate is-unpacked-goods
    kindOfPackages is ('NE', 'NF', 'NG')

packages must have numberOfPieces
    when is-unpacked-goods
    error "When goods are unpacked, number
        of pieces must be present"
```

Burula

```
predicate is-sea-transport
    transportModeAtBorder is ('1', '8')

const ship-number '[0-9]{7,8}'

idOfTransport idOfTransport is like ship-number
    when is-sea-transport
    error "Identity of transport vehicle
        must be IMO ship number"
```

Burula: Features

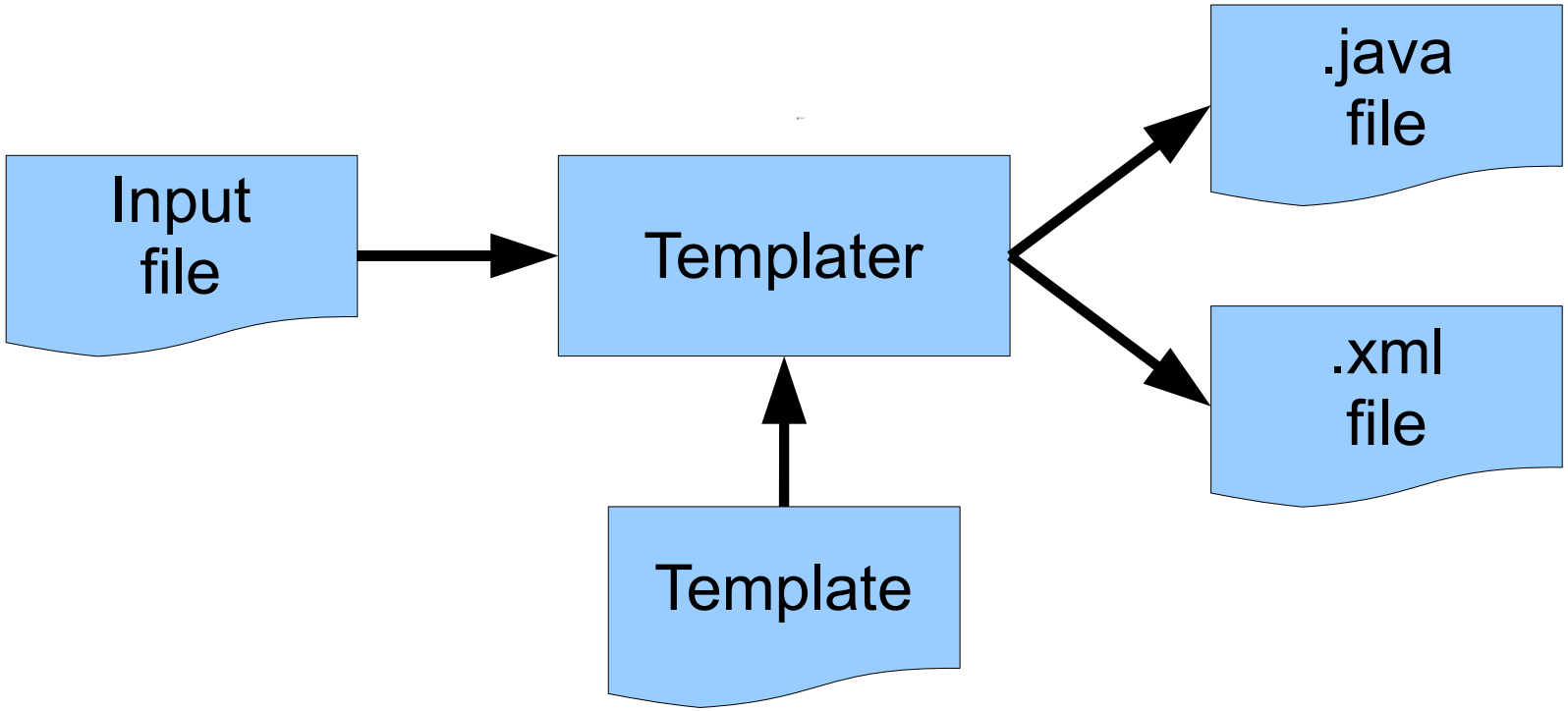
- (hopefully) Intuitive syntax
- Simple use of document fields
- Implicit iteration
- Other implicit „magic”
  - For example, recording location of the error

Burula: Implementation

- Compiled to Java bytecode
- Compiled programs are stored in database and loaded when needed
- Older versions of programs are retained for use with old documents
- Burula programs can call Java methods

Lightweight Languages

- We use templating system



Templater

- Input files use S-expressions as syntax
- Templater can generate Java or XML files

```
(message jms.complex-exit-notificaton-receiver
        ee.cyber.complex.bean.ExitNotificationReceiverBean)

(local
        ee.cyber.complex.service.RemoteDeclarationService
        ee.cyber.complex.bean.RemoteDeclarationServiceBO
        (transaction RequiresNew)
        (anonymous-user))

(remote ee.cyber.coal.client.SadIntegrationService
    ee.cyber.complex.bean.SadIntegrationServiceBO)
```

Evaluation

- The modules are very flexible and easy to change
  - Creating new modules is more about describing the functionality in languages provided by the platform
- Analysts' work is different
  - Instead of documents, they write programs
  - Very short round-trip – immediate feedback to analyst
  - Using formal language for requirements exposes problems early

Evaluation

- Programmers have less routine tasks
  - e.g. less UI tweaking
  - Need to fill gaps left by the DSLs
- Because analysts write directly in formal language, there tends to be less documentation
  - „Why?” is not documented
  - Formal rules not readable by users

Conclusion

- Our overall experience with language-oriented programming is positive
- I would recommend it when
  - There is lot of complex business logic
  - Project is big enough to justify building languages
  - There are good people and tools

We shape our tools and
thereafter our tools shape us.

-- Marshall McLuhan