DSM 2013

On the Way of Bottom-Up Designing Textual Domain-Specific Modelling Languages

Bastian Roth, Matthias Jahn and Stefan Jablonski

Bastian Roth
Telefon: +49 921 - 55 7625
Fax: +49 921 - 55 7622
E-Mail: bastian.roth@uni-bayreuth.de
Motivation

- Designing a Domain-Specific Modelling Language (DSML) is a complex and time-consuming task, especially when restricting to the wide-spread formal method.

- It requires to “pre-think” the language.

• An alternative approach has arisen which is often called “bottom-up (meta) modelling” or “by-demonstration approach.”

• First models are sketched diagrams which are used to (semi)-automatically derive a DSML.

- Current solutions only address graphical DSMLs.

• Our focus.
  Tool support for building textual DSMLs the bottom-up way.

1: understand real world instances
2: create models bottom-up
3: generate language
4: create models top-down
Information about source- and target language and speaker

Ideally, this is achieved the same way as reconstructing a model

There is no jump involved, levels are reconstructed from bottom to top

Formal description of a model instance

Model reconstruction

Information about modeling patterns

Language reconstruction

Formal model

Modeling language

Informal, textual description

Information Extraction Natural Language Processing

Larger Scope
**Example**

- Simplified variant of Fowler’s state machine
- Exemplary state machine

```
events  START, WORK, STOP

state  Idle
   START  =>  Active

state  Active
   WORK  =>  Active
   STOP  =>  Idle
```

- How to achieve this syntax highlighting?
  - Split text into tokens
  - Tag those tokens

- During sketching textual models it is expedient that the user directly sees the resulting hierarchical structure (concrete syntax tree, CST)

- Suggestion for an adequate structuring

  ![Concrete Syntax Tree (CST)](image)

- The text’s hierarchical structure acts as basis for the later derived abstract syntax. How to identify such a structure?
The identification procedure is performed every time a token is modified. This modification occurs by simply entering free text.

Many DSMLs and also general purpose programming languages (GPLs) consist of various recurring **token types**
- **Keyword**
- **Identifier**
- **Reference**
- **String literal**
- **Integer literal**
- **Float literal**
- **Comments**
- **Delimiter**
  - Further subdivision is required
  - Special case: delimiters may directly affect the CST’s structure

Explicitly set a token’s type using **shortcuts**

Identification by means of **regular expressions**
- Those regular expressions need to be determined beforehand
- It is expedient to provide defaults
**Structure identification**

- Containers as structuring nodes within the CST, Tokens as leaf nodes

- According to many DSMLs and GPLs, differentiation between following **container types**
  - *Statement*
  - *Block*
  - *Expression*

- **Example** with all container types

```
public int compute ( int a, int b ){
    return a * 100 / b
}
```

- Container creation and rebuilding is triggered during the identification of certain tokens (primarily delimiters)
Categorization of delimiters according to their impact on the CST’s structure

- **Token separator**
  - Default type of each delimiter
  - Separates tokens from each other (within a statement)
  - No impact on the CST

- **Statement separator**
  - Separates statements from each other
  - Inserted into the parent block
  - Successive tokens are wrapped by a new statement

- **Opening brace**
  - Adds a successive block and creates a corresponding closing brace
  - Familiar feature of modern IDEs

- **Binary operator**
  - Wraps the previous token or expression and the inserted operator token with a new expression

Extensions are possible and even recommended!
modelXworkbench – a Web-based modelling environment

![Web-based modelling environment](image-url)
Future challenges

• Provide further delimiter types which occur in other languages (e.g. unary operators, end-of-line symbols)

Configurability of the editor’s behaviour to fit different users’ needs

• Improving the editor’s user experience (e.g. content assist for references and recurrent statements)

• Inducing the abstract syntax (meta model) from such textual example models