TCS, Textual Concrete Syntax

A DSL for the Specification of Textual Concrete Syntaxes in Model Engineering

Anas Abouzahra & Frédéric Jouault
Ivan Kurtev

6, rue de Cornouaille
BP 91 941
44319 NANTES Cedex 3, France
Tel.: +33 (0) 2.28.23.60.60
Fax: +33 (0) 2.28.23.60.57
Email: info@sodius.com

ATLAS group, INRIA & University of Nantes, France
http://www.sciences.univ-nantes.fr/lina/atl/
Prerequisites

To be able to understand this lecture, you should be familiar with the following concepts, languages, and standards:

- Basic programming concepts
- The role of model transformations in MDE
- Model Driven Engineering (MDE)
- MOF
- UML
Outline

• Introduction
  • AMMA Context
  • TCS: Bridging Metamodels and Grammars
• A case study: BibTeX
  • Basic constructs: templates, properties, and literals
  • More constructs: conditionals, symbol table handling (KM3 example)
  • Advanced constructs: operators, function templates (KM3 and ATL examples)
  • Other features (from ATL.tcs)
• Textual Generic Editor
• Conclusion
Introduction

• Two important views of Model Driven Engineering
  • Using GPLs (General Purpose Languages):

    Consisting in starting with a well known standard
    Universal language like C++, C#, and Java and modeling
    language like UML

  • Using DSLs (Domain Specific Languages):

    Using small, well-focused languages to model each
    system and deal with the coordination between these

    With DSLs domain concepts are directly represented by syntactic constructs
    more concise and precise specifications
    Sentences expressed in a DSL usually make use of higher-level constructs
    than equivalent sentences in a GPL
Introduction, continued

• There are, however, issues limiting the usage of DSLs:
  A major one is the reduced availability of tools for DSLs compared to GPLs.

So, How to implement DSLs?

• Several Ways:
  Using XML engineering  Using GrammarWare
  Using MDE (Model Driven Engineering)
Introduction, continued

• We study the problem in the MDE approach:
  • There is a growing interest in using MDE for this purpose
  • MDE solution for an MDE problem

• In this approach, the different aspects of a DSL are captured by different models:

- The domain concepts are represented in a metamodel that we call a Domain Definition MetaModel.
- The concrete syntax facet of a DSL used to generate tools for model-to-text and text-to-model transformations.
- A possible solution for DSL-to-DSL and even DSL-to-GPL translations.
The AMMA DSLs Building framework architecture
AMMA Context, continued

• The AMMA DSLs Building framework architecture
TCS: Bridging Metamodels and Grammars

• We present here the AMMA solution for implementing concrete syntaxes of DSLs:

  TCS: Textual Concrete Syntax

• TCS contributes a significant capability to AMMA
  • Bridging the modeling and syntax worlds.
  • Implementing the concrete syntax of AMMA core languages like KM3 (Kernel MetaMetaModel), ATL (ATLAS Transformation Language), and TCS itself.
  • Specifying the concrete syntax of other DSLs.
TCS: Bridging Metamodels and Grammars, continued

• TCS works by providing means to associate syntactic elements to metamodel ones

• Both model-to-text and text-to-model translations can be performed using a single specification

A grammar can be generated from both the metamodel and the TCS model to perform text-to-model translation

Grammar annotations that build the model while parsing can be automatically generated.

Model-to-text translation can be performed with the same information

A generic interpreter has been defined to traverse the model following the syntactical path specified in TCS and keywords and symbols are written alongside model information.
Case Study: BibTeX

- We want to build the BibTeX DSL.
- Assume we need a simplified BibTeX metamodel which only deals with the mandatory fields of each BibTeX entries (for instance, author, year, title and journal for an article entry)
BibTeX: Requirement

- We express this metamodel in KM3
- Then we have to define its concrete syntax in TCS
- Assume that we want to be in compliance with the BibTeX syntax:
  - A BibTeX file must look like:

```bibtex
@article { dayan96exploration,
    author = « Peter Dayan »,
    title = « Dual Control »,
    journal = « Machine Learning »,
    volume = « 25 »,
    number = « 1 »,
    pages = « 5-22 »,
    year = « 1996 »
}
```

```bibtex
@misc{ jong-hierarchical,
    author = "Edwin De Jong",
    title = "Hierarchical Genetic Algorithms",
    url="citeseer.ist.psu.edu/705095.html"
}
```
BibTeX: Defining the TCS file

• Basic constructs: templates, properties, and literals

Metamodel excerpt:

```java
class BibtexFile {
    reference entries[*] container : BibtexEntry;
}

abstract class BibtexEntry extends LocatedElement {
    attribute key : String;
    reference bibtexAttrs[*] container :
        BibtexAttribute;
}

class Article extends BibtexEntry {
    ...
}

abstract class BibtexAttribute {
    attribute value : String;
}

class Authors extends BibtexAttribute {
}
class Title extends BibtexAttribute {
    ...
}
```

Corresponding TCS excerpt:

```tcs
template BibtexFile main : [ entries ]
;
template BibtexEntry abstract;
template Article : "@" "article" "{"
["key","bibtexAttrs{separator = ","}"
     "}"
;
...

template Authors : "author" "=" value 
;
template Title : "title" "=" value 
;
...
```
Basic constructs: remarks

• There is little redundancy between the metamodel and the TCS:
  • The links between metamodel and TCS elements are done by name (e.g. Class & Template, properties, etc.),
  • The (primitive) type of the value attribute is known from the metamodel whereas its position in the text is known from the TCS,
  • The (complex) type and multiplicity of the bibtexAttrs reference are known from the metamodel whereas its position in the text is known from the TCS.

• Structural elements are defined in KM3.
• Syntax elements are defined in TCS:
  • Keywords as alpha-numeric strings between double quotes,
  • Symbols as non-alpha-numeric strings between double quotes.
More constructs: symbol table handling

• TCS handles the symbol table:
  • Elements are marked as being contexts using the “context” keyword,
  • Elements are added in the current context using the “addToContext” keyword,
  • Elements are referred to by the value of one of their properties using the “refersTo” keyword.

• KM3 case study:
More constructs: symbol table handling (KM3 example)

```
syntax KM3 {
    template ModelElement abstract;
    template Package main context:
        "package" name "{" contents "}";
    template Classifier abstract addToContext;
    template DataType context:
        "datatype" name ";;"
    template Class context:
        (isAbstract ? "abstract") "class" name
        (isDefined(supertypes))?
        "extends" supertypes{refersTo = name, separator = ",", autoCreate = never}
        "{" structuralFeatures }"
... ... ...
```
More constructs: conditionals, Separators

- **Conditional elements:**
  - Using the value of Boolean properties (e.g. `isAbstract`),
  - Testing the value of a property (will be shown later),
  - Testing whether a property is set for multiplicities 0-n, $1 \leq n$ (e.g. `supertypes`).

- **Miscellaneous:**
  - Separators can be specified for multi-valued properties (e.g. `supertypes`).
More constructs: conditionals, Separators (KM3 example)

Syntax KM3 {
    template ModelElement abstract;

    template Package main context
        "package" name "{" contents "}" ;

    template Classifier abstract addToContext;

    template DataType
        "datatype" name ";;"

    template Class context
        (isAbstract ? "abstract") "class" name
        (isDefined(supertypes))?
            "extends" supertypes{refersTo = name, separator = ",", autoCreate = never}
        
        "{" structuralFeatures "}
        ...
        ...
}
Advanced constructs: operators, function templates

- TCS can also deal with operators:
  - Operators and their priorities are first defined,
  - Operator Templates are used to specify their use,
  - The notation can be chosen:
    - Infix: \( 1 + x \times 4 \)
    - RPN: \( (+ 1 (* x 4)) \)
- More complex symbol table handling can be performed:
  - Context importation (e.g. to deal with class inheritance),
  - Search for a target element in another context than the current one using the “lookIn” keyword (see the Reference template later in this presentation).
- Functions can be defined to factorize code
Advanced constructs: operators, function templates (KM3 example)

```plaintext
template StructuralFeature abstract;

template Reference addToContext
  :  "reference" name $multiplicity (isContainer ? "container") ":" 
    type(refersTo = name, autoCreate = never)
    (isDefined(opposite) ? 
      "oppositeOf"
      opposite(refersTo = name, lookIn = type, autoCreate = never) 
    ) 
  
  ";"
;

template Attribute -- Function call
  :  "attribute" name $multiplicity ":" 
    type(refersTo = name, autoCreate = never) 
  
  
  ";"
...
```

-- Uses lookIn

---

DDMM

-- Uses lookIn

```plaintext
TCS
```

---

TCS, Textual Concrete Syntax
Advanced constructs: operators, function templates (KM3 example), continued

```plaintext
function multiplicity(StructuralFeature) -- function definition
  :   (lower = 1 and upper = 1 ?
      -- nothing
    :
      (lower = 0 and upper = -1 ?
        "[" "*" "]"
      :
        (upper = -1 ?
          "[" lower "-" "*" "]"
        :
          "[" lower "-" upper "]"
        )
      )
  )
  (isOrdered ? "ordered")
  ;
... ... ...
```

-- test of integer properties

---

**Diagram**

- ModelElement
  - name: String
  - contents
- Class
  - +type
  - +isAbstract: Boolean
  - +superclasses
- Reference
  - +isContainer: Boolean
  - +opposite
- Attribute
  - +isUnique: Boolean
  - +isAbstract: Boolean
- StructuralFeature
  - +lower: Integer
  - +upper: Integer
  - +isOrdered: Boolean
- Classifier
  - +type
  - +structuralFeatures
- Package
  - +package
  - +contents
- DataType
  - +isAbstract: Boolean
  - +superclasses
- DDMM
  - -- function definition
  - -- test of integer properties
Advanced constructs: operators, symbols (from ATL.tcs)

symbols {
  lsquare = "["; rsquare = "]" : rightSpace;
 ...
-- operator symbols
  point = "." : leftNone;
  rarrow = ">" : leftNone;
  minus = "-" : leftSpace, rightSpace;
 ...
} -- end of symbols
operators {
  priority 0 {
    opPoint = point, 2;
    opRarrow = rarrow, 2;
  }
  priority 1 {
    opNot = "not", 1;
    opMinus1 = minus, 1;
  }
-- no corresponding symbol to the « not »
-- operator, so symbol is the keyword
-- defined by the quoted string
 ...
} -- end of operators
... ...
Other features

• **Model-to-text actually needs more than the syntax:**
  • Indentation blocks can be defined,
  • Specific separators can be used (new line, blank, tab, etc.).

• **Text-to-model traceability is provided:**
  • The location attribute of each generated element is set with:
    • Line and column numbers of beginning,
    • Line and column numbers of ending,
  • Comments may be kept (and serialized).
Other features: Indentation example (from ATL.tcs)

```
... ...

template Module context
  :   "module" name ";" <newline>
    "create" outModels{separator = ","} (isRefining ? "refining" : "from")
    inModels{separator = ","} ";"

  [ libraries elements ] {nbNL = 2, indentIncr = 0}

... ...

template OclFeatureDefinition
  :   (isDefined(context_) ? context_) "def" <no_space> ":" feature

... ...

template Operation context
  :   name "(" parameters{separator = ","} ")" ":" returnType ";"
    [ body ] {endNL = false}

... ...
```

- Put a new line
- Put in a bloc
- Put in a bloc
- 2 new lines and No indentation incrementation (by default = 1)
- To oblige no space between «def» and «:»
- No new line at the end of the bloc
Textual Generic Editor

- Eclipse Editor Plugin:

```java
class Class extends Classifier {
    attribute isAbstract : Boolean;
    reference supertypes[*] : Class;
    reference structuralFeatures[*] ordered container : StructuralFeature oppositeOf owner;
    reference operations[*] ordered container : Operation oppositeOf owner;
}

class TypedElement extends ModelElement {
    attribute lower : Integer;
    attribute upper : Integer;
    attribute isOrdered : Boolean;
    attribute isUnique : Boolean;
    reference type : Classifier;
}

class StructuralFeature extends TypedElement {
    reference owner : Class oppositeOf structuralFeatures;
    reference subsetOf[*] : StructuralFeature
    reference derivedFrom[*] : StructuralFeature oppositeOf owner;
```
Textual Generic Editor, continued

- **Language definition** is an EMF model specifying:
  - Comment blocks,
  - Keywords list,
  - Highlighting format (font and color),
  - Can be generated by *TCS2Editor.atl*.

- **Outline definition** is an EMF model specifying:
  - Nodes to display,
  - Label format,
  - Can be generated by *KM32Outline.atl*.

- **Uses TCS-generated parser**:
  - To populate the outline,
  - To provide text hovers and hyperlinks.
Conclusion

• Modeling using a DSL can be done textually.
• Specifying a bidirectional mapping between a metamodel and a textual syntax is possible and rather straightforward using TCS.
• TCS has been used for several DSLs: KM3, TCS, ATL, ACG, AM3, Editor, SQLDDL, etc.
• HUTN has been partially implemented with a KM32TCS.atl transformation.
• Eclipse-based textual editor (TGE) for free.
Conclusion, continued

- Although only the textual syntax is presently used to create TCS models, it could be defined in some other way (for instance using weaving).

- Some current limitations:
  - Code formatting is specified with the syntax,
  - Text-to-model traceability is added in the target model,
  - Grammar ambiguities are not traced back to TCS constructs,
  - TCS is meant to provide textual syntaxes for DSLs, when one can make compromises on the textual syntax,
  - It may not be usable to parse Java or C++ code, although it may be usable to serialize such code (there are less constraints in this direction),
  - More complex cases could probably be dealt with using weaving between grammars and metamodels.
End of the presentation

- Thanks
  - Questions?
  - Comments?