Parsing and Code Generation Techniques to Deal with Uncertainty

- Cedric Lemaire lemaire_cedric@yahoo.fr
DSM at BNP Paribas

- **Equity Derivatives**
  - Trading on electronic markets
  - Exotic financial products
  - Core software: the pricer, which holds a part of the model

- **Capture of the Business Knowledge**
  - Extended UML modeler in the past
  - ... replaced by textual DSL today

- **Intensive use of parsing and code generation**
  - Tool: [Code Worker]
DSM at BNPParibas (2)

- Model: contributors are composed of developers exclusively, in ease with a DSL
- Meta-model: a BNF parse script,
- Parse tree: more precisely, a graph and not necessarily a hierarchical structure
- Code generation: a set of template-based scripts
- Mixed source code: to better dispatch commonalities between source code and code generation
Instability in Requirements

Model

Meta model

Parse tree

Code generator

Mixed source code

Output

```
interface BankAccount {  
    // inherits from a parent int
    if(!this.inheritance.empty())
        // this.inheritance$from
    {
        // attributes:
        if(!this.attributes.empty()) {
            foreach i in this.attribute
                if i.documentation{
                    @i.documentation;
          }
        }
    }  
    @this.name  
}
```

```
module bank {
    interface Account {
        // readonly_long_balance
        void makeDeposit();
        void makeWithdraw();
    }
}
```

```
package

// An instance of this class
// It updates the list of eve
// to avoid navigating an obj
class NavigatorData {
    // cache keeping all
    public java.util.Set<
        // the custom visitor
        public Visitor visitor;
        
        public NavigatorData(
            visitor = v;
        )
    }
```
Instability in Requirements
changing a model

Model

Model

Changes are propagated automatically up to the source code

Meta model

Parse tree

Code generator

Output

Mixed source code

// Parsing of a slot: the name and
slot(theSlot : node) ::= 
"<slot" #continue 'name' '=' #readCString:theSlot
-> '>

[property]? value(theSlot.value)
"</slot>"

// Don't care about the property property ::= "<property" -> '

@ interface @this.name

// inherits from a parent int
if !this.inheritance.empty()
  @ this.inheritance$from
else

// attributes:
if !this.attributes.empty() {
  foreach i in this.statement
    if

// documentation.

module bank {
  interface Account {
    readonly long balance
    void makeLodgement(...)
    void makeWithdrawal(...)
  }

  public java.util.Set
    // the custom visitor
    public Visitor visitor

  public NavigatorData(...)
    visitor = v;
}
Instability in Requirements
Changing the metamodel

Model

Multiple DSL for a separation of concerns

Parse tree

Meta model

Parameterized fonctions to enhance the synchronization with the parse tree

Code generator

Instability in Requirements
Changing the metamodel

Impact demanding reworking: must facilitate the propagation of changes

Multiple DSL for a separation of concerns

Parameterized rules to enhance the extension of the grammar and its maintenance

Code works

Output

Mixed source code

Mixed source code

// Parsing of a slot: the name and the slot
slot theSlot : node) ::=<slot>
  #continue
  'name' ':=' #readCString theSlot

  [property]?
  value (theSlot.value)

  ''</slot>''

  // Don't care about the property property ::= '<property' -> '>',

// interface @this.name@ { // inherits from a parent interface if !this.inheritance.empty() { #this.inheritance$from

// attributes:
if !this.attributes.empty() {
  foreach i in this.attribute' 
    if i.documentation' 
      #this.documentation

module bank {
  interface Account { readonly long balance
    void makeDeposit()
    void makeWithdrawal()
    } } //endif

  class NavigatorData {
    // cache keeping all
    public java.util.Set<
      // the custom visitor
      public Visitor visitor

      public NavigatorData(
        visitor = v;
    } }
Evolving Source Code that Catches a part of the Model

Resistant to changing their way of working

// Parsing of a slot: the name and slot(theSlot : node) ::= "<slot" #continue "name" '=' '#readCString:theS' ->''
    [property]? value(theSlot.value) "</slot>"
;
// Don't care about the property property ::= "<property" ->'"';

strategy BusinessDayNight {
    start time() > (17*60 - 30) => deactivate;
    vehicles_hour(place_open) => duration(scri)
    vehicles_hour("rue de " => activate(Riv)
    vehicles_hour(auber->") => deactivate:
    time() > (22*60 + 30) => deactivate;

Model

Parse tree

Source code

Code generator

Evolving Source Code that Catches a part of the Model
Flexibility in parsing and code generation

- Building DSLs tolerant to rearranging
  - Parameterized rules,
  - Reusability by overloading of production rules,
- Building code generators tolerant to changes
  - Parameterized functions,
  - Code Expansion
  - Preserved areas
- Maintenance of multiple translations
  - Program transformation
  - Source-to-Source translation
Parameterized Rules and Overloading

- `instruction<"if"> ::= ... /* BNF production rule*/`
- `instruction<"while"> ::= ...`
- Add of a new instruction: non intrusive, just by adding a new production rule:
- `instruction<"for"> ::= ...`
- Overloading of a production rule: change the behavior of an existing one
- `#overload instruction<"if"> ::= ...`
Mixing Code Generation and Manufactured Code

- **Code Expansion**
  - Hand-typed code and some markups
  - Generated code is inserted at the markup place

- **Preserved areas**
  - Code generation erases the precedent content
  - Hand-typed code is preserved at some locations

- **Program transformation**
  - A translation script scans the file
  - It replaces some parts according to rules
Conclusion

- DSM accelerates the implementation of software
- The effort of raising the abstraction level moves complexity in transforming models to code
- This process must tolerate reworking at any time
- Some parsing + code generation features have revealed themselves being useful in practice:
  - Improve reaction against changing requirements in the metamodel and in the source code commonalities
  - Help in spreading these changes along parse tasks and code generation