Parsing and Code Generation Techniques to Deal with Uncertainty

- Cedric Lemaire `lemaire_cedric@yahoo.fr`

DSM at BNPParibas

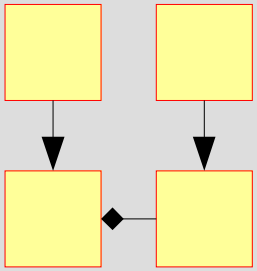
- Equity Derivatives
  - Trading on electronic markets
  - Exotic financial products
  - Core software: the pricer, which holds a part of the model
- Capture of the Business Knowledge
  - Extended UML modeler in the past
  - ... replaced by textual DSL today
- Intensive use of parsing and code generation
  - Tool: CodeWorker

DSM at BNPParibas (2)

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

Instability in Requirements

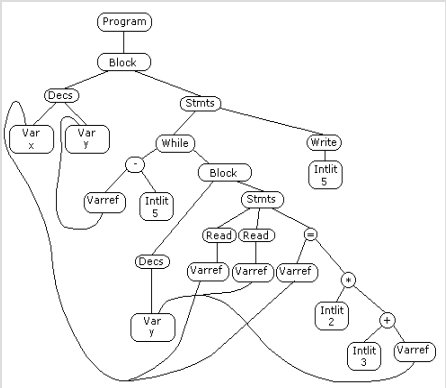
Model



Meta model

```
// Parsing of a slot: the name ar
slot(theSlot : node) ::=
    "<slot" #continue
    "name" '=' #readCString:theSl
    ->'>'
    [property]?
    value(theSlot.value)
    "</slot>"
    ;

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

Parse tree



Code generator

```
@    interface @this.name@ @
// inherits from a parent int
if !this.inheritance.empty()
    @: @this.inheritance#fron
}
@{
@
// attributes:
if !this.attributes.empty() {
    foreach i in this.attribu
        if i.documentation {
            @        /**
            @i.documentation.
```

Mixed source code

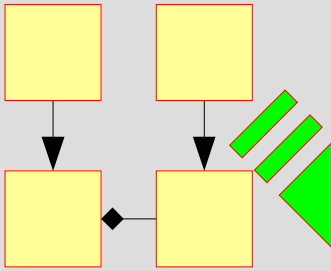
```
//##markup##"package"

// 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 visito

        public NavigatorData(
                visitor = v;
        }
```

Output

```
#ifndef _bank_Account_idl
#define _bank_Account_idl


module bank {
    interface Account {
        readonly long bala

        void makeLodgement

        void makeWithdrawa
    }
}

#endif
```

Instability in Requirements
changing a model

Model

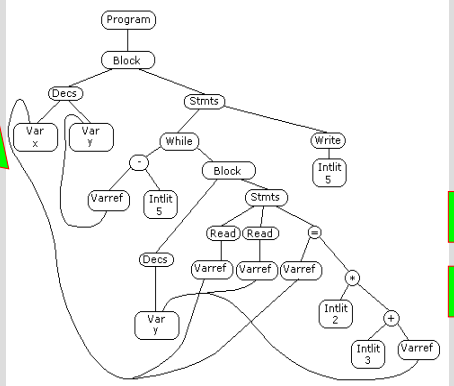


Changes are propagated automatically up to the source code

Code generator

```
@    interface @this.name@ @
// inherits from a parent int
if !this.inheritance.empty()
    @: @this.inheritance#fron
}
@{
@
// attributes:
if !this.attributes.empty() {
    foreach i in this.att
        if i
                .documentation.
```

Parse tree



Output

```
#ifndef _bank_Account_idl
#define _bank_Account_idl


module bank {
    interface Account {
        readonly long bala

        void makeLodgement

        void makeWithdrawa
    }
}

#endif
```

Meta model

```
// Parsing of a slot: the name ar
slot(theSlot : node) ::=
    "<slot" #continue
    "name" '=' #readCString:theSl
    ->'>'
    [property]?
    value(theSlot.value)
    "</slot>"
    ;

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

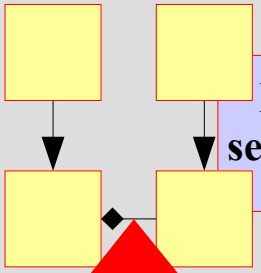
Mixed source code

```
//##markup##"package"

//          e of this class
//  l           the list of eve
//  to          rating ar
class Nav
        // cache keeping
        public java.util.Set<
        // the custom visitor
        public Visitor visito

        public NavigatorData(
                visitor = v;
        }
```

CodeWorker

Instability in Requirements
Changing the metamodel

Model



Impacts demanding reworking: must facilitate the propagation of changes

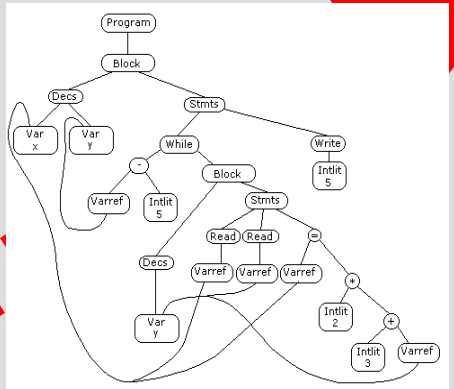
Multiple DSL for a separation of concerns

Code generator

```
@    interface @this.name@ @
// inherits from a parent int
if !this.inheritance.empty()
    @: @this.inheritance#from
}
@{
@
// attributes:
if !this.attributes.empty() {
    foreach i in this.attribu
        if i.documentation {
            @        /**
            @i.documentation.
```

Parameterized fonctions to enhance the synchronization with the parse tree

Parse tree



Output

```
#ifndef _bank_Account_idl
#define _bank_Account_idl


module bank {
    interface Account {
        readonly long bala

        void makeLodgement

        void makeWithdrawa
    }
}

#endif
```

Meta model

```
// Parsing of a slot: the name ar
slot(theSlot : node) ::=
    "<slot" #continue
    "name" '=' #readCString:theSl
    ->'>'
    [property]?
    value(theSlot.value)
    "</slot>"
    ;

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

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

Code Worke

Mixed source code

```
//##markup##"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 visito

        public NavigatorData(
                visitor = v;
        }
```

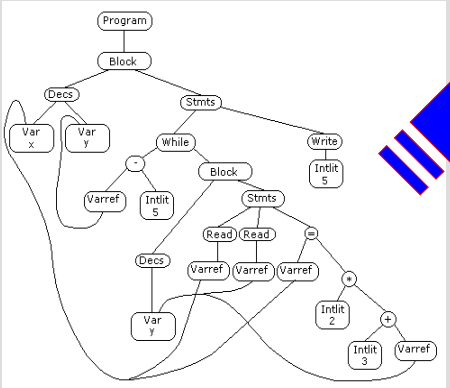
Evolving Source Code that Catches a part of the Model

Source code

```
#ifndef _bank_Account_idl
#define _bank_Account_idl


module bank {
    interface Account {
        readonly long bala

        void makeLodgement

        void makeWithdrawa
    }
}

#endif
```

Resistant to changing their way of working

Parse tree



Code generator

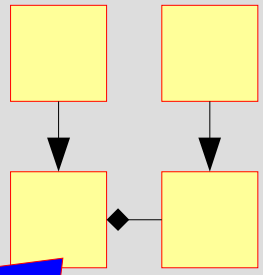
```
@    interface @this.name@ @
// inherits from a parent int
if !this.inheritance.empty()
    @: @this.inheritance#fron
}
@{
@
// attributes:
if !this.attributes.empty() {
    foreach i in this.attribu
        if i.documentation {
            @        /**
            @i.documentation.
```

Mining process

```
// Parsing of a slot: the name ar
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 -

        vehicles_hour(place_ope
                => duration(scrib
        vehicles_hour("rue de
                => activate(Riv
        vehicles_hour(auber->"
                => desactivate;

        time() > (22*60 + 30)*
                => desactivate;
}
```

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