CVL to Clafer transformation

Tom Wijsman

22 Januari 2015

University of Antwerp

Overview

1. CVL to Clafer transformation steps
   1. Identify common features for both languages
   2. Create abstract and concrete visual syntax for CVL in AToMPM
   3. Export metamodel and models from AToMPM to metaDepth
   4. Transform (ETL) from CVL to Clafer in metaDepth
   5. Generate (EGL) from Clafer in metaDepth to concise Clafer
2. Example
3. Verification
4. Conclusion
5. Future work

Chapter 1

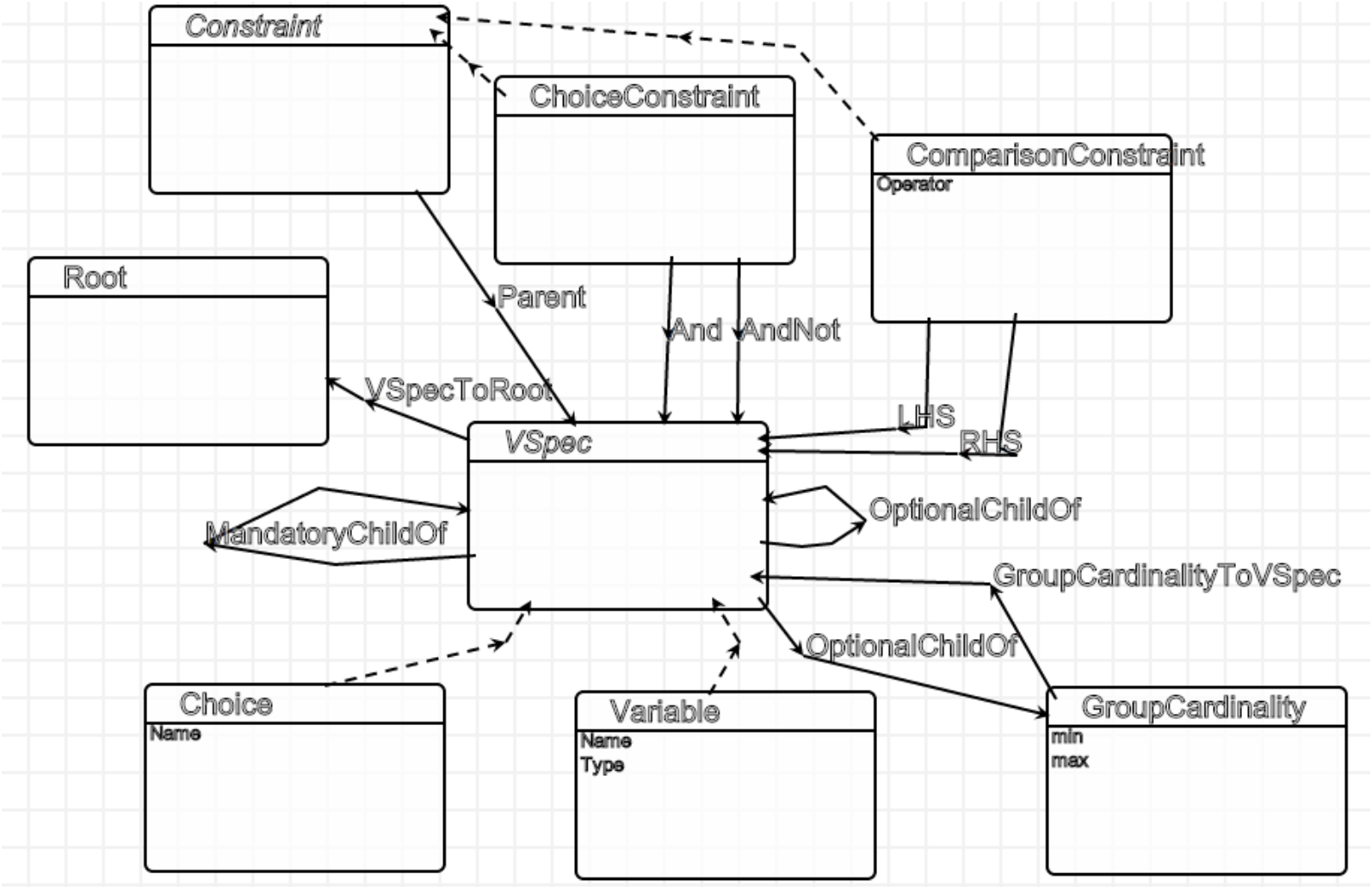
CVL to Clafer transformation steps

Step 1: Identify common features for both

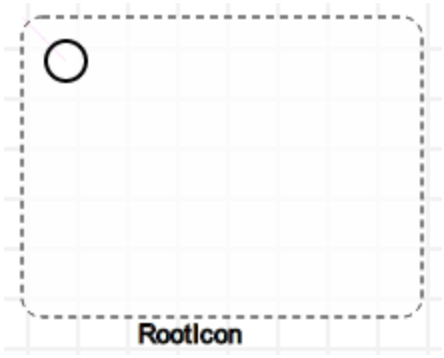
CVL	Clafer
Variability Specification	Class/Feature Type
Variability Choice	Feature
Variability Variable	Type Definition
Optional VSpec	“?” behind the Class/Feature Type
Type of Variable	“: Type” behind the Class/Feature Type
Group Cardinality	“xor” (1..1), “or” (1..*) or “x..y” (xy) before the Class/Feature Type
Children of a VSpec	Indented one level deeper than the parent

The constraints are custom made and not part of these languages.

Step 2: Create **abstract** and concrete visual syntax

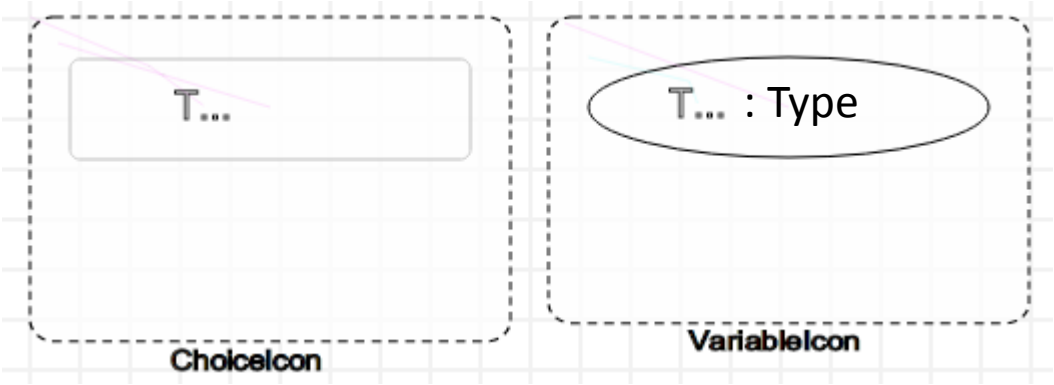


Step 2: Create abstract and **concrete visual** syntax



Step 2: Create abstract and **concrete visual** syntax

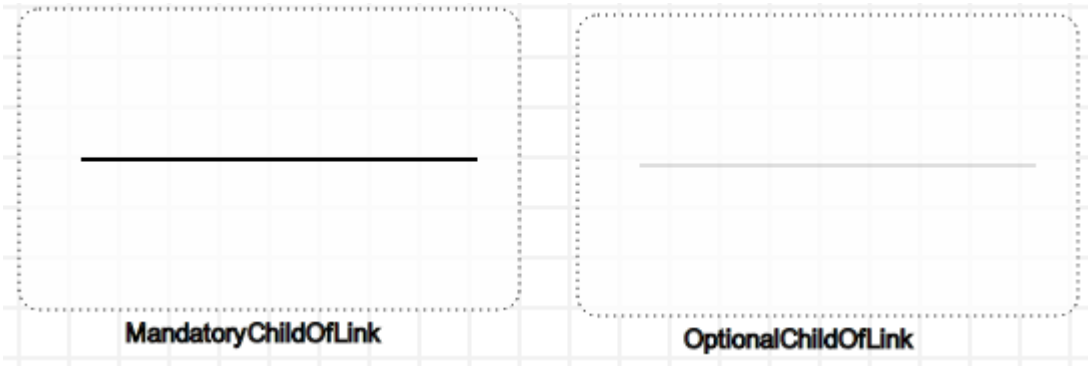
CVL:



Clafer: Example Example : Type

Step 2: Create abstract and **concrete visual** syntax

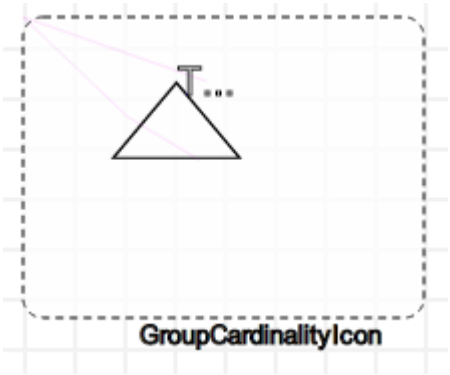
CVL:



Clafer: Example Example?

Step 2: Create abstract and **concrete visual** syntax

CVL:

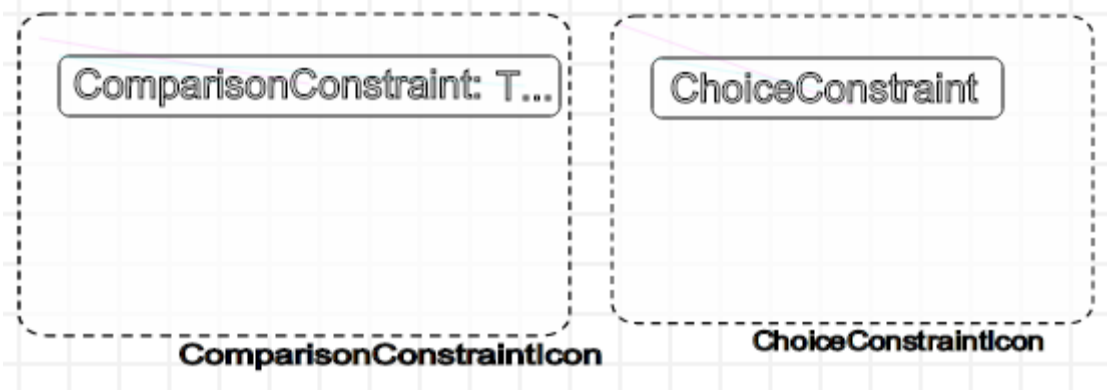


Clafer:

```
xor Example
    …

or Example
    …

x..y Example
    …
```

Step 2: Create abstract and **concrete visual** syntax

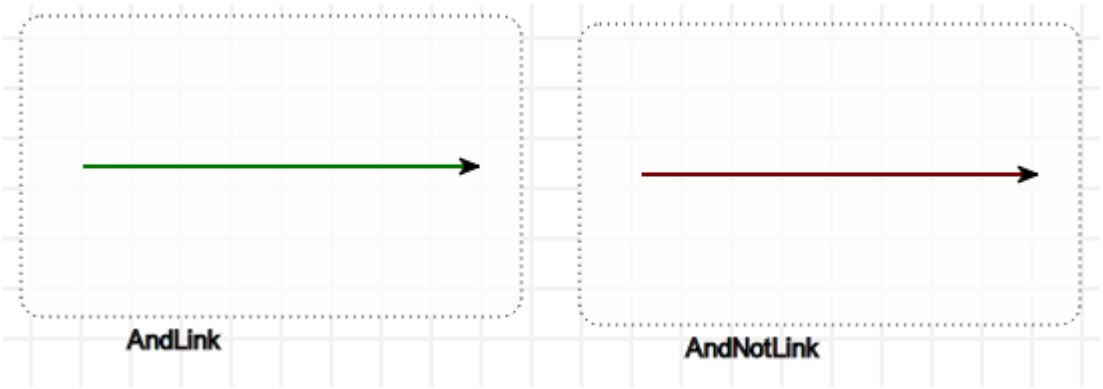
Custom CVL:



Custom Clafer: [x OPERATION y] [x && y && !z]

Step 2: Create abstract and **concrete visual** syntax

Custom CVL:



Custom Clafer: x !x

Custom CVL:

Custom Clafer: [lhs OPERATION rhs]

Step 3: Export metamodel to metaDepth

```
Model CVLMM {
        Node Link {
                src : Node;
                dst : Node;
        }
        Node Root {}
        Node VSpec {}
        Node Variable : VSpec {
                Name:String = "Name";
                Type:String = "Type";
        }
        Node Choice : VSpec {
                Name:String = "Name";
        }
        Node GroupCardinality {
                min:String = "0";
                max:String = "*";
        }
        Node Constraint {}
        Node ComparisonConstraint :
                Constraint{
                Operator:String = "=";
        }
        Node ChoiceConstraint : Constraint {}
        Node VSpecToRoot : Link {}
        Node Parent : Link{}
        Node MandatoryChildOf : Link {}
        Node OptionalChildOf : Link {}
        Node GroupCardinalityToVSpec : Link
                                                {}

        Node LHS : Link {}
        Node RHS : Link {}
        Node And : Link {}
        Node AndNot : Link {}
                                                }
```

Step 4: Transform from CVL to Clafer in metaDepth

```
Model ClaferMM {
        Node Link {
                src : Node;
                dst : Node;
        }

        Node Root {}

        Node Clafer {
                Name:String = "";
                Optional:boolean = false;
                Type:String = "";
                CardType:String = "";
                IsRootElement:boolean = false;
        }

        Node Constraint {
                Constraint:String = "";
        }

        Node ClaferToRoot : Link {}
        Node ClaferToParent : Link {}
        Node ConstraintToClafer : Link {}
}
```

- Enumerate all root VSpecs
- Recursively enumerate their children
- Recursively enumerate group cardinalities too
- For each VSpec, create a Clafer clafer
- For each link between VSpecs, create a Clafer link
- For each CVL Constraint, create a Clafer constraint
- Set the Clafer elements' parameters accordingly

Step 5: Generate the concise Clafer representation

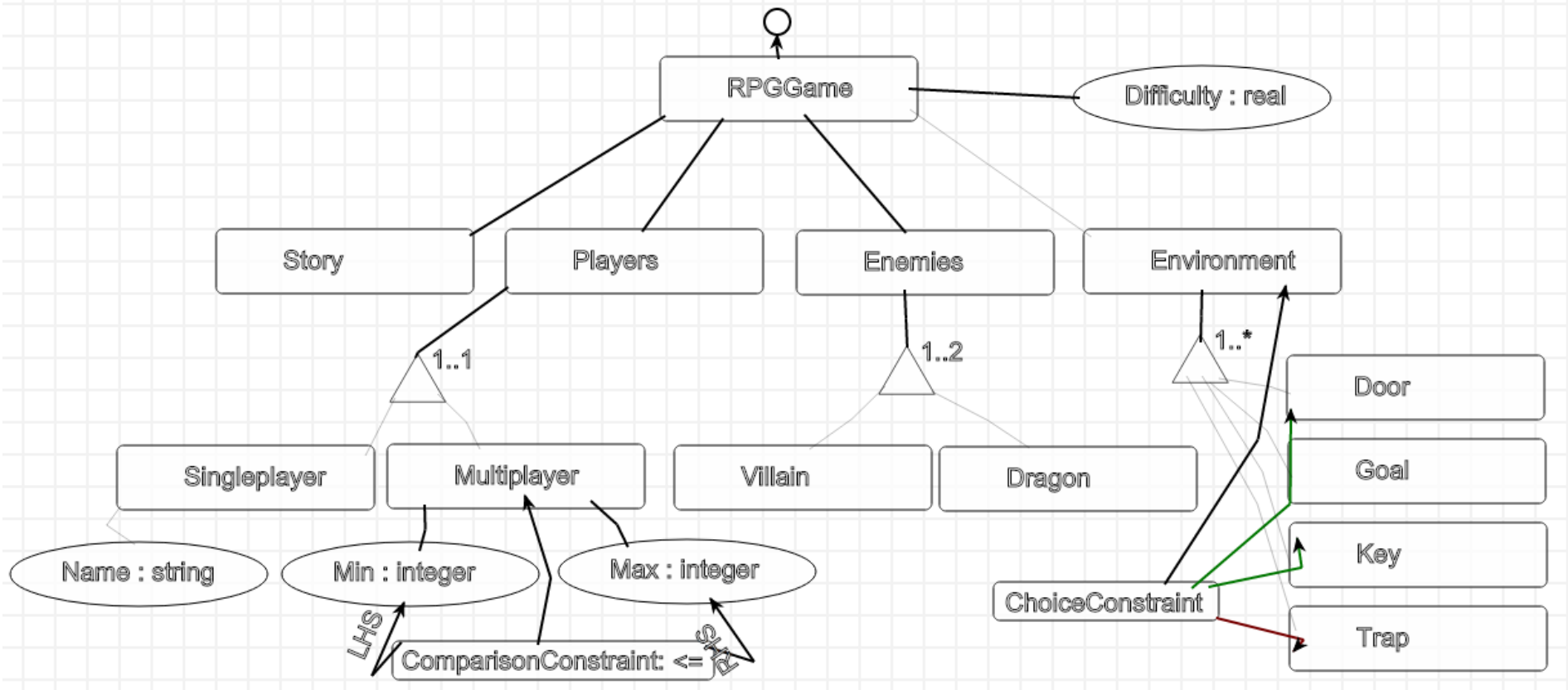
```
[% for (c in Clafer.all.select(x | x.IsRootElement = true)) { %]
[%=c.printClafer() %]
        [%=c.enumChildren() %][% } %]
[%
@template
operation Clafer enumChildren() {
        for (cp in ClaferToParent.all.select(x | x.dst = self)) {
                %][%=cp.src.printClafer() %]
        [%=cp.src.enumChildren() %]
                [%
        }
        for (cc in ConstraintToClafer.all.select(x | x.dst = self)) {
                %][ [%=cc.src.Constraint %] ][%="\n" %][%
        }
}

@template
operation Clafer printClafer() {
        %][%=self.printCardType() %][%=self.Name %][%=self.printType() %][%=self.printOptional() %][%
}
```

- Enumerate the root clafers
- Recursively enumerate the child clafers and constraints
- For each clafer, print parameters

Chapter 2

Example

Example (CVL)


RPGGame
Difficulty : real
Story
Players
Enemies
Environment
1..1
1..2
1..*
Singleplayer
Multiplayer
Villain
Dragon
Door
Goal
Key
Trap
Name : string
Min : integer
Max : integer
ChoiceConstraint
LHS
ComparisonConstraint: <=
RHS

Example (Clafer)

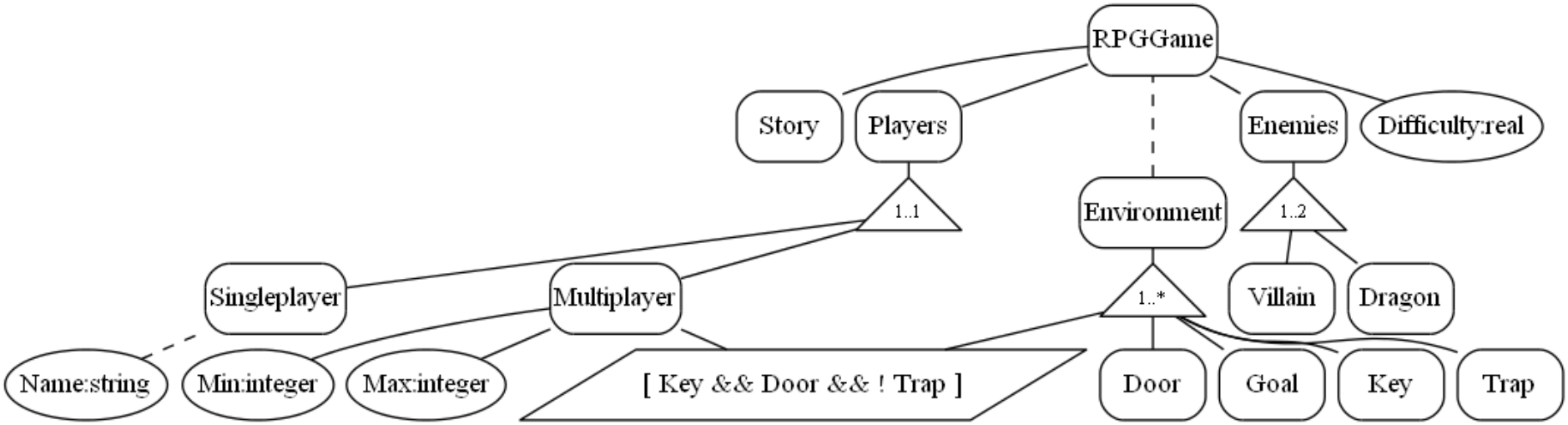
```
RPGGame
    Story

    xor Players
        Singleplayer
            Name : string?

        Multiplayer
            Min : integer

            Max : integer

            [ Min <= Max ]

    1..2 Enemies
        Villain

        Dragon
```

```
Difficulty : real

or Environment?
    Door

    Goal

    Key

    Trap

    [ Key && Door && !Trap ]
```

Chapter 3

Verification

Verification



Chapter 4

Conclusion

Conclusion

A common subset of features between CVL and Clafer exists that is large enough for a successful transformation of a large set of CVL models to Clafer.

Chapter 5

Future work

Future work

- Study similarities and difference of **instantiation** of CVL and Clafer, transforming CVL's variation points to concepts (constraints) or methods (IG manipulation) in Clafer.

- Study CVL **constraints** (OCL based) and Clafer constraints (Alloy based) with a comparison in order to create a more **complete transformation** for constraints.

These suggestions can be considered projects on their own.

Thank you for your attention!

Comments and questions are welcome.