A Formal and Sound Transformation from Focal to UML

An Application to Airport Security Regulations

David Delahaye, Jean-Frédéric Étienne,
and Véronique Viguié Donzeau-Gouge

David.Delahaye@cnam.fr, etiennje@cnam.fr,
donzeau@cnam.fr

CEDRIC/CNAM, Paris, France

UML&FM'08

Kitakyushu-City, Japan
October 27, 2008

Motivations

EDEMOI Project

- Integrate and apply several RE and FM techniques to analyze airport security regulations;
- Use of the Focal specification language to build the formal models of the Annex 17 and Doc 2320 standards:
  - D. Delahaye, J.-F Étienne, and V. Viguié Donzeau-Gouge. *Certifying Airport Security Regulations using the Focal Environment* (FM'06);
  - D. Delahaye, J.-F Étienne, and V. Viguié Donzeau-Gouge. *Reasoning about Airport Security Regulations using the Focal Environment* (ISoLA'06).

Purpose of the UML Diagrams

- Graphical documentation of the formal models for developers.
- Higher-level views pertinent to certification authorities.

Graphical Documentation for Developers

Our Major Concern

A formal framework for an automatic transformation from Focal to UML:

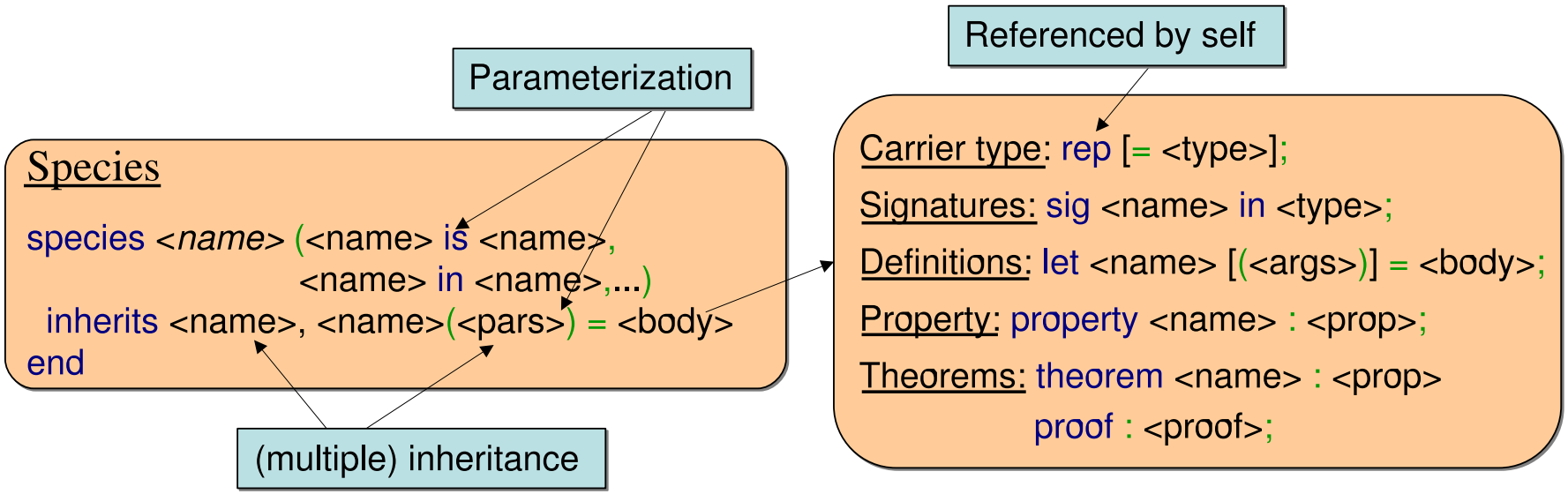
1. Formalize a subset of the UML 2.1 static structure constructs (BNF syntax derived from UML 2.1/XMI schema);
2. Extend the UML metamodel (via profile mechanism) to cater for the semantic specificities of the Focal specification language;
3. Describe the transformation rules from Focal to UML (formal translation using a denotational style);
4. Establish the soundness of the transformation (validating the profile and the generated UML model).

The Focal Environment

What is Focal?

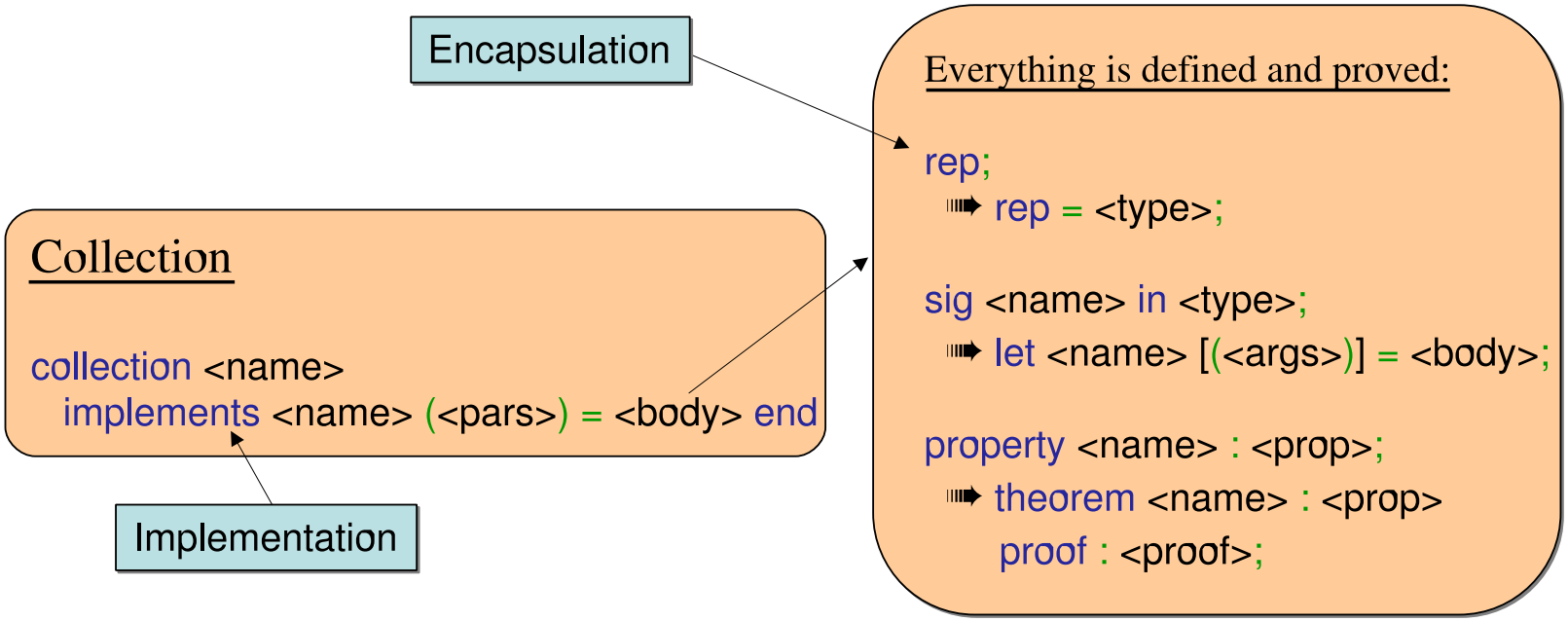
- Specification and proof development system;
- Object-oriented features (inheritance, parameterization);
- Algebraic specification flavor (representation);
- Automatic proof construction (Zenon), verification (Coq).

Specification: Species



The Focal Environment (continued)

Implementation: Collection



Focal Compiler: Outputs

- OCaml code for execution;
- Coq code for certification (with Zenon providing the Coq proofs);
- Documentation in FocDoc (XML format), with options for LaTeX and HTML;
- Inheritance and dependence graphs.

Abstract Syntax of the UML Static Structure

UML Model

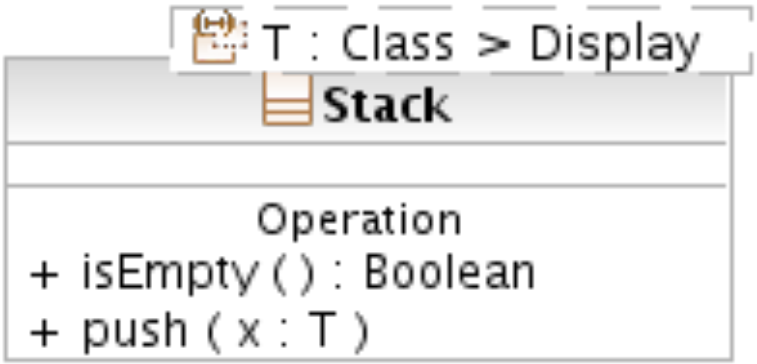
$$\begin{array}{rcl} \mathit{Um} & ::= & \mathit{decl}^* \\ \mathit{decl} & ::= & \mathit{class} \mid \mathit{constraint} \mid \mathit{opaque} \mid \mathit{dep} \end{array}$$

Class

$$\begin{array}{rcl} \mathit{class} & ::= & \mathit{option}\ \textbf{class}\ \mathit{ident}\ [\ \textbf{(}\mathit{cl\text{-}param}\ \{, \mathit{cl\text{-}param}\}^*\textbf{)}\ ] \\ & & \quad [\ \textbf{binds}\ \mathit{bind}\ \{, \mathit{bind}\}^*]\ [\ \textbf{inherits}\ \mathit{ident}\ \{, \mathit{ident}\}^*\ ]\ \textbf{=} \\ & & \quad \mathit{constraint}^*\ \mathit{attr}^*\ \mathit{opr}^*\ \mathit{class}^* \\ & & \textbf{end} \\ \mathit{option} & ::= & [\ \mathit{visibility}\ ]\ [\textbf{final} \mid \textbf{abstract}] \\ \mathit{visibility} & ::= & \textbf{public} \mid \textbf{private} \mid \textbf{protected} \\ \mathit{cl\text{-}param} & ::= & \mathit{ident}\ :\ \textbf{class}\ [> \mathit{class\text{-}type}] \mid \mathit{ident}\ :\ \textbf{opaqueExpr}\ [> \mathit{type}] \\ \mathit{class\text{-}type} & ::= & \mathit{ident} \mid \mathit{bind} \\ \mathit{type} & ::= & \mathit{class\text{-}type} \mid \textbf{Integer} \mid \textbf{Boolean} \mid \textbf{UnlimitedNatural} \mid \textbf{String} \\ \mathit{bind} & ::= & \mathit{ident}\textbf{<}\mathit{subs}\ [, \mathit{subs}^*]\textbf{>} \\ \mathit{subs} & ::= & \mathit{ident} \rightarrow \mathit{ident} \end{array}$$

An Example: Stacks

UML Notation



Abstract Syntax

public class *Stack* (*T* : **class** > *Display*) **=**

  public operation *isEmpty* **(return** *ret* : **Boolean)**

  public operation *push* **(in** *x* : *T* **)**

end

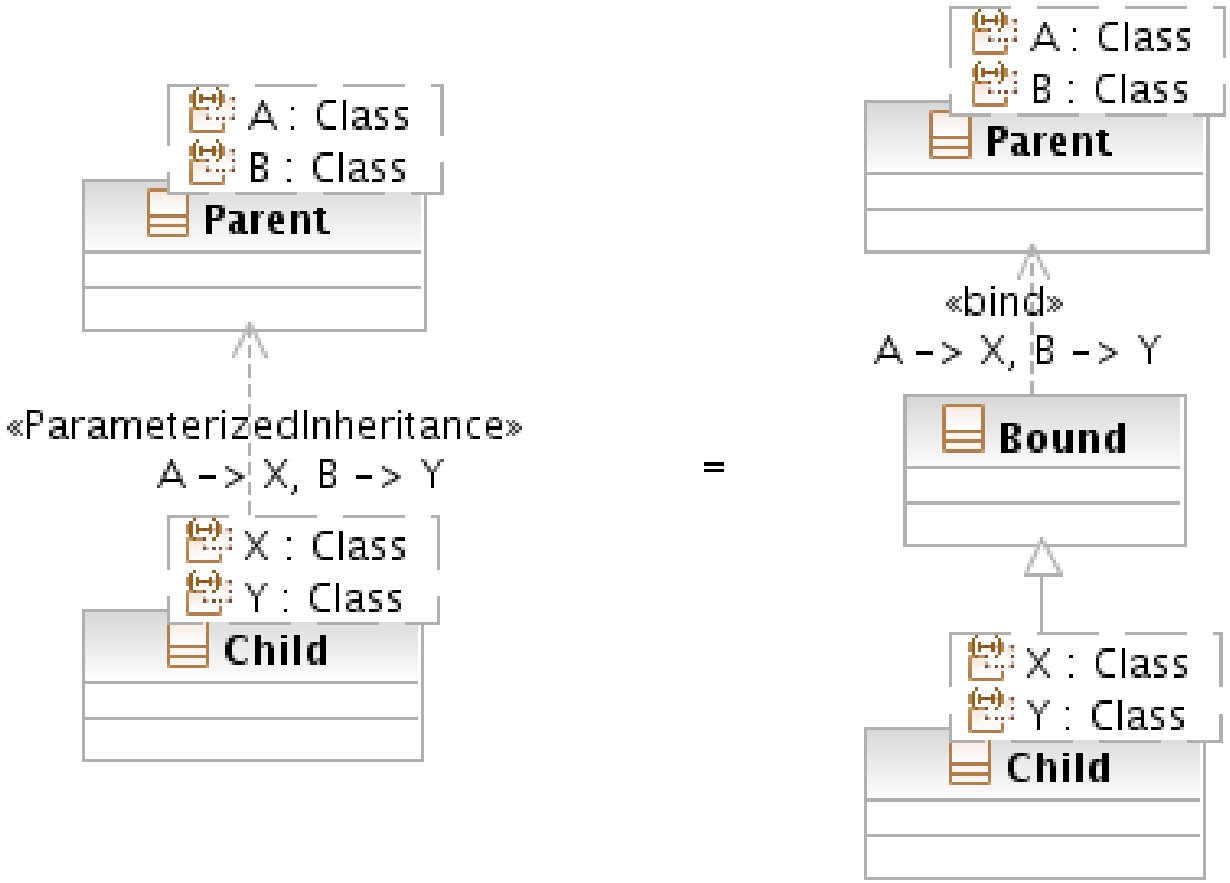
Extending the UML Metamodel

A Profile for Focal

- Need to consider the semantic specificities of the Focal specification language to properly document Focal models in UML;
- Use of the profile mechanism to tailor the UML metamodel:
  - Define appropriate stereotypes to reflect the semantics of each Focal constructs («Species», «Collection», «ParameterizedInheritance», etc);
  - Encode the semantics relative to the template binding construct:
    O. Caron et al. *An OCL Formulation of UML2 Template Binding* (UML04);
    Extension to consider nested *bound* classes and inherited members;
  - Introduce the parameterized classes Fun and Pair to model function and product types.

«ParameterizedInheritance» Stereotype

Extending the Dependency Metaclass



An Example: Finite Stacks

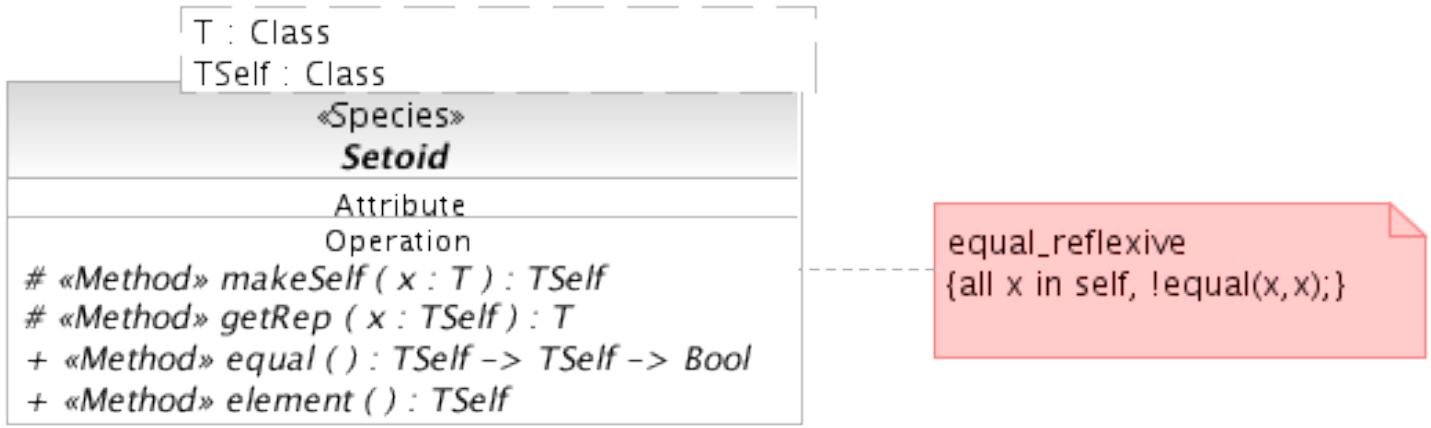
- Need to be able to compare two items on a stack and also two stacks;
- Make use of the predefined species setoid (root node).

Species setoid

```
species setoid =

  rep;

  sig equal in self → self → bool;
  sig element in self;

  property equal_reflexive : all x in self, !equal (x, x);

  property equal_symmetric : all x y in self, !equal (x, y) → !equal (y, x);

  property equal_transitive : all x y z in self,
    !equal (x, y) → !equal (y, z) → !equal (x, z); ...

end
```

Root Node and Representation

Setoid Class



Translation

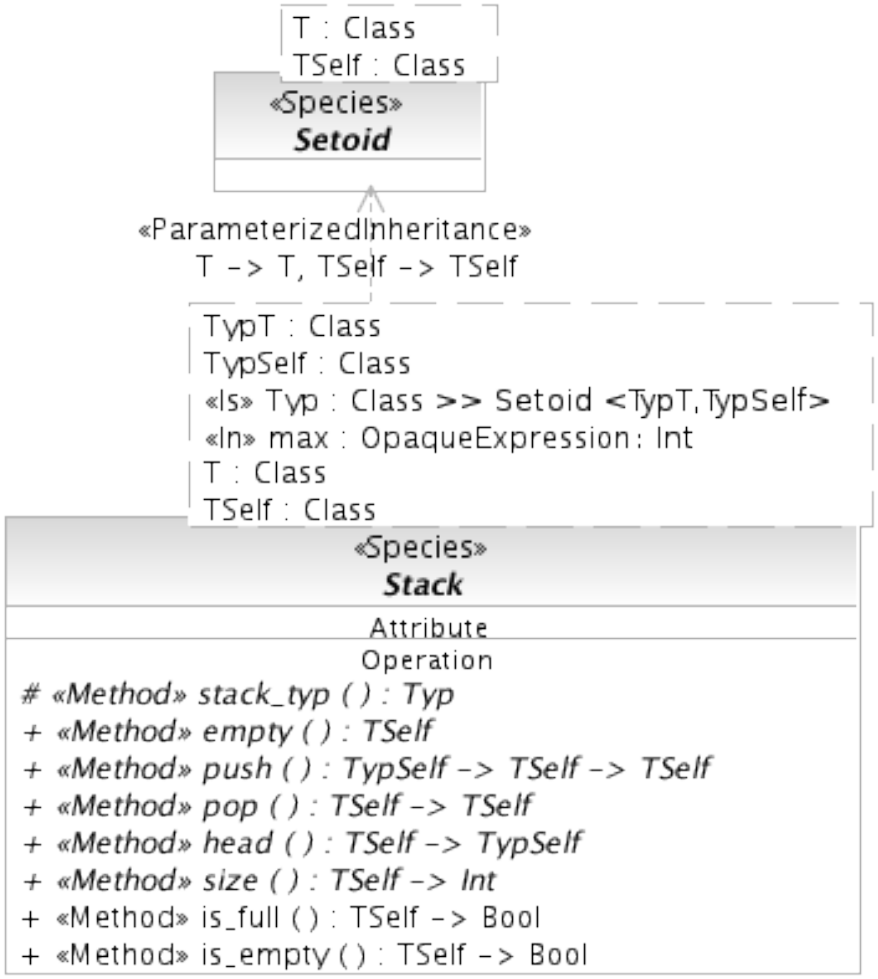
- Species: abstract factory class («Species»)
  manipulating immutable value objects of a given type;
- Representation: two type parameters T and TSelf («FocalType») where:
  - T represents the type of the entities;
  - TSelf represents the class in which T is encapsulated.
- The correlation between T and TSelf is specified by two protected factory methods makeSelf and getRep (generated only for root nodes).

Finite Stacks

```
species stack (typ is setoid, max in int) inherits setoid =

  sig empty in self;
  sig push in typ → self → self;
  sig pop in self → self;
  sig head in self → typ;
  sig size in self → int;

  let is_full (s) = #int_eq (!size (s), max);
  let is_empty (s) = !equal (s, !empty);

  property size_max : all s in self, #int_leq (!size (s), max);

  property ie_empty : all s in self, !is_empty (!empty);

  property hd_push : all e in typ, all s in self,
    not (!is_full (s)) → typ!equal (!head (!push (e, s)), e);

  property id_ppop : all e in typ, all s in self,
    not (!is_full (s)) → !equal (!pop (!push (e, s)), s); ...

end
```

Inheritance and Parameter Declarations

Stack Class



Translation

- Collection parameter declaration *c* **is** *S*: three type parameters cT, cSelf and c, with:
  - cT and cSelf characterizing the representation of species *S*;
  - c constrained by the factory class generated for *S*.
- Entity parameter declaration *e* **in** τ: non-type parameter, opaque expression;
- Inheritance: dependency relation stereotyped with «ParameterizedInheritance».

Concrete Representation and Collection

An Implementation Based on Lists

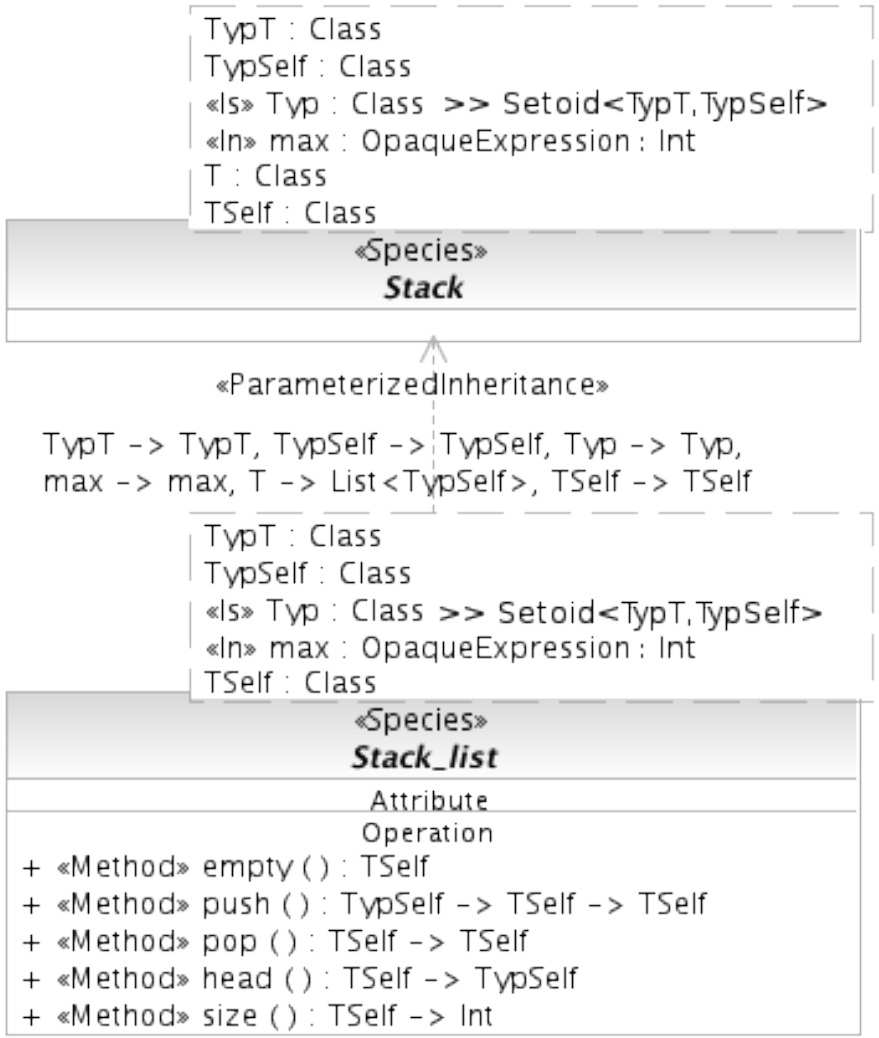
```
species stack_list (typ is setoid, max in int) inherits stack (typ, max) =

  rep = list (typ);

  let empty = #Nil;
  let push (e, s) = if !is_full (s) then #foc_error ("Full␣stack!")
                    else #Cons (e, s);
  let pop (s) = if !is_empty (s) then #foc_error ("Empty␣stack!")
                else #tl (s);
  let head (s) = if !is_empty (s) then #foc_error ("Empty␣stack!")
                 else #hd (s);
  let size (s) = #length (s);

  proof of ie_empty = ...; ...

end
```

Finite Stacks of Integers

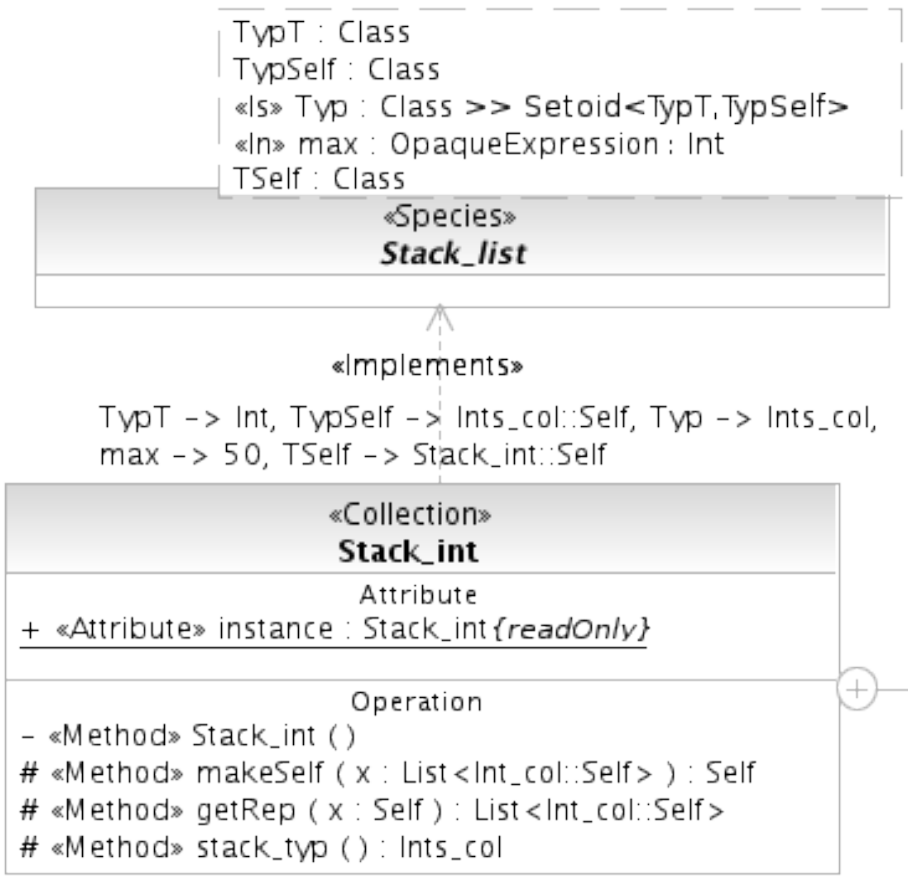
```
collection stack_int implements stack_list (ints_col, 50) = end
```

Concrete Representation

Stack_list Class



Stack_int Class



- Collection: concrete singleton factory class, with:
  - a static read-only attribute instance for the singleton instance;
  - a private constructor (to prevent uncontrolled instantiation).
- Abstraction of the concrete representation: inner class Self.

Soundness of the Transformation

Type Preservation (Semantics)

Established by showing that:

1. The constraints specified in the Focal profile do not invalidate the well-formedness rules of the UML metamodel;
2. The UML model generated from a well-typed Focal specification satisfies:
   - The well-formedness rules of the UML metamodel;
   - The constraints in the Focal profile.

Structure Preservation (Isomorphism)

- Type preservation not enough;
- Need of structure preservation theorems (e.g., preservation of the number of methods);
- Possible to show that the transformation is bijective (up to renaming and except for proofs): work in progress.

Implementation

FocDoc

XML format used by the Focal compiler for documentation.
The information are extracted from:

- Focal abstract syntax;
- Structured comments annotating a Focal specification;
- Type inference and dependency analysis performed by the compiler.

Two Parts

1. UML profile for Focal specified with the UML2 Eclipse plug-in:
   - Use of the integrated OCL checker to validate the constraints in the profile;
   - Use of static profile definition to provide implementation for the operations and derived attributes characterizing each stereotype in the profile.
2. XSLT stylesheet that encodes the transformation rules.
   From a Focal specification in FocDoc to a UML model in XMI.

Conclusion

Formal Framework for Graphical Documentation for Developers

- Formal syntax for a subset of the UML 2.1 static structure constructs;
- Extension of the UML metamodel (via profile mechanism):
  - Semantic specificities of the Focal specification language;
  - Semantics relative to the template binding construct.
- Formal description of the transformation rules from Focal to UML:
  - A design pattern for the representation of complex algebraic structures and algorithm within an OO paradigm;
  - The UML models produced can be used to map a Focal specification to any appropriate OO programming language (e.g., Java or C#).
- Soundness of the transformation (type preservation).

Future Work

- Another notion of soundness (structure preservation);
- Higher-level views more pertinent to certification authorities;
- Dynamic views of the formal models (i.e., sequence and state-transition diagrams) through static analysis.

Thank you!

ありがとう！