A Metamodel-based Approach to Reverse Engineering Ada source code to UML

Xavier Sautejeau

Ada Europe 2006

Contents

- Rationale
- Difficulties
- A metamodel based solution
- Conclusion

Rationale

- Reusing software assets written in Ada
  - With high size, complexity and age
  - Already developed, tested, certified…
- In new projects
  - Where UML is used in the development process
- Or for maintenance activities
  - Where UML is used as a documenting tool
- Requires
  - An automated means of reverse-engineering the Ada code to UML models

A challenging task

- UML and Ada intersect to a large extent
  - Same « business domain » => SW engineering
- But the mapping remains difficult
  - Pattern-based
  - Divergent approaches
  - Other issues
  - Gap

“Natural Mapping”

Ada construct	UML model element
Package	Class | Package
Operation	Operation
Variable	Attribute
Enumeration	Enumeration

Some possible equivalences from Ada to UML

Pattern-based mapping

- Some Ada patterns/idioms may represent a single UML construct
  - E.g. An Ada package with a single record type and operations may map to a UML class
  - E.g. subprogram_specification + subprogram_body_stub + subunit => operation
- And conversely
  - E.g. An Ada record (as opposed to a record type) may map to a UML class with a <<Singleton>> stereotype

Divergent approaches
(some examples)

- Out passing mode for current instance parameter
  - The current instance parameter of a UML non-static operation has the passing mode of its operation, derived from its « isQuery » property.
  - Can be overcome through using a profile, but it is not an Ada specific issue and should be part of "native" UML
- Orthogonality of types and namespaces
  - In Ada, types and packages are separate concepts
  - In UML, a classifier **is a** namespace
- Operations and namespaces
  - In Ada, an operation is a namespace, not in UML
  - In UML, an operation can only belong to a class, not to a package or to another operation as in Ada

Divergent approaches

- Issues mainly arise from the fact that UML is
  - Class-centric
  - Biased towards Java and C++

Comments

(other issues)

- Not unique to Ada
- May contain valuable information
  - Configuration management info
  - Requirements traceability
  - SPARK annotations
- Main problem is assigning correct ownership to comments
  - Is a comment related to the element **before** or **after** it in the syntax tree ?
    - (Assuming syntactical proximity is a factor of relevance)

Declaration order

(gap)

- Required for
  - Compilation
  - Flow analysis (cf SPARK)
- Inherent to Ada Grammar
  - Built-in total ordering
    - i.e. order is defined for heterogeneous element kinds
- Foreign to UML
  - Only partial ordering
    - For elements of the same kind (e.g. attributes)

Declaration order
(gap)

- MOF : OMG's Meta Object Facility
  - Defines the rules governing the construction of UML-related Meta Models
  - The rough equivalent of BNF for languages
  - **But without total ordering of elements !**

Declaration order

```
package Stacks is
  type Stack is private;

private
  Stack_Size : constant := 100;
  type Pointer_Range is range 0..Stack_Size;
  subtype Index_Range is Pointer_Range range
     1..Stack_Size;
  type Vector is array (Index_Range) of Integer;

  type Stack is record
      Stack_Vector: Vector;
      Stack_Pointer: Pointer_Range;
  end record;
end Stacks;
```

Declaration order

```
package Stacks is
  type Stack is private;

private
  Stack_Size : constant := 100;
  type Pointer_Range is range 0..Stack_Size;
  subtype Index_Range is Pointer_Range range
     1..Stack_Size;
  type Vector is array (Index_Range) of Integer;

  type Stack is record
     Stack_Vector: Vector;
     Stack_Pointer: Pointer_Range;
  end record;
end Stacks;
```

== declaration order dependencies ==

Pointer_Range (type) => Stack_Size (constant)

Index_Range (type) => Pointer_Range (type)

Vector (type) => Index_Range (type)

Stack_Vector(attribute) => Vector(type)

Stack_Pointer(attribute) => Pointer_Range(type)

Metamodelization

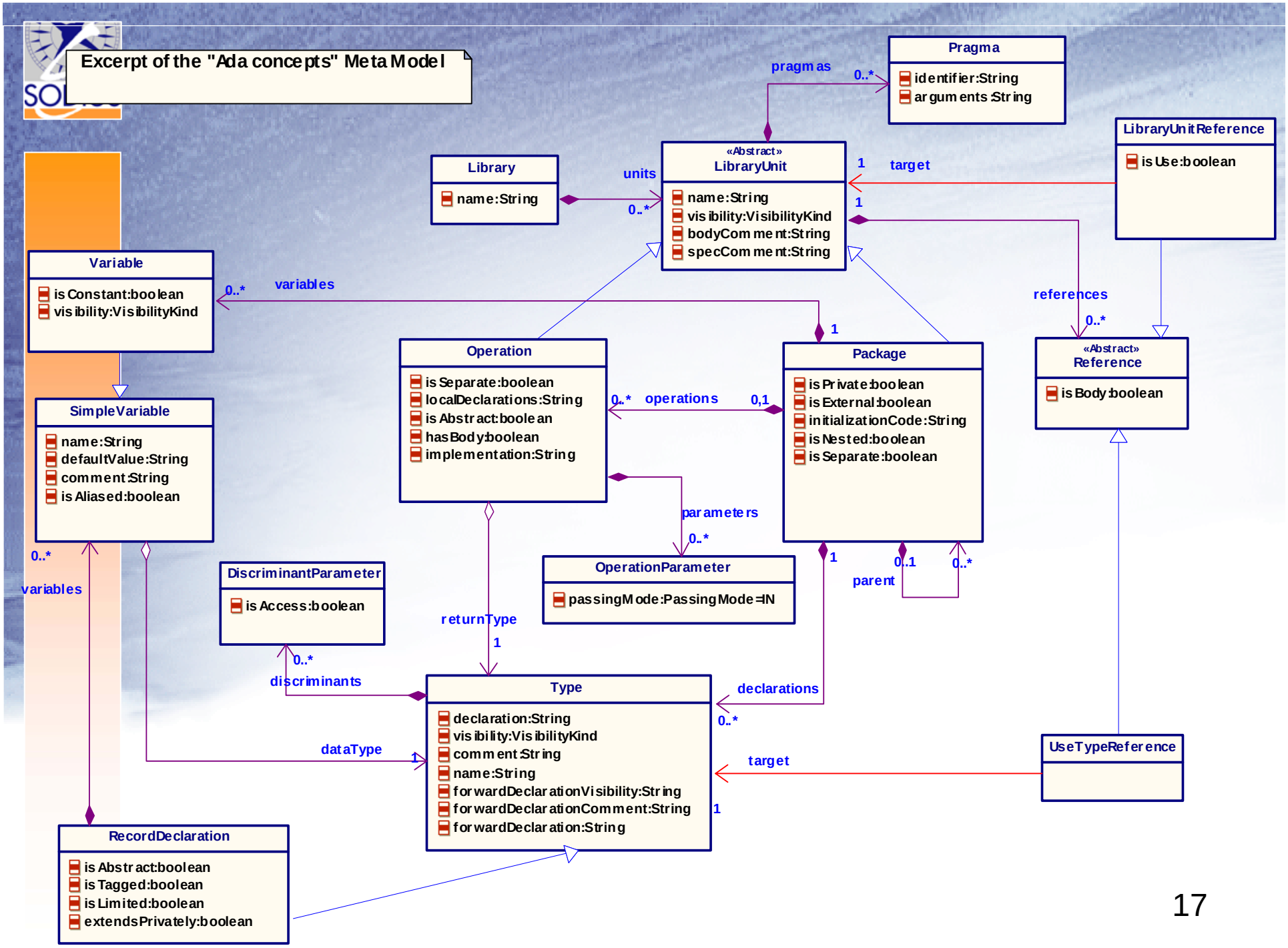
- Rules for reverse engineering Ada to UML should rely on same mapping as the one used for generating Ada code from a UML model.
- This mapping is embodied in a UML profile :
  - set of stereotypes, tagged values and constraints
  - to bring additional descriptive power to the UML notation

Model transformation

- 3 steps process
  - Ada syntax => Ada concepts MM => UML MM + Ada profile
- Ada concepts MM
  - MOF compliant representation of Ada syntax rules
- Model Driven Architecture
  - OMG Framework for mapping between two MOF instantiations

Ada syntax => Ada concepts MM

- Conversion with appropriate level of details
  - Some Ada non-terminal syntactic constructs represented as string properties without further analysis
    - e.g. operations bodies, attributes initial values
  - Granularity of concepts is similar to the one of the UML MM


Excerpt of the "Ada concepts" Meta Model
Pragma
identifier:String
arguments:String
pragmas
0..*
LibraryUnitReference
is Use:boolean
«Abstract»
LibraryUnit
name:String
visibility:VisibilityKind
bodyComment:String
specComment:String
Library
name:String
units
0..*
1
target
1
Variable
is Constant:boolean
visibility:VisibilityKind
0..*
variables
1
references
0..*
«Abstract»
Reference
is Body:boolean
Operation
is Separate:boolean
localDeclarations:String
is Abstract:boolean
hasBody:boolean
implementation:String
0..*
operations
0,1
Package
is Private:boolean
is External:boolean
initializationCode:String
is Nested:boolean
is Separate:boolean
SimpleVariable
name:String
defaultValue:String
comment:String
is Aliased:boolean
parameters
0..*
0..*
variables
DiscriminantParameter
is Access:boolean
OperationParameter
passingMode:PassingMode=IN
1
0..1
0..*
parent
returnType
1
0..*
discriminants
Type
declaration:String
visibility:VisibilityKind
comment:String
name:String
forwardDeclarationVisibility:String
forwardDeclarationComment:String
forwardDeclaration:String
declarations
0..*
dataType
1
target
1
UseTypeReference
RecordDeclaration
is Abstract:boolean
is Tagged:boolean
is Limited:boolean
extendsPrivately:boolean

Ada concepts MM => UML MM + Ada profile

- Rules based
  - Dedicated Model Transformation Syntax
  - Facilitate patterns description
- Customizable
  - To adapt to project specific coding conventions
    - E.g. More than one class per package

A customizable process

- Refine
  - UML Ada profile
  - Ada concepts MM
- Update
  - Parser
  - Transformation rules

Applicability

- Static dimension of the system
  - Requires syntactical analyis of the code
    - Affordably automatable
    - Direct matches or simple patterns recognition
- Dynamic dimension
  - Requires semantic analysis of the code
    - More difficult to automate
    - More complex patterns and more open to interpretation

Exploiting the result

- Regenerate code for legacy system "L"
  - With limitations
    - Such as total declaration order preservation
    - May require some adjustments to the model
- Abstract the structure of L
  - For documentation purposes
- Model interfaces from new system "N" to L
  - For reuse of L
    - No need to regenerate L for generating interfaces to it
  - And then optionally generate code for N

Implementation

- Part of the I-logix Rhapsody in Ada product
  - Rules editor licensed separately

Future directions

- Support for SPARK reverse-engineering
  - To match our SPARK code generation UML Profile
- Target other Meta-Models
  - Bridging better the declaration order gap between formal grammars and MOF
  - Declaration order essential to
    - Compilation (implementation, low-level concern)
    - Proofs (system-level concern)
  - Hood, AADL ?

Questions ?

Xavier Sautejeau

xsautejeau@sodius.com