Ensuring Component Models’ Consistency by Means of Metamodel Level WFRs

Vladiela PETRAȘCU

Computer Science Department
Faculty of Mathematics and Computer Science
Babeș-Bolyai University, Cluj-Napoca, Romania
vladi@cs.ubbcluj.ro

November 25th, 2008, Prague
Outline

- Modeling the Abstract Syntax of DSMLs
  - MDE
  - Anatomy of a DSML
  - Abstract Syntax Modeling
  - “Meta”-tools requirements

- ContractCML Abstract Syntax Modeling in OCLE
  - ContractCML
  - Test Model
  - OCLE Metamodeling
  - OCLE Demo
  - Conclusions
Modeling the Abstract Syntax of DSMLs
MDE (Model Driven Engineering) - state of the art approach in SE, trying to formalize/automatize a major part of the development process

- Models become *first class citizens*

- MDE approach
  - start with a model of the problem at hand
  - apply a series of transformation steps (M2M & M2T), in order to reach a final system implementation

- At the core of MDE
  - DSMLs
  - Transformation engines & generators
Anatomy of a DSML

- DSML (Domain Specific Modeling Language)
  - Abstract syntax
  - Concrete syntax
  - Semantics

- Abstract syntax
  - Metamodel
    - Describes concepts in the language and their interrelationships
    - Expressed by means of a class model, using a UML subset or a specialized metamodeling language (MOF, Ecore, XCore)
  - Well Formedness Rules (WFRs)
    - Rules stating whether a model written in the language is valid or not
    - Provide a more detailed description of syntactical rules than possible by concepts and relationships alone
    - Written in natural / formal language (OCL or OCL-like)
    - Particularly useful when implementing a tool to support the language, as they can be used to validate the correctness of models as they are created
Abstract Syntax Modeling

- **Stages**
  - Concepts’ identification
  - Concepts’ modeling
    - Uses standard OO modeling features: classes for describing concepts, associations for capturing their interrelationships, generalization for defining categories of concepts, packages for grouping
  - Well-formedness rules (WFRs)
    - Identified by constructing both valid and non valid models in the language
    - Implications of their combined use (possible conflicts) should be carefully investigated
  - Query operations (QOs)
    - Useful in constraints, as well as model validation
  - Testing and Validation
    - Uses snapshots that match example models
Challenges in specifying the abstract syntax and implementing the associated repository:

- Support for specifying a metamodel compliant to the abstract syntax (meta-metamodel + WFRs) of its metamodelling language
- Support for specifying, compiling and evaluating constraints written in OCL or OCL-like languages
- Support for an easy specification of snapshots, needed for testing the abstract syntax
- Support for a quick identification of metamodel instances causing the failure of different WFRs
- Complete repository code generation, including code for WFRs and query operations
ContractCML
Abstract Syntax Modeling in OCLE
ContractCML

- Four levels for component contracts
  1. basic (syntactic) - signatures of required/provided services
  2. behavioral (semantic) - functional description of required/provided services by means of pre/postconditions (individual behavior)
  3. synchronization - dependencies between required/provided services (sequencing, parallelism) in a distributed concurrent context (global behavior)
  4. quality-of-service - non functional properties

- ContractCML (Contract Component Modeling Language)
  - a DSML imagined as basis of a framework able to support component contracts’ specification at all four levels
  - hierarchical
  - for now it only covers the first two levels, but its modular, extensible architecture allows adding new ones in a non invasive way
ContractCML Metamodel

Diagram showing the relationships between InterfaceSpec, SyntacticSpec, SemanticSpec, BlackBoxComponent, WhiteBoxComponent, Architecture, and Basic.
ContractCML::Basic
ContractCML::InterfaceSpec::SyntacticSpec
ContractCML::InterfaceSpec::SemanticSpec
ContractCML::BlackBoxComponent
ContractCML::WhiteBoxComponent
ContractCML::Repository

Diagram:

- Repository
  - +repository
    - 0..1
  - +types
    - 0..*
  - +interfaces
    - 0..*
  - +componentTypes
    - 0..*
  - +components
    - 0..*
  - +architectures
    - 0..*
Test Model (1/2)

CoCoME subset:
- TradingSystem
  - CashDeskLine
  - Inventory
    - GUI
    - Application
    - Data
    - Database
Test Model (2/2)
OCLE Metamodelling

- Metamodel (meta-classes & relationships)
  - represented as a UML class model; the metamodeling language subsets UML 1.5

- Well-formedness rules
  - written as OCL 2.0 invariants
  - grouped within .bcr files => they do not pollute the metamodel

- Query operations
  - defined using the OCL def mechanism

- Metamodel testing and validation
  - accomplished by means of snapshots
  - tool assets: precise error messages & valuable debugging aid
OCLE Demo

...
Simple WFR example

```plaintext
package ContractCML::Basic

-- [WFR1] Every NamedElement should have a valid, non empty name

context NamedElement
    inv nonEmptyName:
        not self.name.isUndefined() and self.name <> ''

endpackage
```
Example WFR using a query operation

```ruby
package ContractCML::Architecture
-- [WFR10] The interfaces that type the two ports linked by a binding should be syntactically compatible (exact signature match for now)
context Binding inv syntacticallyCorrectBinding:
  self.fromEnd.port.interf.exactSignatureMatch(self.toEnd.port.interf)
endpackage

package ContractCML::InterfaceSpec::SyntacticSpec
-- [QO2] Checks if two interface signatures match exactly
context Interface
def: let exactSignatureMatch(interf:Interface):Boolean =
  self.name = interf.name and
  self.services->size() = interf.services->size() and
  self.services->forAll(s:Service | interf.services->exists(s1:Service | s.exactSignatureMatch(s1) ))
endpackage
```
WFRs’ shape affects the efficiency of the debugging process

```plaintext
package ContractCML::Repository

-- [WFR16] No name collisions for components inside a repository
-- simplest form, providing no useful debugging info

context Repository
inv noComponentNameCollisions:
    self.components->forAll(c1,c2:Component | c1 <> c2 implies
c1.name <> c2.name)

endpackage

-- context changed, more debugging info

package ContractCML::BlackBoxComponent

context Component
inv noComponentNameCollisions:
    self.repository.components->select(c:Component |
c <> self and c.name = self.name)->isEmpty()

endpackage
```
More shape issues ...

-- [WFR8] An interface cannot be provided/required by multiple ports of the same component type

context ComponentType
inv sameInterfProhibited:

  -- most common (simple) shape, not so useful
  -- smallest amount of debugging info (only the ComponentType instance breaking the rule)
  self.allPorts->forAll(p1,p2:Port | p1<>p2 implies p1.interf <> p2.interf)

  -- more useful
  -- returns, by partial evaluation, all interfaces associated to at least two different ports

  self.allInterfaces->select
    ( interf:Interface | self.allPorts->select
      (p:Port | p.interf = interf)->size() > 1
    )->isEmpty()
-- the most useful
-- returns all interfaces associated to at least two different ports,
together with the corresponding ports

let wrongInterfs:Set(Interface) =
    self.allInterfaces->select(interf:Interface |
        self.allPorts->select(p:Port|p.interf=interf)->size()>1 )
in
wrongInterfs->iterate( i;
    s:Set(TupleType(interf:Interface,ports:Set(Port))) = Set{} | s->including(Tuple{interf = i, ports = self.allPorts->select(p:Port|p.interf = i)})->isEmpty() )
Query operations used in constraints

package ContractCML::Architecture

-- [QO9] Provisions of a certain ComponentInstance
context ComponentInstance
def: let providedPorts:Set(Port) =
    self.component.componentType.providedPorts
endpackage

package ContractCML::InterfaceSpec::SyntacticSpec

-- [QO1] Checks if two service signatures match exactly
context Service
def: let exactSignatureMatch(s:Service):Boolean =
    self.name = s.name and
    self.type = s.type and
    self.arguments->size() = s.arguments->size() and
    Sequence{1..self.arguments->size()}->forAll(i:Integer |
                  self.arguments->at(i).type = s.arguments->at(i).type)
endpackage
Querying the model for useful info

```ruby
package ContractCML::Repository

-- [Q011] Returns all components providing a certain interface from within a repository

context Repository
  def: let AllComponentsProviding(interf:Interface):Set(Component) =
    self.components->select(c:Component |
      c.componentType.allProvidedInterfaces->includes(interf))

endpackage
```
Conclusions

- For most DSMLs, the metamodel alone is unable to capture all domain constraints => WFRs are needed in order to represent them
- In addition to the facilities offered for specifying, compiling and evaluating such constraints, tools should provide valuable support in model debugging
- Beyond using powerful tools, using efficient specifications is highly important
Questions?