STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVII**, Number 1, 2002

UML MODEL CHECKING

DAN CHIOREAN, ADRIAN CARCU, MIHAI PASCA, CRISTIAN BOTIZA, HORIA CHIOREAN, AND SORIN MOLDOVAN

Abstract. Correctness against the UML definition has to be a prerequisite for every UML model. In terms of programming languages this requirement is stated: "the precondition for every application is to be syntactically and semantically correct against the language specification". The objective of this paper is to go over the state of the art in this domain highlighting some drawbacks in the UML 1.4 AO and WFR[1]. The XMI adoption as a standard format for UML models transfer opened the way to verifying the level at which different UML tools comply with the UML semantics. Taking into account that existing OCL tools do not implement all the functionalities required for efficient UML model checking, we have designed and implemented an OCL evaluator[2]. The possibility to check every UML Model stored in XMI format, a repository fully compliant with UML 1.4, including all the AO, the possibility to evaluate the WFR, MR[3] and BCR[4], are among the main features of our tool.

Key words: UML 1.4, OCL, UML model checking, CASE Tools, AO, WFR, OCL evaluator

1. The UML model correctness

UML model correctness is certainly a very important aspect unfortunately ignored by many specialists in the modeling domain. How else could we possibly explain a series of errors found in different UML models, which the user is not warned about after the check?

By model correctness, we understand the correctness of the model against the modeling language. For UML this means satisfying the WFR. Further, after the WFR have been passed, the BCR have to be syntactically and semantically correct. Asking that different kinds of applications comply with a set of Methodological Rules, may extend the concept of "correctness". An aspect related with model checking is the moment at which the check is performed. In this paper, we are referring to key moments in the application's life cycle; for example the moment

2000 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.3 [**Software**] : Software Engineering – *Coding Tools and Techniques*; D.2.7 [**Software**] : Software Engineering – *Distribution, Maintenance and Enhancements* .

when the design model is automatically turned into code, the moment right before the end of modeling, the moment before exporting the model to another tool etc. The importance of model validation in the above-mentioned key moments is unanimously recognized. For example, in his paper "Consistency Checking" [Moore2000], Michael Moors states: "... during model editing, the model will frequently be syntactically incorrect, and the tool needs to be able to allow for syntactical incorrectness in this mode." The Amigos also call this "Inconsistent models for work in progress," because "the Final model must satisfy various validity constraints to be meaningful."

To understand better the value of UML models correctness, we suggest you reflect over the syntactic and semantic correctness of an application against the implementation language. In this later case, the existence of errors does not allow you to build the application. The process will be interrupted at compile or link time. Using the existent CASE Tools, the errors identified in the analysis or design models, do not interrupt the development process immediately. Consequently, errors are transferred to the next model produced in the software lifecycle. This implies a bigger amount of time and money spent on application development.

2. State of the art

To test UML model correctness, at least all the WFR must be evaluated. Naturally the "precondition" for the above statement is: The WFR have to be correct and complete (in other words, they have to cover at least all the important semantic features of the UML model elements).

The usage of formalisms such as script languages or formal languages (different from OCL) has some drawbacks. In order to be rigorous, we have to demonstrate for each WFR, the equivalence between its specification in OCL and its specification in the used formalism. (Supposing that WFR are correctly specified in OCL). Another problem, even more embarrassing, can appear when the checks are done using a UML CASE tool. This is because even in the best case (when the CASE Tool repository fully implements the UML metamodel), the tools don't allow the user to create some model elements or to declare certain relationships among the existent model elements. The most used CASE Tools – Rational Rose, Together, Poseidon, etc. haven't yet implemented: the Inheritance Relationship among packages, the Permission Relationship between packages (including standard stereotypes for this relationship), the Collaboration ModelElement, and other concepts defined in UML 1.4. Moreover, the Repository Interface for the above-mentioned tools is pretty different from the interface formed by joining the `get` and `set` operations defined for the UML metamodel classes and the AO. (Our position is that a minimal UML repository interface should include at least the AO and the set and get operations)

Among the existent CASE Tools offering OCL support (`http://www.klasse.nl/ocl`) neither Argo (Poseidon) nor Use or ModelRUN don't provide user access

to the tool repository by means of AO. Consequently they do not support UML model checking in a straightforward manner.

In some UML papers, and particularly in [Richters 2000] there are mentioned different drawbacks of the UML AO and WFR. Most of them are syntactic and semantic errors. Unfortunately, there are also some conceptual errors. In the following, we will try to focus on this category of errors.

We noticed that the UML 1.4 "static semantics", expressed using OCL expressions, contains a lot of errors. Consequently, we will try to find the rationale of this situation and propose some solutions.

3. The LCI OCL Evaluator

Because the main objective of this paper is OCL, AO and WFR, we will not insist on our tool architecture. Below we roughly present how to use our checker.

As we can see in Figure 1, the main components of this tool are: the repository, the XML reader, the OCL/UML Type System, the syntactic analyzer, the semantic analyzer, the evaluator and the GUI.

First, the user has to load the model, the UML 1.4 metamodel (both expressed in XMI format) and the WFR or other constraints or operations specifications, expressed in a text with the ".ocl" extension. The succession of these three operations does not matter.

Before beginning the checking process, the user has to verify the OCL expressions syntactically and semantically. In case of semantic errors, the tool offers the possibility to do a partial evaluation. In this process, the type of expressions located before the error can be calculated.

The next step consists in identifying the model element(s) to be checked next against an OCL constraint. Finally, the last step consists in constraint evaluation. As we mentioned before (in case of semantic errors), the user has the possibility to evaluate the whole expression or parts of them.

For the moment we are, the user has just the possibility to modify the OCL expressions. To correct (change) the UML models, he has to use the UML CASE Tools and to save the modified models in XMI format, in order to do a new check. To evaluate dynamically the OCL constraints we intent to translate the OCL specifications in a programming language (Java, C++ etc.) and to generate automatically the code for the UML models. Concerning the methods code, this will be write by hand or generate automatically using the State Transition Diagrams or Object Diagrams.

Taking into account the aspects mentioned in the previous section, one of our main objectives was to support the user in checking UML models. In order to do this, the OCL constraints and specifications can be evaluated both at the metamodel and model level.

4. Errors in AO specifications

First of all we will analyze the operation pair contents, allContents, defined in the Namespace context.

As is very well mentioned in [Richters 2000] both operations have the same specification in English and in OCL. A first observation we made is that the specification below, can not be evaluated because the stop condition is not explicitly mentioned.

contents: Set(ModelElement)
contents = self.ownedElement->union(self.namespace.contents)

In order to evaluate the contents AO defined in the Namespace context, we propose:

contents = **if** self.namespace->isEmpty **then** self.ownedElement **else**
self.ownedElement->union(self.namespace.contents) **endif**

We found this specification clearer even in case of implementing manually in the UML 1.4 API repository. The second remark concerns the expressiveness of the operation's name, directly connected to the contents "specification" in English language. "The operation contents results in a Set containing all ModelElements contained by the Namespace".

For us, the above operation, return the Set of ModelElements visible (potential servers) in a Namespace if we do not take into account the dependency relationship among the Namespace and other server Namespaces.

Analyzing the UML AO, we notice that the above operation is redefined in the Classifier context and in the Package context, where the inherited elements are tacked into consideration. The specifications are identical in both cases. Taking into account that both Classifier and Package are descendents of GeneralizableElement, our opinion is that it would be better to define allContents operation only in GeneralizableElement. In this case, the conflict existent in Subsystem, do to a multiple inheritance of allContents operation both from Classifier and Package disappear.

The operation is equally redefined in Collaboration, in order to rejected the elements specialized in descendants. In this case, the potential conflict due to a multiple inheritance of allContents operation in the GeneralizableElement is solved due to the above mentioned redefinition.

Another example is provided by the specification of allFeatures AO. In this case, there is discordance between the "specification" (description) made in English language and those made in OCL. In [UML 1.4] is stated: "The operation allFeatures results in a Set containing all Features of the Classifier itself and all its inherited Features.", the OCL specification being:

allFeatures =
self.feature->union(self.parent.oclAsType(Classifier).allFeatures)

We can simply notice that in the OCL specification the ancestors' private features had not been eliminated. The correct specification can be (the reject operation can also tacked into consideration):

```
allFeatures = self.feature->union(self.parent.oclAsType(Classifier)
.allFeatures->select(f | f.visibility=#public or f.visibility=#protected))
```

The allFeatures is a very important specification because she is used in allOperations, allMethods and allAttributes AO and in different WFR.

```
allContents = self.contents->union(self.parent.allContents->select(e |
e.elementOwnership.visibility = # public or e.elementOwnership.visibility =
# protected))
```

5. Errors in WFR

In the following we will analyze the WFR using the above mentioned AO, beginning with the WFR[4] defined in the Association context.

The connected Classifiers of the AssociationEnds should be included in the Namespace of the Association, or be Classifiers with public visibility in other Namespaces to which the Namespace of the Association has "access" Permissions.

```
self.allConnections->forAll(r | self.namespace.allContents->includes
(r.participant)) or
self.allConnections->forAll(r | self.namespace.allContents->excludes
(r.participant) implies
self.namespace.clientDependency->exists(d | d.oclIsTypeOf(Permission) and
d.stereotype.name = 'access' and
d.supplier.oclAsType(Namespace).ownedElement->select(e |
e.elementOwnership.visibility = #public)->includes(r.participant) or
d.supplier.oclAsType(GeneralizableElement).
allParents.oclAsType(Namespace).ownedElement->select(e |
e.elementOwnership.visibility = #public)->includes(r.participant) or
d.supplier.oclAsType(Package).allImportedElements->select(e |
e.elementImport.visibility = #public)->includes(r.participant)))
```

Apart from the specification form, we notice that the classifiers inherited were taken into consideration twice. These because the association's Namespace has to be a Package and, as we mentioned in the previous section, in the Package, the allContents operation had been redefined in order to include the inherited elements.

```
allContents = self.contents->union(self.parent.allContents->select(e |
e.elementOwnership.visibility = #public or e.elementOwnership.visibility =
#protected))
```

More, the Permission relationships having the stereotype 'friend' and 'import' have not been taken into consideration. In this case, our proposal is to define in the Package context an allVisibleElements AO in order to return all the elements able to be used as servers in that Package. This AO will be useful to check all the

relationships defined in that Package and to calculate the potential servers for the features of Classifiers defined in this Package.

```
context Package::allVisibleElements():Set(ModelElement)
post:
let clDepSuplElem(d: Dependency): Set(ModelElement)= d.supplier->asSequence
->first.oclAsType(Package).ownedElement
let clDepStName(d: Dependency):String = d.stereotype->asSequence->first
.name in
allVisibleElements() = self.allContents->union(self.clientDependency->
select(oclIsKindOf(Permission))->iterate(cD ; acc:Set(ModelElement)=Set{} |
    if (clDepStName(cD)='import' or (clDepStName(cD)='access'))
        then acc->union(clDepSuplElem(cD)->select(e | e.oclIsTypeOf(Classifier)

            and e.elementOwnership.visibility=#public))
        else if (clDepStName(cD)='friend')
            then acc->union(clDepSuplElem(cD)->select(oclIsTypeOf(Classifier)))
            else acc->union(Set{})
        endif
    endif))
```

In this case, the Association WFR[4] will be:

```
context Association
inv WFR_4:
(self.namespace.allVisibleElements.oclAsType(Classifier)->asSet)
->includesAll(self.connection->iterate(ae ; acc:Set(Classifier)=Set{} |
acc->including(ae.participant)))
```

In order to support our proposal, we will analyze now, the BehavioralFeature WFR[2].

"[2] The type of the Parameters should be included in the Namespace of the Classifier."

```
self.parameter->forAll(p | self.owner.namespace.allContents->includes
(p.type))
```

It is very clear that in this case, the classifiers visible by means of relationships declared among the classifier's package and other packages were not be tacked into consideration. The above proposed allVisibleElements AO is proving to be useful in this context also.

Another aspect worth to be analyzed is the Class WFR[2].

"A Class can only contain Classes, Associations, Generalizations, UseCases, Constraints, Dependencies, Collaborations, DataTypes, and Interfaces as a Namespace."

```
context Class
inv WFR_2:
self.allContents->forAll->(c | c.oclIsKindOf(Class) or c.oclIsKindOf(Association)
or c.oclIsKindOf(Generalization) or
```

```
c.oclIsKindOf(UseCase) or c.oclIsKindOf(Constraint) or
c.oclIsKindOf(Dependency) or c.oclIsKindOf(Collaboration) or
c.oclIsKindOf(DataType) or c.oclIsKindOf(Interface))
```

Concerning this WFR there are at lest two observations to do. Firstly, supposing that this WFR really has a role, the invariant is incomplete because at least the Realisation relationships between an interface and a Classifier, the Derivation relationships and the AssociationClasses were forgotten.

Due to the above mentioned lacks (errors) if we will evaluate this WFR[2], for classes included in simple UML models, like the model presented in Figure 2, the evaluation result will be fail as you can see in Figure 3.

We suppose that the followings WFRs are straightly connected with the above mentioned WFR. "[1] A Component may only contain other Components in its namespace."

```
context Component
inv WFR_1:
self.allContents->forAll(c | c.oclIsKindOf(Component))
```

"[2] A DataType cannot contain any other ModelElements."

```
context DataType
inv WFR_1:
self.allContents->isEmpty
```

Taking into account that both Component and DataType are Classifier's descendants the evaluation of these WFR will fail even for the simplest UML models. Analyzing the last three WFR, the following question arises: "What is the role, of these invariants?" In order to eliminate this kind of unpleasant situations, we suggest that in similar cases to describe (using the English language) the meaning of each WFR.

The lasts WFR we will analyze in this paper, are the WFR[4] and WFR[5] defined in the Classifier context.

"[4] The name of an Attribute may not be the same as the name of an opposite AssociationEnd or a ModelElement contained in the Classifier."

```
context Classifier
inv WFR_4:
self.feature->select(a | a.oclIsKindOf (Attribute))->forAll(a |
not self.allOppositeAssociationEnds->union(self.allContents)->collect(q | q.name)
->includes(a.name))
```

"[5] The name of an opposite AssociationEnd may not be the same as the name of an Attribute or a ModelElement contained in the Classifier."

```
context Classifier
inv WFR_5:
self.oppositeAssociationEnds->forAll(o | not self.allAttributes->
union(self.allContents)->collect(q | q.name)->includes(o.name))
```

Even a brief inspection shows us an abusive use of allContents AO. Our statement is based on the fact the WFR do not have to forbid legal situations accepted in object-oriented programming languages like those showed in Figures 4 & 5. More, the last two WFRs, offer us the possibility to highlight the importance of an explicit rule for naming features in UML. This rules have to state explicit that in descendants, the features names are formed prefixing the name feature with "NamespaceName::" for example, if we define in the class A an attribute named "a1", in the class B, this attribute will be called "A::a1". We supposed that the class A was defined in the same package with the class B. If the class A was defined in another package P2, in the class B, the attribute a1 inherited from the class A will be called "P1::A::a1". If in the class B, there are not other attributes named a1, than the attribute defined in the class A can receive the alias a1. Taking into account that a possible name conflict among the attributes defined in a class and the associationEnds attached to the associations connecting this class, can appear only in the execution model (obtained by translating in code the UML design model), the two above mentioned WFR, can be joined in the following WFR:

context Classifier
inv WFR_4:
self.feature->select(a | a.oclIsKindOf (Attribute))->forAll(a |
not self.oppositeAssociationEnds.name->includes(a.name))

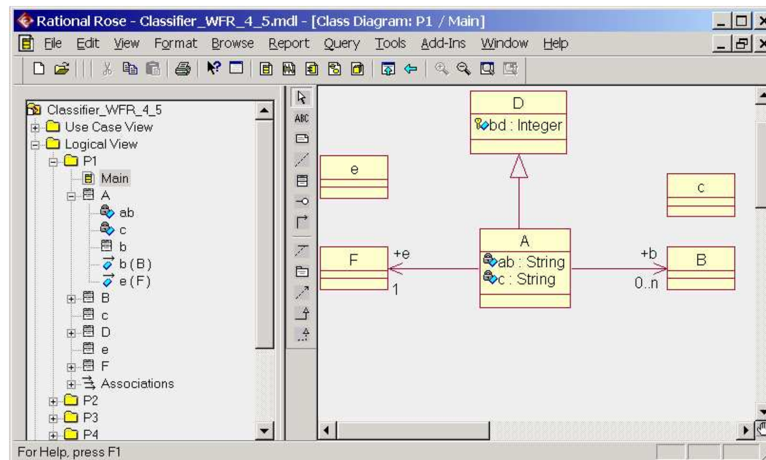

FIGURE 1. The UML model

In order to illustrate the above-mentioned statements, we will evaluate the Classifier WFR [4] and [5]: "The name of an Attribute may not be the same as the name

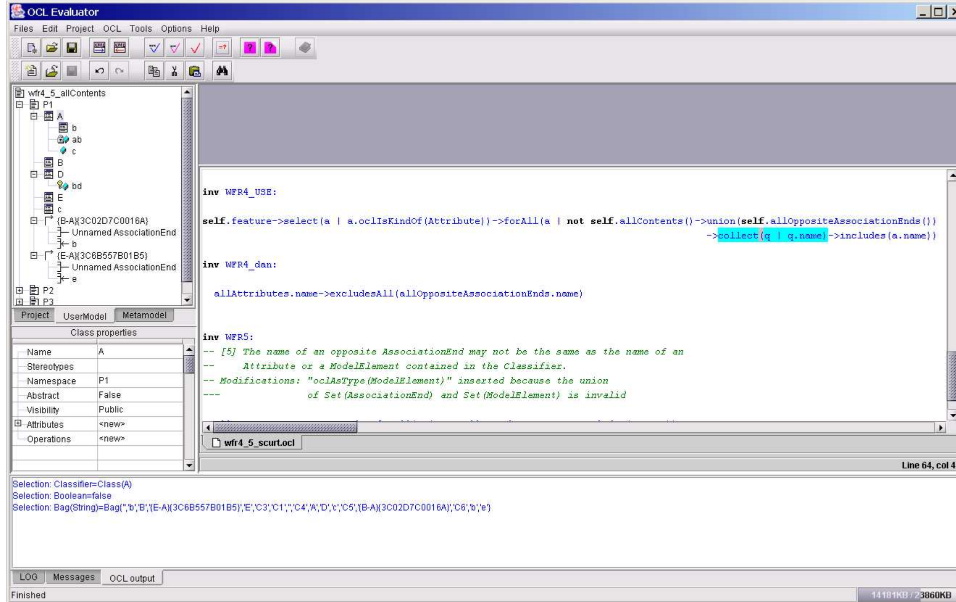

FIGURE 2. WFR_4 Evaluation in the context of class A

of an opposite AssociationEnd (or a ModelElement contained in the Classifier)[1]" using two different formalisms: Rose Script language and OCL.

This because the above-mentioned WFR can be expressed using the Rational Rose Script Language (see next section). If we try to specify (using the Rose Script Language) the Stereotype, Interface, Component, Collaboration WFR, this will not be possible due to the fact that the REI[2] dose not offer access to their required information.

In the case of "access violation", the situation is somewhat similar. As mentioned in [Moore2000]: "An access violation occurs when a class in one package references a class in another package in the absence of an import relationship between the two packages. An access violation will also occur when a package references a class from another package, whose export control is set to Protected, Private, or Implementation. In such cases, the presence of an import relationship between the two packages has no bearing. All references to implementation classes from different packages are sited as violations. Rose provides a very nice GUI for these kinds of inconsistencies – Report: Show Access Violations shows a

[1]The brackets are used to highlight that the inclusion of ModelElements different from attributes is incorrect, as we shall prove in Section 5

[2]Acronym for Rational Rose Extensibility Interface

list of violations created by generalizations, "realize" relationships, dependencies, and "instantiates" relationships. **But it does not show violations created by association classes or types**."

In order to explain the above-mentioned Rose function (menu) limits, let us consider two different packages P1 and P2, placed at the same level in the model. Let's define class A in package P1, and class B in package P2. Set the visibility of A in package P1 as being private and assume that B uses A. Irrespective of the existence of a dependency relationship between P2 and P1, the function "Report Access Violation" doesn't mention this error. Moreover, all the WFR that use the Permission relationship cannot be expressed in Rose Script Language because Rose does not implement this UML relation.

Consequently, the Rose Script Language supports evaluation only of some WFR. Unfortunately even in cases when the WFR specification can be implemented in the script language, it is inefficient as compared to its OCL specification.

6. WFR evaluation using Rose Script Language

The OCL Tools mentioned at `http://www.klasse.nl/ocl`, offer a little part of the functionalities needed to check UML Models. Consequently, in UML CASE tools, checks are usually done using different script languages. The model information accessible by means of script languages is arbitrarilydetermined by each tool provider.

The typical example is offered by Rational Rose, the UML most widely used and known CASE Tool. Consequently, in order to compare the script language support against OCL language support in checking UML models, we wrote the following script, used to check the UML Classifier WFR [4&5] in Rose.

```
'--------------------------------------------------------------------------
'Classifier_WFR_[4&5]
'Verifies the above Well-Formedness Rule
'Description:
'[4] The Name of an Attribute may Not be the same As the Name of an opposite
'AssociationEnd.
'OCL Expression:
'    allAttributes.name->excludesAll(allOppositeAssociationEnds.name)
'--------------------------------------------------------------------------
Sub AllNonPrivateAttributes(theClass As Class,
                            ByRef resultAttrs As AttributeCollection)
        'add the non private attributes of the current class
        Dim attrs As New AttributeCollection
        Set attrs = theClass.Attributes
        For i = 1 To attrs.Count
                If (attrs.GetAt(i).ExportControl <> rsPrivateAccess) And
                   (Not resultAttrs.Exists(attrs.GetAt(i)))
                    Then resultAttrs.add attrs.GetAt(i)
```

```
                End If
        Next i
        'now the attributes of the superclasses
        Dim sc As ClassCollection
        Set sc = theClass.GetSuperclasses
        If sc.Count > 0 Then
                For i = 1 To sc.Count
                        Call AllNonPrivateAttributes(sc.GetAt(i), resultAttrs)
                Next i
        End If
End Sub

Sub AllOppositeAssociationEnds(theClass As Class,
                               ByRef resultAssocEnds As RoleCollection)
        'add the opposite association ends of the current class
        Dim assocEnds As New AssociationCollection
        Set assocEnds = theClass.GetAssociations
        For i = 1 To assocEnds.Count
          If Not
    resultAssocEnds.Exists(assocEnds.GetAt(i).GetOtherRole(theClass))
  Then resultAssocEnds.add assocEnds.GetAt(i).GetOtherRole(theClass)
          End If
        Next i
        'now the opposite association ends of the superclasses
        Dim sc As ClassCollection
        Set sc = theClass.GetSuperclasses
        If sc.Count > 0 Then
                For i = 1 To sc.Count
                  Call AllOppositeAssociationEnds(sc.GetAt(i), resultAssocEnds)
                Next i
        End If
End Sub

Sub AllAttributes(theClass As Class, ByRef resultAttrs As AttributeCollection)
        Dim theAttrs As AttributeCollection
        'get all non private attributes of this class and it's superclasses
        Call AllNonPrivateAttributes(theClass, resultAttrs)
        Set theAttrs = theClass.Attributes
        'add the private attributes of the current class
        For i = 1 To theAttrs.Count
                If (theAttrs.GetAt(i).ExportControl = rsPrivateAccess) Then
                    resultAttrs.add theAttrs.GetAt(i)
                Next i
End Sub
```

```
Function checkWFR(theClass As Class) As Boolean
        Print "Class: " & theClass.Name
        'Get allAttributes
        Dim allAttrs As New AttributeCollection
        Call AllAttributes(theClass, allAttrs)
        Print "Attributes: "
        'print allAttributes - additional operation result;
        For i = 1 To allAttrs.Count
                Print ,allAttrs.GetAt(i).name
        Next i
        'Get allOppositeAssociationEnds
        Dim allOpEnds As New RoleCollection
        Call AllOppositeAssociationEnds(theClass, allOpEnds)
        'print allOppositeAssociationEnds - additional operation result;
        Print "Association Ends: "
        For i = 1 To allOpEnds.Count
                Print ,allOpEnds.GetAt(i).name
        Next i
  'compare the names of the attributes and of the opposite association ends
        Print
        For i = 1 To allAttrs.Count
                For j = 1 To allOpEnds.Count
                        If allAttrs.GetAt(i).name = allOpEnds.GetAt(j).name Then
                                Print "WFR[4&5] Failed !"
                                Print ,"Attribute '" &  allAttrs.GetAt(i).name &
                                       "' = Association End '" &
                                       allOpEnds.GetAt(j).name & "'"
                                checkWFR = false
                                Exit Function
                        End If
                Next j
        Next i
        Print "WFR[4&5] Passed ! "
        checkWFR = True
End Function

Sub Main
        Dim selectedClasses As ClassCollection
        Set selectedClasses = RoseApp.Currentmodel.GetSelectedClasses()
        'check the rule against the selected classes
        If selectedClasses.Count = 0 Then
                MsgBox "No classes selected!"
                Exit Sub
        End If
        'open the viewport and print the results
```

```
        Viewport.Open
        For i = 1 To selectedClasses.Count
                Print "--------------------------------------------------"
                Dim aClass As Class
                Dim ruleResult As Boolean
                Set aClass = selectedClasses.GetAt(i)
                ruleResult = checkWFR(aClass)
                Print "--------------------------------------------------"
                Print
        Next i
End Sub
```

Evaluating the above script, for the UML Model presented in Figure 2, when class A is selected, we will obtain:

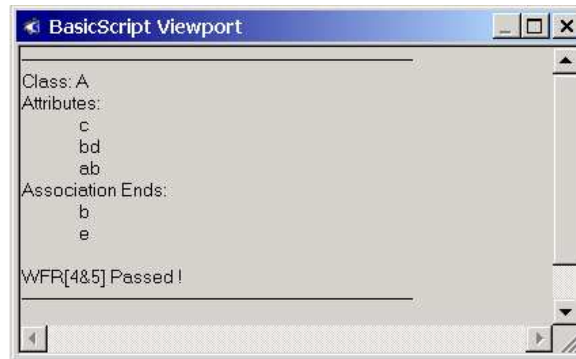

FIGURE 3. Script Evaluation Result

As we can notice, the result is correct. Even if we take into account just the "Function checkWFR(theClass As Class) As Boolean" (considering that the information furnished by subroutines is offered by REI), we notice that the WFR OCL specification (see Section 5) is by far more compact and clear than the above WFR script code.

As mentioned in the previous section, using script languages has some drawbacks. First of all, we cannot express all the WFR specified in UML. More, REI doesn't offer all the information accessible by means of AO. Even in cases when the specification is possible by means of script language, this proved to be cumbersome.

Of course, the WFR semantic correctness represents the precondition which needs to be fulfilled in order to do the above-mentioned checks. This "precondition" cannot be evaluated in the absence of adequate OCL Tool support. This doesn't have to be restrained to the syntactic and semantic WFR checking. It is

mandatory that the OCL tools support the full evaluation of the OCL expressions expressed at the metamodel level. In the next section, we will present the results obtained using our OCL evaluator.

7. WFR evaluation using OCL

In the NEPTUNE IST 1999-20017 Research Project framework, we designed and implemented a tool having as first goal UML model checking. In order to support the tool's independence with respect to UML CASE tools, two main decisions were taken: to use the exchange format for UML models (XMI) and OCL as the rule language. The tool takes the UML models saved in XMI format and checks their correctness against WFR. Compared with USE tool and experience presented in [Richters2000] our tool supports complete UML model checking, enabling the user to take into account all the WFR. (The WFR and AO semantic errors reported in [Richters2000] are type-checking errors). The errors discussed below are "design" errors.

In order to provide the opportunity to do some comparisons between the Rose Script Language and OCL, we will analyze the Classifier WFR [4] and [5].

[4] The name of an Attribute may not be the same as the name of an opposite AssociationEnd or a ModelElement contained in the Classifier.

```
self.feature->select(a | a.oclIsKindOf(Attribute))->forAll(a |
not self.allOppositeAssociationEnds->union(self.allContents)->collect(q | q.name)
->includes(a.name))
```

[5] The name of an opposite AssociationEnd may not be the same as the name of an Attribute or a ModelElement contained in the Classifier.

```
self.oppositeAssociationEnds->forAll(o | not self.allAttributes->
union (self.allContents)->collect(q | q.name )->includes(o.name))
```

A simple analysis of these two rules gives us the opportunity to notice their similarity. From the informal point of view, in the description made in natural language only the phrase topic is changed. Concerning the OCL specification, in [4] only the attributes defined in the Classifier are taken into account. On the contrary, in [5] the attributes defined in the ancestors are also included. Our opinion is that this second solution is correct. However some corrections have to be made.

First of all, both in WFR[4] and [5], as mentioned in [Richters2000], we notice that from the type checker point of view, the expression self.allAttributes->union(self.allContents) is erroneous because we try to join a Set(Attributes) with a Set(ModelElements). In order to solve this semantic error, the solution proposed at `http://www.db.informatik.uni-bremen.de/∼mr` is to change the order of operation. A more careful analysis of the above WFR shows us that the OCL expressions contain redundancies. The expression:

```
forAll(a | not self.allContents->union(self.allAttributes)->collect(q | q.name)->
includes(a.name))
```

is equivalent with:

forAll(a | **not** allAttributes.name->union(allContents.name) ->includes(a.name)).

Now the WFR is semantically correct and can be evaluated. For the UML model presented in Figure 2, the evaluation will fail (see Figure 3). Analyzing Figure 3, we find the reason. The Boolean attribute b, declared in class C1, is included in the set allAttributes, computed for class A, but the visibility of b is private, so it shouldn't be included in that set. As we can see, in Figure 4, this mistake was discarded; removing all private attributes defined in the classifier's ancestors. In fact the correct strategy is to do this correction in the Classifier's allFeatures() AO. Taking into account its usage, this represents a very important correction in the AOs. The change will be automatically propagated in allAttributes() and allOperations().

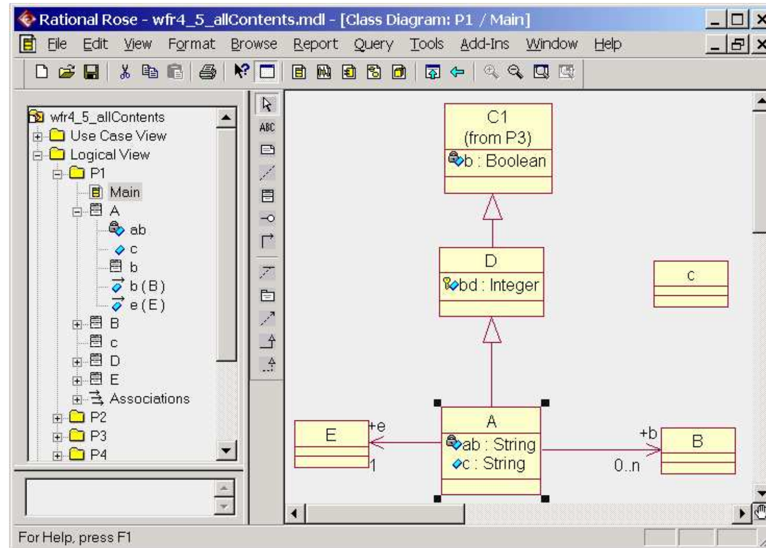

FIGURE 4. Another UML model

Solving the above-mentioned problem, another error will occur. This new error is determined by the reunion allAttributes.name->union(allContents.name). As you can see, the set allContents.name includes 'b', the name of the inner class declared in class A(see Figure 2 – the Rational Rose browser). As a consequence, the WFR evaluation will fail due to the name conflict between the associationEnd b and the inner class b. The above reunion is erroneous because it forbids modeling legal situations found in programming languages such as Java, C++, etc. Moreover, oppositeAssociationEnds has to be replaced by allOppositeAssociationEnds because,

due to inheritance, the associationEnds declared in the classifier's ancestors are accessible.

Taking into account all the above suggested corrections, we can now write the Classifier's WFR[4]:

allOppositeAssociationEnds.name->excludesAll(allAttributes.name).

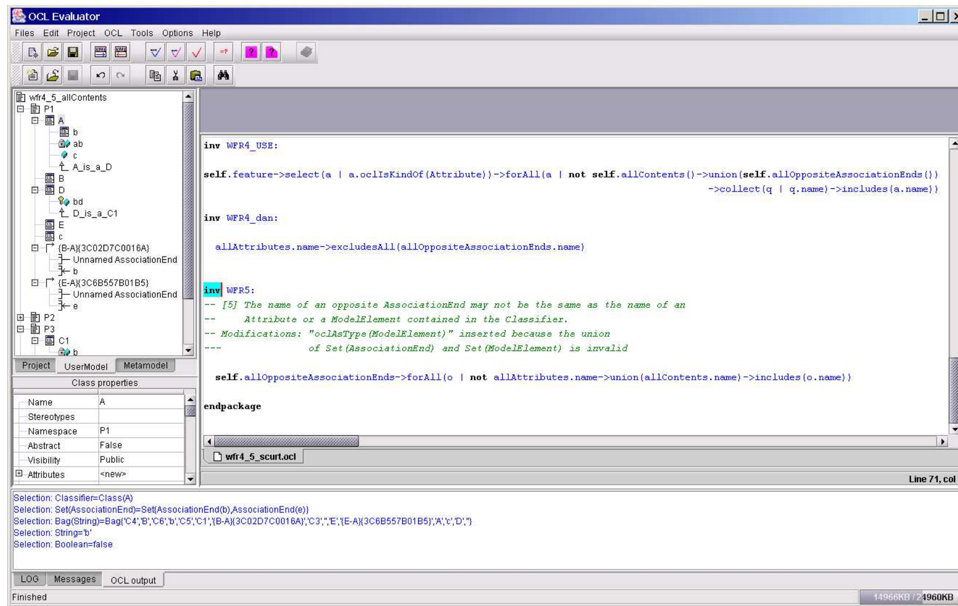

FIGURE 5. WFR [5] Evaluation

This last proposed specification covers equally the initial Classifier's WFR[4] and WFR[5]. It is even more concise and clear as compared to the textual specification and can be directly evaluated; a big advantage.

In order to understand the use of our OCL evaluator, we mention that the UML 1.4 metamodel is automatically loaded every time when the evaluator is launched. The next mandatory activities are the loading of the OCL file and the UML user model. Their order does not matter. The third mandatory activity is the OCL file (expression) compilation, followed by the context specification. In Figures 3 and 4, the first two lines of the output pane mark successful compilation. In the third line, successful loading of the UML model is signaled. Finally in the fourth line, the context specification is noticed. In both the above-mentioned figures, the last output pane line shows the results of the invariants evaluation. In Figure 3 the lines 5-8 illustrate the evaluation of oppositeAssociationEnds, allAttributes.name, allContents.name allAttributes.name->union(allContents.name), respectively. In Figure 4,

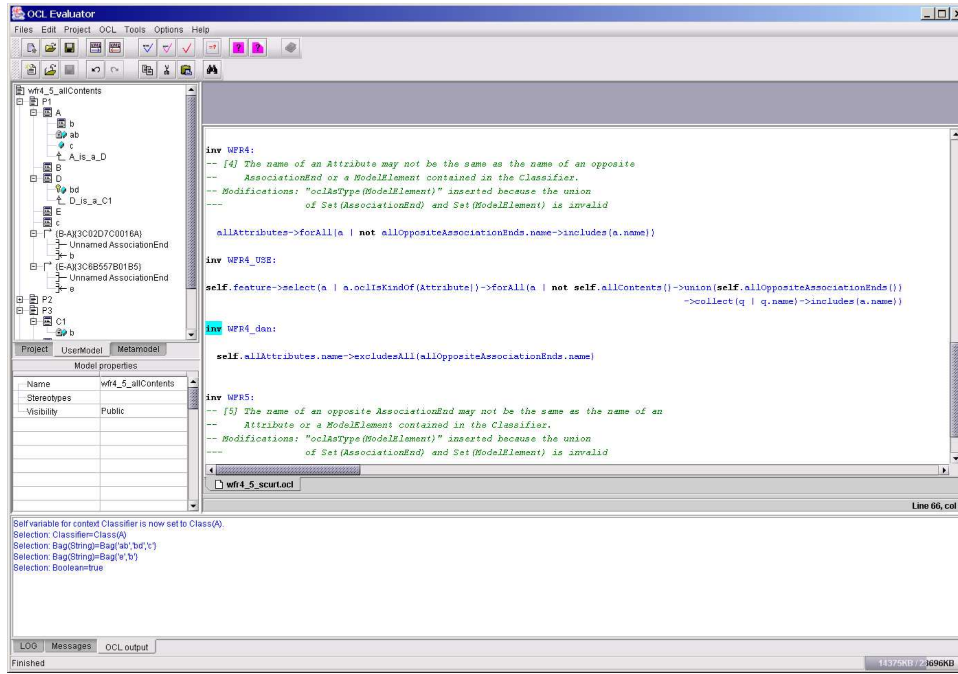

FIGURE 6. Proposed WFR [4&5] Evaluation

the lines 5-6 show the evaluation result for: allOppositeAssociationEnds.name and allAttributes.name. The possibility to evaluate different specification "chunks" is very useful in debugging OCL expressions.

8. Conclusions and future work

In this paper we tried to analyze the state of the art in the UML model-checking domain. Our research demonstrated that today, for this activity, the appropriate formalism should be by far OCL. Using script languages is also possible. As we showed, this last formalism has different drawbacks. The experience we acquired in the NEPTUNE Research Project showed that in order to be efficient, the OCL tools have to support the evaluation of expressions. The syntactic and semantic errors like those mentioned in other papers are not sufficient. Moreover, the access to the UML metamodel is mandatory. Evaluating UML 1.4 AO and WFR gives us the opportunity to find a lot of conceptual errors. The space of this paper does not allow us to mention all of them. Equally important, we identified redundant OCL specifications and some WFR not yet mentioned in the UML specification. Specifying AO and WFR has many solutions. It is very important to find the

simplest and clearest ones. In order to check UML models, the AO and WFR completeness and correctness represent a precondition. Consequently, our results offer support for the accomplishment of this “precondition”. At least, as far as we are concerned, we don’t know similar results published in other scientific papers. We are now trying to identify all the errors in the AO and the WFR and to describe solutions for all of them.

References

[Moors2000] Michael Moors, Consistency Checking – Rose Architect, Spring Issue, April 2000, http://www.therationaledge.com/rosearchitect/mag/ index.html

[Richters 2000] Mark Richters, Martin Gogolla, Validating UML Models and OCL Constraints, “Proc. 3rd International Conference on the Unified Modeling Language (UML)”, Springer-Verlag, 2000

[Chiorean01] Dan Chiorean, Using OCL beyond specification, Lecture Notes in Informatics 7, “Practical UML-Based Rigorous Development Methods – Countering or Integrating the eXtremists, 2001, pag. 57-69

[UML 1.4] UML 1.4 Draft Specification, February 2001, `http://uml.sh.com`

[Warmer1999] Warmer J, Kleppe A., “The Object Constraint Language”, Addison Wesley, 1999

[DresdenOCL] `http://dresden-ocl.sourceforge.net/index.html`

Babeş-Bolyai University, Computer Science Research Laboratory, RO 3400 Cluj-Napoca, Str. Kogălniceanu 1, Romania

E-mail address: chiorean@cs.ubbcluj.ro