Electronic Communications of the EASST
Volume 9 (2008)

Proceedings of the Workshop Ocl4All: Modeling Systems with OCL at MoDELS 2007

Using OCL in Executable UML

Ke Jiang, Lei Zhang, Shigeru Miyake

11 Pages

Guest Editors: David H. Akehurst, Martin Gogolla, Steffen Zschaler
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Using OCL in Executable UML

Ke Jiang, Lei Zhang, Shigeru Miyake

Hitachi (China) Research & Development Corporation
301, Tower C Raycom infotech Park, 2 Kexueyuan Nanlu, Hai Dian District,
Beijing 100080, China

Abstract: Executable UML allows precisely describing the software system at a higher level of abstraction. The executable models can be translated to a less abstract programming language completely or executed directly. Object Constraint Language (OCL), as a formal specification language, is a standard published along with UML. It is primitively used to describe constraints for UML models. In this paper, we explore some general features of executable UML and propose using OCL in executable UML. We extend OCL to support actions with side-effect in order to precisely model behavior. We also discuss the some open issues.

Keywords: OCL, executable UML

1 Introduction

Unified Modeling Language (UML) [1] was originally designed to sketch the structure of object-oriented software systems. But UML is not precise enough for model execution. Executable UML [2] bridges the gap between the UML-based design models and the implementation. Executable UML allows user to precisely describe a software system. The executable models can be compiled or translated to a less abstract programming language, which can be deployed on various platforms for specific implementation. Executable UML also allow directly executing UML models. With executable UML, we can clearly grasp what the system is doing early, enable testing of the system as the system is built, and find the flaws in the design which can be immediately corrected, rather than later when it is too costly to correct them. OCL is a formal language used to describe expressions on UML models [3]. It rises to meet part of the requirements for constraint, query and checking of UML models. However, OCL is still not sufficient for model execution, since it is designed to be a specification language and is side-effect free.

In this paper, we explore some general features of executable UML and propose using OCL in executable UML. We extend OCL to support actions with side effects in order to model behavior. We also discuss the some open issues in practice.

2 Executable UML

Executable UML is at the next higher layer of abstraction of the problem space based on the object-oriented programming language. It provides an evolutionary model-driven solution to express software. Rather than elaborate an analysis product into a design product and then write code, application developers of executable UML will use tools to translate abstract application constructs into executable entities. With executable UML, what the developers do is just modeling, the automatically transformation from the models to the implementation is

supported by the executable UML tools. The code generated from an executable UML model will be as uninteresting and typically unexamined as the assembler pass of a third generation language compile is today. The Developers only care about the models, so the gap between the design and the implementation is eliminated.

An executable UML should firstly define a compact subset of UML that comprises a computationally complete language for executable models. For semantics aspect, an Executable UML should have sufficiently precise, unambiguous, well-grounded execution semantics. The executable models can be executed given a runtime environment, which also means that they can be validated early in the development lifecycle, as well as be translated to target code achieving near 100% code generation. This results in the import of Action Semantics Language (ASL), which is used for precisely describing the conceptualization and behavior. It concerns about the algorithm and deals with "how to do". Secondly, an executable UML should explicitly state where it refines the semantics for the constructs in the foundational subset as defined in the UML. Besides, some new constructs may be defined by using a profile of UML or extending the UML meta-model, together with their syntax and semantics. Moreover, an executable UML should have a standard model library, which contains basic data type and basic computational functions such as basic arithmetic, comparison functions, logical functions, and I/O functions. Finally, it should be independent of software organization and be translatable to multiple implementations and a broad range of languages.

Existing executable UMLs can be divided into two kinds. The first kind of executable UML defines an Object Management Group (OMG) action-semantics-compliant language for well-defined, computationally complete formalism. This kind includes executable and translatable UML (xtUML) [4] and xUML [5]. The second kind of executable UML provides action language using simply C, C++, Ada, Java or VBA code. XIS-xModels [6] and Rhapsody [7] provides such executable UML.

As OMG does not recommend a specific language so far, it results in the lack of a standard ASL. ASL with fully new syntax may somewhat reduce its usability from the user view. Other the other hand, the programming-based action languages may lead to the problem of platform dependence, which means the capability and the implementation of this kind may be constrained by a specified platform.

3 OxUML

We propose an OCL-based executable UML (OxUML). From the thirteen diagrams of UML, we choose class diagram, state machine diagram and activity diagram as the essential diagrams of OxUML. Class diagram is used to describe the static structure of the target software system. It concerns about the data. State machine diagram and activity diagram are behavior view. The state machine diagrams focus on the control as well as interaction. Beside state machine diagram, activity diagram is adopted in OxUML as a dynamical diagram to express the behavior, which is different from the prior arts.

The remarkable distinction between OxUML and the above executable UMLs is the ASL. Because of the overlap between ASL and OCL, we suggest that OCL can be partly used for ASL, and the capability of model execution can be provided by extending OCL. We define OCL for Execution (OCL4X) [8] as ASL. In OxUML, we make use of OCL's capability such as basic data type, standard lib, redefine some OCL elements for practice and provide an

extended OCL library. As a specification language, OCL can be translatable to multiple implementations and a broad range of languages. Thus, OCL can contribute a lot to executable UML.

3.1 OCL4X

OCL covers large parts of functionality in the Action Semantics (AS) of UML. For example, in UML action semantics, the action *CallOperationAction* transmits an operation call request to the target object where it may cause the invocation of associated behavior [1]. The expression *OperationCallExp* in OCL denotes invoking an operation defined in a Classifier [3]. Such actions are *SendSignalAction, ReadSelfAction, ReadStructuralFeatureAction, ReadExtentAction, ReadIsClassifiedObjectAction*. By mapping from actions defined in UML to OCL, we use the OCL syntax to express the action which has similar description in both two specifications, as shown in table 1.

For the actions which do not have similar definition in OCL, new syntax constructs are defined on the basis of standard OCL, together with their semantics. The expressions of *AssignExp, PropertyAssignExp, VariableAssignExp, CreateObjectExp, DestroyObjectExp* and *OpaqueExp* are defined in OCL4X to express the actions of *WriteAction, AddStructuralFeatureValueAction* /*ClearStructuralFeatureAction, AddVariableValueAction* /*ClearVariableAction, CreateLinkAction*/ *CreateObjectAction, DestroyLinkAction*/ *DestroyObjectAction, OpaqueAction* respectively.

Table 1. Mapping from action to OCL expression

Action	**Expression in OCL**	**Extended Expression**
AcceptCallAction	N/A	N/A
AddStructuralFeatureValueAction	N/A	AssignExp
AddVariableValueAction	N/A	AssignExp
BroadcastSignalAction	N/A	N/A
CallOperationAction	OperationCallExpCS	
ClearStructuralFeatureAction	N/A	AssignExp
ClearVariableAction	N/A	AssignExp
CreateLinkAction	N/A	CreateObjectExp
CreateObjectAction	N/A	CreateObjectExp
DestroyLinkAction	N/A	DestroyObjectExp
DestroyObjectAction	N/A	DestroyObjectExp
OpaqueAction	N/A	OpaqueExp
RaiseExceptionAction	N/A	N/A
ReadExtentAction	allInstances in Classifier	
ReadIsClassifiedObjectAction	oclIsTypeOf	
ReadLinkAction	simpleNameCS or OclExpression	
ReadSelfAction	simpleNameCS	
ReadStructuralFeatureAction	simpleNameCS or OclExpression	
ReadVariableAction	simpleNameCS	
RemoveStructuralFeatureValueAction	N/A	AssignExp
RemoveVariableValueAction	N/A	AssignExp
ReplyAction	N/A	WhileExp,ActionExp
SendSignalAction	OclMessageExpCS	
TestIdentityAction	OclAny in Standard Library	
ValueSpecificationAction	Returns in Standard Library	
WriteStructuralFeatureAction	N/A	AssignExp

3.1.1 Abstract Syntax

The abstract syntax of OCL4X resides on layer two of the OMG four-layered architecture. We extend OCL expression package by adding new syntax constructs for actions, including assignment expression, creating object expression, and deleting object expression. The expressions and their corresponding actions are shown in table 1. The inheritance relationships of OCL4X expressions from UML and OCL are shown as Fig. 1.

The elements are defined using a Meta Object Facility (MOF) [9] compliant meta-model. ModelElement is imported from UML. *OclExpression*, *PropertyCallExp* and *LoopExp* are defined in OCL. Expression *BlockExp*, *WhileExp*, *ActionExp*, *AssignExp*, *PropertyAssignExp*, *VariableAssignExp*, *OpaqueExp*, *CreateObjectExp* and *DestroyObjectExp* are newly added elements in OCL4X.

The semantics of an OCL4X expression is given by association: each value defined in the semantic domain is associated with a type defined in the abstract syntax; each evaluation is associated with an expression from the abstract syntax. The value yielded by an OCL4X expression in a given environment is the result value of its evaluation within a certain name space environment.

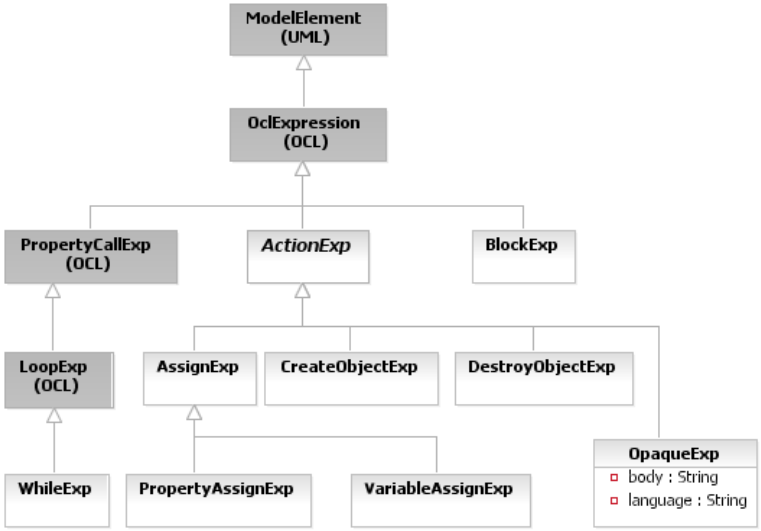

Fig. 1. Inheritance relationships of OCL4X expressions

BlockExp

A *BlockExp* is a block, which is composed of a list of *OclExpression*. A *BlockExp* contains a list of *BlockExp*. *SubOclExpression* and *subBlock* in *BlockExp* are in the sequence order. While executing the *BlockExp*, *SubOclExpression* and *subBlock* are executed according to the order. The abstract syntax is shown in Fig. 2.

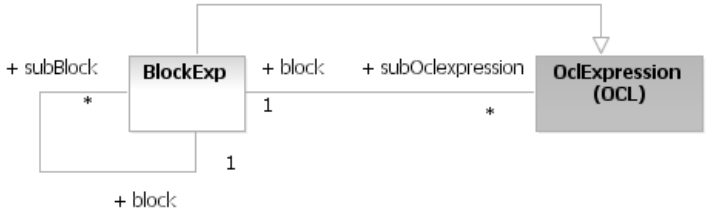

Fig. 2. Abstract syntax of *BlockExp*

WhileExp

A *WhileExp* is a specified *loopExp*, which is defined in OCL. It judges the condition result from an *OclExpression*. If the evaluation of the *OclExpression* is true, it executes the *loopBody* and judges the condition again before beginning the next loop. Otherwise, it does nothing. The abstract syntax is shown as Fig. 3.

Fig. 3. Abstract syntax of *WhileExp*

ActionExp

ActionExp is an abstract class, whose execution represents some processing in the modeled system. *AssignExp*, *CreateObjectExp*, *DeleteObjectExp* and *OpaqueExp* are inherited from it. The abstract syntax is shown as is shown as Fig. 4.

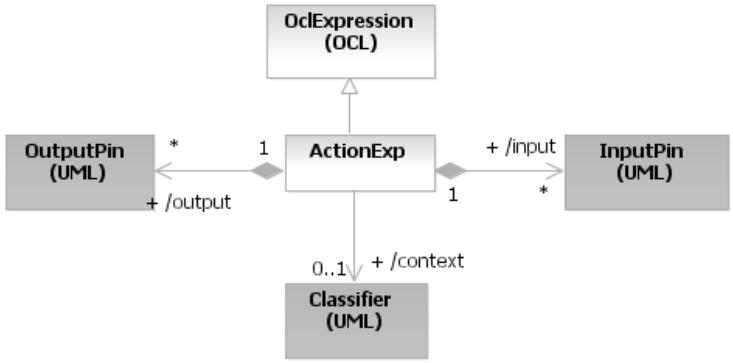

Fig. 4. Abstract syntax of *WhileExp*

AssignExp

An *AssignExp*, derived from *ActionExp*, results in an input value assignment to output value. *PropertyAssignExp* and *VariableExp*, inherited from *AssignExp*, are specified for property assignment operation and variable assignment operation. The abstract syntax is shown as Fig. 5.

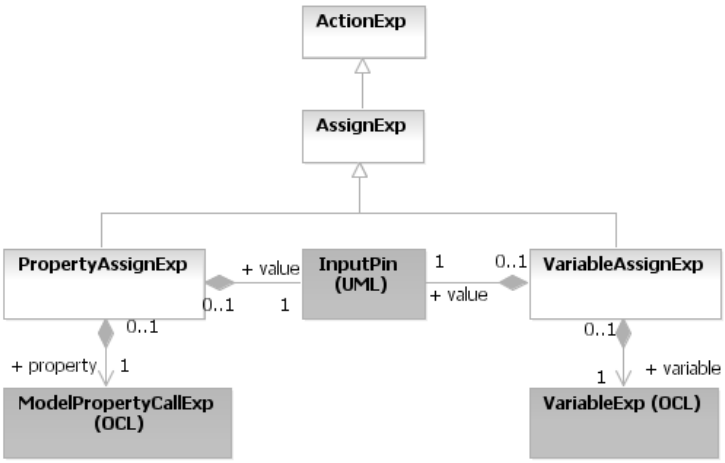

Fig. 5. Abstract syntax of *AssignExp*

PropertyAssignExp is an expression assigns the value of the *InputPin* to value of the referred property, which results from the *ModelPropertyCallExp*. It implements the function of action *AddStructuralFeatureValueAction* and *RemoveStructuralFeatureValueAction* derived from *WriteStructuralFeatureAction* in AS of UML.

VariableAssignExp assigns the value of the *InputPin* to the referred variable of a *VariablelExp*. It shares the same semantics definition as *AddVariableValueAction* and *RemoveVariableValueAction* which are derived from *WriteVariableAction*.

CreateObjectExp

CreateObjectExp implements the *CreateObjectAction* in AS in UML and provides a function for creating object. A *CreateObjectExp* with a given classifier and arguments results in a created Object expressed by *OutputPin*, as is shown in Fig. 6.

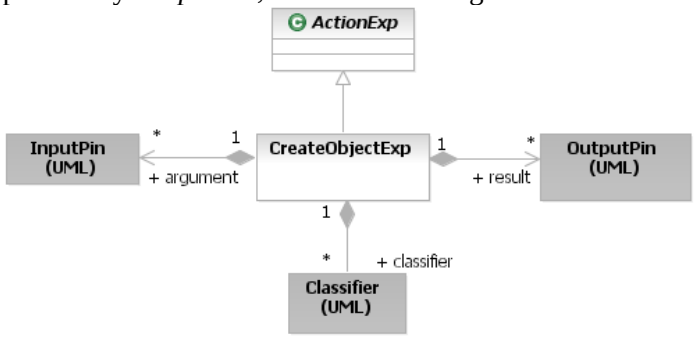

Fig. 6. Abstract syntax of *CreateObjectExp*

DestroyObjectExp

DestoryObjectExp implements the *DestroyObjectExp* in AS in UML and provides a function for destroying object. A *DestoryObjectExp* destroy the object that *InputPin* refers to. The abstract syntax definition is shown in Fig. 7.

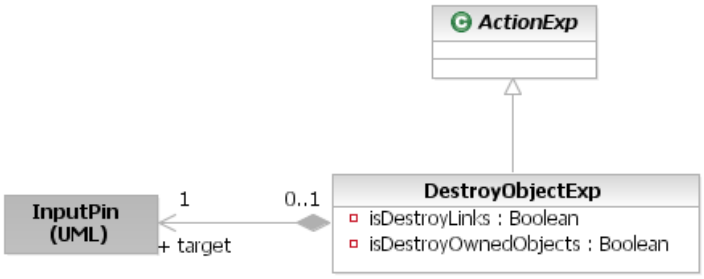

Fig. 7. Abstract syntax of *DestroyObjectExp*

OpaqueExp

An *OpaqueExp* is an operation with implementation-specific semantics, which implements the function of *OpaqueAction* in AS of UML. It executes the given body according to the given language. The abstract syntax is shown in Fig. 1.

3.1.2 Concrete Syntax

The concrete syntax definition is shown as follows.

```
BlockExpCS ::= OclExpressionCS ';' | 'begin' OclExpressionCS ';'
BlockExpCS 'end'

PropertyAssignExpCS ::=PropertyCallExpCS ':=' OclExpressionCS

VariableAssignExpCS ::= simpleNameCS ':=' OclExpressionCS

WhileExpCS ::= 'while' OclExpressionCS 'do' BlockExpCS 'endwhile'

CreateObjectExpCS ::= 'new' pathNameCS '(' argumentsCS? ')'

DestroyObjectExpCS ::= 'delete' simpleNameCS

OpaqueExpCS ::= '!script(' simpleNameCS ',' simpleNameCS ')'
```

Detailed definition is shown in appendix. The execution of OCL4X starts at the first statement in the action and as directed by control logic structures, proceeds successively through the subsequent lines and stops when the final statement is completed.

3.2 OCL in executable UML

OCL can be used to express the constraint for executable UML models as for UML. It can express the class operation, pre- and post-condition of behavior, condition guard. It can be used to express signal call and operation call with side-effect free. Here, we redefine the operation call in OCL as side-effect. That is the operation call using OCL can change the state of the system. Besides, OCL provides some standard lib and support some basic data types as well as some basic computational functions.

In the executable UML, we use OCL4X as a ASL to express the operation body in class diagram, action effect in activity diagram or behavior of state in state machine diagram. We also define an extended OCL lib. It is used to deal with some functions which are not provided by OCL standard lib and UML actions, such I/O functions, bridge API which provides interfaces to bridge different domain of the models.

4 Discussion

Even though the OCL is a first-order language, it is very much in the spirit of being executable. Beyond using OCL in specifying assertions and operations, new approaches and visions reveal beneficial usage of OCL to specify model behavior, model transformation, model compilation, and code generation.

xUMLi [10], UMLAUT [11] and the OCL4Java library of the Kent Modelling Framework (KMF) [12] combine OCL with programming language to provide the ability of model transformation and execution. The problem is that these languages may be operation system and hardware independent. Developers may be constrained by the limitations of the programming language provided. Other extensions of OCL try to extend OCL grammar for model execution, such as Action Semantics Surface Language based on OCL Queries (ASOQ) [13] and OCL4X. In [14] OCL actions are also suggested.

ASOQ embed OCL expressions in new syntax constructs for actions and give the syntax definition of the new elements. This approach keeps the interface between both languages minimal and allows keeping both languages relatively separate, not tainting OCL by introducing side effects to OCL itself. It is similar with OCL4X. But it dose not implement all the needed actions, such as *OpaqueAction*, *CreateObjectAction*, *DestroyObjectAction*, etc. In OCL4X, we map most actions to OCL constructs, and add new syntax constructs for actions that are required, but not covered by OCL. All the actions that are needed for execution are defined in OCL4X, so the language is more expressive than that of ASOQ. Besides, *OpaqueExp* in OCL4X provides some extended lib.

OCL4X is a rich textual language to describe the behavior of system. UML coupled with an executable sub-language of OCL4X will be expressive enough to describe any possible computation function.

But there are still some issues left for discussion. Question of how to access low level API provided by operating systems, and other issues of this kind will somewhat be solved by the introduction of *OpaqueExp* for domain specified operation, but it is still limited and may bring some imprecise semantics. One possible solution is to provide hand-coded modeling libraries for these special needs. Providing modeling libraries for reusable code would be useful. Moreover, exception handling, *AcceptCallAction* and *RelayAction* are not defined in OCL4X. Finally, while the OCL specification states that it is “a formal language that remains easy to read and write” [3], in fact it may be difficult to use for most modelers who are familiar with popular programming language. Although short, straight-forward expressions are very easy to understand, the clarity and readability decreases radically when the complexity (and size) grows. There are probably several factors involved. For example, the users are not as versed in the use of OCL as they are in their preferred programming languages. Finally, as there is a lack of regard of efficiency in current OCL, the implementation of OCL interpreter may be a complex work. The execution of OCL expressions in an interpreter may be inefficient. The OCL specification is still being improved. We will happy to see that action features is supported in future OCL.

5 Conclusion

Using OCL in executable UML, we provide a method to precisely describing behavior in a relative formal way. The basic idea is that using and extending OCL to express actions. OCL4X replaces ordinary programming by high-level modeling. It also enriches OCL by

providing extended lib. An implementation of OCL4X is used in the prototype of Hitachi UML Virtual Machine, which is an abstract computing machine for UML model debugging and execution.

6 References

1. OMG: UML 2.0 Super Structure Specification. http://www.omg.org/cgi-bin/doc?ptc/2004-10-02
2. Mellor, S.J., Balcer, M.J.: Executable UML: A Foundation for Model Driven Architecture. Addison Wesley Professional, 2002
3. OMG: Object Constraint Language Specification Version 2.0, 2006,http://www.omg.org/cgi-bin/apps/doc?formal/06-05-01.pdf.
4. Mentor Graphics, BridgePoint Development Suite, http://www.mentor.com/products/embedded_software/nucleus_modeling/index.cfm, 2007.
5. Kennedy Carter: Executable UML (xUML), http://www.kc.com/xuml.php, 2007.
6. Luz, M.P., da Silva, A.R.: Executing UML Models. 3rd Workshop in Software Model Engineering (WiSME 2004), IEEE Computer Society, Lisbon, Portugal, (2004)
7. Gery, E., Harel, D., Palachi, E.: Rhapsody: A Complete Life-Cycle Model-Based Development System. In: Third International Conference on Integrated Formal Methods (IFM)(2002)
8. K. Jiang, L. Zhang, and S. Miyake: OCL4X: An Action Semantics Language for UML Model Execution, In: Proceding of 31st Annual IEEE International Computer Software and Applications Conference (COMPSAC 2007), Beijing, China (2007) 633-634.
9. OMG: MetaObject Facility Specification. http://www.omg.org/docs/ptc/04-10-15.pdf.
10. Airaksinen, J., K. Koskimies, J. Koskinen, J. Peltonen, P. Selonen and T. Syst a, xUMLi: Torwards a Tool-independent UML Processing Platform, in: K. _sterbye, editor, Proceedings of the Nordic Workshop on Software Development Tools and Techniques, 10th NWPER Workshop, Copenhagen, Denmark (2002) 1-15.
11. Wai-Ming Ho, F. Pennaneac'h, N. Plouzeau, UMLAUT: a framework for weaving UML-based aspect-oriented designs, in: In Proc. of 33rd International Conference on Technology of Object-Oriented Languages, (2000) 324-334.
12. KMF, Kent modelling framework. http://www.cs.kent.ac.uk/projects/kmf/index.html.
13. S. Haustein, J. Pleumann, OCL as Expression Language in an Action Semantics Surface Language, Seventh International Conference on UML, Modeling Languages and Applications (UML 2004), OCL and Model Driven Engineering Workshop, Lisbon, Portugal, (2004)
14. D. Pollet , D. Vojtisek , J.-M. Jézéquel: OCL as a Core UML Transformation Language, Position paper, Workshop on Integration and Transformation of UML models (WITUML 2002), Malaga, Spain, (2002)

Appendix: Concrete Syntax

BlockExpCS
[A] BlockExpCS ::= OclExpressionCS ‘;’
[B] BlockExpCS::= ‘begin’ OclExpressionCS ‘;’ BlockExpCS[2] ‘end’
Abstract syntax mapping
BlockExpCS.ast: BlockExp
Synthesized attributes
[A] BlockExpCS.ast.subOclExpression = OclExpression.ast

[B] BlockExpCS.ast = BlockExpCS [2].ast->prepend(OclExpressionCS.ast)
OclExpressionCS.ast.consequent : ControlFlow
OclExpressionCS.ast.consequent.successor = BlockExpCS [2].ast->first
Inherited attributes
OclExpressionCS.env = BlockExpCS.env
Disambiguating rules
None
WhileExpCS
WhileExpCS ::= 'while' OclExpressionCS 'do' BlockExpCS 'endwhile'
Abstract syntax mapping
WhileExpCS.ast: WhileExp
Synthesized attributes
WhileExpCS.ast.condition = OclExpressionCS.ast
WhileExpCS.ast.body = BlockCS.ast
Inherited attributes
OclExpressionCS.env = WhileExpCS.env
BlockExpCS.env = WhileExpCS.env
Disambiguating rules
None
PropertyAssignExpCS
PropertyAssignExpCS ::= PropertyCallExpCS ':=' OclExpressionCS
Abstract syntax mapping
PropertyAssignExpCS.ast: PropertyAssignExp
Synthesized attributes
PropertyAssignExpCS.ast.isReplaceAll = true
PropertyAssignExpCS.ast.value = InputPin
PropertyAssignExpCS.ast.value.flow = DataFlow
PropertyAssignExpCS.ast.value.flow.source = OclExpressionCS.outputPin
Inherited attributes
PropertyCallExpCS.env = PropertyAssignExpCS.env
OclExpressionCS.env = PropertyAssignExpCS.env
Disambiguating rules
None
VariableAssignCS
VariableAssignExpCS ::= simpleNameCS ':=' OclExpressionCS
Abstract syntax mapping
VariableAssignExpCS.ast: VariableAssignExp
Synthesized attributes
VariableAssignExpCS.ast.variable = env.lookUpASVariable(simpleNameCS)
VariableAssignExpCS.ast.value = InputPin
VariableAssignExpCS.ast.value.flow = DataFlow
VariableAssignExpCS.ast.value.flow.source = OclExpressionCS.outputPin
Inherited attributes
simpleNameCS.env = VariableAssignExpCS.env
OclExpressionCS.env = VariableAssignExpCS.env
Disambiguating rules

None

CreateObjectExpCS

CreateObjectExpCS ::= ‘new’ pathNameCS ’(’ argumentsCS? ’)’

Abstract syntax mapping

CreateObjectExpCS.ast: CreateObjectExp

Synthesized attributes

CreateObjectExpCS.ast.arguments = argumentsCS.ast
CreateObjectExpCS.ast.classifier = env.lookupPathName(pathNameCS.ast).referredelement
CreateObjectExpCS.ast.result = OutputPin

Inherited attributes

argumentsCS.env = CreateObjectExpCS.env

Disambiguating rules

[1] The pathNameCS refers to a defined type in the environment.

DestroyObjectExpCS

DestroyObjectExpCS ::= ’delete’ simpleNameCS

Abstract syntax mapping

DestroyObjectExpCS.ast: DestroyObjectExp

Synthesized attributes

DestroyObjectExpCS.ast.target = env.lookup(simpleNameCS.ast) .referredelement

Inherited attributes

simpleNameCS.env = DestroyObjectExpCS.env

Disambiguating rules

[1] The simpleNameCS refers to an object.

OpaqueExpCS

OpaqueExpCS ::= ‘!script(’ simpleNameCS[1] ’,’ simpleNameCS[2] ’)’

Abstract syntax mapping

OpaqueExpCS.ast: OpaqueExp

Synthesized attributes

OpaqueExpCS.ast.body = simpelNameCS[1]
OpaqueExpCS.ast.language = simpelNameCS[2]

Inherited attributes

SimpleNameCS[1].env = OpaqueExpCS.env
SimpleNameCS[2].env = OpaqueExpCS.env

Disambiguating rules

[1] The simpleNameCS[1] and simpleNameCS[2] refer to String.