Open Distributed Processing

Specifying OCL2.0 Constraints on Interfaces of the ODP Applications

Oussama Reda* — Bouabid El Ouahidi* — Daniel Bourget **

* Université Mohammed-V Agdal
Faculté des Sciences, Département d'Informatique
B.P. 10 14 Rabat Maroc
ouahidi@fsr.ac.ma, redaoussama@gmail.com
** Telecom Bretagne Technopole
Département d'informatique CS 83818 29238 Brest
Daniel.Bourget@telecom-bretagne.eu

RÉSUMÉ. Dans ce travail, nous analysons les concepts ODP de signatures d'interfaces de traitement et les règles de typages associées; le but étant de redéfinir les signatures de manière concise et compacte. Pour cela, nous modélisons les signatures par des concepts UML équivalents. Ensuite, nous spécifieons des contraintes en OCL 2.0 sur ces signatures d'interfaces de traitement ODP liées aux règles de typages et sous-typages.

ABSTRACT. In this work, we model the ODP interaction signatures concepts in a consistent and compact manner as well as their related type checking rules. We begin by literally analysing those concepts in order to bring unambiguous definitions out of them, and we shall formalize those concepts by mapping them into UML language constructs. Then we specify constraints imposed on interfaces interaction signatures related to the computational language typing and subtyping rules. We shall show how we can literally redefine those rules in order to steadily formalize them. After rewriting those rules in a compact way, we make use of OCL 2.0 which provides the means to exploit those new definitions.

MOTS-CLÉS : ODP, point de vue traitement, UML, OCL, Meta-modélisation, signature d'interface, règles de typages

KEYWORDS : ODP, Computational Viewpoint, UML, OCL, Meta Modeling, Interaction Interface signature, Type Checking Rules

1. Introduction

The expansion of distributed processing field has led to the ODP standardization initiative [1] which consists of a framework by which distributed systems can be modelled using five viewpoints. The computational viewpoint is concerned with the description of the system as a set of objects that interact at interfaces constrained by rules, among others typing rules. Researches in [4], [5],[6], have been particularly interested in applying UML as a formal notation for the specification of the computational viewpoint, [7] has focused on how to consistently present concepts of the ODP computational viewpoint and clarified some ambiguities found while aiming to express them formally. The solutions proposed were given on a semantic level. We have [9],[10] [11] also noted those issues, then, provided solutions and presented them on a syntactic level without the need to relegate them on a semantic one, as well as specifying constraints related to computational interface signatures typing and subtyping rules. However, the OCL specifications of those rules are not easy to write, and are complicated to read and understand. This comes from the fact that OCL 1.1 doesn't provide any means in order to write easily comprehensible expressions. OCL 2.0 has known significant enhancements, especially with the provision of expressions that allow the definition and reuse of variables/operations over multiple OCL expressions. This fits well the specification of OCL constraints on typing/subtyping rules associated to interaction signatures, since those rules are redundant and have the same literal description pattern within their definitions. Our attempt is to model concepts of the ODP computational viewpoint and our main focus is the formalization of the interaction signature part as well as specification of their associated typing and subtyping rules. In this respect we use OCL 2.0 [13] to specify clear and understandable constraints.

In Section 2, we present concepts of interaction signatures concepts provided by RM-ODP. In Section 3, we discuss the literal definitions of those concepts ; the result of is a consistent UML model. We discuss in section 4 how to re-define the literal definitions of type checking rules associated to computational interfaces. In Section 5, based on those new definitions we treat the complete specification in OCL 2.0 of type checking rules. A conclusion and perspectives end the paper.

2. RM-ODP Interaction Signatures Concepts

An interface template is defined in RM-ODP as an interface template for a signal interface, a stream interface or an operation interface. Each interface has a signature : (1) A signal interface signature comprises a finite set of action templates, one for each signal type in the interface. Each action template comprises the name for the signal, the number, names and types of its parameters and an indication of causality (initiating or responding, but not both) ; (2) An operation interface signature comprises a set of announcement and

interrogation signatures as appropriate, one for each operation type in the interface, together with an indication of causality (client or server, but not both) for the interface. Each announcement signature is an action template containing the name of the invocation and the number, names and types of its parameters ; (3) Each interrogation signature comprises an action template with the following elements : the name of the invocation ; the number, names and types of its parameters, a finite, non-empty set of action templates, one for each possible termination type of the invocation, each containing the name of the termination and the number, names and types of its parameters ; (4) A stream interface comprises a finite set of action templates, one for each flow type in the stream interface. Each action template for a flow contains the name of the flow, the information type of the flow, and an indication of causality for the flow (i.e. producer or consumer but not both).

3. Re-Definition of Interaction Signatures Concepts

When trying to formalize these concepts, we have met with the issue concerning interaction signature concepts and how they are currently used and defined. In other works such as [7] discussions have focused on whether an action template concept lays on a syntactic level or a semantic one. Here, we do not take sight of these considerations as the solution we propose lies on a syntactic level. We analyze how all these concepts are linked to each others, and bring a consistent description out from their definitions.

Announcement signatures definition is clear and easy to understand when taken apart and separately from the other definitions. However, it becomes ambiguous when we shall join it to the definition of interrogation signatures. This is due to the fact that the invocation and announcement concepts are indistinguishable. Interaction signatures other than interrogations and announcement signatures are unambiguous. The new literal definition of interrogation signatures is as follows : *Each interrogation signature comprises at least two action templates which are an invocation and its corresponding termination. An invocation can possibly have more than one associated termination. invocations and terminations are action templates and they are statically described by their name and their number of parameters. Each parameter is characterized by its name and its type*. Based on their definitions, announcement signatures and interrogations signatures are two different concepts. An announcement signature is an action template ; so, it is formalized as shown in figure 1. Now, interrogation signatures do comprise action templates. invocations and terminations are also both kind of action templates ; and, since invocations and announcements describe the same concept from a practical point of view, it is preferable to merge them in one term. Thus, invocations are now absorbed by announcements, and, consequently, the announcement term present both invocation and announcement concepts (see figure 1). Interrogation signatures comprise one and only one invocation and to each invocation there is a corresponding finite non-empty set of terminations. Terminations (Parameterized Action Templates) are packed in interrogation signatures.

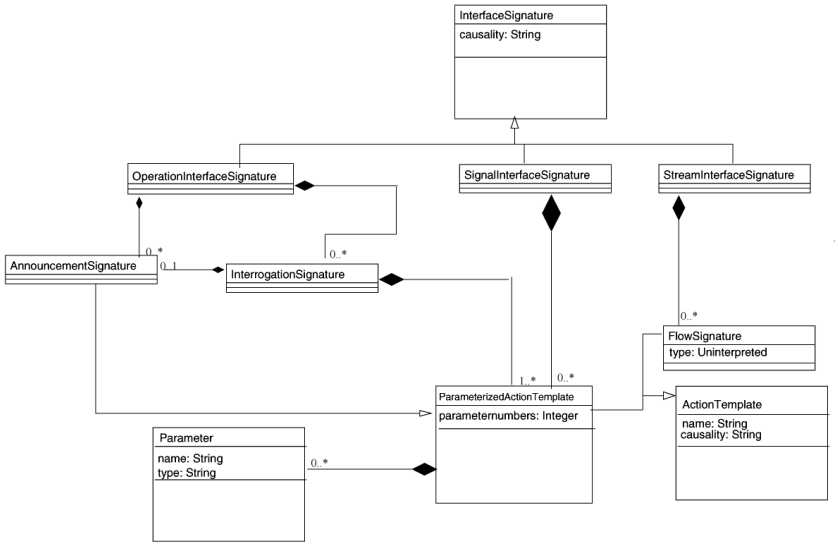

Figure 1. *Interface signatures Model*

4. OCL Typing Rules Re-Definition

In this section we specify semantics of interaction signatures related to subtyping rules. We rewrite those literal rules and present them under a new form. First, we give the rules as they are presented in the ODP computational language, and, then provide a clearer and compact description of them. We shall just concentrate on interrogation signatures as the other rules are already compact and easy to understand. Typing rules in the computational language corresponding to interrogation signatures are defined as follows. Operation interface X is a signature subtype of interface Y if the conditions below are met : (1) for every interrogation in Y, there is an interrogation signature in X which defines an interrogation with the same name ; (2) for each interrogation signature in Y, the corresponding interrogation signature in X has the same number and names of parameters ; (3) for each interrogation signature in Y, every parameter type is a subtype of the corresponding parameter type of the corresponding interrogation signature in X ; (4) the set of termination names of an interrogation signature in Y contains the set of termination names of the corresponding Interrogation signature in X ; (5) for each interrogation signature in Y, a given termination in the corresponding interrogation signature in X has the same number and names of result parameters in the termination of the same name in the interrogation signature in Y ; (6) for each interrogation signature in Y, every result type associated with a given termination in the corresponding interrogation signature in X is

a subtype of the result type (with the same name) in the termination with the same name in Y ; (7) for every announcement in Y, there is an announcement signature in X which defines an announcement with the same name ; (8) for each announcement signature in Y, the corresponding announcement signature in X has the same number and names of parameters ; (9) for each announcement signature in Y, every parameter type is a subtype of the corresponding parameter type in the corresponding announcement signature in X.

The new definitions are rewritten as follows : Operation interface X is a signature subtype of interface Y if the conditions below are met : (1) for every interrogation in Y, there is an interrogation signature X with the same name, with the same numbers and names of parameters and that each parameter in the interrogation signature in Y is a subtype of the corresponding parameter in the interrogation signature in X ; (2) for every termination in an interrogation signature in Y, there is a corresponding termination in interrogation signature X with the same name ; with the same numbers and names of parameters and that each parameter in the termination of the interrogation signature in X is a subtype of the interrogation signature in Y ; (3) or every announcement in Y, there is an announcement signature X with the same name, with the same numbers and names of parameters and that each parameter in the interrogation signature in Y is a subtype of the corresponding parameter in the interrogation signature in X.

For signal interface types that are not defined recursively, the rules are are defined as follows : Signal interface signature type X is a subtype of signal interface signature type Y if the conditions below are met : (a) For every initiating signal signature in Y there is a corresponding initiating signal signature in X with the same name, with the same number and names of parameters, and that each parameter type in X is a subtype of the corresponding parameter type in Y, (b) for every responding signal signature in X there is a corresponding responding signal signature in Y with the same name, with the same number and names of parameters, and that each parameter type in Y is a subtype of the corresponding parameter type in X.

Now, that we have reorganised the verbal description of these rules in a compact form, we realize they do share the same description pattern. Indeed, interaction signatures which are related by a Type/Subtype relation must have the same names, the same names and numbers of parameters, the latter having to satisfy a Type/Subtype relation. We can break these rules in order to bring out OCL sub-expressions which can be used in the context of all kinds of interaction signatures. We shall exploit these similarities between these definitions and come up with general formal expressions which can be used to specify OCL constraints on all interaction signatures. OCL 2.0 provides the means to realize this.

The different type checking rules related to computational interfaces contain similarities in their literal definitions. That is, all interaction signatures of all computational interfaces which are related by a Type/Subtype relation must have the same names, the same names and numbers of parameters and that parameters have to satisfy a Type/Subtype relation. Since all interaction signatures derive from the Parameterized Action Template term,

we explore this fact in order to specify those similarities mentioned above in the context of the Parameterized Action Template classifier. In what follows we give the identified OCL sub-expressions to be used in Typing/Subtyping relation specification constraints :
Context ParameterizedActionTemplate **inv :**
def : hasSameName(PAT : ParameterizedActionTemplate) : Boolean = self.name= PAT.name)
def : hasSameParametersNumber(PAT : ParameterizedActionTemplate) : Boolean = (self.parameternumbers= PAT. parameternumbers)
def : hasSameParametersNames(PAT : ParameterizedActionTemplate) : Boolean = self.Parameter → forAll(Px : Parameter | ParameterizedActionTemplate → Exists(Py : Parameter | Px.name = PAT.Py. name))
def : isSubTypeOf (PAT : ParameterizedActionTemplate) : Boolean = self.Parameter → forAll(Px : Parameter | ParameterizedActionTemplate → Exists(Py : Parameter | PAT.Py.oclIsKindOf(Px)))
def : isOfCausality(c : String) : Boolean = (self.causality=c)
Context ActionTemplate **inv :**
def : hasSameName(AT : ActionTemplate) : Boolean = self.name= AT.name)
def : isOfCausality(c : String) : Boolean = (self.causality=c)

5. OCL Typing Rules Specification

OCL Constraints On Signal Interface Signatures The literal definition of signal interface subtyping rules was given in the previous section.This constraint is described using OCL as follows :
Context SignalInterfacesignature **inv :**
SignalInterfacesignature.allInstances →forAll(X,Y |
ParameterizedActionTemplate.allInstances →forAll(PY |
ParameterizedActionTemplate.allInstances→exists(PX |
PX.isOfCausality('initite') and PY.isOfCausality('initite') and
Y.PY.hasSameName(X.PX) and Y.PY.hasSameParametersNumbers(X.PX) and
PY.hasSameParametersNames(X.PX) and X.PX.isSubTypeOf(Y.PY))))
and
ParameterizedActionTemplate.allInstances→forAll(PX |
ParameterizedActionTemplate.allInstances→exists(PY ||
PX.isOfCausality('respond') and PY.isOfCausality('respond') and
Y.PY.hasSameName(X.PX) and Y.PY.hasSameParametersNumbers(X.PX) and
Y.PY.hasSameParametersNames(X.PX) and Y.PY.isSubTypeOf(X.PX)))
implies X.oclIsKindOf(Y)

OCL Constraints On Operation Interface Signatures
The rules for Operation interface types that are not defined recursively were given in the

previous section. This constraint is described using OCL as follows :
Context OperationInterfacesignature **inv :**
OperationInterfacesignature.allInstances → forAll(X,Y |
(Interrogationsignature.allInstances → forAll (Iy |
Interrogationsignature.allInstances → exists(Ix |
Announcementsignaturee.allInstances → forAll (Ay |
Announcementsignature.allInstances → exists(Ax |
Y.Iy.Ay.hasSameName(X.Ix.Ax) and Y.Iy.Ay.hasSameParametersNumbers(X.Ix.Ax)and
Y.Iy.Ay.hasSameParametersNames(X.Ix.Ax)and X.Ix.Ax.isSubTypeOf(Y.Iy.Ay)))))))
and
(Interrogationsignature.allInstances → forAll(Iy |
Interrogationsignature.allInstances → exists(Ix |
ParameterizedActionTemplate.allInstances → forAll (Ty |
ParameterizedActionTemplate.allInstances → exists(Tx |
Y.Iy.Ty.hasSameName(X.Ix.Tx) and Y.Iy.Ty.hasSameParametersNumbers(X.Ix.Tx) and
Y.Iy.Ty.hasSameParametersNames(X.Ix.Tx) and Y.Iy.Ty.isSubTypeOf(X.Ix.Tx))))))
and
(Announcementsignature.allInstances → forAll(Ay |
Announcementsignature.allInstances → exists(Ax |
Y.Iy.Ay.hasSameName(X.Ix.Ax) and Y.Iy.Ay.hasSameParametersNumbers(X.Ix.Ax) and
Y.Iy.Ay.hasSameParametersNames(X.Ix.Ax) and X.Ix.Ax.isSubTypeOf(Y.Iy.Ay))))
implies X.oclIsKindOf(Y)

OCL Constraints on Stream Interface Signatures

Stream signature subtyping rules are defined as follows. Stream interface X is a signature subtype of stream interface Y if the conditions below are met for all flows which have identical names : If the causality is producer, the information type in X is a subtype of the information type in Y. If the causality is consumer, the information type in Y is a subtype of the information type in X. This constraint is described using OCL as follows :
Context StreamInterfacesignature **inv :**
StreamInterfacesignature.allInstances → forAll(X,Y |
(Flowsignature.allInstances → forAll(Fxp,Fyp |
Fxp.isOfCausality('produce') and Fyp.isOfCausality('produce') and
X.Fxp.hasSameName(Y.Fyp) implies X.Fxp.type.oclIsKindOf(Y.Fyp.type)))
and (Flowsignature.allInstances → forAll(Fxp,Fyp |
Fxp.isOfCausality('consume')and Fyp.isOfCausality('consume') and
X.Fxp.hasSameName(Y.Fyp) implies Y.Fyp.type.oclIsKindOf(X.Fxp.type))))
implies X.oclIsKindOf(Y)

6. Conclusion and Perspectives

In the present work, we specified OCL constraints associated to type checking rules for distributed applications. In the first main part of this work, we analysed the interaction signatures concepts. We then raised inconsistencies in their verbal description ; and finally provided an UML model of those concepts. In the second major part of the work, we specified in OCL, semantics of interaction signatures relating to subtyping rules. We showed that those literal rules provided by the ODP computational language can be aggregated in a more compact definition. We then reorganized them and gave an equivalent description in a clearer manner. Now, we have done that, our work aim to serve as a contribution within the field of applying UML as a formal notation for the specification of the ODP computational viewpoint, and we are investigating how to express computational interface signatures typing rules in terms of signal signatures typing rules

7. Bibliographie

[1] ISO/IEC , « Basic Reference Model of Open Distributed Processing-Part1 : Overview and Guide to Use », *ISO/IEC CD 10746-1*, 1994.

[4] R. ROMREO ET AL, « Modelling the ODP Computational Viewpoint with UML 2.0 », *IEEE International Enterprise Distributed Object Computing Conference*, 2005.

[5] D.H.AKEHURST ET AL, « Addressing Computational Viewpoint Design », *Seventh IEEE International EDOC, IEEE Computer Society*, 2003.

[6] BEHZAD BORDBAR ET AL , « Using UML to specify QoS constraints in ODP », *Computer Networks Journal pp.279-304*, 2002.

[7] R. ROMERO ET AL, « Action templates and causalities in the ODP computational viewpoint », *WODPEC'04 pp. 23-27*, 2004.

[8] J. RUMBAUGH AND AL, « OMG Document ptc/03-10-14 », *Addison Wesly*, 2003.

[9] O.REDA ET AL, « Interaction signatures and Action Templates in The ODP Computatinal Viewpoint », *Proc. of SEPADS'07, Greece, pp 127-131*,2007.

[10] O. REDA ET AL, « Towards Refinement of The ODP Computational Viewpoint Interaction signatures », *WSEAS Transactions On Telecommunications Journal, pp 601-606*, May 2007.

[11] O.REDA ET AL , « Specification of OCL Constraints on ODP Computational Interfaces », *Proc. of Applied Applications & Communication Conf AIC'07, Greece, pp 305-311, Aout*,2007.

[12] OMG, « UML2.0 Superstructure Specification », *OMG document formal/05-07-04*,2005.

[13] OMG, « UML 2.0 OCL Final Specification », *OMG Document ptc/03-10-14*,2003.