Specification Of OCL Constraints On ODP Computational Interfaces

OUSSAMA REDA, BOUABID EL OUAHIDI
Mohammed-V University, Faculty of Sciences
Dept of Computer Sciences
Ibn Battouta P.O Box 10 14, Rabat
MOROCCO

DANIEL BOURGET
ENST Bretagne
Dept of Computer Sciences
Technopôle Iroise - CS 83818, 29238 Brest
FRANCE

Abstract:- Open Distributed Processing systems are constructed in terms of five viewpoints. The computational viewpoint which supports three sorts of interaction models imposes constraints on their corresponding computational interfaces. Computational interfaces are strongly typed so as to sustain meaningful object interaction. We address in this work the need to re-verbalize both interaction signature concepts, and, typing rules for computational interfaces in order to steadily formalise them; while, preserving semantics of their initial definitions. This need comes from the fact that those initial definitions are ambiguous, because, they are described in natural language. Based on their new definitions, we shall present a consistent UML model for interaction signatures, as well as, OCL specification of typing rules concerning computational interfaces supporting those interactions.

Key–Words: ODP, Computational Viewpoint, UML, OCL, Meta-modelling, Interaction Interface Signature, Typing Rules

1 Introduction

The ODP standardization [1],[2],[3]), initiative has led to 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. A computational specification describes the functional decomposition of an ODP system in distribution transparent terms and is constrained by the rules of the computational language. These comprise, among others, interaction rules. Works within the computational viewpoint such as [4], [5],[6], have mainly addressed the specification of the functional decomposition of an ODP system using UML. Some of these works [7] have 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. Works such as [16] have also noted those issues, then, they provided solutions and presented them on a syntactic level without the need to relegate them on a semantic one. Other works [17] have used those solutions as a laying ground for specifying refinements on all kind of interaction signatures into signal signatures. However, none of them has ever aimed to specify constraints related to computational interface signatures typing and subtyping rules. In the same spirit of these works our attempt is to model concepts of the ODP computational viewpoint and our main focus is the formalization of concepts of the interaction signature part, as well as, specification of their associated typing and subtyping rules. Over the past years, there has been a considerable amount of research [8], [9] ,[10] within the field of applying the UML Language citeBoochUnified98, [12] as a formal notation to the ODP viewpoints, and particularly to the ODP computational viewpoint [4], [5], [6], [16],[17]. In this respect, we use the UML language to discuss and present our proposals. We also use the OCL language [13] to specify constraints associated to computational signature interfaces typing rules. The remainder of the paper is organized as follows. Section 2 presents concepts of interaction signatures provided by RM-ODP, as well as, we discuss how to construct a consistent UML model of interaction signatures. In section 3, we address the issue of of re-varbalizing the litteral description of computational interfaces typing rules. In section 4, we specify the typing rules of interaction signatures using the OCL language. A conclusion and perspectives end the paper.

2 UML Description of Interaction Signature Concepts

In this section, we present the *Interaction Signatures* concepts as they are defined in the computational viewpoint. These definitions will serve us to discuss the ideas of the rest of the paper. the definitions are given as follows:

A computational interface template is an interface template for either a signal interface, a stream interface or an operation interface. Each interface has a signature:

- 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) with respect to the object which instantiates the template.
- 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 as a whole, with respect to the object which instantiates the template. Each announcement signature is an action template containing both the name of the invocation and the number, names and types of its parameters. 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 both the name of the termination and the number, names and types of its parameters.
- 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) with respect to the object which instantiates the template.

As we look at the definition of interrogation signatures, and, since they might be interpreted in different ways, we realize that we have several options and choices to formalize them) [**?**]. Right now, let's rewrite those definitions in a clearer manner, especially for interrogation signatures. The new 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 could 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 described by its name and its type.* 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. Furthermore, the typing rules prescribed by the computational language (see section 3) never mention the *invocation* concept in their rules; letting it to be implicitly mixed up with either the *interrogation* or the announcement concept. Thus, Invocations are now absorbed by Announcements, and, consequently, the Announcement term present both invocation and Announcement concepts. The natural way they are to be formalized is as in Figure 1(see figure 1).

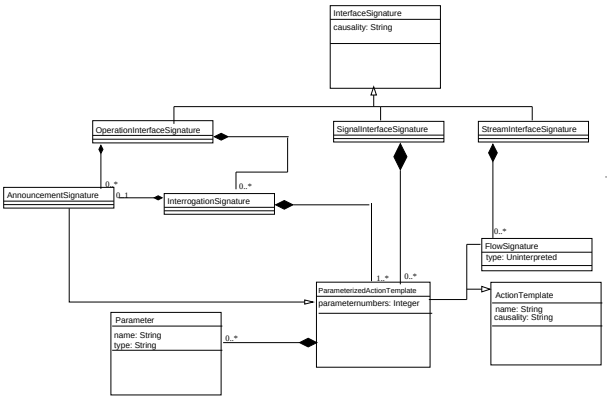

Figure 1: Computational Interface Signatures Model

Terminations are packed in an interrogation signature. Now, we can either choose to explicitly model the Termination term, or, let it be implicitly presented by Action Templates (Parameterized Action Template) term, as a Termination is an Action Template. After we just come to put an invocation into an interrogation signature, it seems obvious that its associated terminations have to be joined to it. And as we are saying this, we can see how easily terminations can go nowhere but get packed into an interrogation signature ((see figure 1). The resulting model (see figure 1)is a coherent model, based on all the pertinent choices made previously.

3 Computational Interfaces Signatures Litteral Description

Now, it's time to specify semantics of interaction signatures related to subtyping rules. As we did in the previous section, we can rewrite those literal rules and present them under a new form. This process will help us a lot when we have to formalize them in OCL as it will facilitate our task. There's plenty of dissipation in the way they actually are written in. 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 written as follows: Operation interface X is a signature subtype of interface Y if the conditions below are met:

- For every interrogation in Y, there is an interrogation signature in X (the corresponding signature in X) which defines an interrogation with the same name.
- For each interrogation signature in Y, the corresponding interrogation signature in X has the same number and names of parameters.
- For each interrogation signature in Y, every parameter type is a subtype of the corresponding parameter type of the corresponding interrogation signature in X.
- The set of termination names of an interrogation signature in Y contains the set of termination names of the corresponding interrogation signature in X.
- 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.
- 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.
- For every announcement in Y, there is an announcement signature in X (the corresponding signature in X) which defines an announcement with the same name.
- For each announcement signature in Y, the corresponding announcement signature in X has the same number and names of parameters.
- For each announcement signature in Y, every parameter type is a subtype of the corresponding parameter type in the corresponding announcement signature in X.

As we look at those literal constraints provided by the ODP computational language, we begin realizing they can be aggregated in a more compact description and especially for operation interface signature typing rules. Right now, look at the new form these definitions are rewritten in. Operation interface X is a signature subtype of interface Y if the conditions below are met:

- 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.
- 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.
- For 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.

Now, that we have reorganised the verbal description of these rules in a clearer manner, we begin to be aware they do share the same description pattern (see the definition of signal interface signature in the coming paragraph). subtyping rules for operation signatures, namely (interrogations, invocations and terminations) and signal signatures share the same formal description in OCL, and therefore, after we specify the rules relating to signal signatures in OCL expressions we can easily deduce the OCL specification of operation signatures subtyping rules with minor changes in the OCL expression body of signal signature typing rules specification.

4 Interface Signatures Typing Rules OCL Specification

Signal interface signature type X is a subtype of signal interface signature type Y if the conditions below are met:

- 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.
- 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.

This constraint is described using OCL as follows:

context SignalInterfaceSignature **inv**:
SignalInterfaceSignature.allInstances→forAll(
X,Y|ParameterizedActionTemplate.allInstances→forAll(
SyI|Y.SyI.causality=initiate
implies
ParameterizedActionTemplate.allInstances→exists(
SxI|X.SxI.causality=initiate
and
Y.SyI.name=X.SxI.name
and
Y.SyI.parametersnumber=X.SxI.parametersnumber
and
Parameter→forAll(
Px|Parameter.allInstances→exists(
Py|X.SxI.Px.name=Y.SyI.Py.name
and
Px.oclIsKindOf(Py))))))
and
(ParameterizedActionTemplate.allInstances→forAll(
SxI|SxI.causality=respond
implies
ParameterizedActionTemplate.allInstances→exists(
SyI|SyI.causality=respond
and
Y.SyI.name=X.SxI.name
and
Y.SyI.parametersnumber=X.SxI.parametersnumber
and
Parameter→forAll(
Py|Parameter→exists(
Px|X.SxI.Px.name=Y.SyI.Py.name
and
Py.oclIsKindOf(Px))))))
implies
X.oclIsKindOf(Y))

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.causality=produce
and
Fyp.causality=produce
and
X.Fxp.name=Y.Fyp.name
implies
X.Fxp.type.oclIsKindOf(Y.Fyp.type)))
and
FlowSignature.allInstances→forAll(
Fxp,Fyp | Fxp.causality=consume
and
Fyp.causality=consume
and
X.Fxp.name=Y.Fyp.name
Implies
Y.Fyp.type.oclIsKindOf(X.Fxp.type))))
implies
X.oclIsKindOf(Y))

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|AnnouncementSignatureе.allInstances→forAll(
Ay|AnnouncementSignature.allInstances→exists(
Ax|Y.Iy.Ay.name=X.Ix.Ax.name
and
Y.Iy.Ay.parametersnumber=X.Ix.Ax.parametersnumber
and
Parameter.allInstances→forAll(
Py|Parameter.allInstances→exists(
Px|X.Ix.Ax.Px.name=Y.Iy.Ay.Py.name
and
Py.oclIsKindOf(Px)))))))
and
(InterrogationSignature.allInstances→forAll(
Iy|InterrogationSignature.allInstances→exists(

```
Ix|ParameterizedActionTemplate.allInstances→forAll(
Ty|ParameterizedActionTemplate.allInstances→exists(
Tx|Y.Iy.Ty.name = X.Ix.Tx.name
and
Y.Iy.Ty.parametersnumber=X.Ix.Tx.parametersnumber
and
Parameter.allInstances→ forAll(
Py|Parameter.allInstances→exists(
Px|X.Ix.Tx.Px.name=Y.Iy.Ty.Py.name
and
Px.oclIsKindOf(Py))))))
and
(AnnouncementSignature.allInstances→forAll(
Ay|AnnouncementSignature.allInstances→exists(
Ax|Y.Ay.name=X.Ax.name
and
Y.Ay.parametersnumber=X.Ax.parametersnumber
and
Parameter.allInstances→forAll(
Py|Parameter.allInstances→exists(
Px|X.Ax.Px.name=Y.Ay.Py.name
and
Py.oclIsKindOf(Px))))))
implies
X.oclIsKindOf(Y)
```

5 Conclusion and perspectives

In our past work [22], we have proposed a UML-Based language for the QoS-aware enterprise specification of ODP systems in which we focused mainly on the specification of QoS from an enterprise viewpoint. When trying to deal with the QoS concepts within the computational viewpoint we have met with some issues as mentioned before. So, we decided to clarify some ambiguities relevant to the computational viewpoint. We have already came across those inconsistencies in other works [16]and have provided reliable solutions to those issues mainly from conceptual point of view. Then, we used those solutions in order to refine all kinds of interaction interface signatures into signal interface signature [17] which can serve as a basis to define end-to-end QoS in open distributed systems, and the operation of multi-party binding and bindings between different kinds of interfaces (e.g. stream to operation interface bindings). Here, in the current work, in addition to those solutions, which are presented in a new and elegant fashion, essentially, for practical considerations, we have also provided formal constraints relating to typing rules, something, which has never been approached actually. Now, we have done that, our work aim to serve as contribution within the field of formalizing the ODP computational viewpoint, at the same time that it helps us to move forward safely and confidently in our coming ones. We are dealing with the matter of refining computational interface typing rules into signal interface typing rules.

References:

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

[2] ISO/IEC, *RM-ODP-Part2: Descriptive Model* ISO/IEC DIS 10746-2, 1994.

[3] ISO/IEC, *RM-ODP-Part3: Perspective Model* ISO/IEC DIS 10746-3, 1994.

[4] R. Romeo 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] M.W.A. Steen and al., *Applying the UML to the ODP Enterprise Viewpoint*, http://www.cs.ukc.ac.uk/pubs/1999/819, 1999.

[9] P.F. Linington et al., *The specification and testing of conformance in ODP systems*, http://citeseer.nj.nec.com/170353.html, 1999.

[10] M. W. A. Steen et al.,*Formalising ODP Enterprise Policies*, IEEE Com. Soc. Press, EDOC'99, 1999.

[11] G. Booch et al., *The Unified Modelling Language Guide* Addison Wesly, 1998.

[12] J. Rumbaugh and al., *The Unified Modelling Language Reference Manual*, Addison Wesly, 1999.

[13] OMG, *UML2.0 OCL Final Specification*, OMG Document ptc/03-10-14, 2003.

[14] pUML group,*The Precise UML* http://www.cs.york.ac.uk/puml

[15] M. Gogolla et al., *State Diagrams in UML- A Formal Semantics Using Graph Transformation*, proceedings of ICSE'98,1998.

[16] B.El Ouahidi et al, *Interaction Signatures and Action Templates in The ODP Computatinal Viewpoint*, Proceedings of the 6th WSEAS International SEPADS'07, Corfu Greece, Feb 16-19, 2007

[17] B. El Ouahidi et al., *Towards Refinement of The ODP Computational Viewpoint Interaction Signatures*, WSEAS Transactions On Telecommunications Journal, pp 601-606, May 2007.

[18] K. Lano et al., *Formalising the UML on Structured Temporal Theories*, Conference ECOOP'98, 1998.

[19] R. Breu et al., *Systems Views and Models of UML*, Physical Verlag, 1998.

[20] A Evans et al., *Core Meta-Modelling Semantics of UML: The pUML Approach*, Proceedings of UML'99, pp 140-155, 1999

[21] J-M. Bruel et al.,*Transforming UML Models to Formal Specifications*, UML'98-Beyond The Notation, 1998.

[22] B. El Ouahidi et al., *An UML-based Metalanguage for the QoS-aware Enterprise Specification of Open Distributed System*, PROVE'02, Kluwer Academic Publishers IFIP series, pp. 255-266, 2002.