On UML Modeling of Computational Interfaces & interactions in the UML4ODP Computational Language

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: We analyze in this work computational interfaces and interactions signatures in order to consistently model them within the UML4ODP computational metamodel. Computational interfaces in the computational language are of three kinds (signals, operations and streams). We show that computational interfaces are classified in two main classes instead of three: Functional and stream interfaces. We also demonstrate that interactions are of two kinds, namely; Parameterized and flowing interactions. Then, we show that only two kinds of parameterized interactions have to be taken into account, Primitive and Compound interactions, primitives are being incoming or outgoing interactions. Based on these, we provide a UML metamodel of interfaces and interactions signatures. Finally, we show how our modeling choices prove to be pertinent to specify OCL constraints on refinements of interactions to define end-to-end QoS and bindings between computational interfaces.

Key–Words: RM-ODP, UML4ODP, Computational language, Meta-modeling, Computational interface, Interaction Signature.

1 Introduction

The ODP framework [1][2][3][4] defines a set of concepts and an architecture for the construction of ODP systems in terms of five viewpoints. The computational viewpoint supports three models of interactions, each of which has an associated kind of computational interface: signals and signal interfaces, flows and stream interfaces, operations and operation interfaces. RM-ODP is not prescriptive about the use of any particular formal description and specification techniques for the specification of ODP systems. Over the past several years, there has been a considerable amount of research [5][6][7][8][9][10][11][12][13][14][15][16][17][18][19][20] in the field of applying the UML Language as a formal notation with the ODP viewpoints, particularly for the computational language. The outcome of these works, amongst others, was the adoption of the UML4ODP FDIS (Final Draft International Standard) [21] which provides the necessary needed framework for ODP systems specification using UML 2.0 [22]. Works [14][15][17] within the computational viewpoint have mainly addressed the specification of the functional decomposition of an ODP system using UML. [16][18][19][20] have shown the UML4ODP computational metamodel contains inconsistencies concerning the semantic relationship between interaction signatures concepts and action templates, then proposed in reliable solutions. In the same perspective we analyze computational interfaces and interaction signatures concepts so as to resolve residual inconsistencies in the UML4ODP computational metamodel.

On the other hand, the second main focus of this work is refinements of interactions into atomic ones. We shall see how to refine any kind of interactions into primitives which are elementary interactions and provide OCL constraints relating to those refinements. In doing so, we are indirectly addressing fundamental QoS issues.

The remainder of the paper is organized as follows. In Section 2, we present concepts of computational interfaces and interactions signatures concepts provided by RM-ODP. We address in section 3 the problem concerning the relationship between Interaction Signatures and Action Templates. In section 4 we lead a conceptual algebraic analysis of computational interfaces and interaction signatures notions in order to steadily define these concepts. We prove interactions are either parameterized or flows, then classify parameterized interactions in primitives and compounds ones. We also define the concept of functional computational interface. The result of this analysis is the definition and introduction of new concepts to the computational
language. Section 5 deals with the introduction of complementary concepts (incoming and outgoing interactions) to those given in section 4 in order to provide the final model of computational interfaces and interaction signatures. In section 6 we discuss interaction refinements rules and provide their specification in OCL 2.0 [23] based on this discussion. A conclusion and perspectives end the paper.

2 The Data Dictionary of computational interfaces & interaction signatures concepts

In this section, we present interaction signatures concepts as they are defined in the computational language. 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.

These concepts are the necessary and sufficient ones for our proposals.

3 What is wrong with interaction signatures?

The matter with interaction signatures concepts is the difficulty of expressing operation signatures in terms of Action Templates since it is not obvious whether operation signatures are kinds of Action Templates or are constituents of Action Templates. Another issue concerning interaction signatures is the way we can describe all of them in terms of Action Templates in one blow. The problem with all this difficulty in modeling is that the definitions of the concepts are not precise and leaves room to plenty of interpretations. To eliminate this ambiguity one have to analyze the definitions on a conceptual level in order to bring out the exact semantic relationships between those concepts. In fact we show interactions are of two kinds: parameterized interactions and flows, Parameterized interactions are composed by primitive and compound interactions, primitives are being incoming and outgoing interactions.

On the other hand, interface signatures are defined in terms of three kind of computational interfaces. However when we analyze interface signatures concepts we show that in fact there are only two relevant categories they are to be classified in. We shall see how interface signatures can principally be classified in two main classes, namely: Functional interface signatures and stream interface signatures.

4 Functional interface signature & Parameterized interaction signatures

We begin by introducing the notation needed to demonstrate our propositions.

Notation:

- The symbol $\cap$ denote the intersection of algebraic sets (it has the same meaning as it is in classical set theory).

- Di, Pi and Ci are respectively the contracture of Definition i, Proposition i and corollary i, where
i is an integer related to the order of their appearance in the text.

- A\B denotes the set of elements which are in A and are not in B.
- bby is the contracture of by and only by.

Let SAinv, SAann, SAint, SAter, SAflo denote the sets of attributes that respectively describe signatures of Invocations, Announcements, Interrogations, Terminations and finally Flows.

**Definition 1:**
An Action Template is defined by the name of the action and its causality.

**Proposition 1:** All Interaction Signatures are Action Templates.

**Proof:**
We have:

SAinv = SAann = SAint = SAter = \{name, numbers of parameters, names of parameters, types of parameters, causality\} and separately SAflo=\{name, causality, information type\}.

The led set of these sets denoted SA which is their intersection SA = SAinv \cap SAann \cap SAint \cap SAter=\{name, causality\} is the set composed by and only by both the name and causality of interaction signatures. Moreover, the Action Template concept is involved in the core description of all interaction Signatures concepts, and since the UML semantic of intersection is a generalization, it follows that all Interaction Signatures are Action Templates.

**Proposition 2:** Interaction Signatures but flows are parameterized (i.e contain finite set of parameters as well as their name and numbers).

**Proof:**
The sets SAinv\SA, SAann\SA, SAint\SA, SAter\SA have the same elements since SAinv\SA=SAann\SA = SAint\SA = SAter\SA ={numbers of parameters, names of parameters, causality}. Consequently, All Interaction Signatures but Flow Signatures are parameterized (i.e described by finite sets of parameters as well as their names and numbers).

Now, when we separately take the set SAflo\SA=\{information type\} we deduce that flow signatures are of different nature than the other Interaction Signatures.

**Flow Signatures** are Action Templates with an (information type) attribute which is not significant to the other interactions. Conversely, all Interaction Signatures have parameters, their name and their numbers as attributes which do not contribute to the description of flows.

**Definition 2:**
A Parameterized interaction signature is an Action Template with a finite set of parameters as well as their numbers.

**Corollary 1:**
From P1, P2 and the definition of interface signatures given in the previous section we have:

1. Interaction signatures are of two kinds: Parameterized interactions signatures and flow interactions signatures.
2. Operation Interfaces signatures and Signal Interfaces signatures are composed bby Parameterized interaction signatures.
3. A stream interface signature is composed bby a set of flow interactions signatures.

**Definition 3:** A Functional Interface Signature is an interface signature composed bby Parameterized interactions signatures.

**Corollary 2:** From D3 and the definition of interface signatures given in the previous section we have:

Interface Signatures are of two kinds, namely: Functional Interface Signature and Stream Interface Signatures.

### 5 UML metamodel for computational Interfaces & interaction signatures

In this section we model computational interfaces and interaction signatures by means of constructs of the UML language.

Interactions in the computational language are of three kinds (signals, operations and flows). We have shown interactions are of two main kinds: Parameterized interactions and flows. Signals in the computational language are defined as being atomic interactions that constitutes the building blocks of the other kinds of interactions. Similarly, parameterized interactions are classified in two main categories: Primitive parameterized interactions (homologous of signals) and compound parameterized interactions (homologous of op-
erations). This classification is necessary to guarantee that the metamodel given (see figure 1) serves 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).

Figure 1: Functional interface signature & Parameterized interaction signatures

In the computational language operations in a computational interface consist of invocations and announcements which are outgoing interactions. To each invocation in the interface corresponds a finite non empty set of terminations which are incoming interactions. In [19] we have shown that invocations and announcements do play the same role conceptually and practically. Thus, We define Compound parameterized interaction signatures as being composed by two kinds of interactions: outgoing interactions and incoming interactions. Indeed, invocations and announcements are identical outgoing interactions. Moreover, invocations and terminations which are (terminations) incoming interactions are associated to each other by a one to many correspondance, and since announcements can be replied to or not during the interaction, we conclude there is a correspondance between outgoing and incoming interactions. That is, for every outgoing interaction corresponds a finite set (possibly empty) of incoming interactions (see figure 1).

Signals in the computational language are the least degree of representation of interactions between computational objects. Since signals do provide the constructing bricks of all other interactions, it is tempting to make use of them in order to refine interactions in their terms. To do so, the computational language imposes rules on these mappings so as to provide for reliable refinements when required. Primitive parameterized interactions do play the role of signals. While operations are represented in terms of signals, compound parameterized interactions can be decomposed in terms of primitive parameterized interactions which are now elementary parameterized interactions. This is exactly the purpose of the following section.

6 A one refinement rule for interaction signatures

In this section, we show how our unification choices in modeling computational interface signatures and interaction signatures concepts help us to specify compact OCL constraints applied on computational interface refinements.

Indeed, an operation or a flow can be resolved in terms of a composition of several individual signals. For instance, we can interpret an interrogation in terms of a sequence of four signals: invocation emission (by the client object), invocation receipt (by the server object), termination emission (by the server), termination receipt (by the client). In opposition, since the computational model do not provide the precise semantics of flows, their mapping on signals is not defined. In fact, a definition of flows using signals depends upon the details of the interactions abstracted in the specification of the stream interface concerned and therefore is beyond the scope of the ODP Reference Model [3].

In [20] we specified those constraints based on their definitions provided in RM-ODP [3] which are given as follows :

- In a signal interface corresponding to a client operation interface there is a signal -invocation submit- corresponding to each invocation with the same parameters. in the case of an interface containing interrogations, a signal - termination deliver - corresponding to each possible termination with the same parameters as that termination.

- In the signal interface corresponding to a server operation interface there is a signal -invocation deliver- corresponding to each invocation with the same parameters. in the case of an interface containing interrogations there is a signal - termination submit- corresponding to each possible termination with the same parameters as that termination.

In the definitions above the correspondance rules do neither depend on the causality of computational
interfaces nor the causality of interactions. What only matters is the existence of a corresponding refining interaction with the same parameters. Thus we can redefine those rules in one unified rule applied to parameterized interactions establishing a correspondence between primitive and compound parameterized interactions. This rule is given as follows:

**Compound into primitive refinement rule:**

For each compound parameterized interaction there is a corresponding primitive parameterized interaction with the same parameters.

We can break down this rule to bring OCL sub-expression out of it which establishes a correspondence between parameterized interactions. That is, two parameterized interactions related by a refinement relationship must have the same parameters. The OCL sub-expression is given in what follows:

**Context** ParameterizedInteractionSignature **inv:**

**def:** hasSameParameters(PIS: ParameterizedInteractionSignature): Boolean = self.Parameter forAll(Px : Parameter | ParameterizedInteractionSignature →Exists( Py : Parameter | Px.name = Py.name and Px.type = Py.type))

The final OCL constraint to refine compound interactions into primitives interactions is given in what follows:

**Context** ParameterizedInteractionSignature **inv:**

**Let** CIS : ParameterizedInteractionSignature = ParameterizedInteractionSignature.oclAsType( CompoundInteractionSignature)

**Let** PIS : ParameterizedInteractionSignature = ParameterizedInteractionSignature.oclAsType( PrimitiveInteractionSignature)

CIS → forAll( c |
PIS → Exists( P |
CIS.hasSameParameters(PIS)))

This constraint establishes a correspondence between primitive and compound interactions of functional computational interfaces, thus providing for end to end QoS characteristics to be defined, as well as allowing for different kind of computational interfaces to be bound (e.g. functional to stream interfaces bindings).

7 Conclusion and perspectives

In this work we model computational interfaces and interactions signatures by introducing new concepts that prove to be relevant to specify minimal OCL constraints on interactions refinements that serves as a basis to the definition of end-to-end QoS and bindings between different types of interfaces. We mainly show a computational interface is either functional or stream. We also show that functional interfaces are composed by parameterized interactions which are divided in two categories: primitives and compounds interactions. Primitives are either incoming or outgoing interactions. Interactions between given computational interfaces are only possible if a binding (i.e. some communication path) has been established between them. Binding in the Reference Model is defined with reference to binding actions. Use of such actions is called explicit binding. There are two kinds of binding actions: primitive binding actions and compound binding actions. Primitive binding actions enable binding of an interface of the object which initiates the action to another interface (of another object, or itself). Compound binding actions enable a set of interfaces to be bound, using a binding object to support the binding [1][3]. On the other hand, a primitive binding action binds two computational object directly. A compound binding action uses primitive binding actions linking two or more computational objects via a binding object. The UML4ODP computational metamodel lacks the specification of binding refinements. Thus, there is a need to provide for such a refinement to be realized. We are looking forward to provide for such refinements, especially by establishing a correspondence between primitive interactions and primitive binding actions from one side and compound interactions and compound binding actions on the other side. Finally, since all kinds of interactions may be mapped into primitive interactions (signals), many rules relating to interactions can be reduced to rules applied on primitive interactions (signals). We are investigating how to define all the rules relating to interactions in terms of rules corresponding only to primitives.
References:


[19] O. Reda et al., Specification of OCL Constraints on ODP Computational Interfaces 7th WSEAS International Conference on Applied Informatics And Communications ( AIC’07), August 24-26, Athens, Greece, 2007


