Using Event B to Specify QoS in ODP Enterprise Language

Hafid Belhaj, Youssef Balouki, Mohamed Bouhdadi, and Said El Hajji

Department of Mathematics & Computer Science,
Faculty of science, Med V University
BP 1014 RP, 4. Av Ibn Batouta - Agdal – Rabat, Morocco
belfid@yahoo.fr, baloukiy@yahoo.fr, bouhdadi@fsr.ac.ma,
elhajji@fsr.ac.ma

Abstract. The ODP enterprise viewpoint focuses on the roles and policies on the enterprise that the system is meant to support. QoS-ODP management is now an important research topic for many universities, institutes and industrial organizations. Several approaches are being used in the field of the formalization of QoS-ODP. The use of formal methods in the design process of ODP systems is explicitly required. In this paper, we explore the benefits provided by using the proof construction approach to define the protocol of negotiating QoS requirements when the enterprise objects perform roles constrained by policies. In this context, we investigate the support for the specification of Quality of Service (QoS) in Event-B when modelling open distributed System in the enterprise viewpoint.

Keywords: RM-ODP, Enterprise Language, Trader function, QOS requirements, event B, RODIN platform.

1 Introduction

RM-ODP [1] provides a framework within which support of distribution, networking and portability can be integrated. It defines a framework comprising five viewpoints provide a basis for the specification of ODP systems. In this paper we treat the need of formal notation of ODP viewpoint languages. The languages Z, SDL, LOTOS, and Esterel are used in RM-ODP architectural semantics part [1] for the specification of ODP concepts. However, no formal method is likely to be suitable for specifying every aspect of an ODP system.

Elsewhere, we used the meta-modelling approach [4], [5] to define syntax of a sub-language for the ODP QoS-aware enterprise viewpoint specifications. We defined a UML/OCL meta-model semantics for structural constraints on ODP enterprise language [6]. We also used the same meta-modelling and denotational approaches for behavioral concepts in the foundations part and in the enterprise language [7]. Furthermore, for modelling ODP systems correctly by construction, the current testing techniques are not widely accepted and especially for the enterprise viewpoint specifications. In this paper, we use the event-B [8], [9] formalism as our formal framework for developing distributed systems. Hence we can benefit from the useful formalism for reasoning about distributed systems given by refinement techniques and from the tool support in B [10], [11], [13], [15]. In this context, we developed the QoS negotiation process using trader function with event B. The Rodin Platform for Event-B provides effective support for refinement and mathematical proof. [10], [14]

The paper is organized as follows. Section 2 presents trader function in the enterprise viewpoint. Section 3 describes and specifies the process of negotiation QoS in ODP enterprise view point by using trader function. In Section 4, we use event B as refinement support to specify this process of negotiation. Section 5 presents the Rodin platform as tool of proving initial and refinement models. A conclusion ends the paper.

2 Trader Function in the Enterprise Viewpoint

The ODP functions [12] are required to support ODP systems. RM-ODP defines three repository functions: Type Repository, Relocator and the trader function [12]. The ODP Trader provides “a dating service for objects”; its purpose is to support dynamic binding by allowing services to be discovered at run-time.

In the enterprise viewpoint, the objectives, the functional requirements, and the policy statements that govern the activities of a trading enterprise are identified. A trader is the centre of a community established for the purpose of trading. The trading activities of the community are service exports and service imports, and are governed by a trading policy of the trader. The community consists of members (objects) that have roles such as: trader, exporter, importer, trading administrator and trading policy maker. Members of the trading community are obliged to obey Policy statements (rules). Some important rules are for Trader Quality of Service requirements. The specification of an enterprise requirement has consequences in the other viewpoints.

3 QoS in the Enterprise Specification

One of the items identified in the work developed by ISO/ITU-T regarding QoS [2], [3] in ODP is the need for a QoS language capable of representing all the QoS information related to all viewpoints specification ODP system. This is the concrete QoS problem this paper focuses on. It presents a QoS language that is expressed by event B in order to take advantage of the support tools such as Rodin platform.[10]

The QoS statements in the enterprise specification are those that relate to objectives, responsibilities and policies of the ODP system in its environment. In general, these statements express QoS requirements, which are taken to include requirements on the system from the outside world, as well as the guarantees or claims its designers make in order to meet the user requirements. QoS requirements are associated with enterprise objectives, responsibilities and policies. These will correspond to requirements expressed on the enterprise objects and their interactions.

3.1 The QoS Enterprise Object Model

We illustrate how trader function is used to specify QoS when modelling an open distributed system in the enterprise viewpoint. The Quality of service may be specified in a contract or measured and reported after the event (Fig 1).

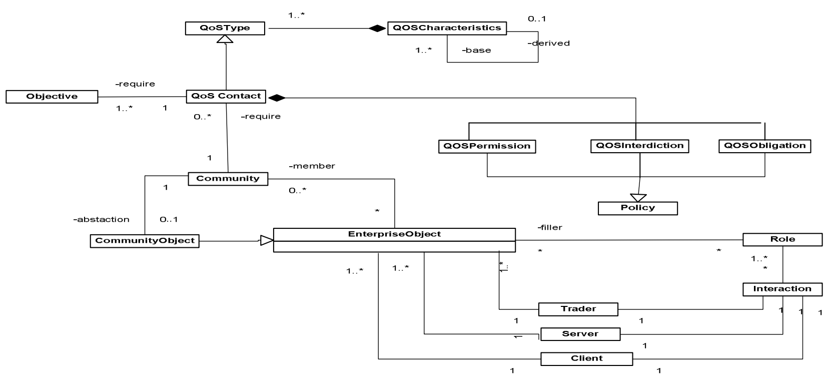

Fig. 1. Modelling QoS by trader function enterprise viewpoint

In this QoS enterprise object model, QoS management activities are driven by a trader object that is responsible to obey the system policies that are in force on objects interactions while filling roles for responding user requirements.

3.2 Single Parameter QoS Negotiation

The concept of QoS negotiation between enterprise objects includes the trader object between them. In this paper we focus on single parameter negotiation:

1) The client (the peer initiating communication) proposes a value P to the trader.
2) The Trader user could refuse the request but, if accepted, could select a new value P' that is not better than the one proposed by the Server, P '<= P.
3) The Server could refuse the request. Instead, when the Server accepts the request, could select a new V value that is not better than the initial value proposed by the client, namely V<= P'. The Server gives this value to the Trader.
5) The Server leaves the V value unchanged.
6) The selected value, V, is returned to the client and it is the value of agreement.

This mechanism is illustrated in Figure 2.

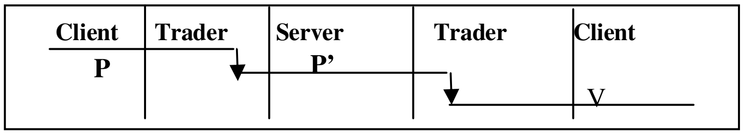

Fig. 2. QoS single parameter negotiation

4 Specifying QoS Negotiation with Event B

4.1 Requirement Document

The requirement document consists in explaining what kind of believe each enterprise object may have at the end of the protocol:

The system is negotiating QoS between enterprise objects	FUN-1

Then we explain what a successively negotiation means:

The final QoS value is published by trader object	FUN-2

We also explain what a proposed QoS:

The initial QoS value is proposed by client object	FUN-3

We describe now what both client and server may believe at the end of the negotiation protocol:

Client and server might either believe if the QoS value is conformably negotiated or failed	FUN-4

We relate the beliefs of both the client and the server:

When the QoS value is published by trader, the client and server believe that the QoS value has been negotiated successively. Otherwise, they believe that negotiation failed.	FUN-5

We explain that it is possible that the client refuse:

However, if the client refuse QoS value, the negotiation is aborted	FUN-6

4.2 Refinement Strategy

In this section, we present our strategy for constructing the QoS single parameter negotiation protocol. This will be done by means of an initial model followed by three refinements.

a) The initial model set up the scene by taking account of requirements FUN-1 to FUN-3. It essentially presents the protocol as done in one shot.
b) In the first refinement, we take care of the remaining requirements FUN-4 to FUN-6 in an abstract way. The QoS is approved step by step, but we are still abstract in that both client and server can have a direct access to the state of the other.
c) In the second refinements, we introduce the trader between client and server so that each of them only reacts to the content of trader messages: it cannot look directly at the value of its partner as was the case in the previous refinements. For these messages, we introduce the trader between client and server (as negotiator). we also introduce the timers in both participants.

4.2.1 Initial Model

Our initial model contains a partial specification of the QoS single parameter negotiation protocol. It deals with requirements FUN-1, FUN-2 and FUN-3. The protocol is executed in one shot.

4.2.1.1 The State

The state is made of the set or real positive numbers IR+ to represent the set of QoS values. The maximum QoS value allowed by server object is denoted by a constant positive value Vserver_max. the QoS values wanted by the client object is expressed by a constant positive value P. These values are defined in axm0_1 and axm0_2.

Sets: IR^+

Constants: Vserver_max
P

Axm0_1 : Vserver_max $\in$ IR+
Axm0_2: P $\in$ IR+

The negotiated and the final QoS values are variables typed in invariants inv0_1, inv0_2, inv0_3, inv0_4, and inv0_5.

variables : Vnegoc
Vfinal

Inv0_1 : Vnegoc $\in$ IR^+
Inv0_2 : Vnegoc $\leq$ Vserver_max
Inv0_3 : Vfinal $\in$ IR^+
Inv0_4 : Vfinal $\leq$ Vserver_max
Inv0_5 : Vfinal $\leq$ P

4.2.1.2 The Events

The initial value of Vnegoc is P. the final value of the protocol is non-deterministically defined by event QoSspnp, that explained that Vfinal value becomes Vnegoc.

Init : Vnegoc := P
Vfinal := 0

QoSspnp
Vfinal :| Vfinal = Vnegoc

4.2.2 First Refinement

The role of this refinement is to specify the final value of the status of client and server as defined in requirements FUN-4 to FUN-6. We shall also see how the QoS value is negotiated step by step between client object and server object.

4.2.2.1 The State

In this first refinement, we introduce the concept of status. For this, we introduce a carrier set named STATUS. It is made of three distinct elements: propose, accept and refuse as shown below.

Sets: STATUS

Constants: Propose
Accept
Refuse

Axm1_1: STATUS ={Propose,Accept, Refuse}
Axm1_2 : Propose $\neq$ Accept
Axm1_3 : Propose $\neq$ Refuse
Axm1_4 : Accept $\neq$ Refuse
Axm1_5 : Vserver $\in$ IR^+
Axm1_6 : Vserver $\leq$ Vserver_max

We introduce the status variables c_st and s_st of the client object and server one respectively. Such variables are member of the STATUS.

variable: c_st
s_st
Vnegoc
Vfinal

Inv1_1 : c_st $\in$ STATUS
Inv1_2 : s_st $\in$ STATUS
Inv1_3 : c_st = Accept $\Leftrightarrow$ P $\geq$ Vserver
Inv1_4 : s_st = Accept ==> c_st =

Requirement FUN-5 is formalized in inv1_4 where it said that the client accept when the server does.

4.2.2.2 The Events

In this refinements, we introduce many new events : Client_accept, Client_refuse, Server_accept and Server_refuse. The events Client_accept and Server_refuse are still very abstract because in the first one, the client is aware of the status of the server and in the second one the server is aware of the status of the client. This is clearly not possible and will be implemented correctly in the next refinement.

```
Client_accept
When
  c_st = Propose
  s_st = Accept
Then  c_st := Accept
      Vfinal :=Vserver
End
```

```
Client_refuse
When
s_st = Propose
Then
 c_st := Refuse
End End
```

```
Server_accept
When
 s_st = Propose
Then
  s_st := Accept
End
```

```
Server_refuse
When
  s_st = Propose
  c_st = Refuse
Then
    s_st := Refuse
 End
```

This is clearly a convenient abstraction but not a final implementation. In fact, this direct access will be removed in the next refinement.

```
Init :    Vnegoc := P
          Vfinal  := 0
          c_st := Propose
          s_st := Propose
```

```
QoSspnp
   When    c_st ≠ Propose
           s_st ≠ Propose
   Then    skip    End
```

The refinement of event QoSspnp must refine its abstraction, which is a non-deterministic event.

4.2.3 Second Refinement

In this refinement, the trader will enter into the scene by cooperating with client and server objects in order to negotiate the QoS value. In fact, the client will not access any more directly the server value as was the case of the previous refinement, this will be done by the trader. We then introduce this trader which is situated between client and server.

4.2.3.1 The State

The state is first enlarged with a constant QoS trader value :Vtrader denoted by the axioms axm2_1 and axm2_2 and a Boolean variable Publish indicated implicitly by inv2_1.

```
Sets:  B
```

```
Constants: Vtrader
           True
           False
           Max_timer
```

```
Axm2_1:   Vtrader ∈ IR+
Axm2_2:   Vtrader ≤ Vserver_max
Axm2_3:   B={True, False}
Axm2_4:   Max_timer ∈ IN
```

```
variables: Publish
           c_timer
           s_timer
```

```
Inv2_1: Publish ∈ B
Inv2_2: Publish=True ==> Vtrader ≤ Vserver
Inv2_3: c_timer  ∈ 0 .. Max_timer
Inv2_4: s_timer  ∈ 0 .. Max_timer
Inv2_5: c_timer = Max_timer + 1 ⇔ c_st = Refuse
Inv2_6: s_timer = Max_timer + 1 ⇔ s_st = Refuse
```

4.2.3.2 The Events

The initialization event is extended in a straightforward fashion as indicated below. The Boolean value publish is set to False at the beginning so that the only two events which can be fired are the ones described next.

```
Init :   Vnegoc := P
         Vfinal  := 0
         c_st := Propose
         s_st := Propose
         Publish := False
         C_timer := 0
         S_timer := 0
```

```
Client_accept
When   c_st = Propose
   Publish = True
   c_timer ≤ Max_timer
Then   c_st := Accept
       Vfinal := Vserver
End
```

```
Client_refuse
When c_st = Propose
   Publish = False
   C_timer  >  Max_timer
Then    c_st := Refuse
End
```

```
Server_accept: When s_st = Propose
               Publish = True
               s_timer ≤ Max_timer
              Then s_st:= Accept   End
```

```
Server_refus:  When  s_st = Propose
                Publish = False
                s_timer > Max_timer
              Then   s_st := Refuse  End
```

5 Conclusion and Perspectives

RM ODP is intended to create an international standard for the design and realization of open distributed systems. The use of formal methods in the design process of ODP systems is explicitly required. In this article the use of the Formal Description Techniques event B is investigated and evaluated. We presented our approach for developing distributed system in Event B. we proposed a visualization of, and guidance for, event B refinement using trader enterprise viewpoint function for negotiating QoS requirements between enterprise objects. Moreover, in order to verify our final model, the refinements models are integrated with the Event B platform developed by the RODIN.

As for future work, we are going to generalize our approach to other methods of reaching QoS agreements in ODP systems in particular using event-B in the design process of ODP systems.

References

1. ISO/IEC: Basic Reference Model of Open Distributed Processing-Parts1,2, 3 and 4, ISO/IEC CD 10746 (1994)
2. ISO/IEC TR 13243 – Information technology – Quality of service –Guide to methods and Mechanisms (November 1999)
3. ITU-T Recommendation G.1000 - Communications quality of service: a framework and Definitions (November 2001)
4. Bouhdadi, M., et al.: A UML-Based Meta-language for the QoS-aware Enterprise Specification of Open Distributed Systems. IFIP Series, vol. 85, pp. 255–264. Springer, Heidelberg (2002)
5. Bouhdadi, M., Balouki, Y.: Semantics of Behavioral Concepts for Open Virtual Enterprises. Lecture Notes in Electrical Engineering, vol. 27, pp. 275–286. Springer, Heidelberg (2009)

6. Balouki, Y., Belhaj, H., et al.: Event B for ODP Enterprise Behavioral Concepts Specification. In: Proceedings of the World Congress on Engineering, WCE 2009, London, U.K., July 1-3. Lecture Notes in Engineering and Computer Science, vol. I, pp. 784–788. Newswood Limited (2009)
7. Balouki, Y., Bouhdadi, M.: Using BPEL for Behavioral Concepts in ODP Enterprise Language, Virtual Enterprises and Collaborative Networks. IFIP, vol. 283, pp. 221–232. Springer, Heidelberg (2008)
8. Abrial, J.-R.: The B-Book: Assigning programs to meanings. Cambridge University Press, Cambridge (1996)
9. Abrial, J.-R.: Extending B without changing it (for developing distributed systems). In: Habrias, H. (ed.) First B Conference (November 1996)
10. Event-B and the Rodin Platform, `http://www.event-b.org/`
11. Joochim, T., Snook, C., Poppleton, M., Gravell, A.: Timing Diagrams Requirements Modelling Using Event-B Formal Methods. In: IASTED International Conference on Software Engineering (SE 2010), Innsbruck, Austria, February 16-18 (2010)
12. ISO/IEC: The ODP Trading Function, ISO/IEC JTC1/SC21 (1995)
13. Butler, M.J., Hallerstede, S.: The Rodin Formal Modelling tool. In: BCS-FACS Christmas 2007 Meeting Formal methods in Industry, London (2007)
14. Abrial, J.-R.: Tutorial - Case study of a complete reactive system in Event-B: A mechanical press controller. In: Proc. 5th International Symposium on Formal Methods (FM 2008), Turku, Finland (2008)
15. Cansell, D., Méry, D., Rehm, J.: Time Constraint Patterns for Event B Development. In: Proc. Formal Specification and Development in B, 7th International Conf. of B (B 2007), Besancon, France, pp. 140–154 (2007)