Formal Specification of QoS Negotiation in ODP System

Abdessamad Jarrar, Youssef Balouki, Taoufiq Gadi
Faculty of Sciences and Technologies Informatics, Imaging and Modeling of Complex Systems Laboratory Settat, Morocco

Article Info

Article history:

Received Feb 9, 2017
Revised Apr 19, 2017
Accepted May 3, 2017

Keyword:

Event-B
Formal method
Negotiation
ODP
Quality of service
Refinement processus

ABSTRACT

The future of Open Distributed Processing systems (ODP) will see an increasing of components number, these components are sharing resources. In general, these resources are offering some kind of services. Due to the huge number of components, it is very difficult to offer the optimum Quality of service (QoS). This encourages us to develop a model for QoS negotiation process to optimize the QoS in an ODP system. In such system, there is a High risk of software or hardware failure. To ensure good performance of a system based on our model, we develop it using a formal method. In our case, we will use Event-B to get in the end of our development a system correct by construction.

Corresponding Author:

Abdessamad Jarrar,
Faculty of Sciences and Technologies,
Computing, Imaging and Modeling of Complex Systems Laboratory,
FST Settat, Km 3, B.P. : 577 Route de Casablanca, Morocco.
Email: Abdessamad.jarrar@gmail.com

1. INTRODUCTION

The evolution of telecommunications technology and the structure of organizations have led to the emergence of complex distributed systems. These systems are distributed structures whose components, both hardware and software, are of different types. In some systems, these components are developed by different actors acting independently of each other. The assembly of such components gives rise to highly heterogeneous systems. The applications that support these systems are themselves composed of distributed components. The interaction between these application components is one of the aspects of the distributed treatment. Specifically, the distributed processing correspond to different aspects of information processing in which specific components can be located in different places, during this process the communication between components may be delays or failures. This means that this kind of systems needs to be developed carefully due to its complexity [1]. This motivates us to model it using a formal method to ensure the correctness of our system and obtain a very strong assurance of bug's absence. Formal methods are a particular kind of mathematically based techniques for the specification [2], [3], development and verification of software and hardware systems [4]. There are a variety of formal methods such as language Z, a specification in Z is a predicate, the specification of invariants and the specification of operations have the form of a predicate. There is also B-method which is a method of software development based on B, a tool-supported formal method based on an abstract machine notation, used in the development of computer software. It was originally developed by Jean-Raymond Abrial [5].

In this paper, we will use Event-B [6] since it allows us to prove that our system is correct by construction basing on proofs obligations. These proofs are done automatically by a tool called Rodin [7]. Event-B is also based on refinement which means creating an abstract model and enriching it in a multiple steps by adding more details to get a more concrete model [8]. In every refinement, we prove that the system is correct and it does not contradict with the previous one, whereby the resulting system is correct by construction [6]. In the beginning, we present the proposed negotiation approach, this approach is based on trader. Then, we define the system requirement along two axes: FUN and ENV. After that, we present our refinement strategy that we will use after that to specify our system. Lastly, we end this paper with a conclusion presenting an abstract about the work done and our expectation about future works.

2. RELATED WORKS

"Using Event B to Specify QoS in ODP Enterprise Language" [1] like our work, is specifying QoS negotiation using event-B, it presents a specification for the different actors of the system and their states and it also present negotiation values. However, it doesn't present a specification of how the system acts exactly such as how the system is able to negotiate with multiple servers.

"End-to-end QoS negotiation in network federations" [9] is another work presenting QoS negotiation, it presents a good specification of the negotiation process, and in addition it presents a mathematical modeling. Yet, mathematical model doesn't prove that the specification is correct. Also, it limits the study in telecommunication domain.

"An example of dynamic QoS negotiation" [10] presents an example of QoS negotiation applied to a video streaming application, this example is presented with mathematical models and statistics results. The same as the previous paper, there is nothing proving the correctness of the system.

Our work is presenting a formal specification using Event-B, this allows us to ensure the correctness of our system using poof obligations, it also presents a modeling of negotiation in more details. Also, our work is proved using Rodin platform [7] which avoid human mistakes during proving proof obligations.

3. NEGOTIATION APPROACH

Quality of Service (QoS) is a management concept that aims to optimize network resources or process and ensure good performance of an Open Distributed Processing (ODP) system, this concept is fundamental in many fields such as transmission protocols [11], routing algorithms [12], resources allocation algorithms [13] and web service [14]. In our negotiation process, we will base our study on the trader concept [1]. This means that in addition of the client and servers, we will have a third actor, it is the trader. The trader is playing the role of a controller who is able to get the best QoS possible for a client. In the beginning, a client propose a value of quality of service P to the trader, the trader may modify the QoS proposed by the client or not, after that the trader send the value P' of QoS to the server, when the server gets the required value it may either refuse the request or may propose the value V that it may offers, at this stage the trader will either modify the value proposed by the server or returns it directly to the client, if the client is satisfied with the proposed value he accepts it or else he refuses it, in this case the trader will automatically start negotiation with another server and proceed in the same way. The Figure 1 below presents the process of negotiation with a server basing on a UIT form [15]:

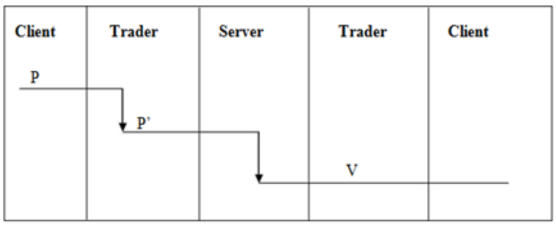

Figure 1. Negotiation Process [1]

4. FORMAL SPECIFICATION OF QOS NEGOCIATION

4.1. Requirement Document

To present the requirement document correctly, we present it along two main axes. The first axis expresses the main functions of system "FUN", and the second describes these functions and provides some details regarding the environment "ENV". We present our requirement document as follows:

The system allows for the negotiation of QoS between a client and one or more servers	FUN 1

ENV 1: The trader is an intermediate between client and server.

Negotiation end if the client accepts the QoS or if he refuses it	FUN 2

ENV 2: A client can be in one of three states (propose, accept or refuse).

The client can only accept negotiation if the trader proposes a value of quality of service	FUN 3

ENV 3: A trader can either propose a value of QoS or refuses to offer the service.

the trader can only refuse negotiation if all the servers refuse to offer any QoS or if the client refuse all the proposed values of QoS	FUN 4

ENV 4: a server can propose or refuse to offer any QoS

The value proposed by the server is always less than or equal the value given by the trader	FUN 5

ENV 5: values of QoS are always positive.

The trader always propose values less than or equal the value proposed by the client	FUN 6
The client propose the value he wants, then the trader seeks the best value less than or equal to that value in all servers	FUN7

4.2. Refinement Strategy

After specifying our negotiation process correctly, we present our refinement strategy. We start by creating an abstract model (first refinement) containing the principal functions of our system, then we will enrich it by adding more function and environment assumptions (second refinement and third refinement). In more details, here is our refinement strategy:
a. First refinement: In the beginning, we present the various actors states (client, trader and servers) and the basic logic of the system (FUN 1, FUN 2, FUN 3, FUN 4, ENV 1, ENV 2, ENV 3 and ENV 4)
b. Second refinement: in this refinement, we include negotiation values and how the actors may change it while negotiating with a server (FUN 5, ENV 5 and ENV 6).
c. Third refinement: lastly, the system is able to negotiate with multiple servers (FUN 7).

4.3. First Refinement: Specifying Actors States

As it is mentioned before, we start by modeling the various states of the system actors. Here is our first context of the first refinement:

```
CONTEXT
        context0
SETS
        STATE
CONSTANTS
        propose
        refuse
        accept
        wait
AXIOMS
        axm1:partition(STATE,{propose},{refuse}, {accept},{wait})
END
```

In this context which is the static part of a refinement, we define the set STATE as a partition of the all the possible states in the system of an actors. In addition, we add an additional state that we have called “wait”; this state is the initialization state. We present the dynamic part of the first refinement (machine); we start by the invariant that are the properties that must be preserved during system occurrence:

```
VARIABLES
        S_st
        C_st
        T_st
INVARIANTS
        inv1:   S_st ∈ STATE
        inv2:   S_st ≠ accept
        inv3:   C_st ∈STATE
        inv4:   T_st ∈ STATE
        inv5:   T_st ≠ accept
        inv6:   C_st = accept ⇒ T_st = propose
        inv7:   T_st = refuse ⇒ C_st ≠ accept
        inv8:   T_st = refuse ⇒ S_st = refuse
        inv9:   S_st = propose ⇒ T_st = propose
```

In these invariants, we define S_st (Server state) as an element of STATE that is not equal accept, the same go for T_st (Trader state) as the server and the trader are not allowed to accept the negotiation, the client is the only actor that may accept the negotiation this is why C_st may be equal any state. In addition, we present some invariant that control the possible state combinations, for example, if the client is in the accept state the trader must be in the state propose (inv6), which mean that a client cannot accepts a negotiation if the trader did not proposes a QoS value. Now we can initialize our variable with “wait” value.

```
INITIALISATION:
        THEN
                act1:   S_st ≔ wait
                act2:   C_st ≔ wait
                act3:   T_st ≔ wait
END
```

Beside the initialization event we have more events that are able to change the states of the actors while preserving all the invariants. Client_propose is the event that starts a new negotiation; we can start a new negotiation only if we end the previous one, which means that the trader is not in the state “propose”.

```
Client_propose:
        WHERE
                grd1:   C_st = wait
                grd2:   T_st ≠ propose
        THEN
                act1:   C_st ≔ propose
        END
```

When a client proposes a value of QoS, the trader switches his state to “propose” to start negotiation with servers.

```
Trader_propose:
        WHERE
                grd1:   C_st = propose
        THEN
                act1:   T_st ≔propose
        END
```

The server proposes or refuses negotiation if and only if the trader proposes.

```
Server_refuse:
        WHERE
                grd1:   T_st = propose
        THEN
                act1:   S_st ≔ refuse
        END
```

The only case where the trader will refuse negotiation is when all servers refuse negotiation, and then the trader ends negotiation and informs client.

```
Trader_refuse:
        WHERE
                grd1:   C_st = propose
                grd2:   S_st = refuse
        THEN
                act1:   T_st ≔ refuse
                act2:   C_st ≔ refuse
        END
```

The client refuses a negotiation if he is not satisfied with negotiation value proposed by the trader.

Client_refuse:
WHERE
grd1: C_st = propose
grd2: T_st = propose
THEN
act1: C_st:= refuse
act2: T_st := refuse
act3: S_st:= refuse
END

Also, if the client is satisfied with the proposed value by the trader, he accepts negotiation.

Client_accept:
WHERE
grd1: T_st = propose
THEN
act1: C_st := accept
END

4.4. Second Refinement: Modeling Negotiation Values

In this refinement, we present the negotiation values. A client propose a value of quality of service P to the trader, the trader may modify the QoS proposed by the client or may keep it, after that the trader send the value P' of QoS to the server, when the server get the required value it may either refuse the request or may propose the value V that it is able to offer, at this stage the trader will either modify the value proposed by the server or return it directly to the client. Before modeling these events, we present the context below presenting the maximum value of QoS that a server may offer Vserver_max:

CONTEXT
context1
EXTENDS
context0
CONSTANTS
Vserver_max
AXIOMS
axm1: Vserver_max ∈ ℕ
axm2: Vserver_max > 0
END

In machine1, we have new variables representing the negotiation values (Vclient, Vtrader, Vserver and Vservice which is the value of QoS offered in case of accepting the negotiation). Moreover, we have additional invariant for the negotiations values:

INVARIANTS
inv1: Vclient ∈ ℕ
inv2: Vserver ∈ ℕ
inv3: Vtrader ∈ ℕ
inv4: Vservice ∈ ℕ
inv5: Vclient ≥ Vtrader
inv6: Vtrader ≥ Vserver
inv7: T_st = propose ⇔ Vtrader ≠ 0
inv8: S_st = propose ⇔ Vserver ≠ 0
inv9: (C_st = propose ∨ C_st = accept) ⇔ Vclient ≠ 0
inv10: Vserver ≤ Vserver_max

Furthermore, we refine the events of refinement 0 by adding actions responsible for dealing with QoS values such as these actions setting negotiation values to 0 in the initialization event:

act4: Vclient := 0
act5: Vserver :=0
act6: Vtrader :=0
act7: Vservice:= 0

The value proposed by the client Vclient must be a not null natural number.

Client_propose:
ANY
val ›
WHERE
...
grd3: val ∈ ℕ
grd4: val ≠ 0
THEN
act1: C_st ≔ propose
act2: Vclient ≔ val
END

The value proposed by the trader must be a positive number less than or equal the value proposed by the client.

Trader_propose:
ANY
val
WHERE
grd1: C_st = propose
grd2: val ∈ ℕ
grd3: val ≤ Vclient
grd4: val ≥ Vserver
grd5: val ≠ 0
THEN
act1: T_st ≔propose
act2: Vtrader ≔ val
END

A server proposes a value less than or equal the value proposed by the trader, and this value is always less than or equal a predefined maximum value of the Qos.

Server_propose:
ANY
val
WHERE
grd1: T_st = propose
grd2: val ∈ ℕ
grd3: val ≤ Vtrader
grd4: val ≠ 0
grd5: val ≤ Vserver_max
THEN
act1: S_st ≔propose
act2: Vserver ≔ val
END

When a trader proposes a value of QoS, the server may be unable to offer any QoS, in this case the server refuses the process of negotiation.

Server_refuse:
WHERE
grd1: T_st = propose
THEN
act1: S_st ≔ refuse
act2: Vserver ≔ 0
END

If a server refuse to offer a QoS, the trader try to negotiate with another server until it find a valid QoS, however in some cases all the servers are unable to offerQoS, which mean that the trader will stop negotiation, which mean that we reset the value of Vtrader and Vclient to 0 in the event Trader_refuse:

act3: Vtrader ≔ 0
act4: Vclient≔ 0

Even if the trader find a server that may offer a Qos, the client may be not satisfied, in this case the client may refuse the negotiation by resetting all the negotiation values to 0 in the event Client_refuse:

act4: Vclient ≔ 0
act5: Vtrader ≔ 0
act6: Vserver ≔ 0

In the other hand, if the client is satisfied with the offered QoS, he may accept negotiation. In this case, the value of service will be the value that the trader proposes, which means that we have one additional action in the event Client_accept:

act2: Vservice ≔ Vtrader

4.5. Third Refinement: Negotiation with Multiple Servers

In this last refinement, we allow the system to negotiate with multiple servers. In other words, when the client gets the QoS value proposed he may be not satisfied with it and refuses the negotiation. In this case, the trader starts negotiating with another server that may offer a better QoS. To model this, we need to define a set of servers in our system illustrated in the context below:

CONTEXT
context2
EXTENDS
context1
SETS
Servers
AXIOMS
axm1: Servers ≠ ∅
END

Similarly, we define new two variables. The first variable is server which represents the current server that we are negotiating with. The second is Servers_Not_Tested which represents the set of servers that we have not negotiated with yet. More than that, we have additional invariant:

inv1: server ∈ Servers
inv2: Servers_Not_Tested ∈ℙ(Servers)

In the same way as the previous refinement, we refine also the events of machine 1. In the beginning, we refine the initialization event:

INITIALISATION:
THEN
...
act8: server :∈ Servers
act9: Servers_Not_Tested ≔ Servers
END

In the beginning of every new negotiation, the client proposes a QoS as mentioned before, at this stage, we initiate the set Servers_Not_Tested as all the servers of our network in the event Client_propose :

act4: Servers_Not_Tested ≔ Servers

The trader is the one responsible for choosing the server to negotiate with; this server is chose from the set Servers_Not_Tested, which mean that we need to check if this set is not empty in the event Trader_propose using the following guard:

grd6: Servers_Not_Tested ≠ ∅

Also, this action picks a server from Servers_Not_Tested:

act3: server :∈ Servers_Not_Tested

When a server refuses to offer a QoS, we remove it from Servers_Not_Tested to ensure that we won't negotiate with the same sever over and over again. The action allowing removing the server from Servers_Not_Tested in the event Server_refuse is the following:

act3: Servers_Not_Tested ≔ Servers_Not_Tested \{server}

The trader refuses negotiation if and only if all the servers refuse to offer a QoS which mean that we removed all the servers from Servers_Not_Tested, this mean that Servers_Not_Tested is empty. This means that the Trader_refuse will never be occurred unless Servers_Not_Tested is empty; this is done by the following guard:

grd3: Servers_Not_Tested=∅

The client accepts the negotiation if there is a server in Servers_Not_Tested that may offer a QoS and the client is satisfied with it. This means that we will have a new guard in the event Client_accept:

grd2: server ∈ Servers_Not_Tested

5. PROVING SYSTEM CORRECTNESS

Proof obligations are a set of evidence that ensures the validity of the system, the most important among them is the preservation of invariants proofs that validates the preservation of all the invariant condition before and after each event, all this can be done manually. Most of the proofs are easy and are not the kind of demonstrations that could interest a mathematician because the difficulty of modeling in event-B is not the complexity of proof but demonstrations of consistency despite the huge number of events and invariants, all events must preserve all the invariant. This mean that the problem is that the amount of proof to prove is very big, in our case we have 21 invariants (9 in machine 0, 10 in the machine 1 and 2 for the machine 2) and we have 10 events this mean that we have 210 proof to be done. Luckily there is a platform to do the most of these proofs automatically, this framework is called Rodin. The Rodin platform is an IDE based on Eclipse for Event- B which provides effective support for refinement and mathematical proof. The platform is open source, contributes to the Eclipse platform and is more extensible with plugging very effective (Atelier, ProB ...). In the Table 1 below the statistics of proofs done by Rodin:

Table 1. Proof Statistics

Elements	Total	Auto	manual	Reviewed	Undischarged
Qosnegot	92	91	1	0	0
Context0	0	0	0	0	0
Context1	0	0	0	0	0
Context2	0	0	0	0	0
Machine0	38	38	0	0	0
Machine1	51	50	1	0	0
Machine2	3	3	0	0	0

6. CONCLUSION

we have developed the process of negotiation of QoS between objects in an ODP system, using the trading function. We have proposed to introduce a dynamic trading assistant in the user's terminal to help, firstly, to choose the best service provider and, secondly, to dynamically negotiate the quality parameters of service (QoS) responsive to the user's requirements and application. The model we have developed is based on the formal method Event-B. The interest of the Event-B in our study lies in its modeling to formally express properties validated by evidence during the design of system models, but also in its refinement principle to master the complexity of the system by progressive and safe development. For future works, we are working on specifying formally aircraft landing process; we also will develop a model of an abstract complex adaptive system.

REFERENCES

[1] Y. Balouki AND Al, “Using Event B to Specify QoS in ODP Enterprise Language”, *Collaborative Networks for a Sustainable World, IFIP Series Advances in Information and Communication Technology*, Vol. 336, pp. 478-485, Springer, 2010.
[2] Cristophe Métayer et Laurent Voisin, “The Event-B Mathematical Language”, Mars 26 2009.
[3] Biggs, N, “Discrete Mathematics”M Second Edition. New Delhi: Oxford University Press, ISBN-10: 0198507178, 2003.
[4] R. W. Butler, “What is Formal Methods?”, Retrieved 2006-11-16, 2001-08-06.
[5] Jean-Raymond Abrial, “The B Tool (Abstract)”, (PDF), In Robin E. Bloomfield and Lynn S. Marshall and Roger B. Jones. VDM — The Way Ahead, Proc. 2nd VDM-Europe Symposium. Lecture Notes in Computer Science. 328. Springer. pp. 86–87. ISBN 3-540-50214-9. 1988.
[6] Jean-Raymond Abrial, “Modeling in Event-B: System and Software Design”, Cambridge University Press, 2008.
[7] Michael Jastram (Editor) Foreword by Prof. Michael Butler, “Rodin User's Handbook” , Covers Rodin v.2.8.
[8] Thai Son Hoang, Hironobu Kuruma, David Basin, Jean-Raymond Abrial, “ Developing Topology Discovery in Event-B”, 2009.
[9] Helia Pouyllau, Richard Douville, “End-to-end QoS negotiation in network federations”, ICT ETICS project, Grant agreement, FP7-248567 Contract Number: INFSO-ICT-248567, 2010.
[10] P. Cremonese , S.Giordano, “An example of dynamic QoS negotiation”, Finsiel S.p.A via Matteucci 34 Pisa Italy.
[11] Augustin WAYAWO, “The Transmission Multicast and The Control of QoS for IPV6 Using the Insfrastructure MPLS”, *International Journal of Information and Network Security (IJINS)*, Vol 1 No 1, pages 9-27. 2012.
[12] Liu Hui, “A Novel QoS Routing Algorithm in Wireless Mesh Networks”, *TELKOMNIKA Indonesian Journal of Electrical Engineering*, Vol 11 No 3, pages 1652-1664. 2013.
[13] Dac-Nhuong Le, “PSO and ACO Algorithms Applied to Optimization Resource Allocation to Support QoS Requirements in NGN”, *International Journal of Information and Network Security (IJINS)*, Vol 2 No 3, pages 216-228. 2013.
[14] Naji Hasan.A.H, Gao Shu, AL-Gabri Malek and Jiang Zi-Long, “An Optimal Semantic NetworkBased Approach for Web Service Composition with QoS”, *TELKOMNIKA Indonesian Journal of Electrical Engineering*, Vol 11 No 8, pages 4505-4511. 2013.

[15] UIT-T, "Série X: Réseaux Pour Données Etcommunication Entre Systèmes Ouverts, Technologies de l'information – Qualité de service – Guide pour les méthodes et les mécanismes Recommandation", X642 09/1998.

BIOGRAPHIES OF AUTHORS

Abdessamad Jarrar is a PhD candidate at IIMCS Laboratory which stands for Informatics, Imaging and Modeling of Complex Systems in Faculty of Sciences and Technologies Hassan 1st University, Settat, Morocco, his research interests center around modeling of complex systems using formal methods, he is currently in his second year of his PhD. his currect research focus on modeling an abstract complex adaptive system, in addition, he is working on a method to solve the problem of infinte cycles in complex systems.

Youssef Balouki is currently a professor of computer and information science at the Faculty of Sciences and Technologies, Hassan Ist University, Settat, Morocco. He holds a PhD in Computer Science (2010) from MohammedV University, Rabat, Morocco and he graduated in Computer Science (1995) in Cadi Ayyad University, Marrakech, Morocco where he got his Master's degree in computer Science. According to Google Scholar, DBLP digital library (as at May 2016) he has several publications which have been cited quite a few times. Finally, his main research interests lie in the Databases, Distributed & Parallel Computing and Scientific Computing.

Gadi Taoufiq is a Professor on computer science at the faculty of science and technologies (Hassan First University of Settat Morocco). Since 2014, he is the Director of the Informatics, Imaging and Modeling of Complex Systems Laboratory. He has conducted more than tens PhD theses and written a fifty of scientific papers in the domain of 3D models analysis, models Driving Architecture, Datamining and Database Analysis, Modeling of Complex Systems. He is recipient of the best paper awards at the IMP Session of IEEE CIST' 2016.