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

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.

Copyright © 2017 Institute of Advanced Engineering and Science. All rights reserved.
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 proof 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]](image)

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 |

IJECE  Vol. 7, No. 4, August 2017 : 2045 – 2053
ENV 1: The trader is an intermediate between client and server.

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

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

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

ENV 5: Values of QoS are always positive.

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
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 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:
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:**

\[
\text{act1: } S_{st} := \text{wait} \\
\text{act2: } C_{st} := \text{wait} \\
\text{act3: } T_{st} := \text{wait}
\]

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:**

\[
\text{WHERE} \\
\text{grd1: } C_{st} = \text{wait} \\
\text{grd2: } T_{st} \neq \text{propose}
\]

\[
\text{THEN} \\
\text{act1: } C_{st} := \text{propose}
\]

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

**Trader_propose:**

\[
\text{WHERE} \\
\text{grd1: } C_{st} = \text{propose}
\]

\[
\text{THEN} \\
\text{act1: } T_{st} := \text{propose}
\]

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

**Server_refuse:**

\[
\text{WHERE} \\
\text{grd1: } T_{st} = \text{propose}
\]

\[
\text{THEN} \\
\text{act1: } S_{st} := \text{refuse}
\]

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:**

\[
\text{WHERE} \\
\text{grd1: } C_{st} = \text{propose} \\
\text{grd2: } S_{st} = \text{refuse}
\]

\[
\text{THEN} \\
\text{act1: } T_{st} := \text{refuse} \\
\text{act2: } C_{st} := \text{refuse}
\]

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

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 $V_{server\_max}$:

\begin{verbatim}
CONTEXT
  context1
  EXTENDS context0
  CONSTANTS
    Vserver_max
  AXIOMS
    axm1: Vserver_max ∈ ℕ
    axm2: Vserver_max > 0
END

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 V C_st = accept) ⇒ Vclient ≠ 0
  inv10: Vserver ≤ Vserver_max
END

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:

\begin{verbatim}
act4: Vclient := 0
act5: Vserver := 0
act6: Vtrader := 0
act7: Vservice := 0
\end{verbatim}

The value proposed by the client Vclient must be a not null natural number.
The value proposed by the trader must be a positive number less than or equal the value proposed by the client.

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

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.

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 offer QoS, 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:

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:

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:


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:

```
act2: Vservice := Vtrader
```

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:
```

```
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>
<thead>
<tr>
<th>Elements</th>
<th>Total</th>
<th>Auto Manual</th>
<th>Reviewed</th>
<th>Undischarged</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qosnegot</td>
<td>92</td>
<td>91</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Context0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Context1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Context2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Machine0</td>
<td>38</td>
<td>38</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Machine1</td>
<td>51</td>
<td>50</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Machine2</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

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 current research focus on modeling an abstract complex adaptive system, in addition, he is working on a method to solve the problem of infinite cycles in complex systems.

Youssef Balouki is currently a professor of computer and information science at the Faculty of Sciences and Technologies, Hassan 1st 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.