

The Description and Calculation of Quality of Composite Services¹

Meng Li, Hao-peng Chen, Nan Wang

School of Software, Shanghai Jiao Tong University, P.R.China, 200240

demien@sjtu.edu.cn, chen-hp@sjtu.edu.cn, wangnan06@sjtu.edu.cn

Abstract

With the widespread of Web Services (WS), more deployed services are being registered into a services registry, consequently giving rise to a series of problems. A conspicuous one is the large amount of functionally equivalent Web Services returned by a service registry. A plausible solution is to involve QoS (Quality of Service) in the services discovery. However, some existing QoS-based discovery mechanisms still fail to consider two distinct points. Firstly, most solutions involve poor semantics so different parties may have different understandings of the same QoS metric. Secondly, existing QoS metrics of WS are so numerous, ambiguous and unpredictable that it limits the flexibility and expansibility. To address these two problems, this paper presents a novel and extensible ontology-based approach for description of the QoS constraints. Further, we propose a QoS calculation method for the composite service to judge whether the set of recommended services is qualified for the QoS constraints.

1. Introduction

Web Services (WS) are self-describing, self-contained, modular and loosely coupled software application. It could be registered, advertised and located across the Internet using a set of standards such as WSDL, SOAP and UDDI. SOA (Service Oriented Architecture) encapsulates the information and interact with standard programmatic interface. It is an advanced mechanism to organize and utilize distributed resources. The outstanding advantage is to enable service dynamically selected and integrated which is a vital autonomic attribute for modern business^[1].

With the proliferation of WS, the number of advertised services according to the consumers' functional constraints is increasing rapidly. Thus a non-functional concept is needed to differentiate the functionally equivalent WS. That is **quality of service** (QoS). However, it is not an easy task to distinguish all the possible QoS metric of WS. Fortunately W3C published a proposal called: " QoS for Web Services: Requirements and Possible Approaches " which aims to identify all the possible QoS requirements such as response time and throughput for web services^[2].

QoS offering is a set of constraints on QoS

metrics. For example, the response time should be less than one second. In order to express the QoS offering in a formal way, some models^{[3][4]} and languages like WSLA^[5] are developed. However, few attempts have been done on semantic facet. Actually, most current models of QoS offerings differ with each other only in the expressiveness of these constraints without semantic aspect. Consequently, the same QoS metric may have different understanding for different parties.

Tentatively it can be suggested to combine Semantic Web and WS technology using ontology. Ontology is a XML-based formal specification of the terms in a certain domain. It defines a common vocabulary for researchers who need to share information in this domain^[6]. There are three advantages for ontology: well-defined syntax and semantics, efficient reasoning support and sufficient expressive power. OWL-Q is the ontology based QoS Meta model aiming to describe WS QoS metric. In addition there are other QoS ontology models like "WS QoS model^[4]".

A composite service is a service whose implementation calls other services. It organizes the web resources to provide powerful and customized services. Before the services registry advice a composite services to consumers according to the functional requirement, it is important to know whether these services are qualified. Nevertheless, we could not deploy these services in real business environment and detect the QoS value. Therefore, a prediction algorithm is essential to calculate the QoS metric of the composited services according every single service's QoS value and the process.

In this paper, we try to make a novel, extensible, semantic and QoS-aware SOA architecture for composite services. We use OWL-Q to model QoS metric and propose WSQC to specify the QoS constraints in chapter 3, and raise a calculation method to predict the composite services at design time in chapter 4. Finally, we make an experiment in chapter 5. The framework is shown in figure 1.

2. Related work

Now most existing services registry are centralized and only function-aware that restricted service registries' ability to publish and discover Web services. We propose a P2P service registry extension named QMC to solve these problems. QMC provides comprehensive support on QoS such as storing QoS

¹This paper is supported by the National High-Tech Research Development Program of China (863 program) under Grant No. 2007AA01Z139.

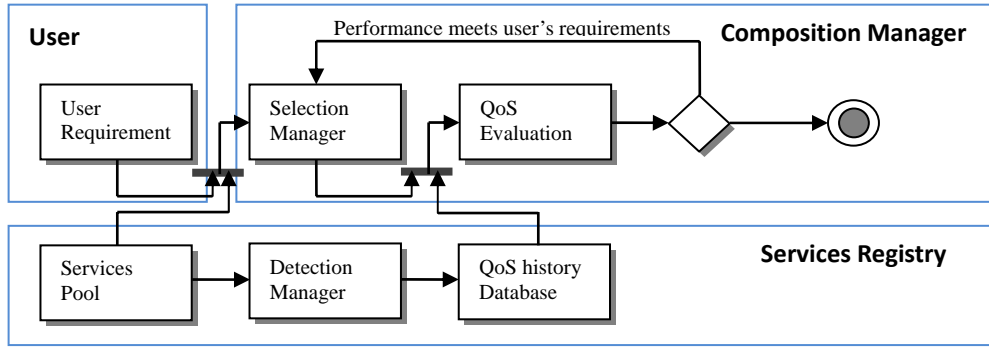


Figure 1: QoS-aware Discovery framework

feedbacks, managing QoS data and handling QoS requests. Moreover, QMC is a system with high scalability and load-balance.

Based on the requirements of QoS-based WS Description, an OWL-S based (syntactical separation) solution is developed, called OWL-Q^[7]. It is a rich, extensible and modular ontology language. Most QoS metric model now is syntactic, poor and not extensible. OWL-Q follows ten rules on WS QoS metric modeling to ensure it is well defined and extensible.

3. WS-QoS Description in Composite Service

In this section, we propose WSQC to describe the QoS constraint for composite services. It's a XML-based formal specification. WSQC references a semantic QoS definition under OWL-Q and supports QoS classification to enhance the flexibility.

3.1 WS-QoS modeling with OWL-Q

As web services turn into a business solution to enterprise application integration, the QoS for web services is becoming increasingly important. However, because of the dynamic and unpredictable characteristics of web services, it is difficult to provide all desired QoS for web service users^[2]. So, we should provide a scalable mechanism which could extend the QoS attribute according to different business domains.

Furthermore, we want to share the domain knowledge with the services providers, register and consumers. Therefore, the model should involve semantic aspect. Ontology was born for that. The use of ontology in computing has gained popularity in recent years for two main reasons: interoperability and machine reasoning^[8].

The Web Ontology Language (OWL) is an xml-base language. It facilitates greater machine interpretability of web content than that supported by XML, RDF, and RDF-Schema by providing additional vocabulary along with a formal semantics^[9].

We use OWL-Q as the Meta model to give a formal definition of our QoS metrics. Following the model, we could extend more QoS metrics. Protégé is a powerful OWL editor created by Stanford and we

use it to write the QoS metrics OWL.

Here we choose some QoS metrics given by the w3c's proposal^[2] and give the calculation method as part of the OWL-Q model:

Performance

Throughput (Tp): The bit number (BN) served in a given time interval (TI).

$$Tp(t)=BN(t)/TI \quad (1)$$

Response Time (RT): Execution time (ET), Delay time (DT)

$$RT(t)=ET(t)+DT(t) \quad (2)$$

Reliability (Re): The number of invoke with response, including abnormal ones (Rli), in the total invoke times (TI) in a give time interval.

$$Re(t)=Rli(t)/TI(t) \quad (3)$$

Availability (Av): The number of invoke with response, not including abnormal ones (RIn), in the total invoke times (TI) in a give time interval.

$$Av(t)=RIn(t)/TI(t) \quad (4)$$

Capacity

Simultaneous Requests (SR): The max simultaneous request number the service could served (MSRN).

$$SR(t)=MSRN(t)/TI \quad (5)$$

3.2 WS-QoS constraints of BPEL

QoS is increasingly in a highlighted position of WS, so a contract between services provider and consumer seems necessary. To satisfy the requirement, QoS specification within Service Level Agreement (SLA) developed. It's a contract between client and service provider. In 2003, IBM released a XML based framework, called Service Level Agreement for web service (WSLA). It's a novel framework for specifying and monitoring SLA for web service^[5]. There are also some other QoS specification languages like WSML, WSOL^[10].

All the QoS-based WS discoveries are facing a common problem: they rely on syntactic or semantically QoS metric description. Different specification occurs for two reasons: a) different

perception for the same concept; b) different system reading for the same metric^[7]. For example, equivalent cost could be associated with different units (e.g. dollar vs. euro). OWL-Q offers the standard WS-QoS description model. We should reference the well defined QoS metrics.

Now, we try to find a solution to support services consumers' QoS constraints of the composite services, like *the price should less than 10 dollars*. In order to make it easy to understand, we use BPEL to represent composite services. Following are the main points of the solution:

- a) The constraints should refer the BPEL in a non-invasive way.
- b) It should provide a standard OWL-Q WS-QoS metrics definition.
- c) It should support the average/worst type of every single QoS metric.
- d) It should support QoS classification.

“Average/Worst type” means it should be applied to the average performance or the worst case. For example, the average response time of the composite service should be less than one second, or every single response time should be less than one second. The latter one means the worst case is one second. Obviously these two cases are quite different and should be treated differently.

The QoS classification aims to deal with the various and unknown QoS metric constraints. We will discuss it in the next section.

We choose a non-invasive way for BPEL to express the constraints rather than extend the BPEL. There are mainly two reasons: firstly, the QoS constraints could be referenced by more than one BEPL. It's reusable. Secondly, the compatibility is better. The BEPL could still works without QoS constraint mechanism support.

Considering these points above, we designed a XML schema to specify users' QoS constraints called Web Service QoS Constraints (WSQC). It mainly divided into three parts: BPEL Reference, Metric Definition and Metric Constraint.

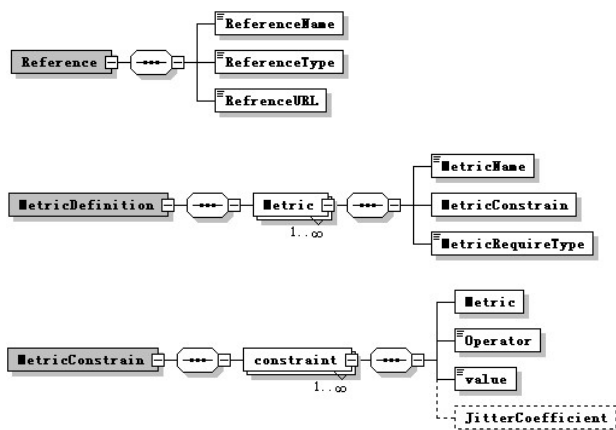


Figure 2: Schema of WSQC

We define “wsqc:MetricCatalog” in other XML schema which refers the WS-QoS OWL definition. “wsqc:MetricClassification”, “wsqc:MetricOperator” in the schema is reusable and extendable.

3.3 WS-QoS Classification

QoS classification is widely used in network, like Streaming Media, to ensure the stability. However, it seldom mentioned in Web Service and the Classification is different from each other according to their own purposes and applications. Jiehan Zhou, Eila Niemelä classifies WS-QoS into five catalogs according to their property: runtime related QoS, transaction support related QoS, configuration management related QoS, cost related QoS and security related QoS^[3].

As the consumers and their business are different, the QoS constraints are various and unpredictable. It conflicts with giving consumers enough flexibility to constraint their demanding QoS.

Fortunately, some QoS metric share the same expression when we calculate the composite service. It means that we can treat some QoS in the same way. We will discuss the detail in chapter 3. Based on that, we give the definition of three catalogs. So customer only need know the catalog of the QoS instead of the exact definition.

There are some concepts to be clarified before give the catalog: *measurable attributes* are measured by specific metrics, like Response Time; *immeasurable rate attributes* cannot be measured directly and is computed by a division operation, like availability; *unique attributes* are not derived by other attributes and are measured by resource metrics, like Response Time; *derived attributes* could be influenced by other attributes and be calculated cooperatively with other metrics, like throughout^[7].

Catalog1: The metric is a **measurable attributes** and **unique**. Like Response Time.

Catalog2: The metric is a **measurable attributes** and **derived**. Like Throughout.

Catalog3: The metric is an **immeasurable rate attributes**. Like Reliability.

4. Composite Web Services QoS calculation

We propose a XML-base method to define the QoS constraints in section 3. The purpose of obtaining the constraints is to filter unqualified services. So in order to build a complete QoS-aware architecture, we discuss “Composite Web Services QoS calculation” in

this chapter.

In the architecture, services registry finds out a set of services according to consumers' functional requirements, and judge whether they satisfy the un-functional requirements. If not, the services registry should replacement some or all of the services till they are in the accepted range.

During the evaluation process in the design time, it is obviously not reasonable to deploy the services in real environment to calculate QoS value. So we design an algorithm to calculate the QoS value of the composite services. We calculate the QoS at design time and inspect the real value at run time.

We consider two different types of users' constraints: one is the **average** response time of the composite services should be less than one second, the other is the response time of the composite services should **strictly** less than one second. Obviously, the two conditions are quite different and should be treated differently. As I mentioned before, in the Web Service QoS Constraints (WSQC) XML file, we used a tag to specify the two different types.

```
<xs:element name="MetricRequireType">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Average"/>
      <xs:enumeration value="Strict"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Figure 3: WSQC fragment of require type

4.1 Composite Service QoS calculation

There are many methods to modeling composite services, and Petri net are wildly used. Stochastic Petri Net (SPN)^[11] is an advanced PN to describe constructs of web service composition. A high-level Petri Net to model workflows at the process level called GWF-net^[12].

We choose stochastic workflow reduction (SWR) algorithm^[13] to calculate the QoS metrics for composite services. The SWR algorithm repeatedly applies a set of reduction rules to a workflow until only one atomic task (Kochut, Sheth et al. 1999) remains. Each time a reduction rule is applied, the

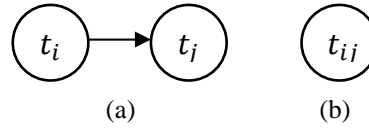
workflow structure changes. After several iterations only one task will remain. When this state is reached, the remaining task contains the QoS metrics corresponding to the workflow under analysis.

If we want to use the reduction, it must base on some assumptions: the user could only use the atomic structure to build the workflow. We recommend BPEL to set up the atomic structure:

- Sequence pattern<sequence>
- Parallel pattern <flow>
- Choice pattern<switch><case><if>
- Iteration pattern <while><repeatUntil><forEach>

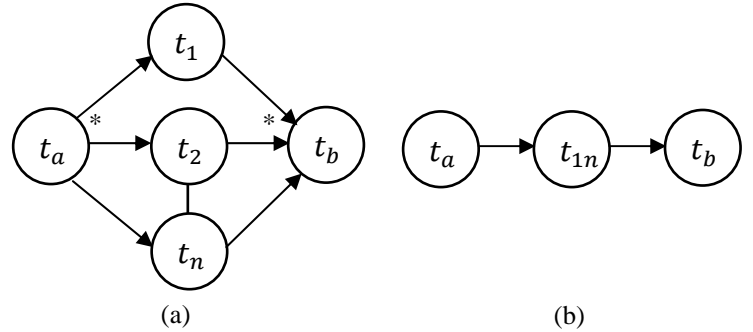
1) Reduction of a Sequential System

Two sequential workflow tasks t_i and t_j can be reduced to a single task t_{ij} . In this reduction, the incoming transitions of t_i and outgoing transition of tasks t_j are transferred to task t_{ij} .



2) Reduction of a Parallel System

A system of parallel tasks t_1, t_2, \dots, t_n , an and split task t_a , and an and join task t_b can be reduced to a sequence of three tasks t_a, t_{1n} , and t_b . In this reduction, the incoming transitions of t_a and the outgoing transition of tasks t_b remain the same.

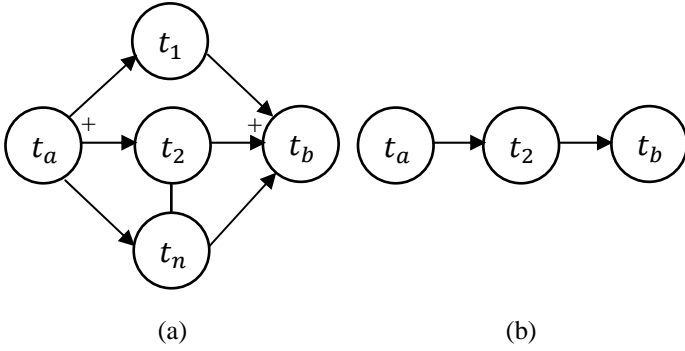


3) Reduction of a Conditional System

A system of conditional tasks t_1, t_2, \dots, t_n , a xor split (task t_a), and a xor join (task t_b) can be reduced to a sequence of three tasks t_a, t_{1n} , and t_b .

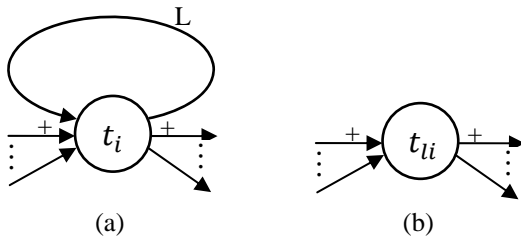
Table 1: function *computeQoS()*

	Catalog1	Catalog2	Catalog3
Sequence	$Q(t_{ij}) = Q(t_i) + Q(t_j)$	$Q(t_{ij}) = \text{Min}\{Q(t_i), Q(t_j)\}$	$Q(t_{ij}) = Q(t_i) * Q(t_j)$
Parallel	$Q(t_{1n}) = \sum_{i=1}^n Q(t_i)$	$Q(t_{1n}) = \text{Max}_{i \in \{1, n\}}\{Q(t_i)\}$	$Q(t_{1n}) = \prod_{i=1}^n Q(t_i)$
Conditional	$Q(t_{1n}) = \sum_{i=1}^n p_{ai} Q(t_i)$	$Q(t_{1n}) = \sum_{i=1}^n p_{ai} Q(t_i)$	$Q(t_{1n}) = \prod_{i=1}^n p_{ai} Q(t_i)$
Loop	$Q(t_{li}) = L * Q(t_i)$	$Q(t_{li}) = L * Q(t_i)$	$Q(t_{li}) = Q(t_i)^L$



4) Reduction of a Loop System

Loop systems can be characterized by simple and dual loop systems. A simple loop system in task t_i can be reduced to a task t_{li} . A dual loop system composed of two tasks t_i and t_j can be reduced to a single task t_{ij}



When the four atomic structures are well defined, we could compute the QoS metrics: we use the *evaluateQoS()* function to get the QoS metrics of an atomic structure according to its atomic type and QoSType. We apply the reduction algorithm to get a new workflow, and treat them recursively until there are only one atomic structure left.

```

1. evaluateQoS(s WS, wsqc WSQC){
2. IF s.structureType ATOMIC THEN {
3.   FOR (each s' WS splitWS(s)) DO
4.     evaluateQoS (s',a);
5.   }
6. var structureType;
7. var QoSType;
8. SWITCH s.structureType {
9.   SEQUENTIAL: structureType='sequential';
10.  BREAK;
11.  PARALLEL SPLIT/JOIN: structureType='parallel';
12.  BREAK;
13.  SLECTIVE: structureType='selective';
14.  BREAK;
15.  LOOP: structureType='loop';
16.  BREAK;
17. }
18. SWITCH wsqc.QoSCatalog{
19.  Catalog1: QoSCatalog =c1; BREAK;
20.  Catalog2: QoSCatalog =c2; BREAK;
21.  Catalog3: QoSCatalog =c3; BREAK;
22. }
23. computeQoS(s,structureType,QoSCatalog);
24.}

```

Figure 4: function *evaluateQoS()*

The function *evaluateQoS()* is recursive. It's shown in figure 4. As we know the BPEL is a tree and we should stop the travelling until we reach the leaf

node such as <invoke>. The function *splitWS(s)* means split the tree into its child trees.

The function *computeQoS()* is the main algorithm to compute the QoS metrics according to different QoS type and structure type. QoS metric of the same catalog will be treated uniformly. It's shown in Table1 *computeQoS()*.

4.2 probability value

We use instantaneous QoS value in the previous section of the prediction algorithm to calculate the composite service. In fact, the QoS value is fluctuant and different metrics may have different amplitude. If the users' require type of QoS constraints is average, not strict, we should not judge the quality of the service by instantaneous value. Every single QoS value is a stochastic variable and the QMC could offer us the historical data of the service's QoS value. We could use these historical data to get the mathematical expectation to compute the stable value in a period.

$$E = \sum_{i=1}^n q_i p_i \quad (6)$$

We substitute the instantaneous QoS value for the mathematical expectation in the table of the *computeQoS()* function. It could represent the quality of the service more objective.

An advanced service should be stable. For example: The response time of the first time is 1s and the second time is 9s. The average response time is 5s. It is worse than the first time is 4s and the second time is 6s. So the QoS metric value of a qualifying service should have a less jitter. We use variance to measure the jitter and the less the jitter is the better the service is.

$$D = \frac{\sum_{i=1}^n (q_i - \bar{q})^2}{n} \quad (7)$$

<xs:element name="JitterCoefficient" minOccurs="0" type="xs:decimal"/>

Figure 5: WSQC fragment of Jitter Coefficient

We have defined a tag in the WSQC to let the user to constrain the variance. The variance must be less than the constraint value but it's not required.

5. Experiment

In order to explain the architecture clearly, we make an example. Firstly, we use Apache ODE tool to create a BPEL which is shown in figure 6. It contains sequence, parallel and loop structures. Secondly, a WSQC file is written to specify the QoS constraint which is shown in figure 7. Further work is publishing five single Web Services and applying them into the workflow (Invoke S1-S5). Finally, the BPEL is deployed on the ODE Engine. Based on the above preparation, we run the BPEL on the engine.

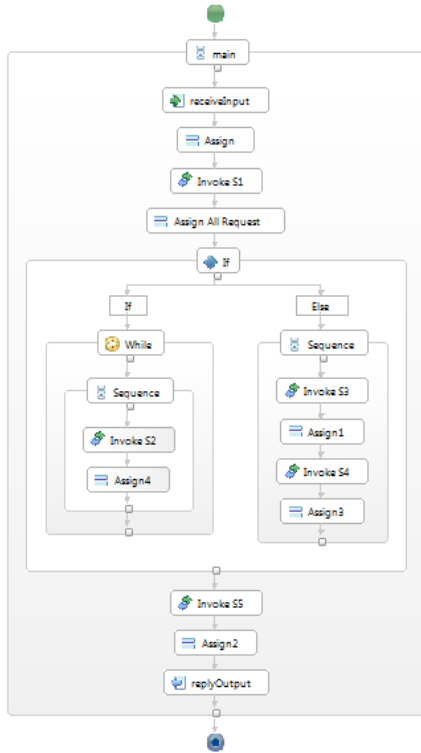


Figure 6: Process of the composite services

```

<Metric>
  <MetricName>responseTime</MetricName>
  <owl-q>WSQC-responseTime</owl-q>
  <MetricRequireType>average</MetricRequireType>
</Metric>

<constraint>
  <ConstraintMetric>ResponseTime</ConstraintMetric>
  <Operator>less</Operator>
  <value>10.0</value>
  <weighing>0.3</weighing>
  <JitterCoefficient>1</JitterCoefficient>
</constraint>

```

Figure 7: WSQC fragment

In the purpose of testing whether the calculation method could get a reasonable result of the QoS value, we detect both every single service and the composite services QoS value. To simplify the problem, we just

detect response time as example.

Then, the function *evaluateQoS()* is used to calculate the composite services' response time according to every single Web Service's response time and the BPEL. After that, we compare the calculated result with the real detected value, and it gets a high accuracy. The result is shown in table 2.

We run every eight times of the BPEL as a round and change the single Web Service's response time every round (R1-R6 in the table). T1-T5 means the response time of these five single services in every round (0-16 second). When the entire single Web Service's response time is zero, the engine still need time to complete the flow. Considering the deviation causing by the engine like print messages in the console, "corrected average" is calculated which means the average response time in the round minus the deviation. The longer the response time is, the more accurate the calculation will be.

As the WSQC constraints: the average response time should less than 10 seconds and the first three rounds' services are qualifying. We successfully filter 50% services.

6. Conclusion and Future Work

The involvement of QoS in WS is becoming more important and it is not an easy task. In this paper, we introduced our ontology-based WS-QoS offering approach and the prediction algorithm for the composite services. Our approach brings an efficient and QoS-aware services selection method in services registry.

The calculation method could treat unknown QoS metrics with the computing type. It also supports two different offering types "strict and average" which should be treating differently. Average offering prefers the service which is relatively stable in a period.

The next step is dynamic replacement of the services. When the composite services' QoS doesn't meet the requirement at design time, we must choose another set of services. But when the number of candidate service is huge, simply traversing is obviously not a good solution. Dynamic replacement is also a big challenge in the whole QoS-aware architecture.

Table 2: response time result

	T1	T2	T3	T4	T5	Real Average	Corrected Average	Prediction	Accuracy
R1	0	0	0	0	0	0.75	0	0	N/A
R2	1.00	1.00	1.00	1.00	1.00	5.31	4.56	4.50	84.9%
R3	2.00	2.00	2.00	2.00	2.00	9.95	9.20	9.00	90.5%
R4	4.00	4.00	4.00	4.00	4.00	18.79	18.04	18.00	95.8%
R5	8.00	8.00	8.00	8.00	8.00	36.79	36.04	36.00	97.9%
R6	16.00	16.00	16.00	16.00	16.00	72.71	71.96	72.00	99.0%

7. References

- [1] Kritikos Kyriakos, Plexousakis Dimitris, "Semantic QoS metric matching", Proceedings of ECOWS 2006: 4th European Conference on Web Services, 2006 December
- [2] W3C QoS for Web Services: Requirements and Possible Approaches, 2003, <http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/>
- [3] Shuping Ran, "A model for web services discovery with QoS", 2003, ACM SIGecom Exchanges
- [4] Tian M., Gramm A., Naumowicz T., Ritter H., Freie J.S., "A concept for QoS integration in Web services" , 2003, Web Information Systems Engineering Workshops, 2003. Proceedings. Fourth International Conference on
- [5] IBM Web Service Level Agreement (WSLA) Language Specification, 2003, <http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf>
- [6] Protege, Ontology Development, 2000, http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html
- [7] Kyriakos Kritikos, Dimitris Plexousakis, "OWL-Q for Semantic QoS-based Web Service Description and Discovery", 2007, First International Joint Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web
- [8] Yan Ha, Hea-Sook Park, "QoS Based on Client information for Semantic Web Service", 2008, Advanced Software Engineering and Its Applications, 2008. ASEA 2008
- [9] W3C: OWL Web Ontology Language, 2004, <http://www.w3.org/TR/owl-features/>
- [10] Jiehan Zhou, Eila Niemela, "Toward Semantic QoS Aware Web Services: Issues, Related Studies and Experience," *wi*, pp.553-557, 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI'06), 2006
- [11] Zhaoli Zhang, Zongkai Yang, Qingtang Liu, "Performance analysis of composite web service", 2008, Granular Computing, 2008. GrC 2008. IEEE International Conference on
- [12] Yunni Xia, Qingsheng Zhu, Yu Huang, Zizhen Wang, "A novel reduction approach to analyzing QoS of workflow processes", 2009, Concurrency and Computation: Practice & Experience
- [13] Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold, Krys Kochut, "Quality of Service for Workflows and Web Service Processes", *Web Semantics*, v 1, n 3, p 281-308, 2004