Service Quality Improvement in Web Service Based Machine Translation

Sapa Chanyachatchawan and Virach Sornlertlamvanich and Thatsanee Charoenporn
National Electronics and Computer Technology Center (NECTEC)
112 Phahon Yothin Rd., Klong 1, Klong Luang, Pathumthani 12120, Thailand
{sapa.cha,virach.sor,thatsanee.cha}@nectec.or.th

Abstract

There are many approaches to increase web service based machine translation result. However, perfect result alone does not guarantee the quality of translation service or user satisfaction. This paper proposes framework to improve translation service by using non functional attributes information. In this paper, we present methodology to measure quality of composite translation service using existing services information and also the guideline for selecting the composition web service which has highest quality of service.

1 Introduction

The advantage of web service based machine translation is the ability to create new language pairs from existing language pairs. This process is based on web service composition in SOA (Service Oriented Architecture). Langrid Project (NICT, 2011) is an example of machine translate service based on web service composition technique. Langrid user can create multihop translation from existing language pairs. The automatic composition process increases accessibility for end users transparently. The most challenging task among the automatic composition processes is the discovery process. Based on W3C, web service description standard (WSDL1.1) defines only input and output name and basic data type for web service with a few descriptions. OWL-S is used to embed semantic into service input and output which enable software agent to discover service. However, translation accuracy does not relate to quality of composite service or user satisfaction. By embedding QoS (Quality of Service) attributes as nonfunctional attributes into web service description, we can improve quality of composite service result. This paper proposes machine translation service framework that can automatically create new language pair from existing resources. The objective of this paper is to provide framework with model to managed nonfunctional attributes and semantic of service. Framework is presented in section 2, where component and overall process are explained in brief. In section 3 discovery process is presented in general idea along with model for evaluate quality of web service and selection method.

2 Framework

In this section, we describe framework for managing web service composition process. The framework is illustrated in Figure 1. From user aspect, our framework is service provider. Users do not need to search and compose web service by themselves. In this paper, existing translation service and composite translation service are called service and composite service respectively.

Component of the framework consists of

- **Proxy Agent** is responsible to interact with users and manage user session.
- **Discovery Agent** is responsible to search and interact with external web service registies.
- **Service Agent** is responsible to invoke external web services.
- **Repository** is responsible for record composite web services and non functional attributes information that do not included in the original WSDL.
- **Compose Agent** is responsible for 1) interact with other component in framework 2) compose web service 3) evaluate web service quality.

The process flow of framework is describe as follow:
Figure 1: Framework

1. User send request to Proxy Agent. User request consists of input, output, and also semantic information.

2. User send request to Proxy Agent. User request consists of input, output, and semantic information.

3. Proxy Agent forwards request to Compose Agent.

4. Compose Agent forwards request parameters to Discovery Agent.

5. Discovery Agent searches services from web service registry.

6. Discovery Agent returns candidate results to Compose Agent.

7. Compose Agent consults Repository if there is additional information about candidate results.

8. Repository returns related information to Compose Agent.

9. Compose Agent evaluates, selects web service, creates workflow, creates instances of Service Agent.

10. Service Agents binds web service.

11. After finish execution, Service Agents returns result to Compose Agent.

12. Compose Agent updates service information in Repository.

13. Compose Agent forwards result to Proxy Agent.

14. Proxy Agent returns output to user.

15. Compose Agent creates WDSL for composite service and forward to Discovery Agent and stores in Repository.

16. Discovery Agent publishes composite service to web service registry.

3 Evaluate quality of services

After discovery process, there are many composite services that satisfy user request. In order to select the highest quality service, quality of service is calculated using non functional attributes of service.

3.1 Non functional attributes of web services

Non function attributes can represent QoS(Quality of Service) which use to differentiate good service from others. QoS values in interval or ratio scale can be concerned as criteria for selecting optimal services. QoS values in nominal scale value use to reduce number of services. Some fundamental attributes will be used in this paper as an example.

For any web service \( S \), the QoS attributes list below:

**Cost of service**: denotes as \( QoS_{\text{Cost}}(S) \), this is the cost to pay for service provider in order to run service. The attributes consist of two parts; first part is taken directly from service provider called direct cost. Second part
is cost for set up and maintenance services called indirect cost which assume that the value is constant for every process throughout the whole system.

**Time of service:** denotes as $QoS_{Time}(S)$, this is the time measure from invoke service to receive respond from service in case of service process successfully. Time of service consists of process time and delay time. *Process time* is time needed to run instance of service, *delay time* is overhead time. This value is kept in Repository and be updated every time that process finish as following equation

$$QoS_{Time}(S) = \frac{((QoS_{Time}(S)_{N-1})*(N-1))+Time_{N}(S)}{N}$$

whereas $QoS_{Time}(S)_N$ is the average time of process after be invoked for $N$ time. This information is kept in Repository.

**Failure ratio:** denotes as $QoS_{Failure}(S)$, is ratio between number of failure and total number of execution. *Failure ratio* initial value is set to 0 and be updated by following formula

In case of service terminate normally;

$$QoS_{Failure}(S) = \frac{((QoS_{Failure}(S)_N)*(N-1))}{N}$$

In case of service terminate abnormally;

$$QoS_{Failure}(S) = \frac{((QoS_{Failure}(S)_N)*(N-1))+1}{N}$$

whereas $QoS_{Failure}(S)_N$ is the failure ratio of process after be invoked for $N$ time. This information is kept in Repository.

**Unavailability:** denotes as $QoS_{Unavl}(S)$, is value to represent the unavailability of services. $QoS_{Unavl}(S)$ is obtained using this formula:

$$QoS_{Unavl}(S) = \frac{T(S)}{C}$$

Whereas, $T(S)$ is total amount of time (seconds) which service $S$ is not available in last $C$ seconds, $C$ is set by framework.

**User satisfaction:** denotes as $QoS_{Sat}$, is cardinal scale value represent satisfaction level of user, this value is variance depend on each users. This information is kept in Repository.

**Security level:** denotes as $QoS_{Sec}$, is cardinal scale value represent security level, this value is taken directly from service or trusted third party providers. This value is taken directly from service providers.

**Bandwidth:** denotes as $QoS_{Band}$, is bandwidth required for running process. This value is taken directly from service providers.

There are number of basic attributes used for measure QoS in streaming application which allow some errors and lossy information.

**Error:** denotes as $QoS_{Error}$, is represent total number of noise and error (in bytes) occur while execute services.

**Delay:** denotes as $QoS_{Delay}$, is total delay and jitter time (in seconds) while execute services.

Some nominal scale non functional attributes that can not be convert to ratio scale, such as user context. These attributes are used to prune web services. Examples of these attributes are:

**Context:** denotes as $QoS_{Context}$, is set of context attributes represent context of users and their environment, such as location, demography information, or user browser environment.

Summarization of non functional attributes is presented in Table 1

### 3.2 Normalize QoS value

In order to compare or measure different attributes, QoS need to be normalized. Each attributes is assigned preference of its value (minimum, maximum). Each attributes are normalized as following:

**Cost of service:** is normalized by using transform table because of cost of service should not be linear function. Table 2 shows the example of normalization of $QoS_{Cost}(S)$.

**Time of service:** is normalized using formula:

$$\frac{QoS_{Time}(S)}{C_{MaxTime}}$$
Table 1: Non functional attributes summarization

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Scale</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QoS(\text{Cost}(S))</td>
<td>Ratio</td>
<td>Service Provider</td>
<td>Service cost</td>
</tr>
<tr>
<td>QoS(\text{Time}(S))</td>
<td>Ratio</td>
<td>Repository</td>
<td>Average execute time</td>
</tr>
<tr>
<td>QoS(\text{Failure}(S))</td>
<td>Ratio</td>
<td>Repository</td>
<td>Failure ratio</td>
</tr>
<tr>
<td>QoS(\text{Unavl}(S))</td>
<td>Ratio</td>
<td>Repository</td>
<td>Unavailability ratio</td>
</tr>
<tr>
<td>QoS(\text{Sat}(S))</td>
<td>Cardinal</td>
<td>Repository</td>
<td>User satisfaction</td>
</tr>
<tr>
<td>QoS(\text{Sec}(S))</td>
<td>Cardinal</td>
<td>Service Provider</td>
<td>Security level</td>
</tr>
<tr>
<td>QoS(\text{Band}(S))</td>
<td>Ratio</td>
<td>Service Provider</td>
<td>Required Bandwidth</td>
</tr>
<tr>
<td>QoS(\text{Error}(S))</td>
<td>Ratio</td>
<td>Repository</td>
<td>Number of error information</td>
</tr>
<tr>
<td>QoS(\text{Delay}(S))</td>
<td>Ratio</td>
<td>Repository</td>
<td>Service delay time</td>
</tr>
<tr>
<td>QoS(\text{Context}(S))</td>
<td>Nominal</td>
<td>User</td>
<td>User context</td>
</tr>
</tbody>
</table>

Table 2: Cost of service transform table example

<table>
<thead>
<tr>
<th>Cost(Dollars)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-0.5</td>
<td>0.0</td>
</tr>
<tr>
<td>0.5-1</td>
<td>0.3</td>
</tr>
<tr>
<td>1-5</td>
<td>0.5</td>
</tr>
<tr>
<td>5-10</td>
<td>0.8</td>
</tr>
<tr>
<td>≥ 5</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Whereas \(C_{\text{MaxTime}}\) is the maximum execute time assigned by framework.

**Failure ratio**: does not need to be normalized, because \(QoS_{\text{Failure}}(S) \in [0, 1]\)

**Unavailability**: does not need to be normalized, because \(QoS_{\text{Unavl}}(S) \in [0, 1]\)

**User satisfaction**: is not normalized and will be used as constraint.

**Security level**: is not normalized and will be used as constraint.

**Bandwidth**: is normalized using formula:
\[
\frac{QoS_{\text{Band}}(S)}{C_{\text{MaxBand}}}
\]

Whereas \(C_{\text{MaxBand}}\) is the maximum bandwidth of framework.

**Error**: is normalized using formula:
\[
\frac{QoS_{\text{Error}}-C_{\text{MinError}}}{C_{\text{MaxError}}-C_{\text{MinError}}}
\]

Whereas \(C_{\text{MinError}}\) and \(C_{\text{MaxError}}\) are minimum and maximum error that framework allow to happen.

**Delay**: is normalized using formula:
\[
\frac{QoS_{\text{Delay}}-C_{\text{MinDelay}}}{C_{\text{MaxDelay}}-C_{\text{MinDelay}}}
\]

Whereas \(C_{\text{MinDelay}}\) and \(C_{\text{MaxDelay}}\) are minimum and maximum delay that framework allow to happen.

**Context**: can not be normalized and will be used as constraint.

Table 3 is the summarization of normalized attributes.

3.3 Quality of composite service

Once the service has been composed, the \(QoS\) of composite service will be calculated. Workflow of composite service determines how \(QoS\) be computed. Workflow of composite service is divided into four types: 1) sequential, 2) parallel, 3) conditional, 4) loop, and 5) complex. Parallel, conditional, loop and complex workflow are reduced into one atomic task. As the result of reduction, new workflow will consist of sequential workflow only, and then sequential workflow is reduced to one atomic workflow. Only ratio scale and interval scale attributes will computed here. Hence, \(QoS\) of composite service is calculated.

3.3.1 Sequential workflow

Sequential workflow \(CS\) (Figure 2) consists of \(n\) sequential process denote as \(S_i; 1 \leq i \leq n\). The work flow start at service \(S_1\) and finish at service \(S_n\). Process \(Si\) must finish before process \(S_{i+1}\) can start.

The \(QoS\) of \(CS\) can be obtained as follows:
\[
QoS_{\text{Time}}(CS) = \sum_{i=1}^{n} QoS_{\text{Time}}(S_i)
\]
Table 3: Normalized functional attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Range</th>
<th>Preferred value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QoS(_{Cost}(S))</td>
<td>[0, 1]</td>
<td>Minimum</td>
<td>Service cost</td>
</tr>
<tr>
<td>QoS(_{Time}(S))</td>
<td>[0, 1]</td>
<td>Minimum</td>
<td>Average execute time</td>
</tr>
<tr>
<td>QoS(_{Failure}(S))</td>
<td>[0, 1]</td>
<td>Minimum</td>
<td>Failure ratio</td>
</tr>
<tr>
<td>QoS(_{Unavail}(S))</td>
<td>[0, 1]</td>
<td>Minimum</td>
<td>Unavailability ratio</td>
</tr>
<tr>
<td>QoS(_{Sat}(S))</td>
<td>-</td>
<td>-</td>
<td>User satisfaction</td>
</tr>
<tr>
<td>QoS(_{Sec}(S))</td>
<td>-</td>
<td>-</td>
<td>Security level</td>
</tr>
<tr>
<td>QoS(_{Band}(S))</td>
<td>[0, 1]</td>
<td>Minimum</td>
<td>Required Bandwidth</td>
</tr>
<tr>
<td>QoS(_{Error}(S))</td>
<td>[0, 1]</td>
<td>Minimum</td>
<td>Number of error information</td>
</tr>
<tr>
<td>QoS(_{Delay}(S))</td>
<td>[0, 1]</td>
<td>Minimum</td>
<td>Service delay time</td>
</tr>
<tr>
<td>QoS(_{Context}(S))</td>
<td>-</td>
<td>-</td>
<td>User context</td>
</tr>
</tbody>
</table>

3.3.3 Conditional workflow

Conditional workflow CS (Figure 4) consists of \( n \) process denoted as \( S_i; 1 \leq i \leq n \), only one process will be execute. \( p_i \) is the probability of process \( S_i \) to be execute and \( \sum_{i=1}^{n} p_i = 1 \), these value store from Repository.

\[
\begin{align*}
QoS_{Cost}(CS) &= \sum_{i=1}^{n} QoS_{Cost}(S_i) \\
QoS_{Time}(CS) &= 1 - \prod_{i=1}^{n}(1 - QoS_{Failure}(S_i)) \\
QoS_{Unavail}(CS) &= 1 - \prod_{i=1}^{n}(1 - QoS_{Unavail}(S_i)) \\
QoS_{Band}(CS) &= MAX_{1 \leq i \leq n}(QoS_{Band}(S_i)) \\
QoS_{Error}(CS) &= \sum_{i=1}^{n} QoS_{Error}(S_i) \\
QoS_{Delay}(CS) &= \sum_{i=1}^{n} QoS_{Delay}(S_i)
\end{align*}
\]

3.3.4 Loop workflow

For loop workflow (Figure 5), there is condition of loop to simplify the calculation. Give CS is composite service that created by repeat execution of service S with \( p \) is the chance that service will be repeat and loop must be execute service S at least one time.

The QoS of CS can be obtained as follows:
\[
\begin{align*}
QoS_{Time}(CS) &= \sum_{i=1}^{n} (p_i \ast QoS_{Time}(S_i)) \\
QoS_{Cost}(CS) &= \sum_{i=1}^{n} (p_i \ast QoS_{Cost}(S_i)) \\
QoS_{Failure}(CS) &= \sum_{i=1}^{n} (p_i \ast QoS_{Failure}(S_i)) \\
QoS_{Unavail}(CS) &= \sum_{i=1}^{n} (p_i \ast QoS_{Unavail}(S_i)) \\
QoS_{Band}(CS) &= \sum_{i=1}^{n} (p_i \ast QoS_{Band}(S_i)) \\
QoS_{Error}(CS) &= \sum_{i=1}^{n} (p_i \ast QoS_{Error}(S_i)) \\
QoS_{Delay}(CS) &= \sum_{i=1}^{n} (p_i \ast QoS_{Delay}(S_i))
\end{align*}
\]
3.3.5 Complex workflow

Complex workflow $CS$ (Figure 6) consists of $N$ processes denoted as $S_i; 1 \leq i \leq n$, it is an acyclic directed graph.

![Figure 6: Complex workflow](image)

The QoS of $CS$ can be obtained as follows:

$$QoS_{Cost}(CS) = \sum_{i=1}^{n} QoS_{Cost}(S_i)$$

$$QoS_{Failure}(CS) = 1 - \prod_{i=1}^{n} (1 - QoS_{Failure}(S_i))$$

$$QoS_{Unavl}(CS) = 1 - \prod_{i=1}^{n} (1 - QoS_{Unavl}(S_i))$$

$$QoS_{Error}(CS) = \prod_{i=1}^{n} QoS_{Error}(S_i)$$

$$QoS_{Delay}(CS) = \prod_{i=1}^{n} QoS_{Delay}(S_i)$$

3.4 Objective function

Objective function is used to evaluate the fitness of a composite service. As many attributes are considered, the single unique value is needed for comparison between each possible combination. In framework, $QoS$ of composite service is defined by formula:

$$QoS(X) = \sum_{i=1}^{n} w_i \cdot QoS_i(X)$$

whereas $N$=number of attributes; $w_i$=weight of attribute $i^{th}$

Our objective is to find composite service that have minimum $QoS$ value, thus objective function will be

$$\min QoS(X) = \min (\sum_{i=1}^{n} w_i \cdot QoS_i(X))$$

some constraints are defined for composite service to represent real life constraints.

$$QoS_i(CS) \leq C_i$$ for each $i \in 1, ..., N$

whereas $N$=number of attributes; $C_i$=constraint of attribute $i^{th}$

3.5 Selecting web service

$QoS$ function from previous section consists of non-linear parameter which make calculation complex. To simplify problem, some assumptions are given 1) suppose that only one possible workflow returned from discovery process 2) composite service is not streaming application 3) user context is irrelevant. 4) workflow consist only sequential processes. Figure 7 show an example of such a workflow.

Cardinal and nominal scale $QoS$ and non-linear composite $QoS$ ($QoS_{Failure}$, $QoS_{Unavl}$, $QoS_{Band}$, $QoS_{Sat}$ and $QoS_{Sec}$) are excluded from objective function and used to prune to discovered services. Hence, workflow objective function and constraints are solely linear function, and then 0-1 linear programming model is applied.

![Figure 7: Output from discovery process](image)
The workflow consists of $n$ processes, each processes there is set or service with size $m_i$; $1 \leq i \leq n$ that can fulfill process requirements.

Then we introduce set of variable $x_{ij}$ to represent decision variable.

\[
\begin{pmatrix}
  x_{11} & x_{n1} \\
  \vdots & \ddots \\
  x_{1k} & x_{nm_i}
\end{pmatrix}
\]

The variable $x_{ij}$ is correspond to $S_{ij}$; $1 \leq i \leq n$ and $1 \leq j \leq m_i$. Whereas $n$ is number of process. $x_{ij} = 1$ iff service $x_{ij}$ has been selected, otherwise $x_{ij} = 0$.

Hence, problem is transformed to linear programming problem:

minimize: $\sum_{i=1}^{n} \sum_{j=1}^{m_i} QoS(S_{ij}) \times x_{ij}$

whereas $QoS(S_{ij}) = (w_{Time} \times QoS_{Time}(S_{ij})) + (w_{Cost} \times QoS_{Cost}(S_{ij}))$

subject to

\[
\begin{align*}
\sum_{j=1}^{m_i} x_{ij} &= 1 \\
x_{ij} &\in \{0, 1\} \\
QoS_{Time}(S_{ij}) &\leq C_{Time} \\
QoS_{Cost}(S_{ij}) &\leq C_{Cost}
\end{align*}
\]

4 Update repository information

After composite that have the best $QoS$ has been selected and executed, there are processes after finish. There are two cases 1) composite service terminate normally 2) composite service terminate abnormally. In later case, we update service information ($QoS_{Failure}$) in Repository. Process will not repeat because of services that makes composite service fail tends to have better $QoS$ value than others. As the result, other combination of this service must be excluded and rediscover web services again. In case of composite service terminate normally, service information ($QoS_{Failure}$, and $QoS_{Time}$) is updated in to Repository, and publish composite service to Web service Registry with $QoS$ information. $QoS$ information can be added to WSDL as extension (Unreaveling the web services: an introduction to SOAP, WSDL, and UDDI)(Curbera, 2002), using Semantic Annotations for WSDL or using OWL-S.

5 Related works

There are many related studies about quality of machine translation notably ones include Automatic Evaluation of Machine Translation Quality Using Longest Common Subsequence and Skip-Bigram Statistics (Lin and Och, 2004) and METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments (Banerjee and Lavie, 2005). There are researches about quality of service in service composition process such as QoS-Aware Middleware for Web Services Composition - A Qualitative Approach (Yeom, Yun and Min, 2006). There are two important processes that we do not focus in our framework. The first process is discovery process which are managed by Discovery Agent in our framework. Algorithms for discovery services are not included in this paper. There are many related studies in searching non perfect match web service such as Automate Composition and Reliable Execution of Ad-hoc Processes (Binder, Constantinescu, Faltings, Haller, and Turker, 2004) and A software Framework For Matchmaking Based on Semantic Web Technology (Li and Horrocks, 2003). Other research work as in Ontology assisted Web services discover (Zhang and Li, 2005) and Web Service Discovery via Semantic Association Ranking and Hyperclique Pattern Discovery (Paliwal, Adam, Xiong and Bornhovd, 2006) use semantic information to discover web services.

The second process is how to search for the optimal quality composite service from all possible combination of services. We can apply linear programming technique as described cutting method in A lift-and-project cutting plane algorithm for mixed 01 programs (Balas, Ceria and Cornuejols, 1993) to perform this search task.

6 Conclusion and future work

The main contribution of this paper is to propose a web service based machine translation framework that enhances quality of translation. We present the concept of embedding quality of service information, method to measurement QoS, and calculation of composite translation QoS.

For future work, we plan to work on simplifying the search space, discovery techniques using semantic, mathematic model for solving integer programming problem, fault tolerance, and implementation of ontology to describe QoS attributes in services.
References

Language Grid. 2011. Language Grid

http://hadm.sph.sc.edu/COURSES/J716/CPM/CPM.html


