Towards QoS-Based Web Services Discovery

Jun Yan and Jingtai Piao

School of Information Systems and Technology
University of Wollongong
Wollongong, NSW, Australia 2522
{jyan, jp928}@uow.edu.au

Abstract. The current UDDI-based web services discovery technologies are designed to discover services which can satisfy consumers' functional requirements. The consumers' non-functional requirements such as quality of services are largely ignored in discovery. This may lead to the problem that the services returned from discovery are ineffective and even useless. To solve this problem, this paper presents an approach to achieving QoS-based Web services discovery. Data structures are proposed for both service providers and service consumers to describe non-functional information about services. A series of algorithms are developed for matching and ranking services according to consumers' non-functional requirements.

1 Introduction

Web services is an emerging technology which provides a machine to machine interaction over a network by using a series of standardized technologies, including WSDL, SOAP and UDDI [1]. Of these technologies, UDDI is designed to be interrogated by SOAP messages, as well as to provide an access to WSDL documents which describe the protocol bindings and messages [3]. With UDDI in place, the advertisement, discovery, and binding of web services through the Internet can be achieved. IBM, SAP and Microsoft established a publicly accessible Universal Business Registry (UBR) and this is recognized as a trial of the UDDI specification. However, the adoption rate of UDDI is still remained at a low level [1]. One of the significant reasons for this stagnation is that the existing Web services discovery framework only supports discovery of services which functionally match with the user request. Discovery with consideration of services' non-functional features has been largely ignored [1].

Services' non-functional features can be best represented with the Quality of Service (QoS) offered by the web services, such as price, response time, reliability, and so on [4]. In the real world, there may be a set of web services which offer the same functional features but current UDDI cannot distinguish these functionally equivalent services. QoS should be a key factor to differentiate the services with similar functionalities [9] [6]. When a service consumer queries for such functional features, only those services that also satisfy the consumer's non-functional requirements are of interests to this consumer [2]. Service discovery based on functional matching only

can lead to severe problems. For example, those services with much longer-than-expected response time or much lower-than-expected availability may be presented to a service consumer as results of service discovery. Subsequently, the performance of the consumer application can be largely deteriorated due to the use of these services. For this reason, QoS-based service discovery which aims at discovering services which meet both functional and non-functional requirements of service consumers becomes critical to the wide adoption of Web services in practice.

To address the above problem, this paper presents an approach towards QoS-based Web services discovery through the extension of the current UDDI service discovery framework. This research primarily focuses on two areas. First, this paper proposes a generic model to represent QoS information in service advertisements as well as QoS requirements in service discovery requests. Based on this model, a matching algorithm and a ranking algorithm are presented. Given a service discovery request, the matching algorithm compares its QoS requirements with the QoS advertisements in the repository and locates those services with matching QoS. The ranking algorithm furthers ranks these discovered services to facilitate the consumers to select services.

The remainder of this paper is organized as follows. Section 2 introduces major related work. Then, Section 3 presents a XML-based model to represent the QoS information, and an extension to UDDI tModel to store QoS information. In Section 4, QoS matching and ranking algorithms for service discovery are discussed based on the proposed QoS model. Finally, Section 5 concludes this paper and outlines our future work.

2 Related Work

Extension of UDDI with consideration of QoS has been recognized as a key solution of distinguishing the web service discovery results with same functionality [1][15][20]. The first step to achieving this is the active development of QoS information representation approaches. To name a few, Mani et al [19] claimed that the major QoS requirements are performance, reliability, security, etc. Ran [1] categorized the QoS information as three domains, namely run-time related QoS, transaction related QoS and security related QoS. Dobson et al [8] proposed a reusable generic QoS ontology by categorizing the QoS information as measurable and immeasurable. Maximilien et al [7] intended to develop a comprehensive QoS ontology including the description of the QoS relationship. Moreover, Zhou et al [6] provided a novel DAML-QoS to establish a three-layer QoS ontology, including profile layer, property layer and metrics layer. Bianchini et al [10] proposed an approach which allows users to submit their QoS requirements semantically, using an ontology which categorizes the QoS attributes as numeric, Boolean and enumeration. Although these representation approaches are successful in capturing QoS information from different perspectives, the flexibility of syntactic QoS model is still lack and the semantic QoS model is relatively complex and time consuming.

Embedding QoS information within UDDI is another important research issue. Adam Blum and Fred Cater [11] stated four different QoS information sorting methods by extend tModel and bindingTemplate within UDDI. D'Melloet et al [16] introduced a QoS repository as a QoS broker to store and interact with QoS information. Similarly, Yu and Lin [17] proposed a QoS capable web service (QCWS) architecture

which is operated as a QoS broker to manage the interaction of QoS information between client and service provider. Furthermore, QCWS is designed have the capability to allocated resource to the clients.

QoS matching and ranking is the key in QoS-based service discovery. The existing work covers both syntactic matching and semantic matching. Yang and Huangcan [12] treated the QoS model as an N dimensional matrix. The priority of QoS can be described as the distance from the spot of the requirement to the provided spot in the QoS space. Wenli [15] suggested utilizing fuzzy logic to calculate a threshold value of QoS information. The services which are out of this threshold will not be discovered. Kritikos and Plexousakis [14] proposed a matching algorithm, which transfers the QoS information into Constraint Satisfaction Problem (CSP) and the matching progress is performed based on the consistency of the CSP. Bianchini et al [10] categorized the QoS information into three types, namely numeric, Boolean and enumeration and the users' QoS requirement of each type can be Good, Poor and Fair etc. The service discovery process is performed by matching those categories. Giallonardo and Zimeo [17] provided an ontology-based matching approach to avoid one-by-one QoS parameter matching failures which are caused by misunderstanding the terminology of QoS attributes.

3 QoS Information Description

This section presents a novel model to represent both service providers' QoS advertisements and service consumers' QoS requirements. This model is generic and can easily be extensible to provide comprehensive and explicit description of QoS information.

3.1 QoS Information Advertisement

A service provider can conventionally publish its service information such as *businessEntity*, *businessService*, and *bindingTemplate*. In order to publish its QoS capabilities, a service provider needs to include an entity called *QoSInformation* in its advertisement. As shown in Figure 1, the *QoSInformation* can contain information of one or more QoS attributes, each of which is described using a five-tuple, *<attributeName, attributeType, attributeValue, attributeUnit, constraints>*.

The element *attributeName* represents the name of the QoS attribute, such as price, response time, availability, and so on. The element *attributeType* represents the type of the QoS attribute. In this research, three types of QoS attributes are introduced, including *Numeric QoS attribute*, *Boolean QoS attribute*, and *Enumeration QoS attribute*. In the matching process, the different algorithm will be invoked according to the type of attribute. The element *attributeUnit* defines the measurement units of the QoS attribute. For example, the *attributeUnit* of the attribute price can be DOLLAR and the *attributeUnit* of the attribute response time can be DAY or HOUR. The element *attributeValue*, represented in different forms according to the type of the QoS attribute, defines the advertised value of the QoS attribute. For Numeric QoS attribute, two numeric values describe the value range of this attribute as an interval. For Boolean QoS attribute, the *attributeValue* can be either TRUE or FALSE. For Enumeration QoS attribute, the *attributeValue* is a discrete value set. The element *constraint* allows service providers to publish the constraints of their services, such as the

server maintenance time, upgrade information, price adjustment and so on. The following sample advertisement <execution time, numeric, [0, 3], hour, (between 9am and 5pm, Monday to Friday)> describes a service provider's QoS promises on attribute execution time. From this advertisement, it is clear that the execution time of this service is no more than 3 hours between 9am and 5pm, Monday to Friday.

Fig. 1. Structure of QoS Advertisement

3.2 Storage of QoS information

Service providers' QoS information needs to be embedded in the current UDDI framework for it to be used in discovery. To achieve this, a new tModel called QoS Information tModel is created, as depicted in Figure 2. The element overviewURL of this tModel is referred to an external file which stores all the QoS information about this service. Such an approach requires minimum modification to the current UDDI data structure. The external file can be hosted by a third party or even by the service provider.

```

- <tModel tModelKey="uuid:xxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx">
  <name>qualityInformation</name>
  <description>Quality of Service Information</description>
- <overviewDoc>
  <description xml:lang="en" />
  <overviewURL>externalQoSURL</overviewURL>
</overviewDoc>
<categoryBag>....</categoryBag>
</tModel>
  
```

Fig. 2. QoS Information tModel

3.3 QoS Requirement

In order for a service consumer to discover "right" services, the service discovery request should contain information to represent the QoS requirements of the service consumer, in addition to functional requirements. As shown in Figure 3, a service consumer can include requirements on one or more QoS attributes in the service discovery request. The requirement on a QoS attribute can either be compulsory or optional, which will be treated differently in matching and ranking.

Fig. 3. Structure of QoS requirement

A compulsory requirement attribute is described by the following eight-tuple:

<attributeName, attributeType, attributeValue, attributeUnit, constraint, weight, direction, relationship>

The first four elements are of the same meanings as those discussed in Section 3.1. The element *constraint* which can be either *belongTo* or *notBelongTo* is introduced to illustrate the restriction of the *attributeValue*. For instance, assuming that a user does not need a service between 1pm to 3pm, this requirement can be described by this model, using an interval [1, 3] and a *constraint* value *notBelongTo*. The element *weight* indicates the importance of this attribute to the service consumer. The value of *weight* should be a number between 0 and 1 and it should be assigned by the consumer. This attribute is introduced to rank the services in the ranking process. The element *direction* only used to describe numeric attributes to show the expected tendency of the attribute. The *positive* direction means that user expects an increasing trend of this numeric attribute, whereas *negative* shows that user expects a decreasing trend of this attribute. Both *weight* and *direction* are used in the ranking algorithm discussed in Section 4 to compare discovered services. Finally, the element *relationship* is used to describe the complex requirements on the relationship of two QoS attributes.

The relationships between attributes are used to trade-off or negotiate with the service discovery system. This paper only provides a way to describe the relationship between two compulsory attributes. In a statement of a relationship, the related two attributes are treated as piecewise function. The value of the dependent attribute is contingent on the independent attribute. Furthermore, the omission of *independentAttributeName* and *independentAttributeType* is because these two attributes map with

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <qosRequirement>
3    <compulsoryRequirement>
4      <qosAttribute name = "availability">
5        <attributeName>availability</attributeName>
6        <attributeType>numeric</attributeType>
7        <attributeValue>0.99.99</attributeValue>
8        <attributeUnit>percent</attributeUnit>
9        <constraint>belongTo</constraint>
10       <weight>0.2</weight>
11       <direction>positive</direction>
12       <relationship>
13         <independentAttributeValue>99.99.99.9999</independentAttributeValue>
14         <dependentAttributeName>encryption</dependentAttributeName>
15         <dependentAttributeValue>false</dependentAttributeValue>
16         <constraint></constraint>
17       </relationship>
18     </qosAttribute>
19     <qosAttribute name = "encryption">
20       <attributeName>encryption</attributeName>
21       <attributeType>boolean</attributeType>
22       <attributeValue>true</attributeValue>
23       <constraint>belongTo</constraint>
24       <weight>null</weight>
25       <direction>null</direction>
26       <relationship>null</relationship>
27     </qosAttribute>
28   </compulsoryRequirement>
29   <optionalRequirement>
30     <attributeName>resolution</attributeName>
31     <attributeType>enumeration</attributeType>
32     <attributeValue>600,800,1024,768,1680,1050</attributeValue>
33     <attributeUnit>pixel</attributeUnit>
34     <constraint>belongTo</constraint>
35     <weight>0.8</weight>
36     <direction>null</direction>
37   </optionalRequirement>
38 </qosRequirement>

```

Fig. 4. A Sample of QoS Requirement

the content of the *attributeName* and *attributeType* which are located in the description of the attribute. Similarly, it is not necessary to reclaim the *dependentAttributeType* because it is mapped to the attribute according to the *dependentAttributeName*.

3.4 A Sample of QoS Requirement

In this section, the data structure of the proposed QoS requirement is realized by a XML document. In the sample shown in Figure 4, the user has two compulsory requirements about availability and resolution and an optional requirement about encryption. Moreover, the requirement includes a relationship between availability

and resolution, namely while the availability more than 99.99 percent and less than 99.9999percent then the resolution can be 600×800 .

4 Matching and Ranking Algorithms

The matching algorithm is to locate the services with matching QoS information, whereas ranking algorithm is in charge of present them to a consumer in an order that may best reflect the consumer' interests.

4.1 Matching Algorithm

Given a service request R which contains compulsory requirements on n QoS attributes $A_1, A_2, \dots, A_n$, the objective of the matching algorithm is to select, among those services that provide the required functions of R , services that satisfy these QoS requirements. The strategy is to examine these requirements one by one to obtain service sets $S_1, S_2, \dots, S_n$ which represent the set of services that satisfy n compulsory QoS requirements, respectively. For a compulsory requirement on a numeric type attribute, the upper bound and lower bound of the interval will be compared between the requirement and the advertisement respectively. If the upper bound of the requested interval is larger than the upper bound of published interval and the lower bound of requested interval is smaller than the lower bound of published interval then this service should be selected, otherwise this service should be discarded. For a compulsory requirement on a Boolean type attribute, if the value of a requested Boolean attribute matches with the published Boolean attribute then this service should be selected, otherwise, the service should be discarded. For a compulsory requirement on an enumeration type attribute, if the published enumeration attribute set E_p is a subset of the request attribute set E_R , then the service should be selected, otherwise, the service should be discarded. Finally, the result of the matching service set S is calculated by the following equation.

$$S = \bigcap_{i=1}^n S_i \quad (1)$$

The Detailed Matching Algorithm Is Shown in Figure 5.

Using symbol S_{R_i} to represent the service set that satisfies the relationship statements of the attribute A_i ; whereas S_C indicates the service set that satisfies the *constraint* statement of this attribute. The service set S_i should be the union of S_{R_i} and S_{C_i} . If there are multiple relationships, the formula can be summarized as follows:

$$S_i = (\bigcup_{i=1}^n S_{R_i}) \bigcup S_C \quad (2)$$

```

Input: serviceSet ( $S_0, S_1, \dots, S_n$ )
Input: requirement
1. for ( $i = 0, i < n, i++$ )
2.   if ( $U_R \geq U_P \parallel L_R \leq L_P$ ) // The  $U_R$  indicates the upper bound of
   // requirement interval, whereas  $U_P$  indicates the upper bound of published
   // service interval.
3.   return  $S_{numeric} = S_n$  // Retrieve all the services which satisfy the
   // numeric attribute
4.   else discard  $S_n$ 
5.   end if
6. end for
7. for ( $i = 0, i < n, i++$ )
8.   if ( $B_R = B_P$ ) // The  $B_R$  indicates the Boolean attribute of requirement,
   // whereas  $B_P$  indicates Boolean attribute of published service.
9.   return  $S_{boolean} = S_n$  // Retrieve all the services which satisfy the
   // Boolean attribute.
10.  else discard  $S_n$ 
11.  end if
12. end for
13. for ( $i = 0, i < n, i++$ )
14.  if ( $E_R \supset E_P$ ) // The  $E_R$  indicates the enumeration attribute of
   // requirement, whereas  $E_P$  indicates enumeration attribute of published
   // service.
15.  return  $S_{enumeration} = S_n$  // Retrieve all the services which satisfy the
   // enumeration attribute.
16.  else discard  $S_n$ 
17.  end if
18. end for
19.  $S = S_{numeric} \cap S_{boolean} \cap S_{enumeration}$  // Final result of matching algorithm.
End algorithm

```

Fig. 5. Pseudocode of Matching Algorithm

Thus, combining formulas (1) and (2), the final matching formula becomes:

$$S = \bigcap_{i=1}^n ((\bigcup_{i=1}^n S_{R_i}) \cup S_C) \quad (3)$$

4.2 Ranking Algorithm

If the matching algorithm returns more than one matched service, it is a desirable feature to rank these services in an order that best represents the consumer's preference. The ranking measures the "goodness" of the matched services and makes recommendation to the consumers according to their needs.

The ranking algorithm takes both the compulsory and optional QoS requirements into consideration. In order to compute the preference of users, QoS metrics are introduced in this approach. Firstly, all of the published QoS attributes of service p are modeled as a vector $A_p = \{A_{p,1}, A_{p,2}, \dots, A_{p,n}\}$ (n is the number of attributes) Secondly, the QoS requirements in the service request can be modeled as a n -dimension binary vector as well, namely, $A_r = ((A_1, w_1), (A_2, w_2), \dots, (A_n, w_n))$,

where $\sum_{i=1}^n w_i = 1, w_i \in [0,1]$. w_i is the weight of the attribute i which is normally given

by the service consumer. Thirdly, a distance $dis(A_p, A_r)$ between the requested

vector A_r and the published vector A_p could be calculated. It is believed that a service that is more close to the requirements would satisfy the consumer better.

Based on the above, the situation that there is m services that each contains n QoS attributes can be modeled as an $m \times n$ matrix Q :

$$Q = \begin{matrix} & A_{1,1} & A_{1,2} & \dots & A_{1,n} \\ \begin{matrix} A_{2,1} \\ A_{2,2} \\ \dots \\ A_{m,1} \end{matrix} & \begin{matrix} A_{2,1} \\ A_{2,2} \\ \dots \\ A_{m,2} \end{matrix} & \begin{matrix} A_{2,2} \\ \dots \\ A_{m,2} \end{matrix} & \begin{matrix} \dots \\ \dots \\ \dots \end{matrix} & \begin{matrix} A_{2,n} \\ \dots \\ A_{m,n} \end{matrix} \end{matrix}$$

Then, the normalization process should be applied. This process is conducted according the value of direction as well. Each element in the Q matrix should be normalized into the interval $[0, 1]$. If the direction is positive, then the formula (4) will be used to normalize. In contrast, if the direction is negative, the formula (5) will be used.

$$Nor(A_{i,k}) = \frac{\max A_{i,k} - A_{i,k}}{\max A_{i,k} - \min A_{i,k}} \quad (i = 1, 2, \dots, m); (k = 1, 2, \dots, n) \quad (4)$$

$$Nor(A_{i,k}) = \frac{A_{i,k} - \min A_{i,k}}{\max A_{i,k} - \min A_{i,k}} \quad (i = 1, 2, \dots, m); (k = 1, 2, \dots, n) \quad (5)$$

After that, matrix Q should be merged with the requirement vector A_r to get a $(m+1) \times n$ matrix Z .

$$Z = \begin{vmatrix} A_{r_1} & A_{r_2} & \dots & A_{r_n} \\ A_{1,1} & A_{1,2} & \dots & A_{1,n} \\ A_{2,1} & A_{2,2} & \dots & A_{2,n} \\ \dots & \dots & \dots & \dots \\ A_{m,1} & A_{m,2} & \dots & A_{m,n} \end{vmatrix}$$

The requirement vector A_r should be optimized in order to indicate a QoS destination in the QoS space, namely all of the attributes of requirement adopt the best value. Rather, *Direction* represents the user preference tendency, the optimized numeric value should be the biggest or the smallest accordingly. For the Boolean and enumeration type attributes, the optimized value should be 1 which is used to indicate the meaning of "true".

Finally, using the formula (6) to calculate the distance between A_p and A_r .

$$dis(A_p, A_r) = \sqrt{\sum_{i=1}^m W(A_{i,j} - A_{r_j})^2} \quad (6)$$

Comparing the distance $dis(A_p, A_r)$, the ranking list can be generated. If the requested optional QoS attribute is absent within the register service then set the distance as 1.

5 Conclusion and Future Work

The lack of consideration of non-functional requirements in service discovery may lead to the problem that the retrieved services cannot guarantee the quality of the services. This paper made an important step towards fully functional QoS-based service discovery by proposing a novel approach to extending UDDI with consideration of QoS, including QoS information description, QoS information storage and QoS information discovery. More specifically, this paper has established a XML data structure to represent QoS information of both service providers and service consumers. A QoS information tModel has been introduced to reference these QoS information documents. In addition, a QoS matching algorithm and a QoS ranking algorithm have been presented to make use of the QoS information in service discovery.

In the future, further research work will be carried out in QoS-based service discovery. The priority will be given to an implementation of the proposed approach for the purpose of proof-of-concept. Based on this implementation, accuracy and efficiency of the matching and ranking algorithms will be evaluated, followed by improvement. In addition, validation of service consumers' discovery request to ensure the consistency between QoS requirements will be investigated.

Acknowledgement

This research is partially supported by University of Wollongong Small Research Grant.

References

1. Shuping, R.: A Model for Web Services Discovery with QoS. SIGecom Exch. 4(1), 1–10 (2003)
2. Andrea, D.A.: A Model-driven WSDL Extension for Describing the QoS of Web Services. In: Proc. of 2006 IEEE International Conference on Web Services (ICWS 2006), Chicago, USA, pp. 789–796 (September 2006)
3. Clement, L., Hatley, A., von Riegen, C., Rogers, T.: UDDI Version 3.0.2, OASIS (October 19, 2004), http://uddi.org/pubs/uddi_v3.htm
4. Kritikos, K., Plexousakis, D.: A Semantic QoS-based Web Service Discovery Algorithm for Over-Constrained Demands. In: Proc. of the 3rd International Conference on Next Generation Web Services Practices (NWeSP 2007), Korea, 29–31 October 2007, pp. 49–54 (2007)
5. Duwaldt and Trees. Web Services A Technical Introduction, DEITEL™ Web Services Publishing. GUNTHER, N.J, The Practical Performance Analyst. McGraw-Hill.ISO, New York (1998), <http://www.iso.ch/iso/en/ISOOnline.frontpage>

6. Chen, Z., Liang-Tien, C., et al.: DAML-QoS ontology for Web services. In: Proceedings. IEEE International Conference on Web Services, San Diego, California, USA, 6-9 July 2004, pp. 472-479 (2004)
7. Maximilien, E.M., Singh, M.P.: A framework and ontology for dynamic Web services selection. *Internet Computing* 8(5), 84-93 (2004)
8. Dobson, G., Lock, R., et al.: QoSOnt: a QoS ontology for service-centric systems. In: 31st EUROMICRO Conference on Software Engineering and Advanced Applications, Porto, Portugal, 30 August -3 September 2005, pp. 80-87 (2005)
9. Gwyduk, Y., Taewoong, Y., et al.: A QoS model and testing mechanism for quality-driven Web services selection. In: The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, 2006 and the 2006 Second International Workshop on Collaborative Computing, Integration, and Assurance. SEUS 2006/WCCIA 2006, 27-28 April 2006, p. 6 (2006)
10. Bianchini, D., De Antonellis, V., et al.: QoS in ontology-based service classification and discovery. In: Proceedings of the 15th International Workshop on Database and Expert Systems Applications, Washington, DC, USA (2004)
11. Blum, A., Carter, F.: Representing Web Services Management Information, <http://www.oasisopen.org/committees/download.php/5144/>
12. Yang, L., Huacan, H.: Grid Service Selection Using QoS Model. In: Third International Conference on Semantics, Knowledge and Grid, Xi'an, China, 29-31 October 2007, pp. 576-577 (2007)
13. Somasundaram, T.S., Balachandar, R.A., et al.: Semantic Description and Discovery of Grid Services using WSDL-S and QoS based Matchmaking Algorithm. In: ADCOM 2006. International Conference on Advanced Computing and Communications, 20-23 December 2006, pp. 113-116 (2006)
14. Kyriakos, K., Dimitris, P.: Semantic QoS Metric Matching. Web Services. In: 4th European Conference on ECOWS 2006 (2006)
15. Wenli, D.: QoS Driven Service Discovery Method Based on Extended UDDI. In: Third International Conference on Natural Computation, ICNC 2007, 24-27 August 2007, pp. 317-324 (2007)
16. D'Mello, D.A., Ananthanarayana, V.S., et al.: A QoS Broker Based Architecture for Dynamic Web Service Selection. In: Second Asia International Conference on Modeling & Simulation, AICMS 2008 (2008)
17. Yu, T., Lin, K.J.: The design of QoS broker algorithms for QoS-capable Web services. In: 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service, pp. 17-24. IEEE, Los Alamitos (2004)
18. Giallonardo, E., Zimeo, E.: More Semantics in QoS Matching. In: IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2007, Newport Beach, California, USA, June 19-20 (2007)
19. Mani, A., Nagarajan, A.: Understanding quality of service for Web services (2002), <http://www-106.ibm.com/developerworks/library/ws-quality.html>
20. Yu-jie, M., Jian, C., Shen-sheng, Z., Jian-hong, Z.: Interactive Web Service Choice-Making Based on Extended QoS Model. *Journal of Zhejiang University - Science A* 7(4), 483-492 (2006)