# Speeding Up Ontology-Based Free-Form Service Request Recognition

**M. J. Al-Muhammed**
**Department of Mathematics, Faculty of Science, Damascus University, Syria**

**Received 07/09/2010**
**Accepted 07/03/2011**

## ABSTRACT

**We offered, in a previous paper, an ontology-based approach to recognize constraints in free-form service requests and provide services for users. Our system handles a service request by finding, from among many ontologies, the domain ontology that best matches the request and then uses the matched ontology to generate the service request constraints. Although our system is powerful in recognizing constraints and therefore servicing requests, the recognition process is a bottleneck due to the number of the tested ontologies and the amount of computations involved. This paper provides a novel approach to speed up the recognition process by (1) using ontology indexing and (2) excluding inapplicable regular expressions early in the process and thus reducing the number of applied regular expressions. Our experiments show that our techniques are effective in significantly reducing the amount of computations and therefore speeding up the recognition process.**

**Key words**: Ontology-based recognition, Free-form service requests, Process efficiency, Ontology indexing, Light-weight process, Heavy-weight process

# تسريع تعرّف طلبات الخدمة المعتمدة على الأنتولوجي

**محمد جاسم المحمد**

**قسم الرياضيات – كلية العلوم – جامعة دمشق – سورية**

**تاريخ الإيداع 2010/09/07**
**قبل للنشر في 2011/03/07**

## الملخّص

**استخدم الباحثون الأنتولوجي لأغراض مختلفة. من الأمثلة على هذا الاستخدام نذكر: استخراج معلومات من الإنترنت Data Extraction، وتسهيل الاتصال بين الكائنات الذكية على الإنترنت Agent Communication، والخدمات الحاسوبية Services. من الأهداف الأساسية لعلوم الحاسبات:**

1. **تمكين المستخدمين العاديين من الحصول على خدمات من الحاسوب بسهولة؛ وذلك من خلال وصف الخدمة المطلوبة فقط.**
2. **تمكين مطوري (مبرمجي) الخدمات من تطوير هذه الخدمات بسرعة.**

**قمنا في بحوث سابقة بتطوير نظام معتمد على الأنتولوجي يلبي هذين المتطلبين ويوفر العديد من الخدمات نذكر منها: أخذ مواعيد من مخدمين (المحامين، الأطباء، إلخ)، إقامة إجتماعات، عمليات البيع والشراء على الوب، تحضيرات السفر (شراء بطاقات طائرات، حجز مكان الإقامة، ...)، وغيرها كثير. يمكّن هذا النظام المستخدمين من توصيف الخدمات التي يرغبون فيها بلغة طبيعية (الإنكليزية) ومن ثم يقوم النظام بتوليد الخدمة المطلوبة آلياً ومن ثم تخديم المستخدم. ويمكّن مطوري الخدمة من تطوير خدمة جديدة من خلال توصيف الأنتولوجي المناسب لهذه الخدمة فقط دون كتابة أي برامج حاسوبية.**

**على الرغم من قوة نظامنا الحاسوبي في توفيرالخدمات بسهولة للمستخدمين، هناك مشكلة كبيرة مع هذا النظام تتعلق بعملية مطابقة طلب خدمة مع الأنتولوجي المناسب. هذه العملية تحتاج إلى وقت طويل بسبب العدد الكبير للأنتولوجيات المتوافرة وعملية المحاكمة المنطقية اللازمة لتحديد الأنتولوجي المناسب للخدمة. ومن ثم تقل كفاءة النظام بشكل كبير عندما تزداد أعداد الأنتولوجيات المتوافرة للنظام (هذه المشكلة معروفة علمياً بـ Scalability Problem). يقدم هذا البحث حلاً فعالاً لهذه المشكلة. نستخدم في هذا الحل تقنيتين لتقليل الزمن اللازم لمطابقة طلب الخدمة مع الأنتولوجي المناسب. التقنية الأولى وتسمى فهرسة الأنتولوجيات (Ontology Indexing) وهدفها تقليل عدد التعابير النظامية (Regular Expression) المستخدمة في أثناء مطابقة طلب الخدمة مع الأنتولوجيات. في هذه التقنية يجري تطبيق التعابير النظامية المشتركة بين الأنتولوجيات مرة واحدة بدلاً من تطبيقها عدداً من المرات مساوياً لعدد الأنتولوجيات التي تحوي هذه التعابير. التقنية الثانية وتسمى المطابقة الثنائية الطور (Two-Pass Process) وهدفها استبعاد التعابير النظامية التي نعلم أنها غير قابلة للتطبيق على طلب الخدمة الحالي ويزمن أقل بكثير من الزمن اللازم للتطبق على كامل التعبير النظامي. أثبتت التجارب التي أجريت أن الحل المطروح في هذا البحث هو حل فعال ويقوم بتسريع عملية المطابقة بشكل كبير؛ مما يزيد من قدرة النظام على التعامل مع أي عدد من الأنتولوجيات. يمكن تلخيص الإسهامات العلمية لهذا البحث بالنقاط الآتية:**

1. **استخدام تقنية فهرسة الأنتولوجيات (Ontology Indexing) لتقليل عدد التعابير النظامية المطبقة على طلب خدمة معين ومن ثم تقليل الزمن اللازم للمطابقة.**
2. **استخدام عملية مطابقة ثنائية الطور (Two-Pass Process) لاكتشاف التعابير النظامية غير القابلة للتطبيق على طلب الخدمة مبكراً ويزمن أقل ومن ثم تفادي تطبيقها الكامل على طلب الخدمة.**

**الكلمات المفتاحية**: تعرف طلبات الخدمة المعتمدة على الأنتولوجي، طلبات الخدمة الموصفة بلغة طبيعية، كفاءة المعالجة، فهرسة الأنتولوجيات، معالجة خفيفة الطور، معالجة ثقيلة الطور.

# 1. Introduction

Researchers have used ontologies for many purposes [9, 10, 11, 12]. Examples include data extraction from the web, agent communications, and services. We described in previous papers [1, 2, 3, 4] a system that allows users to specify service requests and invoke services. This approach is strongly based on domain ontologies and supports a particular type of service whose invocation involves establishing an agreed-upon relationship in the ontology. Examples of these types of services include scheduling appointments, setting up meetings, selling and purchasing products, making travel arrangements, and many more.[1]

To help readers understand the problem we address in this paper, we give a brief description about the system. Figure 1 shows a simplified overview of the system architecture. The system automatically generates software for different types of services. The underlying technique that enables this automatic generation of a service is domain knowledge known as domain ontology (ontology described in Section 2). One of the interesting characteristics of our system is it both (1) enables service providers to deliver services by creating only domain ontologies describing these service *without writing any line of code* and (2) it allows users to invoke services using free-form specification rather than having to first find services and invoke them.

When the system receives a service request from a user, it handles the request as follows. It first matches a service request with a set of domain ontologies known to the system, and finds a domain ontology that best matches the request through a process called the recognition process. Secondly, it utilizes the best matching ontology to generate a formalism, which is a predicate calculus formula. Thirdly, it satisfies the constraints in the formalism and provides a solution for the request. In cases where there are too many solutions that satisfy all the constraints or no solution that satisfies all the constraints, the system negotiates solutions with users to reach an agreement on a solution (satisfies all the constraints) or a near solution (violates at least one constraint). The details of these processes are not the focus of this paper and can be found elsewhere [3].

[1] We intend the word "service" to be thought of in accordance with its typical meaning—"an act of assistance or benefit." Technically, we define a very special type of service (as described herein). We do not intend our services to be thought of in other technical ways such as registering services with a broker so that they can be found by expressing their functionality in terms of inputs, outputs, and capabilities.

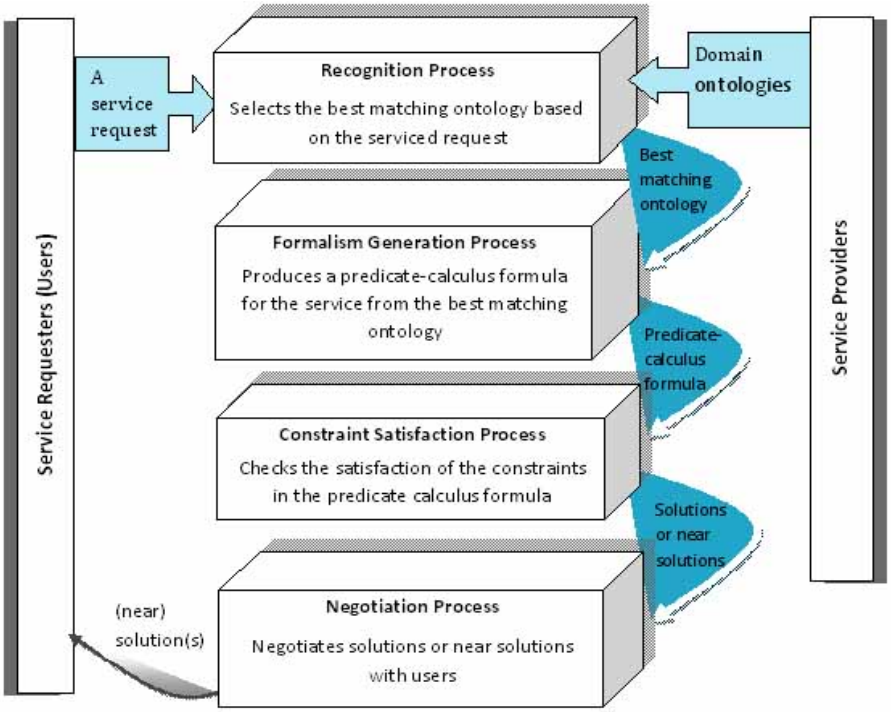


**Figure 1. A simplified overview of the system**

Unfortunately, as the number of registered ontologies increases, the recognition process suffers from a scalability problem and is a bottleneck. As pointed out in [1, 3], the recognition process is the most difficult and time-intensive process due to the huge amount of computations required to find a domain ontology that best matches the request. This paper addresses this problem and proposes a novel approach that highly reduces the amount of required computations and thus speeds up the recognition process. Our approach uses two techniques to achieve this goal. First, it uses **ontology indexing** to factor out shared regular expressions among the domain ontologies. Consequently, a regular expression contained in many ontologies is applied once instead of as many times as the number of ontologies containing this regular expression. Second, it uses a **two-pass process** to exclude inapplicable long, time-consuming regular expressions early in the recognition process.

To this end, the paper makes the following contributions. First, it provides an ontology indexing technique to ensure that each regular

expression is applied *only* once regardless of how many ontologies contain this regular expression. Second, it discovers inapplicable regular expressions with significantly less time than applying full regular expressions. Both techniques work synergistically with the objective of reducing the time required by the recognition process.

We present our contributions as follows. Section 2 gives a brief introduction to the notion of ontology and data frames. Section 3 briefly describes the recognition process. Section 4 presents our new way to improve the time complexity of the ontology recognition in Subsections 4.1 and 4.2, and presents the results of our evaluation of the new techniques in Subsection 4.3. We conclude in Section 5 and give directions for future work.

## 2. Domain Ontology

A domain ontology encodes domain knowledge through defining among other things a set of concepts and relationships among these concepts. In the context of our system, a domain ontology specifies precisely the information required to provide a specific service in the domain. For instance, to schedule an appointment with a service provider, say a dermatologist, you would generally need information about the appointment such as the date and the time of the appointment along with the address of the service provider. As a consequence, the domain ontology for scheduling appointments must define a set of concepts including: *Time*, *Date*, and *Address*. (More on ontologies can be found in [1, 2, 3]).

***Distance***
internal representation: real

**text representation: \d+(\.\d+)?|(\.\d+)**
**Right-context: miles?|kilometers?|...**

DistanceLessThanOrEqual(d1: Distance, d2: Distance)
returns (Boolean)
context keywords/phrases: (within|...)\s+{d2}|...
…

**Figure 2. A data frame for the concept *Distance*. Part of the data frame is shaded because of its relevancy to our discussion.**

Each concept in a domain ontology has an associated data frame [8], which describes instances for the concept. Data frames capture the

information about concept instances in terms of regular expressions and other descriptors. Figure 2 shows a data frame for the concept *Distance*. For the purpose of this paper, we focus only on the highlighted part of the data frame in Figure 2. A regular expression has a structure composed of three slots: **left context**, **text representation**, and **right context**, which we call **LTR-structure**. Generally speaking the text representation captures the actual instances of a concept whereas the left and right contexts provide information to more precisely describe the instances. Any of the left context or right context can be an empty string (null). For example, in Figure 2, the *LTR-structure* for the regular expression that describes instances of the concept *Distance* is: the left context is null, text representation is "\d+(\.\d+)?|(\.\d+)", and the right context is "miles?|kilometers?|...". (The ellipses "..." indicate that there may be more contextual keywords.)

To identify instances of a concept in a service request, the system forms a full regular expression (also called an *instance recognizer*) using the information in the data frame associated with the concept. In particular, the system forms a full regular expression describing the instances of a concept from a data frame by concatenating the three parts left context, text representation, and right context respectively. For instance, the full regular expression that can be formed to recognize the instances of the concept *Distance* in Figure 2 is "\d+(\.\d+)?|(\.\d+)\s*(miles?|kilometers?)".[2] Note this regular expression identifies the appearance of distance instances such as "5 miles" or "8 kilometers" in a service request to which it is applied.

## 3. The Recognition Process

The ontology recognition process selects, from among potentially many domain ontologies registered with our system, the (correct) domain ontology for a service request. The recognition process takes the set of available domain ontologies and a service request as input, and returns the domain ontology that best matches with the service request as output. The recognition process works in two steps. First, for each domain ontology, the recognition process applies full regular expressions formed from the data frames to the service request and marks every concept that matches a substring in the service request.

[2] The creation of full regular expressions from the LTR-structure is straightforward and is done automatically by the system. The system pulls the left context, text representation, and right context from the ontology and concatenates them in the same order.

Second, the process computes a rank value for each domain ontology with respect to the service request and then selects the domain ontology with the highest rank value.

Details about how the process marks concepts of an ontology and selects the best matching one is beyond the scope of this paper and can be found elsewhere [2, 3, 4]. What is important for this paper is how the recognition process works. As discussed above, the recognition process applies all the regular expressions in the currently processed ontology regardless whether these regular expressions from previously applied ontologies have been already applied. This means that a regular expression, say $r_i$, contained in, say $n$ ontologies, is applied $n$ times. In addition, there is *no* mechanism that helps identify whether a full regular expression is applicable before applying this regular expression. It appears in both cases that the recognition process performs unnecessary computations.

We address this problem in Section 4 and show how our techniques speed up the recognition process.

## 4. Improving Time Complexity for Recognition Process

According to the insights from Section 3, it appears that speeding up the recognition process, requires reduction in the amount of computations required by the recognition process. To reduce the amount of computations, we need to discover the redundant computations and eliminate them. As discussed in Section 3, there are two main problems that negatively affect the performance of the recognition process. First, the recognition process applies a regular expression as many times as the number of ontologies containing the regular expression. Second the recognition process applies full regular expressions although these regular expressions may *not* be applicable.

In Subsection 4.1, we introduce a simple, but powerful, technique called ontology indexing to factor out regular expressions and avoid applying them more than one time. In Subsection 4.2 we develop two-way processing to discover and exclude inapplicable regular expressions

### 4.1. Ontology Indexing

First, before trying any ontology, the system loads available ontologies and creates an ontology index for them. The ontology index is a data structure similar to the one in Figure 3. Each entry in the structure has a *key*, which is a regular expression; and a *value*,

which is a set of one or more ontologies containing this regular expression. For instance, in Figure 3, the regular expression Regular expression$_1$ is a key corresponding to the value Ontology 1, ontology 3, ontology 100. The ontology index is built once at system configuration time. When a new ontology is registered with our system, the system inserts the newly added ontology to the index.

| **Regular Expressions** | **Ontologies to which these regular expressions belong** |
|---|---|
| Regular expression$_1$ → | Ontology 1, ontology 3, ontology 100 |
| Regular expression$_2$ → | Ontology 3, ontology 10, ontology 30 |
| . | . |
| . | . |
| Regular expression$_n$ → | Ontology 1, ontology 50, ontology 70, ontology 120, … |

**Figure 3. An ontology index**.

The system currently uses a semi-automatic way to create the ontology index. It automatically populates the index by extracting the regular expressions from each ontology. If a regular expression is contained in more than one ontology, the system lists all theses ontologies containing this regular expression as a value for the regular expression. For instance, as shown in Figure 3, Regular expression$_1$ belongs to ontologies 1, 3, 100 and hence the system inserts these ontologies as a value corresponding to this regular expression. With this index, a regular expression, say regular expression$_n$, which belongs to four different ontologies, is applied only once rather than **four times** as performed in the previous process.

Determining whether two or more regular expressions are equivalent is currently based on the string matching procedure. This of course is prone to matching errors because equivalent regular expressions can be written in different ways; a problem that string matching cannot discover. Hence, we likely end up enumerating the same regular expression many times in the index; the thing we should avoid. Therefore, after the system automatically populates the ontology index, we manually examine the index for potential errors.

Before leaving this section, we point out a situation in which each ontology contains different regular expressions than the others (i.e. no regular expression shared among ontologies). In this case, the ontology index gives apparently no time improvement. Philosophically, this cannot happen in practice, however, as domains do have common information and likewise do ontologies. In fact, our experience shows that for even moderate applications there is always a

significant number of regular expressions shared between ontologies representing the domains of these applications.

### 4.2. Two-Pass Process: Inapplicable Regular Expression Exclusion

Applying regular expressions to documents with the objective of finding matches is time consuming. It has been shown in [7] that the time for applying regular expressions greatly increases with both the length of these regular expressions and length of a service request.[3] As such, we should avoid applying long regular expressions if there is a way to discover in advance that they do not match anything in the text.

Our strategy to avoid applying inapplicable regular expressions is based on performing a two-pass process. The first pass is called **light-weight process** in which we exploit the *LTR-structure* of a regular expression. Specifically, we apply either left context or right context of a regular expression to a service request. If one of the contexts (left or right) fails to match with a substring in the service request, we exclude the whole regular expression to which this context belongs from the following process since we know that the full regular expression will be inapplicable. Since the time required for applying regular expressions significantly increases with their lengths and the contexts are far shorter than the full regular expression, the processing time most likely decreases. The second pass, which we call **heavy-weight process**, applies full regular expressions to the service request. The idea is to create the full regular expressions, which are formed from regular expressions that have not been excluded in the light-weight process. This way we apply only full regular expressions that are likely to match as opposed to arbitrarily applying all of them.

### 4.3. Performance Analysis

We have conducted many experiments to validate our techniques for speeding up the ontology recognition process. We tried 20 different service requests. The average length of our requests is 130 characters.[4] These requests cover the domains of the five ontologies that we used in our experiments. The domain ontologies include scheduling appointments, renting apartments, scheduling meeting, purchasing cars, and purchasing electronic devices. Every ontology

[3] The length of a regular expression (or a service request) is the number of characters and the special symbols included in the regular expression.

[4] For experimental purposes, we opened our system for people to use for 6 months. The log information showed that most of the requests are of average of around 130 characters.

includes roughly 30 regular expressions on average. (In real-world applications, ontologies may include more.)

In the first experiment, we used five ontologies. We gradually increased the length of the service request by merging 2, 5, 10, 15, and finally 20 service requests. The objective here is to study the performance as the length of a service request increases (consequently increasing the number of matches). The timing starts when the recognition process is called and ends when this process successfully returns.

Table 1 shows the time required for processing the different input sizes and Figure 4 depicts graphically the required time. Generally speaking the time numbers show that our new techniques performed remarkably better than the old technique. Our new techniques have significantly reduced the processing time. As seen in Table 1 (consequently also in Figure 4), the time required for processing service requests is far less using the new techniques. For instance, while the old technique needs 920 milliseconds to process an input of two service requests, our new techniques need *only* 183 milliseconds for processing the same number of requests; almost 5 times less. On other end of the spectrum, Table 1 (Figure 4) shows that the old techniques need 10100 milliseconds to process 20 service requests, while our new techniques need only 1860 milliseconds to process the 20 service requests; again almost 8 times less.

It is probably worth noting also that for 2 requests the new techniques were 5 times less while for 20 service requests our techniques were 8 times less. This probably indicates a serious problem with the old technique. Namely, the larger the request (in terms of number of characters) to be processed is the longer the required time becomes. In any case, we believe that the time gains in our techniques are easily justifiable.

**Table 1. The processing time for varying numbers of requests. The time measurement starts when the system calls the recognition process and ends when the recognition process successfully returns.**

| Number of Requests | Time Old (in milliseconds) | Time New (in milliseconds) |
|---|---|---|
| **2** | **920** | **183** |
| **5** | **2305** | **324** |
| **10** | **2905** | **677** |
| **15** | **6000** | **918** |
| **20** | **10100** | **1860** |

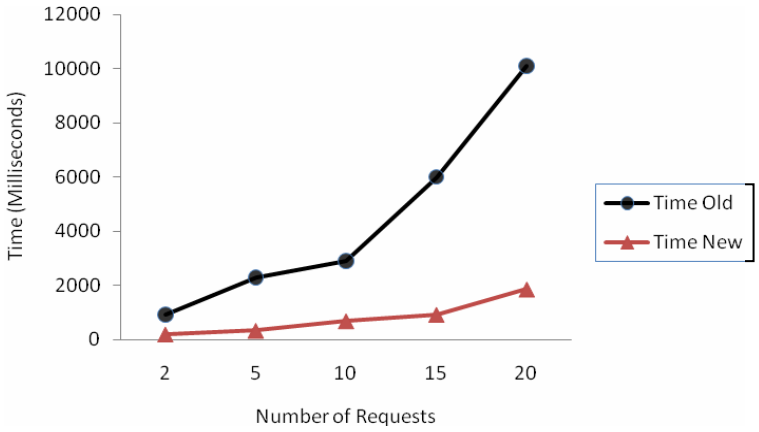


**Figure 4. Time complexity for processing services requests in the old process (Time Old) and the new techniques (Time New).**

Specifically our techniques apply a smaller number of regular expressions due to the ontology indexing and exclude, *early in the process*, many *inapplicable* regular expressions due to two pass processing technique.

For the sake of further discussing the performance of the new techniques, we conducted another experiment. In this experiment, we used an input of 10 service requests while gradually incrementing the number of ontologies by one every time we run the recognition process. Our objective here is to examine the processing time changes as the number of ontologies gradually increases. The time figures are shown in Table 2 and illustrated graphically in Figure 5.

**Table 2. The processing time as the number of ontologies increases.**

| Number of Ontologies | Time Old (in milliseconds) | Time New (in milliseconds) |
|---|---|---|
| 1 | 453 | 398 |
| 2 | 819 | 422 |
| 3 | 1346 | 461 |
| 4 | 2000 | 499 |
| 5 | 2898 | 668 |

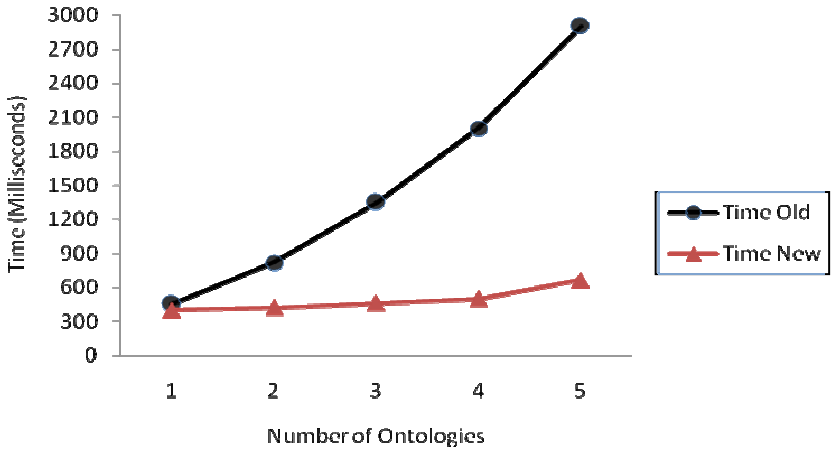


**Figure 5. Time complexity for processing services requests in the old process (Time Old) and the new techniques (Time New).**

As it can be seen, the old techniques and the new techniques show different time-increase patterns. In the old techniques, the processing time sharply increases as the number of ontologies gradually increments. In contrast, in the new techniques, the processing time slowly increases as the number of ontologies gradually increments.

Before leaving this section, we make the following point. Despite the fact that our experiments by no means are enough for proving the superiority of our new techniques, the time figures indicate better performance than the old techniques. Generally speaking, the ontology indexing insures that no matter how many ontologies involved in the recognition process, our new techniques will not apply more regular expressions than the old techniques. Actually, in the worst case scenario the old techniques and our new techniques will apply the *same* number of regular expressions. Additionally, it is likely that some of the regular expressions will be excluded early in the recognition process with *less time* through the light-weight process. As such, we believe that there is clear evidence that our techniques can significantly speed up the recognition process and the time reduction will be even more significant as the number of ontologies increases.

## 5. Conclusions and Future Work

We have proposed a new way to enhance the speed of the recognition process. Experiments with our new techniques showed that they improve the time complexity for the recognition process. Thus, we believe that the time for servicing any request is much better than before.

We have two objectives for our future work. First, we should find ways to determine whether two or more regular expressions are equivalent. This helps us to automatically create the ontology index. Knowing that two regular expressions are equivalent, however, is well known to be a hard problem and it is a topic of our ongoing research. Second, we want to conduct more experiments on the new techniques before we integrate them into our service architecture.

We believe that it is possible to build ontologies to recognize requests in Arabic. This also can be another objective to pursue in the future work.

**Acknowledgement**

I appreciate Dr. James H. Matis in A&M University, Texas, USA, for his valuable comments and suggestions that significantly improved this paper.

# REFERENCES المراجع


1. **Muhammed J. Al-Muhammed and David W. Embley. Resolving Underconstrained and Overconstrained Systems of Conjunctive Constraints for Service Requests. In *Proceedings of the 18th International Conference on Advanced Information Systems Engineering (CAiSE06)*, pages 223–238, Luxembourg, June 2006.**
2. **Muhammed J. Al-Muhammed and David W. Embley. Ontology-Based Constraint Recognition for Free-Form Service Requests. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE 2007)*, pages 366–375, Istanbul, Turkey, April 2007.**
3. **Muhammed J. Al-Muhammed, David W. Embley, and Stephen W. Liddle. Conceptual Model Based Semantic Web Services. In *Proceedings of the 24th International Conference on Conceptual Modeling (ER 2005)*, pages 288–303, Klagenfurt, Austria, October 2005.**
4. **Muhammed J. Al-Muhammed, David W. Embley, Stephen W. Liddle, and Yuri A. Tijerino. Bringing Web Principles to Services: Ontology-Based Web Services. In *Proceedings of the 4th International Workshop on Semantic Web for Services and Processes (SWSP 2007)*, pages 73–80, Salt Lack City, Utah, USA, July 2007**
5. **Yuri. Tijerino, Muhammed J. Al-Muhammed, and David W. Embley. Toward a Flexible Human-Agent Collaboration Framework with Mediating Domain Ontologies for the Semantic Web. In *Proceedings of ISWC-04 Workshop on Meaning Coordination and Negotiation(MCN-04)*, pages 131–42, Hiroshima, Japan, November 2004.**
6. **Muhammed J. Al-Muhammed and David W. Embley. Towards Enabling Communication among Independent Agents in the Semantic Web. In *Proceedings of the 3rd International Workshop on Information Systems Technology and their Applications*, Salt Lake City, Utah, USA, July 2004.**
7. **R. Sidhu and V. K. Prasanna. Fast Regular Expression Matching Using FPGAs. In *Proceedings of the the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2001)*, pages 227–238,Washington, DC, USA, 2001. IEEE Computer Society.**
8. **D. W. Embley. Programming with Data Frames for Everyday Items. In D. Medley and E. Marie, editors, *Proceedings of AFIPS Conference*, pages 301–305, Anheim, California, May 1980**
9. **D.W. Embley, S.W. Liddle, D. Lonsdale, G. Nagy, Y. Tijerino and et al. Conceptual-Model-Based Computational Alembic for a Web of Knowledge. The Conceptual Modeling Conference (ER08), October, 2008.**
10. **S. Kubicki, E. Dubois, G. Halin, and A. Guerriero. Towards a Sustainable Services Innovation in the Construction Sector. *Proceedings of the 21st International Conference on Advanced Information Systems Engineering,* pages: 319-338, Amsterdam, Netherlands. 2009.**

11. **M. Sabesan and T. Risch. Adaptive Parallelization of Queries over Dependent Web Service Calls. *Proceedings of the 24rd International Conference on Data Engineering (ICDE 2007)*, pages 1725-1732, Shanghai, China, March - April 2009.**
12. **C. Tao, Y. Ding, and D. Lonsdale. Automatic Creation of Web Services from Extraction Ontologies. The First International Workshop on Semantic Web Applications: Theory and Practice (SWAT 2006) in conjunction with ER 2006, Tucson, Arizona, November 2006.**