Matching Independent Global Constraints for Composite Web Services

Nalaka Gooneratne Zahir Tari
School of Computer Science and Information Technology
RMIT University
Melbourne, VIC 3001, Australia
{ngoonera, zahirt}@cs.rmit.edu.au

ABSTRACT

Service discovery employs matching techniques to select services by comparing their descriptions against user constraints. Semantic-based matching approaches achieve higher recall than syntactic-based ones (as they employ ontological reasoning mechanisms to match syntactically heterogeneous descriptions). However, semantic-based approaches still have problems (e.g. lack of scalability as an exhaustive search is often performed to located services conforming to constraints). This paper proposes two approaches that deal with the problem of scalability/performance for composite service location. First, services are indexed based on the values they assign to their restricted attributes (the attributes restricted by a given constraint). Then, services that assign "conforming values" to those attributes are combined to form composite services. The first proposed approach extends a local optimisation technique to perform the latter, since identifying such values is NP-hard. However, this approach returns false negatives since the local optimisation technique does not consider all the values. Hence, a second approach that derives conforming values using domain rules is defined. The used rules are returned with each composite service so that a user can understand the context in which it is retrieved. Results obtained from the experiments that varied the number of available services demonstrate that the performance of the local optimisation-based approach is 76% better than existing semantic-based approaches and recall is 98% higher than syntactic-based approaches.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search process, Selection process; H.3.5 [**Online Information Services**]: Web-based services

General Terms

Performance, Theory, Algorithms

1. INTRODUCTION

Web services are autonomous and modular applications that are located, accessed and executed over the Internet. Service discovery enables such services to be located based on functional requirements [2], non-functional requirements [12] or both [10]. It employs matching techniques to compare user requests against service descriptions to retrieve appropriate services. A service discovery approach is either syntactic-based [11] (where descriptions are represented as a set of strings and "string matching" is used) or semantic-based [2] (where ontological relationships are used to perform mappings between terms of user requests and service descriptions). It is well accepted that semantic-based approaches achieve higher recall than syntactical ones [2]. Therefore the focus of this paper is on semantic-based matching and we will show the limitations of existing approaches (especially when dealing with complex/composite services).

Before providing further details about existing semantic-based approaches, we will first consider the following scenario which will be used in the rest of this paper (to illustrate the use of various concepts). In this scenario, we want to model the "purchase service", where a user wishes to purchase a (Macintosh) computer online and wants it delivered home. A shipper should be able to pick up this computer from the dispatch location of a seller, and it should be insured from the point at which the computer is picked up. The computer must not be insured until it is assembled (so that the insured period can be maximised), and it should not be picked up by the shipper until it is insured. The (composite) service shown in Figure 1 is formed by locating, co-ordinating and collaborating the constituent services, and executed to satisfy this request.

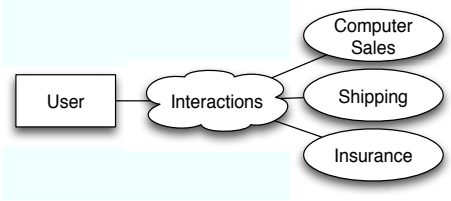


Figure 1: A Composite Service

Locating a composite service (like the one shown in Figure 1) would require accurate specifications of both service descriptions and user requests. Constraints are used in user requests to accurately describe the services that need to be located [2]. They are of two types: local and global constraints. The former restricts the values of a particular attribute of a single service (e.g. $type.Computer = MACINTOSH$), whereas the latter simultaneously restricts the values of two or more attributes of multiple constituent services. Global constraints can be classified based on the complexity of solving them (i.e. determining the values that should be assigned to their attributes) as either strictly dependent or

WWW 2008, April 21–25, 2008, Beijing, China.
ACM 978-1-60558-085-2/08/04.

independent. A (global) constraint is strictly dependent if the values that should be assigned to all the remaining restricted attributes can be uniquely determined once a value is assigned to one. $location.Dispatch = location.Pickup \in validRegion.Insurance$ is an example of a strictly dependent global constraint. Once *MELBOURNE* is assigned to *location.Dispatch*, the same value has to be assigned to *location.Pickup* and included in the set of values assigned to *validRegion.Insurance*. Services that conform to strictly dependent global constraints can be easily located in polynomial time [5].

Any global constraint that is not strictly dependent, is **independent**. For example, $availableDate.Computer \leq approvalDate.Insurance \leq date.Pickup$ is an independent global constraint. In our opinion, the class of composite services that conform to independent global constraints is probably the "most" interesting to study, as their location is known to be NP-hard [3, 12]. Indeed, if a given independent global constraint restricts q attributes, and if service descriptions assign p values to each attribute, then a technique locating a conforming service may have to consider p^q combinations of values. As a consequence most of the existing matching techniques (for locating composite services) do not consider independent global constraints [1, 8]. Nonetheless, there are some that consider them [9, 11, 12] and they use integer programming solutions focusing on local optimisations [12] and AI planners [9, 11] to efficiently locate conforming composite services. However, all the techniques of the latter type are syntactic-based approaches. None of the current semantic-based composite matching approaches consider independent global constraints.

This paper proposes a semantic-based matching technique that locates services conforming to independent global constraints. A two dimensional data structure, called a *slot list*, is designed to efficiently find services. Available services are indexed based on the values they assign to the attributes restricted by a global constraint. This index enables the quick retrieval of services that assign a particular value to their restricted attribute. Then, tuples of conforming values are determined using either a greedy approach or derived using domain rules [2]. Finally, services that assign these conforming values to their restricted attributes are retrieved from the slot list and combined to form conforming composite services. Experiments were performed to compare the proposed techniques against a syntactic-based composite matching technique [11] and a relevant semantic-based matching technique [2]. The proposed technique that incorporates a greedy algorithm performs 76% better than the existing techniques. Experimental results also show that the proposed approaches achieve a higher recall than syntactic-based approaches.

The rest of his paper is organized as follows. In the remaining sections we will refer to composite services simply as "services" (because the focus of this paper is only on the discovery of such services). Section 2 overviews some background needed to understand the various parts of the paper. In Section 3 we provide concise guidelines on how requests for services should be specified. Section 4 details the proposed composite matching techniques and Section 5 provides a summary of advantages/limitations of the proposed approaches. Details of the experiments are given in Section 6. In Section 7, we review existing composite matching techniques. Finally, we summarize and conclude in Section 8.

2. BACKGROUND

Meta-Ontology

In this paper we will use the meta-ontology proposed by Elgedawy et al. in [2]. As pointed out in [5], in this ontological structure relationships can be accurately modeled and reasoned using a "simple" mechanism. A meta-ontology consists of four types of elements (i.e. concepts, operations, roles and rules). Concepts are basic entities of a domain. Domain elements with common attributes are abstracted into concepts (e.g. - *Computer*, *Finance*, *Insurance*). Transactions that are permitted in a domain are specified with operations (e.g. - *Sales()*, *Insure()*, *Ship()*). Roles are used to specify the actors such as *Sales_Assistant*, *Telemarketer*. A description of rules will be provided later in this section.

The relationships between the elements of an ontology are specified with *substitution graphs* and *transformation graphs*. In a substitution graph, the contextual aspect of a relationship is specified with a substitution condition. Each (substitution) graph defines a set of attribute mappings. Figure 2 depicts an example of a substitution graph which specifies the relationship between *Computer* and *Laptop*. A transformation graph models a relationship requiring the meta-model of an element to be changed. It is specified with one or more consecutive operations. Figure 3 shows a transformation graph modeling the relationship between *Dell* and *Intel* (which are brand names of a Computer and a Processor). This is specified with an operation that lists components and another that returns the type of one.

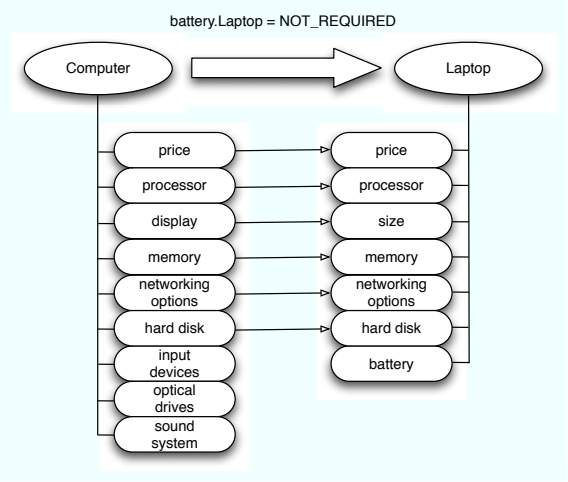


Figure 2: Substitution graph

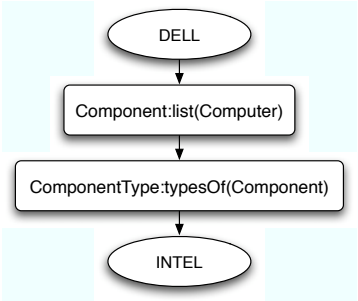


Figure 3: Transformation graph

These two data structures (i.e. substitution and transformation graphs) are specified for every element of an ontology (i.e. concept, operations, roles). The meta-ontology requires any transitive relationship to be explicitly specified with the data structures. For example, if *Calculator* can be substituted with *Laptop*, and *Laptop* can be substituted with *Computer*, there is a transitive substitution relationship between *Calculator* and *Computer*. Then, a substitution graph which allows the concept *Calculator* to be substituted with *Computer* should be explicitly included in the ontological descriptions.

WS-$\mathcal{ALUE}$

The proposed composite matching technique assumes that the functional descriptions of services are specified with WS-$\mathcal{ALUE}$ [6]. In such a language, the functionality of a service is described by three elements; its purpose (goal), state transitions and data transformations. Current description frameworks represent the purpose with an operation [2, 6]. The state transitions are described with *pre-conditions* (describing the state that the execution environment of a service needs to be in before commencing its execution) and *post-conditions* (describing the state after its execution has been successfully completed). States are described by the values that can be assigned to the attributes of a service according to the pre-conditions or the post-conditions. Data transformations are described with inputs and outputs. Unlike other functional description frameworks, WS-$\mathcal{ALUE}$ models **all three** functional aspects **together** [6]. For the approach proposed in this paper, descriptions that specify the purpose and the state transition performed by services would be sufficient. However, WS-$\mathcal{ALUE}$ is selected because we intend to develop techniques that verify the composability [7] and compatibility [7] of (composite) services, in the future.

An WS-$\mathcal{ALUE}$'s operation (used to describe the purpose of a service) is constrained by the role it performs in a domain and the concepts it affects. For example, a *Computer Sales* service performs the operation *Sales()*, affects the concept *Computer* and performs the role of an *Sales_Assistant*. The data transformations are described with inputs and outputs. "in-constraints" and "out-constraints" restrict the values used as inputs and outputs. State transitions are described with pre-conditions and post-conditions.

The conditions (specifying the in-constraints, out-constraints, pre-conditions and post-conditions) model a relationship between an attribute and a value. A condition explicitly reduces the scope of an attribute to a particular domain. The scope of an attribute is the set of values that can be assigned to the attribute. It is defined by associating an attribute with a concept. For example, the scope of the attribute *dispatchTime* cannot be determined unless it is specified as *dispatchTime.Computer*. Then any value (specifying the time taken to dispatch a computer) can be assigned to the attribute *dispatchTime*. When an attribute is used in a condition, its scope is reduced to a particular domain. For example, in a condition of the form *dispatchTime.Computer*$<$*24_HOURS*, the scope of *dispatchTime.Computer* is reduced to a value less than 24 hours.

Rules

Rules of a meta-ontology [2] define the legitimate derivations of a domain (e.g."Insurance should be approved immediately according to standard procedures", "A computer should be picked up a day after it is made available according to standard procedures", "A customer is categorized as VIP if more than 100 purchases are made within a week"). The second matching approach proposed in this paper will use these rules to derive values that conform to independent global constraints. An exact specification of how these rules are represented is not provided in [2]. Therefore, we assume that a rule is specified as a function, which derives a value that should be assigned to a derived attribute based on a value that is assigned to a determinant attribute. The context in which these assignments should be made to the attributes is specified with an operation, an affected concept and a role. Figure 4 depicts graphical representations of the first two sample rules described above.

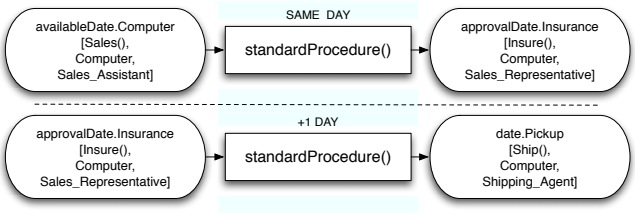


Figure 4: Sample Rules

Context Matching Technique

Elgedawy's approach [2] matches the contexts of two similar goals. Their notion of "semantically related attributes" will be used in the proposed technique to identify the list of attributes related to a given attribute according to ontological relationships.

Definition 1 (Semantically Related Attributes). *Given two attributes a_i and a_y where their scopes are defined with the concepts c_i and c_y respectively, a_i is semantically related to a_y if c_y can be substituted with c_i using either a substitution graph or a transformation graph, and the substitution results in a_i being mapped to a_y.*

The attribute *Computer.display* is semantically related to *Laptop.size* according to the substitution graph given in Figure 2, because *Computer* substitutes *Laptop* and the substitution results in *display* being mapped to *size*.

3. USER REQUEST

Concise guidelines (for creating requests for composite services) are given in this section. The proposed matching technique assumes that a request is structured according the given guidelines. A request consists of a composite service template and a constraint model. The former specifies the types of services that need to be aggregated to form a composite service. It is defined as a collection of service types. A service type is described with the triplet $[O, C, R]$, where O is an operation, C is an affected concept and R is a role. A description of the composite service template of the service given in Figure 1 follows.

$$\{[Sales(), Computer, Sales\_Assistant],\\ [Ship(), Computer, Shipping\_Agent],\\ [Insure(), Computer, Sales\_Representative]\}$$

Constraints are included in a request to provide an accurate description of the required services. For example, the constraint "*type.Computer* = *Macintosh*" (which is applied on

services that sell computers) states that only those that sell Macintosh computers are required. The constraint model described here contains an independent global constraint. Such constraints are formed with a collection of binary attribute comparisons (binary constraints). For example, *availableDate.Computer* $\leq$ *approvalDate.Insurance* $\leq$ *date.Pickup* is an independent global constraint. Typically, a global constraint is specified with a non-empty set of attributes. Additionally, a type which indicates whether a constraint restricts attributes that describe either pre-conditions (of services), post-conditions or both is included in a specification.

The proposed approach employs both a local optimisation technique and derivation-based technique (that uses domain rules) to determine values that conform to a given constraint. The reasons for using these techniques will be described in the next section. Both of these techniques determine the values that should be assigned to all the attributes based on a value assigned to one. Hence, they use the values assigned to one particular attribute as a starting point. This attribute is referred to as the "free attribute". Unlike the values assigned to the non-free attributes, all the values assigned to this attribute will be considered by the proposed approach when determining conforming values.

Since an independent global constraint is a collection of binary attribute comparisons, the relationships between the attributes are specified with a set of binary operators. Each one of them is specified with two attributes and relational operator, which can be either $=, <, >, \leq, \geq, \neq, \in, \subset, \subseteq$ or $\notin$. Following is the description of *availableDate.Computer* $\leq$ *approvalDate.Insurance* $\leq$ *date.Pickup*.

$\{availableDate.Computer, approvalDate.Insurance,$
$date.Pickup\}, POST, availableDate.Computer$
$\{availableDate.Computer \leq approvalDate.Insurance,$
$approvalDate.Insurance \leq date.Pickup\},$

4. THE MATCHING APPROACH

This section provides details of the proposed composite matching technique. This technique requires: (i) WS-$\mathcal{ALUE}$ [6] service descriptions, (ii) user requests structured according to the guidelines given in Section 3, and (iii) the terms in descriptions (service descriptions and user requests) to be defined in a meta-ontology [2]. It locates services that conform to independent global constraints. Let *gc* be a constraint that restricts the attributes $[a_1, \ldots, a_n]$ of services of types $[S_1, \ldots, S_m]$. A service tuple $[s_1, \ldots, s_m]$ is a composite service that **conforms** to *gc* if:

1. $\forall s_i$, s_i is of type S_i,
2. $\forall s_i$, s_i is described using an attribute a'_i, where $a'_i \in \{a_1, \ldots, a_n\}$, and
3. $[v_1, \ldots, v_n]$ assigned to $[a'_1, \ldots, a'_n]$ of $[s_x, \ldots, s_y]$ conform to *gc*.

$[a'_1, \ldots, a'_n]$ of $[s_1, \ldots, s_y]$ are referred to as **restricted attributes**.

The proposed approach consists of three phases: (1) candidate acquisition, (2) service indexing and (3) composite service acquisition. The first phase locates services of the different types in a composite service template. The second phase identifies restricted attributes of services and indexes them based on the assigned values. The final phase retrieves the values that conform to a given constraint and combines services that assign those values to their restricted attributes (to form conforming composite services).

(1) Candidate Acquisition

This phase describes a way of locating candidate services. A candidate service is a service of a particular type, and the proposed approach locates such services for all the types in a composite service template. By locating candidate services, it ensures that the constituent services of a composite service are of appropriate types.

Let S_m be a service type and s_n a given service. We also denote by o_m, c_m and r_m the operation, the affected concept and the role that describe S_m. For s_n we denote its elements by o_n, c_n and r_n. The service s_n is a candidate of type S_m if o_n substitutes o_m, c_n substitutes c_m, and r_n substitutes r_m. The substitutions are performed with the substitution graphs in a meta-ontology [2]. Each candidate s_i of type S_i is placed in a list of candidates $candidates(S_i)$. Given the available services W and a composite service template CST (defined with the service types $S_x, \ldots, S_y$), this phase generates a set of candidate lists *Candidates*, where *Candidates*=$\{candidates(S_x), \ldots, candidates(S_y)\}$.

(2) Service Indexing Phase

This phase identifies the restricted attributes of candidate services and indexes them in a two dimensional structure based on the values they assign to these attributes. An attribute is restricted, if the assigned values are restricted by a user constraint. Such attributes need to be identified because whether a particular candidate service can be included in a conforming composite service is determined based on the values assigned to them. Since this technique is semantic-based, an attribute of a service is considered as a restricted attribute if it is semantically related to one that is used to specify a global constraint.

Consider the constraint *availableDate.Computer* $\leq$ *approvalDate.Insurance* $\leq$ *date.Pickup* described in Section 3. This restricts the attributes *availableDate.Computer*, *approvalDate.Insurance* and *date.Pickup* of services of types *Sales(Computer)*, *Insure(Computer)* and *Ship(Computer)*[1]. A service *ComputerSales-I* of type *Sales(Computer)* described with *date.Dispatch* exists. *date.Dispatch* is semantically related to *availableDate.Computer* according to the ontological relationships. In such a situation, the restricted attribute *time.Assemble* of *ComputerSales-I* is a restricted attribute, because the values assigned to it are restricted by *date.Dispatch* $\leq$ *approvalDate.Insurance* $\leq$ *date.Pickup*.

Candidate services need to be indexed based on the values assigned to such attributes because of two reasons.

1. The proposed approach first determines the values that conform to a given constraint. Then, it combines services that assign those values to their restricted attributes to form composite services. Such an approach requires the list of values assigned to each restricted attribute by the candidate services. These lists can be easily obtained by indexing services based on the assigned values. Let us assume we have the services described in Table 1. If these services are indexed in the way shown in Figure 5, then the lists of values assigned to the restricted attributes can be easily obtained. We assume that the values assigned to the attributes *avail-*

[1] *Sales(Computer)*, *Insure(Computer)* and *Ship(Computer)* are the abbreviated forms of [*Sales()*, *Computer*, *Sales_Assistant*], [*Insure()*, *Computer*, *Sales_Representative*] and [*Ship()*, *Computer*, *Shipping_Agent*] respectively.

ableDate.Computer, *approvalDate.Insurance* and *date.Pickup* are specified in date format and that the required ontological descriptions are available.

Table 1: Sample services

Service	Type	Assigned Values
ComputerSales-I	Sales(Computer)	183-2007, 185-2007
ComputerSales-II	Sales(Computer)	4TH_JULY
Insurance-I	Insure(Computer)	3RD_JULY, 4TH_JULY
Insurance-II	Insure(Computer)	1ST_JULY, 5TH_JULY
Shipping-I	Shipping(Computer)	4TH_JULY
Shipping-II	Shipping(Computer)	2ND_JULY, 3RD_JULY

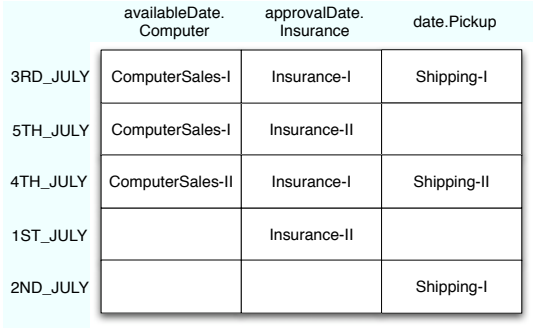


Figure 5: Indexed Services

2. When combining services to form composite services, those that assign a particular value to a restricted attribute can be retrieved quickly. Let us assume that tuple [*3RD_JULY*, *3RD_JULY*, *4TH_JULY*] of conforming values is given for the above scenario. In such a situation if the available services are not indexed, 10 operations (3+4+3) may have to be performed to locate those services that assign these values to their restricted attributes. However, it would only take *3* operations if the services are indexed (see Figure 5).

The proposed approach generates a set of Slot Lists to index candidate services based on the values they assign to restricted attributes. Figure 6 provides a global view of this data structure. It depicts a set of Slot Lists $\{L_1, L_2, L_3, L_4\}$ generated for a global constraint that restricts the attributes $[a_1, a_2, a_3, a_4]$. Note that a Slot List L_i is generated for every attribute a_i. A slot l_{ij} is included in a list L_i for each value v_{ij} that can be assigned to attribute a_i. Each slot contains a set of services. A service s_k is included in a slot l_{ij} if it is able to assign the value v_{ij} to a_i. That means, the service description of s_k is specified using either

- a_i: v_{ij} is assigned to a_i, or
- an attribute a'_i to which a value v'_{ij} is assigned: a'_i is semantically related to a_i and the relationship causes v'_{ij} to be mapped to v_{ij}.

For example, in *Shipping-I* in Table 1 the restricted attribute is *date.Dispatch* and the assigned values are not specified in "date-month" format. However, since the ontological relationships semantically relate *date.Dispatch* to *availableDate.Computer*, and map the assigned values to *3RD_JULY* and *5TH_JULY*, *Shipping-I* is placed in the slots that correspond to ⟨*availableDate.Computer*, *3RD_JULY*⟩ and ⟨*availableDate.Computer*, *5TH_JULY*⟩. When generating a set of Slot Lists it is assumed that the domain of values that can be assigned to a particular attribute and the number of attributes semantically related to a particular attribute are bounded.

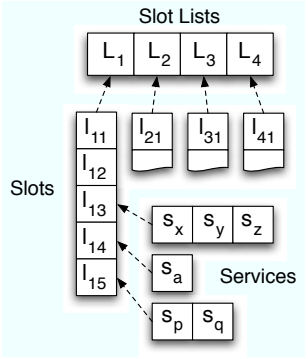


Figure 6: Set of Slot Lists

(3) Composite Service Acquisition

The task performed in this phase is divided into two steps. First, values that conform to a given independent global constraint are identified. Then, services that assign these conforming values are combined to form composite services.

The proposed approach models an independent global constraint as a directed acyclic graph *G=(V, E)*, where *V* is a set of nodes and *E* is a set of arcs. The nodes of such a graph represent the attributes and the arcs model the relationships between the attributes. Each arc and the two connected attributes model a binary attribute comparison of a constraint. Figure 7 depicts a graph modeling a constraint that restricts a_1-a_6, where the binary attribute comparison are specified with the operators $\{o_{12}, o_{13}, o_{24}, o_{34}, o_{35}, o_{46}\}$.

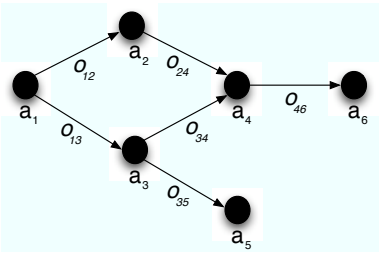


Figure 7: Directed Acyclic Graph modeling an Independent Global Constraint

Values that conform to a constraint are identified by generating instances of such graphs. A tuple of values assigned to the nodes of a graph is an instance. For example, the values $[v_1, v_2, v_3, v_4, v_5, v_6]$, where each value v_i is assigned to an attribute a_i, forms an instance of the graph depicted in Figure 7. Such an instance conforms to a given constraint if each of its values conform to the binary attribute comparisons represented by the arcs. A value that conforms to the binary attribute comparisons is referred to as an *Arc Consistent Value*.

Definition 2 (Arc Consistent Value). *Let gc be an independent global constraint that restricts the values assigned to attributes* $\{a_1, \ldots, a_n\}$, *where* a_i *is a non-free attribute and* $a_i \in \{a_1, \ldots, a_n\}$. $\{a_j, \ldots, a_k\}$ *is the set of attributes connected to* a_i *with arcs* $\{o_{ij}, \ldots, o_{ik}\}$, *where* $\{a_j, \ldots, a_k\} \subseteq \{a_1, \ldots, a_n\}$. $[v_j, \ldots, v_k]$ *is a tuple of values*

assigned to $\{a_j, \ldots, a_k\}$. *A value* v_i *assigned to* a_i *is* ***arc consistent****, if the relationships represented by the arcs* $\{o_{ij}, \ldots, o_{ik}\}$ *exist between* v_i *and the values in* $\{v_j, \ldots, v_k\}$.

An instance of a graph that consists of arc consistent values is a tuple of conforming values. Such tuples are referred to as *Arc Consistent Instances*.

DEFINITION 3 (ARC CONSISTENT INSTANCE). *Let gc be an independent global constraint that restricts the values assigned to attributes* $\{a_1, \ldots, a_n\}$, *where* a_i *is a non-free attribute and* $a_i \in \{a_1, \ldots, a_n\}$. *A tuple of values* $[v_1, \ldots, v_n]$ *assigned to* $\{a_j, \ldots, a_k\}$ *is an* ***arc consistent instance*** *if each value* v_j *in* $\{a_j, \ldots, a_k\}$ *is locally arc consistent.*

Identifying an arc consistent instance is NP-Hard. If a given relational constraint restricts n attributes and m arc consistent values can be assigned to each attribute, then m^n combinations of values may have to be considered to identify an arc consistent instance. The proposed technique defines two separate approaches to perform this task. The first approach uses a basic greedy algorithm (backtrack-free search algorithm [3]) to identify arc consistent instances. The second approach derives them using domain rules [2]. The reason for proposing two separate approaches will be given later. We will refer to the first one as the *optimised approach*, and the second as the *derivation-based approach*.

(1) The Optimised Approach

This approach employs a greedy algorithm to locate arc consistent instances in polynomial time. It attempts to generate an arc consistent instance for each value that is assigned to the free attribute. When generating these instances, first, a value assigned to the free attribute is selected. Then, based on that value, arc consistent values are selected for the remaining attributes. If such a value cannot be located for a particular attribute, then the technique moves to the next value that is assigned to the free attribute. It does not perform any backtrack to consider alternative (arc consistent) values for attributes. Figure 8 shows the way in which values are selected by this approach when services are indexed as in Figure 5. A solid line indicates an arc consistent instance, whereas a dashed line indicates that the assignments have led to a dead-end (i.e. an arc consistent instance is not located for the corresponding value that is assigned to free attribute). When *3RD_JULY* is selected for *availableDate.Computer*, the arc consistent instance [*3RD_JULY*, *3RD_JULY*, *3RD_JULY*] is located. However, when *5TH_JULY* is considered, an arc consistent value cannot be located for *date.Pickup*.

Once an instance is located, the services in the slots that correspond to the arc consistent values are combined. Let us consider a constraint *gc* that restricts the attributes $a_1, \ldots, a_n$ of services of types $S_1, \ldots, S_n$. We also assume that a set of Slot Lists *SL* has been generated for *gc*. A slot at a location $\langle a_i, v_i \rangle$, which corresponds to an attribute a_i and a value v_j, contains services of type S_i which assign the value v_j to attribute a_i. Therefore, if $[v_1, \ldots, v_n]$ is an arc consistent instance, then the services at the slots $\langle a_1, v_1 \rangle, \ldots, \langle a_n, v_n \rangle$ would form conforming composite services. In the above scenario, when [*3RD_JULY*, *3RD_JULY*, *3RD_JULY*] is located, the services at the slots ⟨*availableDate.Computer.3RD_JULY*⟩, ⟨*approvalDate.Insurance.3RD_JULY*⟩ and ⟨*date.Pickup.3RD_JULY*⟩ are combined to form the conforming composite service [*ComputerSale-I*, *Insurance-I*, Shipping-I].

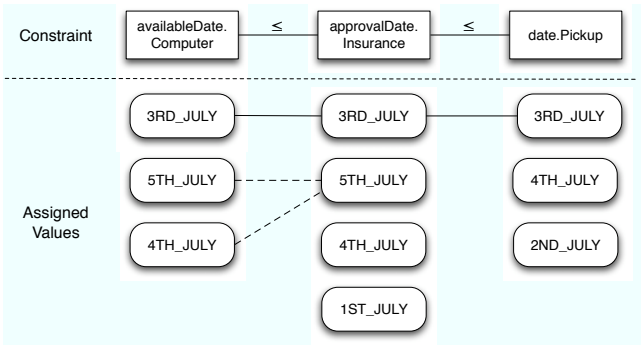


Figure 8: Value Selection

The algorithm for the optimised approach is given in Algorithm 1. This requires a set of Slot Lists and an independent global constraint.

```
1  locateServices(SL[][], gc)
2  removeEmptySlots(SL);
3  {c0.a0, ..., cn.an}← restrictedAttributes(gc);
4  {atcp, ..., atcq}← attributeComparisons(gc) ;
5  for i←0; i<SL[0].length; i++ do
6      values[0]←SL[0][i].getValue();
7      for j←1; j<SL.length; j++ do
8          {atcx, ...,
           atcy}← includes(cj.aj, {atcp, ..., atcq});
9          for k←0; k<SL[ j].length; k++ do
10             values[j]←SL[j][k].getValue();
11             if isArcConsistent(⟨values[0], ..., values[j]⟩,
               {actx, ..., acty}) then
12                 break;
13             end
14             else
15                 reset(values[j]);
16             end
17         end
18     end
19     if containsInitialValues(values) then
20         continue;
21     end
22     for each service s0 in SL[0][values[0]].getServices() do
23         for each service s... in
           SL[...][values[...]].getServices() do
24             for each service sn in
               SL[n][values[n]].getServices() do
25                 conforming_services.add(s0, ..., sn);
26             end
27         end
28     end
29 end
```

Algorithm 1: Locating Services - Greedy Approach

First, the empty slots are removed from the Slot Lists (line 2). These slots need to be removed because the corresponding values are not assigned to the relevant restricted attributes by any candidate service. Then, this algorithm attempts to generate an arc consistent instance for each value that is assigned to the free attribute (lines 5-17). Once the instances are generated, the services at the indicated slots are retrieved (lines 20-22) and combined to conforming composite services (lines 23-28). The time complexity of Algo-

rithm 1 is exponential. If a given constraint restricts p attributes, there are m conforming value tuples, and each slot (in the set of Slot Lists) contains q services, then $m^*(p^q)$ composite services would be generated. However, this technique is "generally" **polynomial** since Algorithm 1 does not retrieve any non-conforming composite services. This algorithm identifies tuples of arc consistent values in polynomial time and only combines services that assign those values to their restrict attributes. Those that do not assign arc consistent values are not considered.

The main drawback of this approach is that it may return false negatives. That means, it may not retrieve all the conforming composite services. In the scenario described in Figure 5, [*ComputerSale-I*, *Insurance-I*, Shipping-I], [*ComputerSale-I*, *Insurance-I*, Shipping-II] and [*ComputerSale-II*, *Insurance-I*, Shipping-II] are conforming composite services, because [*3RD_JULY*, *3RD_JULY*, *3RD_JULY*], [*3RD_JULY*, *3RD_JULY*, *4TH_JULY*], [*3RD_JULY*, *4TH_JULY*, *4TH_JULY*] and [*4TH_JULY*, *4TH_JULY*, *4TH_JULY*] are consistent instances. However, only [*3RD_JULY*, *3RD_JULY*, *3RD_JULY*] is located by the optimised approach since it employs a greedy algorithm.

(2) The Derivation-based Approach

This approach uses domain rules to derive arc consistent instances. Like the previous approach, this one may also return false negatives. However, this approach returns the used domain rules along with each composite service so that the context in which it is located can be understood by a user. Let us consider the constraint *availableDate.Computer* $\leq$ *approvalDate.Insurance* $\leq$ *date.Pickup* and the services given in Table 1. If the technique that identifies arc consistent instances selects *3RD_JULY* for *availableDate.Computer*, then the rules in Figure 4 can be used to derive the tuple [*3RD_JULY*, *3RD_JULY*, *4TH_JULY*][2]. Even though the discovery technique only returns [*ComputerSale-I*, *Insurance-I*, *Shipping-I*] (and not [*ComputerSale-I*, *Insurance-I*, *Shipping-I*] and[*ComputerSale-II*, *Insurance-I*, *Shipping-II*]), a user would be able to understand the context in which it was retrieved if the rules in Figure 4 are returned.

When generating an arc consistent instance, values are derived for the non-free attributes based on one that is assigned to the free attribute. Since we assume that a rule is a function that derives a value for a derived attribute based on one that is assigned to a determinant attribute, a set of rules can be used to uniquely determine the values that should be assigned to the non-free attributes based on one that is assigned to the free attribute. That means, the domain rules are used to approximate an independent global constraint to a strictly dependent global constraint.

Definition 4 (Approximatable Constraint). *Let gc be an independent global constraint which restricts the values assigned to attributes* $\{a_x, \ldots, a_y\}$*, where* a_x *is its free attribute and* V_x *is the set of values assigned to* a_x*.* $\{r_m, \ldots, r_n\}$ *is the set of available rules. gc is approximatable if* $\exists v_x \in V_x$ *and* $\exists\{r_p, \ldots, r_q\}\subseteq\{r_m, \ldots, r_n\}$*, which can derive the values* $v_{x+1}, \ldots, v_y$ *based on* v_x*, such that* $[v_x, v_{x+1}, \ldots, v_y]$ *forms an arc consistent instance.*

Let us consider the constraint depicted in Figure 7. This could be considered as an **approximatable** (see Figure 9). The constraint is approximatable if the values $[v_2\text{-}v_6]$ assigned to the attributes $[a_2\text{-}a_6]$ are derived using the rules $\{r_{12}, r_{13}, r_{14}, r_{35}, r_{46}\}$, based on the value v_1 assigned to a_1, and $[v_1\text{-}v_6]$ form an arc consistent instance.

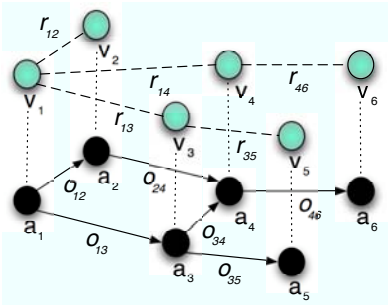


Figure 9: Approximatable Constraint

Approximating an independent global constraint using domain rules is NP-hard. Let us consider a constraint *gc* that restricts the attributes $\{a_x, \ldots, a_y\}$, where a_x is its free attribute and $|x, \ldots, y|=m$. For each attribute a_i in $\{a_{x+1}, \ldots, a_y\}$ there is a set of rules R_i where each r_i is able to derive a value v_i for a_i based on a value v_x assigned a_x. In such a situation $(m\text{-}1)^n$ combinations of rules may have to be considered to establish if *gc* is approximatable. Hence, the proposed, uses a greedy approach to select rules. If a particular rule r_i is able to derive a value v_i for an attribute a_i in $\{a_{x+1}, \ldots, a_y\}$ based on a value v_x assigned to a_x, such that v_i is arc consistent, then it is selected. Then, the technique proceeds to select a rule that derives a value for a_{i+1} in $\{a_{x+1}, \ldots, a_y\}$. Any other rule that derives a value for a_i based on value v_x assigned to a_x is not considered.

The algorithm that determines whether a given constraint is approximatable is given below (see Algorithm 2). This algorithm requires an independent global constraint, the available domain rules and a set of Slot Lists. A given constraint is considered to be approximatable if at-least one value tuple (i.e an arc consistent instance) to be derived based on a value that is assigned to its free attribute. This algorithm attempts to generate such an instance for each value assigned to the free attribute (lines 4-6). When generating an instance, it iterates through the available rules (line 9) and attempts to derive an arc consistent value (lines 10-20) for each non-free attribute (line 7). If a derived value is arc consistent, both the value and the rule used to derive it are stored in a temporary array (lines 13 and 16). If not, the corresponding elements of the two arrays are reset and another value is derived using a different rule (line 19). Finally, every derived arc consistent instance is placed in a list along with the rules used for the derivations (line 26). The complexity of Algorithm 2 is polynomial (i.e. $v^*x^*y^*z$, where v is the number of values assigned to the free-attribute, x is the number of non-free attributes, y is the number attributes semantically related to a given restricted attribute and z is the number of rules). This algorithm assumes that (i) each attribute is semantically related to a bounded number of attributes, and (ii) a finite number of domain rules are available.

Once Algorithm 2 is executed, the derived arc consistent instances are obtained from *derived_values_list*. If it is not empty, the services in the slots (of the Slot Lists) that cor-

[2]According to defined rules, an insurance policy has to be approved on the same day and computers have to be picked up a day later for shipping.

```
1  approximate(gc, R, SL)
2  {c_0.a_0, ..., c_n.a_n}← restrictedAttributes(gc);
3  {atc_p, ..., atc_q}← attributeComparisons(gc);
4  for i←0; i<SL[0].length; i++ do
5      initialize values[], rules[];
6      values[0]←SL[0][i];
7      for each attribute c_j.a_j ∈ {c_1.a_1, ..., c_n.a_n} do
8          for k←0; k<j; k++ do
9              for each rule r_m ∈ R do
10                 c_x.a_x ← determinantAttribute(r_x);
11                 c'_x.a'_x ← derivedAttribute(r_x);
12                 if (c_k.a_k==c_x.a_x || isSemanticallyRel-
                   ated(c_k.a_k, c_x.a_x)) && (c_j.a_j==c'_x.a'_x ||
                   isSemanticallyRelated(c_j.a_j, c'_x.a'_x))
                   then
13                     values[j]←
                       derive(c_k.a_k, c_j.a_j, values[k], r_m);
14                     {atc_c, ..., atc_d}←
                       includes(c_j.a_j, {atc_p, ..., atc_q});
15                     if isArcConsistent(⟨values[0], ...,
                       values[j]⟩, {act_c, ..., act_d}) then
16                         rules[j]←r_m; go to line 9;
17                     end
18                     else
19                         reset(values[j]); rules[j]← NULL;
20                     end
21                 end
22             end
23         end
24     end
25     if ¬containsInitialValues(values) then
26         derived_values_list.add(values, rules);
27     end
28 end
```

Algorithm 2: Approximating Independent Global Constraints

respond to the values in the instances are combined (like the proposed optimised approach) to form composite services. The rules used to derive an arc consistent instance are returned with each corresponding composite service.

5. DISCUSSION

Both the optimised and derivation based approaches have better complexity than existing techniques, are they are both semantic-based and they locate conforming services in polynomial time. Hence, they are able to locate conforming composite services efficiently in situations where user requests and service descriptions are syntactically heterogeneous. However, the main drawback of our approaches is that they retrieve duplicate entries. A duplicate entry occurs if a composite service that includes a particular sequence of constituent services is retrieved more than once.

Let *gc* be a global constraint that restricts the attributes $a_x, \ldots, a_y$, where the services $s_x, \ldots, s_y$ are able to assign the values $[v_{x1}, \ldots, v_{y1}]$ and $[v_{x2}, \ldots, v_{y2}]$ to the restricted attributes. If both tuples of values conform to *gc* (i.e. they are arc consistent instances), one duplicate entry would be retrieved. In a situation in which n conforming value tuple can be assigned to the restricted attributes by a tuple of services, where $n>1$, *n-1* duplicate entries would be retrieved by these techniques.

However, the number of these duplicate entries only increases at a polynomial rate. If there are p conforming composite services, the number of duplicate entries that can be generated would be $p*q$, where q is the number of conforming value tuples assigned to each tuple of restricted attributes.

6. EVALUATION

In this section the experiments we performed to compare the proposed approaches against those in [2] and [11]. The metrics used are performance and recall. Recall was calculated based on the number of unique conforming composite services retrieved. The technique proposed in [11] is implemented with JSHOP[3] and others were done in Java (JDK 1.5.0). A MySQL 5.0 Community Server database is used as a registry to store service descriptions and ontological descriptions. Experiments were conducted on Pentium IV 3.0 GHz machines with 1 Gigabyte of memory.

Up to our knowledge, there is no standard data set that can be used to evaluate semantic-based service discovery techniques. We were unable to obtain a real-world data set of semantic-based web service descriptions. None of the composite matching techniques that we came across [1, 8, 9, 11, 12] defined an approach to generate test data. We have therefore tried to generate our own test data set using a "appropriate" methodology which uses the following steps. Even though this may not be the best way, we believe that it will give us good indication about the quality of the proposed approaches (when compared to existing ones).

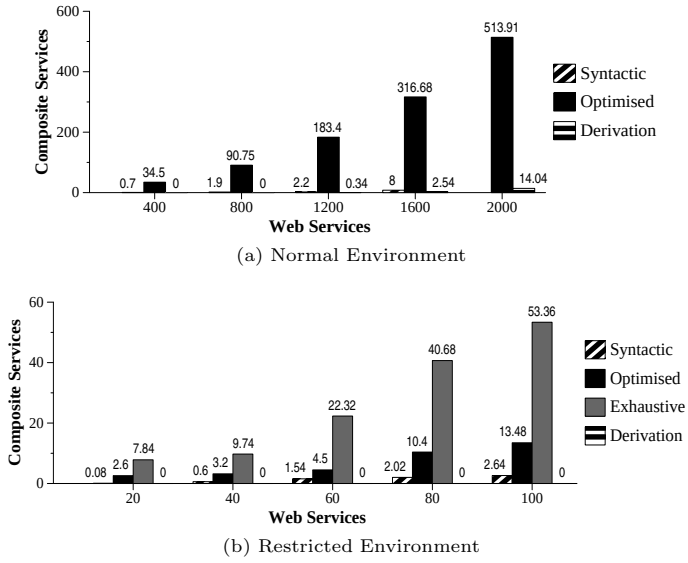
Details were extracted from sample ontologies and an ontology that consists of 157 concepts, 19 roles and 27 operations was randomly generated. 2000 substitution graphs and transformations graphs were generated to define ontological relationships. Service descriptions were created with elements that were randomly selected from the ontology. Each description had an operation, an affected concept, a role and between 20 to 30 pre-conditions and post-conditions.

A similar approach was used in [2] to generate test data to evaluate implementations of their semantic-based matching techniques. Experiments were conducted by varying the number of available service descriptions and service types in composite service templates. Between 400 to 2000 service descriptions and composite service templates which include between 5 to 25 service types were considered. Since the technique proposed in [2] employed an exponential algorithm, it was not feasible to execute it in such an environment. Hence, the experiments that included the exhaustive approach were conducted in a restricted environment, which contained between 20 to 100 service descriptions and composite service templates that include between 2 to 10 service types.

The recall levels achieved by the exhaustive and the proposed optimised approach were higher than that achieved by the syntactic approach. Unlike the later approach, the semantic-based approaches matched syntactically heterogeneous service descriptions and user requests using ontological relationships. The exhaustive approach considered all the possible combination of services; and for each combination it employed an exhaustive algorithm to evaluate the values assigned to the restricted attributes (i.e. it checks if the values assigned to the restricted attributes conform to a given constraint). The proposed approach only considered

[3]A Java-based implementation of SHOP.

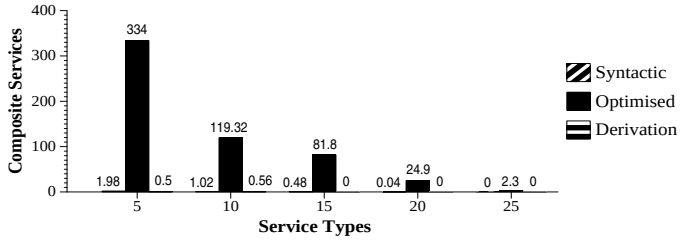
a limited number of values. Hence, the exhaustive approach retrieved more services than the proposed approach. The number of services retrieved by all the techniques decreased, as the number of service types increased. The values assigned to the restricted attributes of candidate services had to conform to an increasing number of binary attribute comparisons. The derivation-based approach did not retrieve many services because the rules required to approximate a given constraint were not available. The used random process did not generate the required rules.



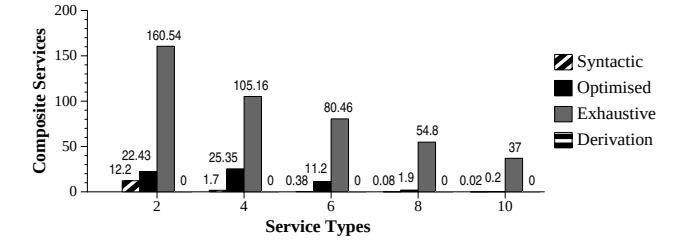
(a) Normal Environment

(b) Restricted Environment

Figure 10: Recall - Varying no. of Available Services



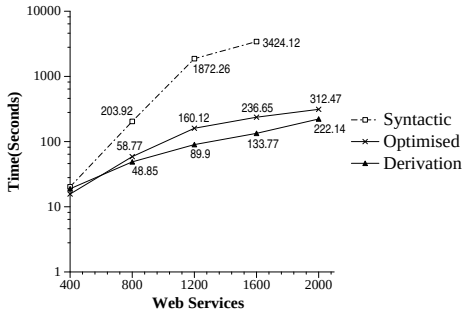
(a) Normal Environment



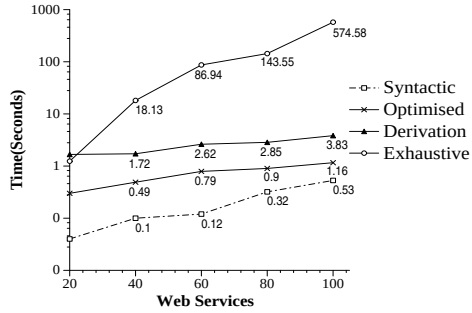
(b) Restricted Environment

Figure 11: Recall - Varying no. of Service Types

The performance of the optimised technique was 76% better than that of the syntactic approach in the experiments varied the number of available services. The syntactic approach used JSHOP, which performs a combinatorial depth-first search to locate conforming composite services. On the other hand, the proposed local optimisation-based approach determined the values that conform to a given constraint in polynomial time and only combined services that assign those values. The performance of the syntactic approach was 32% better in the experiments that increased the number of service types in a composite service template. Since the optimised approach is semantic-based it retrieved more candidate services than the syntactic approach. The time taken by the devised approach to index services in the Slot Lists increased at a rapid rate with an incrementing number of service types. The exhaustive approach was the least efficient in the restricted environment. In the two worst cases (with 100 available services and 10 service types) it took around 9 minutes 35 seconds, and and 46 minutes and 25 seconds respectively, whereas the proposed approach only took 1.16 seconds and 1.47 seconds.



(a) Normal Environment



(b) Restricted Environment

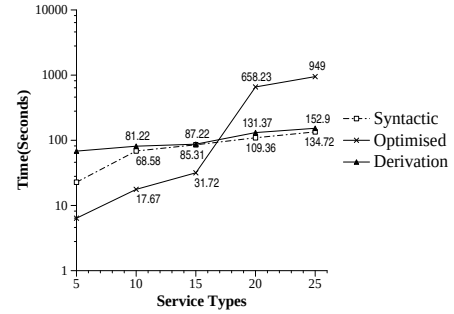
Figure 12: Time taken - Varying no. of Available Services

7. RELATED WORK

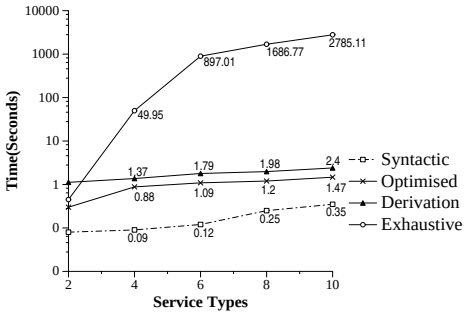
Semantic-based approaches retrieving composite services were introduced in [1, 4, 8]. However, the technique in [4] is an approximate approach which does not consider the values assigned to the attributes. None of the remaining techniques neither consider global constraints nor locate composite services that conform to such constraints.

Wu et al. in [11] and Sirin et al. in [9] considered both local and global constraints. Their approaches require service descriptions and user requests to be specified with OWL-S. SHOP-2 (Simple Hierarchical Planner-2) is used to locate an orchestration of services that form a conforming composite service. First, a "SHOP method" is generated according to the OWL-S process in a request. Then, the request and available service descriptions are converted to operator instances. Both approaches directly used the unification functions of SHOP-2 to match domain instances to operators. Therefore basic string matching functions were used to match user requests to service descriptions. The

approach in [9] extends [11] by incorporating a Description Logic reasoner. However, this is only used to check whether the pre-conditions of a given operator (in the hierarchy) are entailed by the state of the planner. Therefore, the approach in [9] locates conforming composite services when a constraint in a request is specified with attributes that have different scopes, but not when requests and service descriptions are syntactically heterogeneous.



(a) Normal Environment



(b) Restricted Environment

Figure 13: Time taken - Varying no. of Service Types

Zeng et al. [12] introduced an approach requiring a request (for a composite service) to be specified with a state chart diagram, where each state models a service type. Services were described with a quality of service model that consists of attributes such as execution price, execution duration, reliability, availability, and reputation. Both local and global constraints were considered in their approach. However, these constraints were limited to those that can be specified with an attribute of the quality of service model. A syntactic-based integer programming technique that focuses on local optimisation is used to match the user requests to the service descriptions.

8. CONCLUSION

This work proposes two service discovery approaches that locate services that conform to independent global constraints. The first approach uses a greedy algorithm to identify conforming values and locate composite services. However, since this approach is not sound, a second approach that approximates a given constraint is proposed. This approach uses domain rules to derive values that conform to a given constraint and combines services that assign those values to their restricted attributes. The domain rules used to derive the conforming values are returned with each service so that context in which it is located can be understood by a user. Services are indexed in a two dimensional data structure (a set of Slot Lists) to improve the performance of these approaches. Experimental results showed that the proposed optimised (initial) approach performs better than any existing technique. It also achieves higher recall than a syntactic-based approach.s As future work, we intend to develop a composite matching technique that considers multiple user constraints of various types.

Acknowledgments

We would like to thank the ARC (Australian Research Council) for the support given towards this work, under the Linkage Project no. LP0667600 titled "An Integrated Infrastructure for Dynamic and Large Scale Supply Chain".

9. REFERENCES

[1] R. Akkiraju, B. Srivastava, A. Ivan, R. Goodwin, and T. Syeda-Mahmood. SEMAPLAN: Combining Planning with Semantic Matching to Achieve Web Service Composition. In *Proceedings of the International Conference on Web Services*, pages 37–44, 2006.

[2] I. Elgedawy, Z. Tari, and M. Winikoff. Exact Functional Context Matching for Web Services. In *Proceedings of the 2nd International Conference on Service Oriented Computing*, pages 143–152, 2004.

[3] E. Freuder. A Sufficient Condition for Backtrack-Free Search. *Journal of ACM*, 29(1):24–32, 1982.

[4] N. Gooneratne, Z. Tari, and G. Craske. Composite Matching Technique for Semantic-based Service Discovery. In *Proceedings of the Australian Undergraduate Conference*, 2004.

[5] N. Gooneratne, Z. Tari, and J. Harland. Matching Strictly Dependent Global Constraints for Composite Web Services. In *Proceedings of the European Conference on Web Services*, pages 139–148, 2007.

[6] N. Gooneratne, Z. Tari, and J. Harland. Verification of Web Service Descriptions using Graph-based Traversal Algorithms. In *Proceedings of the ACM Symposium on Applied Computing*, pages 1385–1392, 2007.

[7] B. Medjahed and A. Bouguettaya. A Multilevel Composability Model for Semantic Web Services. *Transactions Knowledge and Data Engineering*, 17(7):954–968, 2005.

[8] M. Paolucci, K. P. Sycara, and T. Kawamura. Delivering Semantic Web Services. In *Proceedings of the International World Wide Web Conference*, 2003.

[9] E. Sirin and B. Parsia. Planning for Semantic Web Services. In *Proceedings of International Semantic Web Conference, Workshop on Semantic Web Services*, November 2004.

[10] K. P. Sycara, S. Widoff, M. Klusch, and J. Lu. Larks: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5(2):173–203, 2002.

[11] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web Services Composition using SHOP2. In *Proceedings of the International Semantic Web Conference*, pages 195–210, 2003.

[12] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-Aware Middleware for Web services Composition. *Trans. on Software Engineering*, 30(5):311–327, 2004.