Semantic Service Composition with QoS End – to – End Constraints via AND/OR Graphs

Xhemal Zenuni[1], Ivan Momtchev[2]

[1] Faculty of Contemporary Sciences and Technologies, South East European University
Tetovo, 1200, Macedonia

[2] Faculty of Computer Systems and Control, Technical University of Sofia
Sofia, 1000, Bulgaria

Abstract

In this paper we present AND/OR graphs as a unifying framework for semantic service composition that considers users QoS constraints. The main virtues of this representation among others are its ability to express semantic inference and to deal with QoS constraints from different perspectives. In addition it correctly handles multiple inputs/outputs of services, and allows high degree of automation. Once service dependencies and QoS features are formalized as AND/OR graph, we apply a search algorithm to discover composite services that considers user QoS end – to – end preferences. The implementation of a prototype system and the experimental results support our underlying hypothesis that AND/OR graphs are not only elegant and expressive formalism for addressing QoS – aware semantic service composition, but efficient as well.

Keywords: *QoS, semantic service composition, AND/OR graphs.*

1. Introduction

In recent years, there has been a significant increase of the number of provided services, resulting in a scale of over 25.000 services available and indexed on Internet [1][2]. This trend has significant ramification to the development of different types of distributed applications, like Grid [3] or Cloud [4] computing systems among others, based on Service Oriented Architectures (SOA) principles.

Despite the diversity and service proliferation, often user requirements are so complex that no single service can satisfy the required functionality alone. Therefore, the need for efficient composition oriented service discovery and ranking is becoming crucial in such settings.

Service composition has gained significant momentum in recent research works, yet current approaches and techniques suffer in several directions. To deal with service proliferation and automation, the composition issue has been akin to Artificial Planning problem [5]. However, in traditional AI approaches, QoS is largely ignored. Apart from functional specification, non - functional requirements such as quality of service (QoS) are becoming major concern as natural discrimination and/or ranking factor when several equivalently functional solutions exist. Most of the works that considers QoS aspects are based on directed acyclic graphs (DAG) and Integer programming (IP) [6][7] for QoS end – to – end optimization. The problem with these approaches is that they require the user to provide pre – defined execution plans as a request, or the determination of service dependencies is assumed. Another problem is the assumption that the graph has to be acyclic, as they cannot deal efficiently with them. On the other hand, IP requires high running complexity, having serious implication on time required to discover composite services efficiently in large systems with many service nodes and potential selections. Finally, most of current methods lack systematic evaluation based on more comprehensive evaluation methodology.

The rest of this paper is organized as follows: Section 2 presents the semantic framework to describe service signatures and formalizes the service dependencies in form of an AND/OR graph and we evaluate the expressiveness of such structure. Section 3 analyzes the basic constructions structures contained in AND/OR graphs, and it presents the objective functions and aggregation patterns of several important QoS parameters in AND/OR graphs. Section 4 explains the search algorithm employed for discovering composite services with QoS constraints. Section 5 presents the system implementation, evaluation metrics and experimental results of the adopted approach. Finally, Section 6 concludes the work and indicates future research directions.

2. Semantic Services and Service Dependency Graph

In semantic services, the description of services is raised at ontological level. This unambiguous description of service signatures, from functional and non - functional aspects is crucial to the determination of service inter – dependencies and to their transformation to an AND/OR graph known as Service Dependency Graph (SDG) [8]. However, as SDG is not tightly coupled with specific semantic service framework, the problem of services is given by the following definitions:

Definition 1: SOA consists of a set of available services offered by different providers and used to build distributed applications. Formally, $S = \{S_1, S_2, \ldots, S_n\}$, where S_i is a specific service. Each service has a set of methods that are the most fined grained piece of functionalities and the most fundamental building blocks that are used to build the distributed application and resolve specific scientific or business problem. For simplicity, service and service methods are used interchangeably.

Definition 2: A service in upper – domain ontology is defined by the triple $S_i = \{S_{in}, S_{out}, Q_s\}$, where S_{in} is a set of input data types defined as concepts in specific domain ontology, S_{out} is a set of output data types defined as concepts in specific domain ontology, and Q_s is a set of QoS attributes, such as response time or availability also defined in some specific domain ontology. In addition, each specific QoS parameter is defined by a metric and a unit (non – measurable parameters, such as security related QoS aspects, may not have a unit).

The most important features of services are described in terms of I/O, which is in the line with the world wide efforts of the community to ontologically describe services and QoS, where each parameter can be mapped to a concept in some domain specific ontology. Once this semantic framework is given, the transformation of service signatures to AND/OR graph (as SDG) is straightforward.

Definition 3: The Service Dependency Graph is AND/OR graph showing the dependencies of services based on I/O and is defined as five tuple:

$$\gamma = (O, T, A, E, S) \tag{1}$$

where O is a set of non – terminal OR nodes. Those are the service I/O not provided directly by the requester. T is a set of terminal OR nodes. Those are the available inputs given by the requester. A is a set of AND nodes. Those nodes represent the available services (service methods) in SOA model. As services can be invoked only if all of its inputs are available at the time of invocation, in services there is AND logical connection from the input parts. E is a subset $(O \cup T) \times A \cup A \times (O \cup T)$ whose elements are directed arcs, showing the connection between services and their I/O. S is an auxiliary (dummy) AND node that is connected to the desired outputs requested in service composition problem. The connections of this node are changed during each request. In addition, theoretically costs can be assigned to arcs and nodes in such graph. However, as QoS implications are manifested only if a service is invoked, the QoS features are assigned as costs to AND nodes (service nodes) in SDG.

Table 1 data shows a simple service repository of services semantically described in terms of I/O and QoS aspects, and Figure 1 presents the Service Dependency Graph of the same.

Table 1: An illustrative service repository

Service	*Input*	*Output*	*QoS*
S1	D1, D2	D3	Response Time (rt) = 4ms.
S2	D2, D4	D5	Response Time (rt) = 1ms.
S3	D3, D5	D6, D7	Response Time (rt) = 2ms.
S4	D1	D8	Response Time (rt) = 6ms.

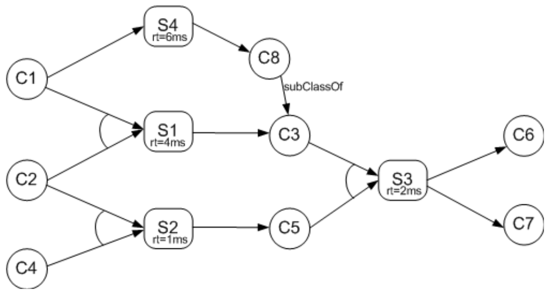

Fig. 1 AND/OR graph showing service dependencies

The following characteristics hold:

- There are no direct edges between two AND nodes [8]. This means that a given service operation cannot serve directly as input to another one, but only through produced output data or attributes.
- A data node, namely an OR node can be connected not only to an service operation node, but to another data node based on the semantic similarity constructs such as subsume or plug – in, synonyms, or other relations defined in the specific domain ontology.
- AND nodes correctly describe the services, as a service operation cannot be invoked until all its inputs are fully satisfied.
- The mapping of service descriptions to an AND/OR graph is straightforward, allowing high degree of automation.

3. Aggregation Patterns and Formulas

The solution to an AND/OR graph, if one exists, is a sub – graph rather than a path. This solution includes sequential and parallel edges, and two types of nodes, which all together form the basic constructs of composition patterns. They determine the QoS aggregations rules of individual services and allow us to verify whether a set of services selected for the composition satisfies the QoS requirements for the whole composition. Moreover they serve as a guide for the search algorithms.
AND/OR graphs support three different concrete aggregation patterns: a) Sequential pattern b) AND – join pattern and c) OR – join pattern, as depicted in Figure 2. They are similar to the abstract composition patterns identified by Jagger et al. [9] from van Der Aalst's pattern catalogue, with the difference that AND/OR graphs in practice are not as rich to support the whole set of patterns.

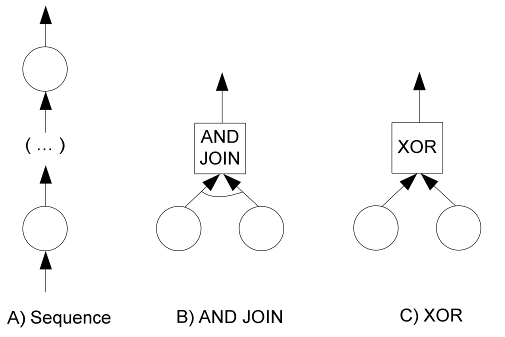

Fig. 2 Aggregation Pattern Structures in AND/OR Graphs

Based on the above mentioned composition patterns, different individual QoS parameters follow different aggregation types of formulas. Table 2 summarizes the aggregation formulas of some of the most important QoS parameters, recursively defined for AND nodes and OR nodes. The aggregation value of any node *n* is recursively defined by the function *h(n)*. In the given formulas, with c_i we denote the incoming nodes to node *n*. If the node *n* is of AND (service) type, then c_i are its inputs. If the node is an OR node (service I/O attribute), then c_i are the operation (AND) nodes that produce the node *n* or some other OR node that points node *n* based on some semantic similarity construct such as subclass, synonyms and so on.
For example, in AND nodes, the larger accumulated response time is used as the composition service response time, as the service has to wait for all its inputs to become available before continuing with its execution. And, as the objective is usually optimization, which means finding the composite service with minimum response time, in OR nodes we tend to minimize the response time.
On the other side, throughput is the maximum amount of information passing through a composite service. Both in sequence and parallel structures, the throughput is the minimum value which presents the bottleneck of the composite service. Moreover, in contrast to response time, the objective is to find the composite service with highest throughput, which is tried to be achieved in OR nodes.
In general, the QoS aggregation formulas fall within one of the following categories.
Certain category of QoS parameters cannot be aggregated. This is true for non – measurable parameters, and they are checked locally, even in end - to – end planning. The second category of QoS attributes follow critical path algorithm. The response time, execution time and several other QoS dimensions fall within this category. They have implications in calculating the overall QoS in parallel executions of different services. In composite services, the QoS of this category are calculated as the longest path from the initial state to the final state. This is important, as in parallel structures there is no pint to further minimize the lower value as there is no effect to the global QoS of the composite service. For the third category, the overall QoS is calculated as the sum of QoS dimensions of all involved individual services, regardless in parallel or sequential manner. The execution price of an execution plan that represents the composite service is one of the QoS attributes that is calculated as the sum of all involved service operations in that plan. And finally, certain QoS attributes such as reliability or availability are calculated as product of individual services involved in the overall composition plan.

Table 2: Aggregation formulas for some QoS parameters in AND/OR graph nodes

QoS Attribute	AND JOIN Structure	XOR JOIN Structure
Response Time (rt)	$h(n) = q_{rt}(n) + max(h(c_i))$	$h(n) = min(h(c_i))$
Execution Time (et)	$h(n) = q_{et}(n) + max(h(c_i))$	$h(n) = min(h(c_i))$
Throughput (thp)	$h(n) = min(q_{tp}(n), h(c_i))$	$h(n) = max(h(c_i))$
Availability Time(at)	$h(n) = q_{at}(n) * \prod_i h(c_i)$	$h(n) = max(h(c_i))$
Reliability (re)	$h(n) = q_{re}(n) * \prod_i h(c_i)$	$h(n) = max(h(c_i))$
Security (se)	local planning	local planning
Cost (price)	$h(n) = q_{price}(n) + \sum_i h(c_i)$	$h(n) = min(h(c_i))$

4. The Search Algorithm for Discovery of Composite Services

Once the service dependencies are formalized as an AND/OR graph, then the issue of composition discovery can be regarded as a search problem in that graph. This opens wide possibilities, as the problem of finding minimum solution graphs has attracted the attention of researchers for a long time, and several algorithms that try to solve this problem has been reported [10][11][12][13]. Earlier algorithms work on implicit graphs and are based on the assumption that the given graph is acyclic. As this assumption is not realistic in many real problems, including the service composition issue, in the second phase that includes the period from the early 90s, the efforts were toward the development of algorithms to solve explicit AND/OR graphs containing cycles. However, as there is no unified framework describing AND/OR graph search algorithms and no real benchmarks exist for their comparison, the task of selecting the right searching algorithm becomes difficult.

Initially, we have chosen to analyze, adopt, extend, and evaluate the REV* [13] search algorithm developed by Chakrabarti. REV* is simple and fast search algorithm that works on explicit graph in bottom – up fashion and can deal with cycles. The pseudo code of the original REV* search algorithm is given as in Algorithm 1.

The user request is given in form of desired outputs, available inputs and the QoS preference, which formally is defined by the triple $R = \{S_{in}(R), S_{out}(R), q_r\}$. The problem is concerned with the optimization of a single QoS attribute. The system can find a solution based on multiple QoS parameters, but then a utility functions need to be applied, as explained in later paragraphs.

In service composition problem represented through AND/OR graphs and using REV* search algorithm, first we connect the auxiliary node *s* to the desired outputs, we put available inputs to list (queue) *O*. To all nodes in the graph, except of the inputs, we set the status to *not found* and the aggregate value to ∞. Then in bottom – up fashion, the status and the corresponding costs are propagated upwards to all nodes for which all immediate successors are declared found. OR nodes not satisfying this condition get their aggregate cost updated, but their status are not changed to *found*. Instead, they are added to the priority queue *O* for subsequent evaluation. When no other propagation is possible, the node with smallest aggregation value is extracted from *O* and declared found and this process continues until the start node *s* is declared *found* or when *O* gets empty. This conservative bottom – up cost revision strategy ensures that it will never loop around a cycle due to cost dominance rule, like in Dijkstra algorithm. In addition, the bug with the "go to" statement presented in original algorithm has been overcome with the use of another list *R*, allowing a cost revision in recursive like scheme.

Algorithm 1: The adoption and extension of REV* Searching Algorithm

```
/* create the dummy node s in G and provide the pointers from
the desired outputs to this node */
/* COST INITIALIZATION */
Create a priority queue O ={ }
Create e list R={ }
for each $n \in G$ do
  if n is terminal node (available input) then
    found[n]=true;
    h(n)=0;
    O.Enqueue(n);
  end
  else
  begin
    found[n] = false;
    h(n)= ∞;
  end;
/* COST REVISION */
while not(found[s]) do
begin
  if O.isEmpty( ) then exit("no solution exists");
  m=O.Dequeue( ); //the node with smallest h(n)
  found[m] = true
  R.append(m); //append m to R
  while(!R.empty( ))
  begin
    n=R.Pop( ); //remove the first node from R to n
    if not(found[s]) then
     for each $p \in P(m)$ and not(found[p]) do
      begin
       if and(found[ $p'$ ]) for each $p' \in S(p)$ then
       begin
         found[p]=true;
         if p is an AND node then
          //calculate the aggregate value as described in Sec.3 //
          //or Eq.(5).Example for the response time //
           $h(p) = q_{rt}(p) + \max(h(p'))$;
         if p is an OR node then
           $h(p) = \min(h(p'))$;
         if $p \in O$ then remove it from O;
          if $p \notin R$ then
           R.append(p);
       end;
     if p is an OR node then
       begin
         $h(p) = \min(h(p'))$;
         O.Enqueue(p);
       end;
    end;
  end;
```

In current model, the uniformity of properties and especially of units used in QoS description is assumed. For example, the property describing the response time should be generally given in milliseconds or seconds, and all given values should be generated by the same definition. Otherwise, a preprocessing phase of data is required before applying the search algorithm.

In addition, the search algorithms are commonly used to find minimal solution graphs. However for certain QoS properties, such as service availability, larger values means better. In this case, the search algorithm has to be changed to search for larger QoS accumulated values, or it needs to transform those QoS parameters first using the inverse variation using Equation 2, change the aggregation formula appropriately and then again searching for minimal solution graph:

$$\hat{q}_p = \frac{1}{q_p} \tag{2}$$

In many cases the user has preferences over multiple QoS variables. If the solution needs to be found on the integration of several QoS attributes, we a two step solution is proposed. First, the unification and normalization of QoS parameters should be conducted. Different QoS parameters have different range of values, so they have to be scaled on the same range, as an important and useful step for their classification. Several techniques exists [14], such as z – score normalization, or decimal scaling, yet we have selected min – max normalization given through Equation 3 as it performs a linear transformation and preserves the relationships among the original data values.

$$\hat{q}_i = \frac{q_i - q_{i_{\min}}}{q_{i_{\max}} - q_{i_{\min}}}\left(new\_\max q_i - new\_\min q_i\right) \quad (3)$$

Once QoS parameters normalized, we apply the following objective function to transform them into one single total cost:

$$\cos t = \sum_{i=1}^{N} w_i * (q_i)^j \quad (4)$$

where N is the number of different QoS parameters (q_i) taken into consideration, w_i is the weight given to QoS parameter i denoting how important is a specific QoS attribute to a user. In addition, the following conditions must hold: $\sum_{i=1}^{N} w_i = 1$ and $j \in [-1,1]$ where j=1 if for the given QoS parameter lower values means better, and j=-1 if for the same parameter greater values means better.
After these preprocessing steps, the multiple QoS are transformed into one single value, and a search algorithm can be applied directly to find the minimal solution graph. The aggregation formula in this case is calculated as:

$$h(n) = \begin{cases} \cos t(n) + \sum h(c_i)\ AND\ node \\ \min(h(c_i))\ OR\ node \end{cases} \quad (5)$$

5. System Implementation and Evaluation

A prototype system that is based on the proposed approach is implemented in C# and the same is evaluated on a test set. The approach is evaluated according to two main criteria's: confusion – based metrics which mainly includes the verification of correctness and completeness, and the time – based performance. In absence of common and widely accepted evaluation methodology, such evaluation is considered to be more comprehensive and in line with the recent research efforts and works [15][16], as the most important metrics related to semantic – based service discovery techniques.
The algorithm is tested against the requirements of the annual WS – Challenge[17]. WS – Challenge provides a significant collection of semantically described services and test tools. Each service in test sets is annotated with the response time and throughput as QoS data. Moreover, it provides evaluation scenarios, which can be used for matching the reference service compositions with the one provided by the composition system, for the given request.
WS Challenge uses software to generate a test set which consists of five basic files: (1) Services.wsdl contains a repository of available services, semantically described by I/O signatures; (2) Servicelevelagreements.wsla gives the response time and the throughput for each service described in the first file; (3) Taxonomy.owl contains the ontology describing the relationships of concepts used as I/O in semantic service descriptions; (4) Challenge.wsdl containing the problem of the requested service and (5) Solution.bpel contains the solution for the given problem.
We used the generator to create 10 test sets. In the first five test sets, the number of services remained constant, and we changed the number of concepts in ontology. In the last five test sets, the number of concepts remained constant, having 10000 concepts as considering this number enough rich to describe variety of different services. The number of services was constantly increased, reaching the number of 20000 services which closely resembles the actual number of real services provided on Internet.
Such settings open the opportunities to see what is the effect of the ontology size and the number of services on the time performance needed to discover the composite services.
Table 3 and Table 4 show the details of the test set, and the last column shows the time performance of the REV* algorithm needed to discover the composite service in this case. The experiment was conducted on Lenovo ThinkCenter A70Z machine, with an Intel Pentium Core 2 Duo E7500 processor, and with 2GB memory running on Windows 7.

Table 3: A test set where the number of concepts increases while the services remain constant

Number of concepts	*Number of services*	*Time (ms)*
2000	1000	48
4000	1000	57
6000	1000	62
8000	1000	66
10000	1000	68

The results of Table 2 shows that on a test set with fix number of services, the time performance of REV* algorithm gracefully changes with the increasing number of concepts in ontology.

Table 4: A test set where the number of services increases while the number of concepts remains constant

Number of concepts	*Number of services*	*Time (ms)*

10000	4000	141
10000	8000	177
10000	12000	198
10000	16000	211
10000	20000	223

The results in Table 3 shows that when the number of concepts is constant, the time performance of REV* changes more linearly with the increase number of services. This is important, as in future when the number and diversity of concepts in ontology will be rich and sufficient to describe variety of things, the size of the ontology will remain constant or it will be changed slightly, but the number of services will increase because of new businesses provided through services on Internet.
Generally, in all test sets, REV* performs efficiently, finding the composition services in no more than 223 ms and discover them correctly. This makes the model an efficient approach to automatic construction of composite services with optimal end – to –end QoS.

5. Conclusions and Future Work

In this paper, we address comprehensively a QoS aware semantic composition model that uses AND/OR graphs as intermediate planning domain. The model is expressive and elegant dynamic environment that enables to find QoS end – to end optimization execution plans from different perspectives and using different searching techniques.
A specific search algorithm was extended and used to find composition plans for a given composition request. Moreover, the algorithm is implemented and its performance is evaluated on a significant size of concepts and services that scale on real number of available services today. The results show that AND/OR graphs are efficient environment for discovery of QoS aware composite services and it can address some of the major problems found on competitive approaches.
The investigation of new search algorithms and efficient data structures for implementation will be conducted. Evaluations on more significant data sets will be performed, to further evaluate the proposed model.

References

[1] E. Al – Masri and Q.H. Mahmoud. “Investigating web services on the world wide web”, in Proceedings of the 17th international conference on World Wide Web, 2008.
[2] Seekda. Web Services Search Engine. http://webservices.seekda.com/. Last accessed on December 2011.
[3] L. Srinavisan and J. Treadwell. “An Overview of Service – Oriented Architecture, Web Services and Grid Computing”, HP Software Global Business Unit, 2005. citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.8488
[4] W. Voorsluys, J. Broberg, and R. Buyya. CLOUD COMPUTING: Principles and Paradigms. John Wiley and Sons, Inc, Hoboken, New Jersey, USA, 2011.
[5] B. George and P. Dimitris. “Automated Web Service Composition: State of the Art and Research Challenges”, Technical Report ICS-FORTH/TR-409, Foundation for Research & Technology – Hellas, Greece, 2010.
[6] Y. Tao, Z. Yue and L. Kwei – Jay. “Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints”, in ACM Transactions.Web, 2007, Vol.1.
[7] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam and H. Chang. “QoS – aware Middleware for Web Services Composition”, IEEE Transactions on Software Engineering, Vol.30, No.5, 2004, pp.311 – 327.
[8] Q. A. Lang. “AND/OR Graph and Search Algorithm for Discovering Composite Web Services”, International Journal of Web Services Research. Vol.2, No. 4, 2005, pp.46 – 64.
[9] M.C. Jaeger, G. Rojec – Goldman and G. Muhl. in The International EEE Conference on e –Technology, e – Commerce and e – Service, 2005, pp.181 – 185.
[10] A.Mahanti, S. Ghose and S. Sadhukhan. “A Framework for Searching AND/OR Graphs with Cycles”, in CoRR journal, vol. cs.AI/0305001, 2003.
[11] A.H. Eric and Z. Shlomo. “Heuristic Search in Cyclic AND/OR Graphs”, in Proceedings AAAI-98, 1998, pp.412 – 418.
[12] A. Mahanti and A. Bagchi. “AND/OR Graph Heuristic Search Methods”, Journal of the ACM, Vol. 32, No.1, 1985.
[13] P.P. Chakrabarti. “Algorithms for Searching Explicit AND/OR Graphs and their Applications to Problem Reduction Search”, Artificial Intelligence, Vol.65, No.2, 1994, pp. 329 – 345.
[14] H. Jiawey and K. Micheline. Data Mining: Concepts and Techniques. Morgan Kauffman, 2nd edition, 2005.
[15] E. Silva, L. Pires and M. Sinderen. “A Framework for the Evaluation of Semantic – based Service Composition Approaches”, in: Seventh IEEE European Conference on Web Services, pp.66 – 74, 2009.
[16] U. Kuster, H. Lausen, and B. Konig – Ries. “Evaluation of Semantic Service Discovery - A Survey and Directions for Future Research”, in: Post-Proceedings of the 2nd Workshop on Emerging Web Services Technology (WEWST07) in conjunction with the 5th IEEE European Conference on Web Services (ECOWS07), Halle (Saale), Germany, November 2007.
[17] C. J. Petrie, H. Lausen, and M. Zaremba, “Sws challenge - first year overview,” in International Conference on Enterprise Information Systems, 2007, pp. 407–412.

Xhemal Zenuni is assistant at SEEU, and PhD student at French Language Faculty of Electrical Engineering, Technical University of Sofia, with primary research interest on the triangle SOA, Grid and intelligent agents.

Ivan Momtchev is dean of French Language Faculty of Electrical Engineering, Technical University of Sofia, from 2005.