Alternate *ACM* SIG Proceedings Paper in LaTeX Format*

[Extended Abstract] †

Ben Trovato‡
Institute for Clarity in Documentation
1932 Wallamaloo Lane
Wallamaloo, New Zealand
trovato@corporation.com

G.K.M. Tobin§
Institute for Clarity in Documentation
P.O. Box 1212
Dublin, Ohio 43017-6221
webmaster@marysville-ohio.com

Lars Thørväld¶
The Thørväld Group
1 Thørväld Circle
Hekla, Iceland
larst@affiliation.org

Lawrence P. Leipuner
Brookhaven Laboratories
Brookhaven National Lab
P.O. Box 5000
lleipuner@researchlabs.org

Sean Fogarty
NASA Ames Research Center
Moffett Field
California 94035
fogartys@amesres.org

Charles Palmer
Palmer Research Laboratories
8600 Datapoint Drive
San Antonio, Texas 78229
cpalmer@prl.com

ABSTRACT

...

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

ACM proceedings, LaTeX, text tagging

1. INTRODUCTION

...

*(Produces the permission block, and copyright information). For use with SIG-ALTERNATE.CLS. Supported by ACM.

†A full version of this paper is available as *Author's Guide to Preparing ACM SIG Proceedings Using LaTeX2ε and BibTeX* at `www.acm.org/eaddress.htm`

‡Dr. Trovato insisted his name be first.

§The secretary disavows any knowledge of this author's actions.

¶This author is the one who did all the really hard work.

WOODSTOCK '97 El Paso, Texas USA

2. SLIDES DESCRIPTION

Problem addressed in our algorithm is: given a set of abstract services, a set of concrete services, a user query and a set of user quality preferences, derive a set of service compositions that answer the query and fulfill the quality preferences.

Our related work relies on two different domains: query rewriting algorithms using views (database domain) [**?**] and query algorithm in the service composition domain [**?**].

We assume that:

- The query expresses an abstract composition that describes the requirements of a user. It is expressed with respect to a catalogue of abstract services
- A concrete service is defined in terms of an abstract composition. It can be associated to a single abstract service or to a composition of abstract services
- Concrete services are tagged quality measures. Not all services are tagged with the same measures. Every measure is defined in a catalogue
- Each measure is written in the form: *constant* (measure name) *operation* ($<$, $>$, $=$, $\leq$, $\geq$) *value* (the static value associated to the measure)
- There are two types of measures: single and composite measures. The single measures is the simplest type. It is a static measure which is has a name associated with an operation and a value. The composite measure is dynamically computed measure. It is defined as aggregations of single measures. All measures we are using are in a catalogue. Example: Single measure: availability, price per call, price per request, response time, location, provenance, etc. Composite measure: total price or cost which are computed by adding price per call and price per request values of all services included in the service composition. Total response time is computed by adding the response time of all services included in the service composition.

```
<measures>
<singlemeasure name="price per request" type="double"/>
<singlemeasure name="price per call" type="double"/>
<compositemeasure name="total price" type="double" action="sum">
<singlemeasure name="price per request" type="double"/>
<singlemeasure name="price per call" type="double"/>
</compositemeasure>
</measures>
```

The parameter "action" in the composite measure specifies how the single measures should be aggregated (adding, average, product, etc).

The table 2 contains the abstract services that will be used in our example.

Abstract Service	***Description***
DiseaseInfectedPatients(d?,p!)	Given a disease *d*, a list of patients *p* infected by it is retrieved.
DiseaseInfectedPatients(d?,p!,op!)	Given a disease *d*, a list of patients *p* infected by it is retrieved, and *op* is an optional boolean output indicating if the operation proceeded well or not.
PatientDNA(p?,dna!)	Given a patient *p*, his DNA information *dna* is retrieved.
PatientPersonalInformation(p?,info!)	Given a patient *p*, his personal information *info* is retrieved.

Table 1: Abstract services description

Let us suppose the following query. *The user wants to retrieve patientâĂŹs personal and DNA information of patients who were infected by a disease ÂńKÂż using services that have availability higher than 98%, price per call less than 0.2 dollars, and total cost less then 1 dollar.*

The query which express the example in terms of abstract services is specified below. The decorations ? and ! are used to specify input and output parameters, respectively.

```
Q(d?, patientinfo!, dna!) := DiseaseInfectedP atient(d?, p!), PatientPersonalInformation(p?, info!),
PatientDNA(p?, dna!){p = "K"}[availability > 98, price per call < 0.2, total cost < 1]
```

The query expressed in terms of abstract services including its constraints and preferences. In the current implementation braces ({}) and brackets ([]) are used to distinguish constraints from preferences.

The first step of the algorithm looks for concrete services that can be matched with the query. We have three matching problems associated to this step: abstract service matching, measure matching and concrete service matching.

- *abstract service matching*: a abstract service A can be matched with a abstract service B only if: (i) they have the same name. In this case we are assuming they perform the same function; and (ii) the number and type of variables should be compatible. This means that the number of input and output variables of A must be equal or higher than the number of input and output variables of B. Consider the example below.

  ```
  1) A1(x?,c!)

  2) A1(a?,b!,c!)
  ```

  In this example, 2 can be matched to 1, because the number of input and output variables of 2 is higher than the number of input and output variables of 1. However 1 can not be matched to 2, because the number of input and output variables of 1 is less than the number of input and output variables of 2.

- *measures matching*: all single measures in the query must exist in the concrete service, and all of them can not violate the measures in the query.

- concrete service matching: one concrete service can be matched with the query if all its abstract services can be matched with the abstract service in the query (satisfying the *abstract service matching* problem) and all the single measures in the query can be matched with the concrete service measures (satisfying the *measures matching* problem). Consider the example below.

  ```
  Q(x?,y!) = A1(x?,c!), A2(c?,y!)[availability > 98, price per call < 0.2, total price < 1]

  S1(a?,b!) = A1(a?,b!) [availability > 98] (It can not be matched. Price per call is missing.)

  S2(a?,b!) = A1(a?,b!) [availability > 98, price per call = 0.2] (It can not be matched. Price per
  the query preference.)

  S3(a?,b!) = A1(a?,c!), A3(c?,b!) [availability > 98, price per call = 0.1] (It can not be matched
  service in the query.)

  S4(a?,b!) = A1(a?,b!) [availability > 98, price per call = 0.1]

  S5(a?,b!) = A1(a?,c!), A2(c?,b!) [availability > 99, price per call = 0.1]

  S5(a?,b!) = A1(a?,c!), A2(c?,b!) [availability > 99, price per call = 0.1, location = ãĂIJcloseãĂ
  it can be matched)
  ```

The principle of the solution for the first step of the algorithm is: given (i) a query and a set of user quality preferences; and (ii) a set of concrete services tagged with quality measures, looks for concrete services which respects the matching rules. The concrete service that matches the rules is a candidate concrete service. The result is a list of candidate concrete services which may be used in the rewriting process.

Related work on web service selection and composition considering QoS aspects [].

To illustrate the selection of candidate services, consider the query in the example presented before and the following concrete services.

```
S1(a?,b!) := DiseaseInfectedPatients(a?,b!)[availability > 99, price per call = 0.1]
S2(a?,b!) := DiseaseInfectedPatients(a?,b!)[availability > 98, price per call = 0.2]
S3(a?,b!,c!) := DiseaseInfectedPatients(a?,b!)[availability > 98, price per call = 0.1]
S4(a?,b!) := PatientDNA(a?,b!)[availability > 98, price per call = 0.1]
S5(a?,b!) := PatientDNA(a?,b!)[availability > 96, price per call = 0.1]
S6(a?,b!) := DiseaseInfectedPatients(a?,k!), PatientDNA(k?,b!)[availability > 93]
S7(a?,b!) := PatientPersonalInfo(a?,b!)[availability > 99, price per call = 0.1]
S8(a?,b!) := PatientPersonalInfo(a?,b!)[availability > 98, price per call = 0.1]
S9(a?,c!,b!) := PatientPersonalInfo(a?,b!),PatientDNA(a?,c!)[availability > 98, price per call = 0.1]
```

In this example, the services S2, S5 and S6 can not be selected as candidate concrete services since they violate user preferences

The second step of the algorithm tries to create *concrete services description* (CSD) to be used in the rewriting process. A CSD maps abstract services and variables of a concrete service to abstract services and variables of the query. A CSD is created according to the following variable mapping rules:

- *Rule 1*: head variables in the concrete service can be mapped to head or local variables in the query if they are from the same type

- *Rule 2*: local variables in the concrete service can be mapped to head variables in the query if they are from the same type

- *Rule 3*: local variable in the concrete service can be mapped to a local variable in the query if: (i) they are

from the same type; and (ii) the concrete service cover all abstract service in the query that depends on this variable. Depends here means that this local variable is used as input in another abstract service.

The principle of the solution for the second step is: given a list of candidate concrete services, looks for a candidate concrete services that satisfies the the variable mapping rules. For the ones who satisfies the rules, a CSD is created. As result a list of CSDs is produced.

To illustrate the selection and creation of concrete service description, consider the concrete services selected in the previous step below.

```
S1(a?,b!) := DiseaseInfectedPatients(a?,b!)[availability > 99, price per call = 0.1]
S3(a?,b!,c!) := DiseaseInfectedPatients(a?,b!,c!)[availability > 98, price per call = 0.1]
S4(a?,b!) := PatientDNA(a?,b!)[availability > 98, price per call = 0.1]
S7(a?,b!) := PatientPersonalInfo(a?,b!)[availability > 99, price per call = 0.1]
S8(a?,b!) := PatientPersonalInfo(a?,b!)[availability > 98, price per call = 0.1]
S9(a?,c!,b!) := PatientPersonalInfo(a?,b!),PatientDNA(a?,c!)[availability > 98, price per call = 0.1]
```

In this example six CSDs are created once all concrete services satisfy the rules to create CSDs. The third step of the algorithm generates all combinations of CSDs. As result we have a list of list of CSDs.

The fifth and final step of the algorithm verifies/identifies which combination of CSDs is a valid rewriting of the user query. A combination of CSDs is a valid rewriting if:

- The number of abstract services in the query is equal to the result of adding the number of abstract services of each CSD
- There is no duplicity/redundancy of abstract services in the list of CSD
- All head variables in the query must be mapped to a variable in one of the concrete services in the list of CSDs
- If the query has a composite measure, this measure is updated for each rewriting produced, and this measure can not be violated

As result of this step we have a list of rewriting of the query. To illustrate let us consider the example used before. In our query we have a preference which is associated to the rewritings (composite measure) and not to a single service. Considering this preference, we have to update its value while producing the rewritings. The value of total cost is this example is updated by aggregating the value of price per call of each service. The rewritings produced that can satisfy the user preference while aggregating these values are below. Note that more than three rewritings can be produced that composite measure did not exists. The rewrintgs are listed in the lexicographical order considering the concrete services .

```
Q(disease?, info!, dna!) := S1(disease?,p!) S7(p?,info!) S4(p?,dna!)
Q(disease?, info!, dna!) := S3(disease?,p!, _) S7(p?,info!) S4(p?,dna!)
Q(disease?, info!, dna!) := S1(disease?,p!) S8(p?,info!) S4(p?,dna!)
```

3. CONCLUSIONS

...

4. ACKNOWLEDGMENTS

optional..