Context-Driven and Service Oriented Semantic Mediation in DaaS Composition

Idir Amine Amarouche[1], Karim Benouaret[2],
Djamal Benslimane[2], Zaia Alimazighi[1] and Michael Mrissa[2]

[1] Université des Sciences et de la Technologie Houari Boumediene
BP 32 El Alia 16111 Bab Ezzouar, Algeirs, Algeria
[2] Université Lyon 1, LIRIS UMR5205
43, bd du 11 novembre 1918, Villeurbanne, F-69622, France
i.a.amarouche@gmail.com, karim.benouaret@liris.cnrs.fr
djamal.benslimane@liris.cnrs.fr, Alimazighi@wissal.dz,
michael.mrissa@liris.cnrs.fr

Abstract. In this paper we present a context driven approach for automatically inserting appropriate mediation services in Data-as-a-Service (DaaS) compositions to carry out data conversion between interconnected services. We propose a context model expressed over Conflicting Aspect Ontology to describe more accurately the semantics of DaaSs. Based on the context model, we specify the mediation services that perform the transformation of DaaS parameter values between contexts. Then, we develop an efficient algorithm to detect and resolve the semantic conflicts between services in DaaS composition. An implementation demonstrates the applicability of our proposal.

Keywords: DaaS composition, semantic mediation, context, semantic conflict.

1 Introduction

A trend emerges towards the use of the Service Oriented Computing paradigm to extract and process data in distributed environments. Data-as-a-Service (DaaS) is a recent specialization of Web services whose main objective is the retrieval of data from existing data sources according to given input parameters. In most cases, querying data sources requires the composition of multiple DaaSs. The automation of DaaS composition requires specifying the semantic relationships between inputs and outputs parameters in a declarative way. This requirement can be achieved by describing DaaSs as views over a Domain Ontology (*DO*) following the mediator-based approach [13]. Thereby, the DaaS composition problem is reduced to a query rewriting problem, which is a well-known problem in the data integration field. In this context, several works [1, 14, 12] consider DaaS as Parametrized RDF[3] Views (*PRVs*) with binding patterns over a *DO*,

[3] RDF: Resource Description Framework

to describe how the input and output DaaS parameters are semantically related. Defined views are then used to annotate DaaSs description files (e.g., WSDL files) and are exploited to automatically compose DaaSs. [1, 14, 12] argue that ontology languages for Semantic Web services (e.g., OWL-S, WSMO) and extension mechanisms (e.g., SAWSDL) do not provide a way to semantically link Web service inputs and outputs which hampers their applicability to DaaS composition. Obviously, [1, 14, 12] focus mainly on the composability of DaaS at the *DO* level. However, the construction of a *DO* unifying all existing representations of real-world entities in the domain is a strong limitation to interoperability between DaaS. This limitation essentially raises semantic conflicts between pieces of data exchanged during the composition process. To this end, the applicability of existing DaaS composition approaches is compromised. Consequently, it is crucial to consider the detection and the resolution of semantic conflicts during the composition process since the contexts of the consumers and the suppliers of DaaSs are different. In this regard, the definition of context as the knowledge allowing to detect semantic conflicts between DaaS parameters values is required. [4, 10] defined context as a set of meta-attributes with values to solve different kinds of semantic heterogeneities. In this paper, we argue that: i) the contextualization of *PRV* is required to handle semantic conflicts in DaaS composition; ii) the resolution of semantic conflicts is assured by mediation services which are used as per the Service Oriented Architecture (SOA) principles.

Motivating example: Let us consider an e-health system where the information needs of health actors are satisfied with DaaS Composition System (DCS) as proposed by [1, 14], which exports a set of DaaSs to query patient data. We assume that a physician submits the following query Q_1: "What are the states indicated by the recent Blood Pressure Readings (BPR) measures for a given patient". The DCS will automatically generate a DaaS composition, as a response to the physician's query, including respectively s_1, s_2 and s_3 as depicted in figure 1.(a). We assume for simplicity, that the DCS generates only one composition. The DCS invokes automatically: 1) "s_1" that provides the recent Vital Sign Exam (Exam-id) performed on his patient; 2)"s_2" to retrieve the BPR measure[4] of the patient; 3)"s_3" that provides the "BPR" state[5] according to the MAP[6] value. However, the DCS exports DaaSs expressed over *DO* which does not take context into account. With the term context, we mean knowledge allowing to detect semantic conflicts between component services in the composition. Then, the physician has to manually : i) detect the existing conflicts in the generated DaaS composition as depicted in 1.(a)[7] ; ii) select, invoke

[4] BPR measure is represented by two concatenated values, e.g., "120/80 mm/HG" where "120" is BPR Diastolic (BPR.D) value and "80" BPR Systolic (BPR.S) value and "mm/Hg" is a measurement unit.

[5] The state represents the classification of BPR measure according to classification BPR value table (e.g., New classification: stage 1,2,3,4; Old classification: severe, moderate, mild)

[6] Mean Arterial Pressure is BPR measure, $MAP = \frac{2}{3}(BPR.D) + \frac{1}{3}(BPR.S)$

[7] cos_1: conflict of codification system; cos_2: conflict of BPR value structure, measurement unit; cos_3 : conflict of classification system scale.

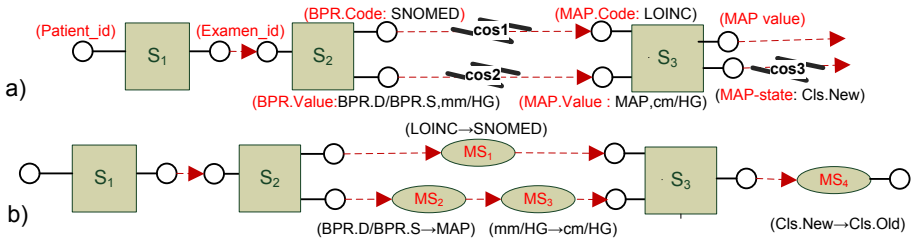


Fig. 1. Physician query scenario: a) DaaS composition generated by the DCS with conflicts ; b) The DaaS composition conflict free.

and compose the appropriate mediation services, assuring the transformation functions, to make the generated composition executable as depicted in figure 1.(b). We assume that a set of atomic mediation services will be provided to the DCS (MS_1,..MS_4). Thereby, the physician has to perform the following tasks: 1) select "MS_1" to map the BPR code returned by s_2(LOINC[8]) to the code acceptable by s_3 (SNOMED[9]); 2) Compose "MS_2" and "MS_3" where : "MS_2" aggregates the two values expressing BPR measure returned by "s_2" to MAP value acceptable by s_3; "MS_3" converts the MAP value expressed with the measurement unit ("mm/Hg") returned by MS_2 to the MAP value expressed with the measurement unit acceptable by s_3 ("cm/Hg"); 3)"MS_4": to map the BPR state returned by "s_3" represented according to the new classification BPR value table to the state acceptable by the physician represented according to the old classification. This is a rather demanding task for non expert users (e.g., physicians). Thus, automating conflict detection and resolution in DaaS composition is challenging.

Contributions: In this paper we propose a context driven approach for automatically inserting appropriate mediation services in DaaS compositions to carry out data conversion between interconnected DaaS. Specifically, we propose 1) a context model expressed over Conflicting Aspect Ontology(CAO) as an extension of "DO". Our model allows to extend the PRV-based DaaS model to more accurately express the semantics of DaaS parameters and to specify the mediation service model ensuring the transformation of DaaS parameters values from one context to another; 2) an approach to detect and resolve semantic conflicts occurring in DaaS composition.

Outline: The rest of this paper is organized as follows. Section 2 presents the overview of our approach. In Section 3, we leverage the background models used throughout the paper. In Section 4, we present our models for context-driven semantic mediation in DaaS composition. In Section 5, we detail our conflict detection and resolution algorithm. Section 6 gives a global view of our implementation and demonstrates the applicability of our proposal. Section 7 reviews related work. Section 8 provides a conclusion and future work.

[8] LOINC : Logical Observation Identifiers Names and Codes
[9] SNOMED: Systematized Nomenclature of Medicine, Clinical Terms

2 Approach overview

Our proposal aims to provide a framework for automatic conflict detection and resolution in the process of DaaS composition. Our approach takes into account the context of component services in DaaS composition and the context of the query. DaaSs are modeled as PRV over a DO and contextualized over a Conflicting Aspect Ontology (CAO) and mediation services are modeled as mapping rule over CAO. Then, at design time, the contextualized PRV and the mapping rule are incorporated into corresponding WSDL description files as annotations. DaaSs and mediation services are stored in two different registries. Figure 2 gives an overview of our approach.

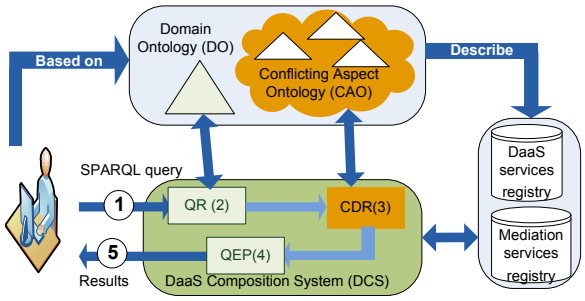


Fig. 2. Approach overview

At query evaluation time, the DaaS composition process starts when the user specifies a query over DO and CAO using SPARQL[10] (Fig. 2 arrow 1). The DCS uses the Query Rewriting (QR) algorithm proposed by [1] and existing PRV to select the DaaS that can be combined, to answer the query (Fig. 2 rectangle 2). Once the DaaS compositions are generated, our Conflict Detection and Resolution Algorithm (CDR) takes into account the context of both the selected $PRVs$ and the query for conflict verification in each generated DaaS composition (Fig. 2 rectangle 3). Then, in case a conflict is detected between: 1) subsequent services in DaaS composition; 2) DaaS composition outputs and the required results by the query, 3) DaaS composition inputs and the constraints specified by the query, our algorithm inserts automatically an appropriate mediation services to resolve semantic conflict. Then, the DCS translates a composite DaaSs conflict free into Query Execution Plan (QEP) describing data and control flow (Fig. 2 rectangle 4). The plan will be executed and returns data to the user (Fig. 2 arrow 5). In this paper, we only focus on conflict detection and resolution.

[10] We adopt SPARQL: http://www.w3.org/TR/rdf-sparql-query/, the *de facto* query language for the Semantic Web, for writing queries.

3 Background

We present in this section the background models used through the paper, namely, the Domain Ontology (DO), the Parametrized Rdf View (PRV) and the Query (Q).

3.1 Domain Ontology (DO)

The DO is 6-tuple < C, D, OP, DP, SC, SP> where C is a set of classes; D is a set of data types; OP is a set of object properties; each object property has its own domain and range in C; DP is a set of data type properties; each data type property has a domain in C and range in D; SC is a relation over $C \times C$, representing the sub-class relationship between classes; SP is a relation over $(OP \times OP) \cup (DP \times DP)$, representing the sub-property relationship between homogeneous properties. In the present work, DO is specified with RDFS. Figure 3 illustrates an excerpt of our Domain Ontology of Vital Sign Exam where Class nodes are represented with ovals and data type nodes are represented with rectangles. DO contains basic shared concepts and their properties with extensions specifying the conflicting aspects. For instance, the $BPReading$ concept has a code expressed in a given health ontology which will be specified using a concept "$CAO : SystemCode$" over the Conflicting Aspect Ontology (CAO). The CAO will be detailed in section 4.

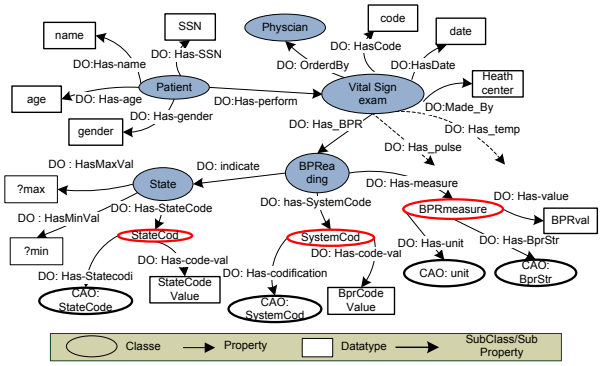
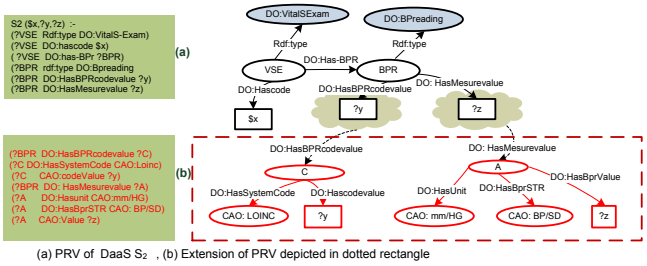


Fig. 3. Domain ontology

3.2 Parametrized RDF View (PRV)

A DaaS S_j is described as Parametrized RDF View (PRV) in a Datalog-like notation over a DO [1]. A DaaS S_j has the form $S_j(\$X_j, ?Y_j) : - < G_j(X_j, Y_j, Z_j), Co_j >$ where: X_j and Y_j are the sets of input and output variables of S_j respectively;



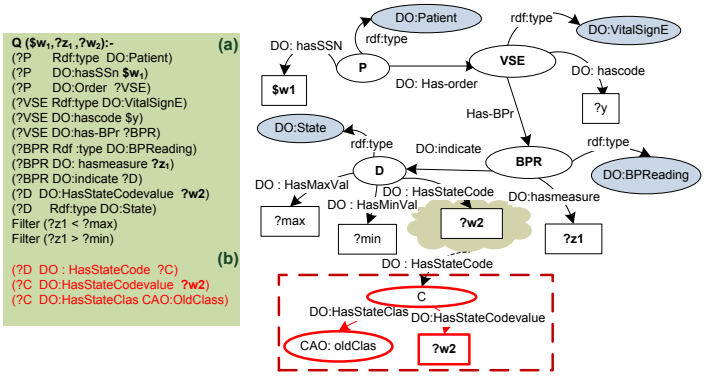
(a) PRV of DaaS S_2 , (b) Extension of PRV depicted in dotted rectangle

Fig. 4. RDF graph and RDF snippet of DaaS model

G_j represents the functionality of the DaaS which is described as a semantic relationship between input and output variables; Z_j is the set of existential variables linking X_j to Y_j; $Co_j = \{Co_{j_1}, ..., Co_{j_n}\}$ is a set of constraints expressed on X_j, Y_j or Z_j variables. Figure 4.(a) gives the PRV of DaaS S_2 depicted in motivating example.

3.3 Query (Q)

We consider conjunctive queries over DO. The Query "Q" has the form: Q(X):- $< G(X,Y), Co_q >$ where $Q(X)$ is the head of Q, it represents the result of query; $G(X,Y)$ is the body of Q, it contains a set of RDF triples where each triple is of the form (subject.property.object); X and Y are called the distinguished and existential variables respectively, X and Y are subjects and objects in the RDF triples; $Co_q = \{Co_{q_1}, ..., Co_{q_n}\}$ is a set of constraints expressed on X and Y variables [1]. Figure 5.(a) depicts the RDF graph of the query Q_1 described in our scenario.



(a) SPARQL query expressed over DO, (b) Query with context extention expressed over CAO depicted in dotted rectangle.

Fig. 5. SPARQL Query graph and snippet

4 Our Models

We present in this section our models used for context-driven semantic mediation in DaaS composition. In the present work, the DaaS Composition $cs = \{s_1..s_n\}$ represents the set of ordered services into DaaS composition ; $First(cs)$ (e.g., s_1) and $Last(cs)$ (e.g.,s_n) denote the first and the last DaaS in "cs". "CSs" denotes the set of compositions generated by the Query Rewriting (QR) algorithm of "DCS" and requiring testing and conflict resolution. We note that each "cs" requires a set of input parameters to be executed and returns a set of output parameters after execution.

4.1 Conflicting Aspect Ontology

Conflicting Aspect Ontology (CAO) is a family of a lightweight ontology, specified in RDFS. A CAO extends DO entities with a taxonomic structure expressing different semantic conflicts between DaaS parameters[11]. The CAO varies in scope, from very broad conflicts to very specific ones. A CAO is a 3 tuple $< AC_g, AC_i, \tau >$, where: *"AC_g"* is a set of classes which represents the different conflicting aspects of a DO entities. Each "ac_g" class in "AC_g" has one super-class and a set of sub-classes. Each "ac_g" class has a name representing a conflicting aspect, such as, "`CAO:Measurement-Unit`" as depicted in Figure 6; *"AC_i"* is a distinct set of instantiable classes having one super-class in "AC_g". By definition, "ac_i" is not allowed to have sub-classes. For instance "mm/HG" and "cm/HG" are two insteancable classes from the "`CAO:BPR-Unit`" class; *"τ"* refers to the sibling relationships on "AC_i" and "AC_g". The relationships among elements of "AC_g" is disjoint. However, elements of "AC_i" of a given "ac_g" are related by the *Peer relationship* which indicates semantic conflicts for similar data semantics.

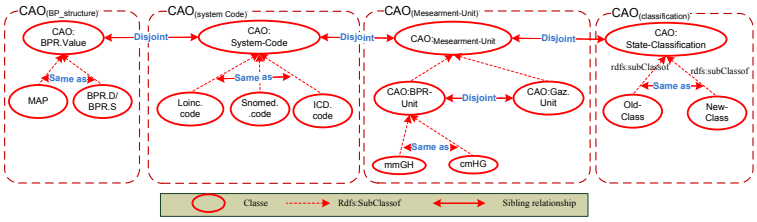


Fig. 6. Conflicting Aspect Ontology

4.2 Context model

A context "Ct" is a set of uniquely identified dimension-value pairsthat are represented under the form $\{(D_i, V_i)|i \in [1, m]\}$ where $D_i \in AC_g$ and $V_i \in AC_i$.

[11] For a classification of the various incompatibility problems in Web service composition see [7]

For instance, the context $Ct_{MU} = \{(CAO : Unit, mm/HG)\}$ indicates that the measurement unit is "mm/HG". Given two contexts "Ct" and "Ct'" where $Ct = \{(D_i, V_i)\}$ and $Ct' = \{(D_i, V_i')\}$. We say that the context "Ct" is in relationship with the context "Ct'" if and only if for each $i \in \{1..n\}$ we have $v_i R_{Di} v_i'$ where $R_{Di} \in \tau_{AC_i}$.

As the PRV based DaaS model expressed over DO does not provide explicit semantics about its input and output parameters, we extend its description with the context describing more precisely how the semantics of the DO entities are described according to the CAO. Then, each DaaS model is extended by annotations on variables appearing in input and output DaaS parameters and expressed in terms of CAO. In this way, it is possible to have variants of the same DaaS, each holding under a different context. In the same vein, the context is used to express more precisely the distinguished variables and the constraints of the query.

1) Contextualized DaaS model : A C-DaaS is defined as $S_j(\$X_j, ?Y_j) : - < V_{DO} > | < X :: Ct_{X_j}, Y :: Ct_{Y_j} >$ where: V_{DO} is the PRV of S_j ; Ct_{X_j} and Ct_{Y_j} are respectively the contexts of the input and the output DaaS parameters expressed over CAO. Ct_{X_j} and Ct_{Y_j} are described by a set of RDF triples over CAO in form of 2-tuple $< AC_g, AC_i >$. Figure 4.(b) gives the graph view of the contextualized DaaS S_2 of our motivating example.

2) Contextualized query model: has the form $CQ(X) : - < Q(X)|X :: Ct_{qx}, Co_q :: Ct_{qc} >$ where $Q(X)$ is the query expressed over DO, and Ct_{qx} is the context of the distinguished variable X and Ct_{qc} the context of query constraint Co_q expressed respectively over CAO. As depicted in figure 5.(b), the query mentioned in our motivation example has for context $Ct_q = ((X :: (CAO : OldClassification)),(Co_q :: \emptyset))$.

4.3 Semantic conflicts : model and classification

In our present work, a semantic conflict occurs in "O/I" operation. The "O/I" operation indicates each data flow occurring between output and input parameters belonging respectively: to subsequent DaaSs "s_i" and "s_j" in "cs"; "First(cs)" and "CQ_{Co_q}"; "Last(cs)" and "CQ_X". Thus, for a given "O_k/I_k" where "O_k" and "I_k" denotes respectively an output and an input parameters which refer to the same DO entity; if their respective contexts "Ct_{O_k}" and "Ct_{I_k}" refer to different "ac_i" entities and having the same "ac_g"; then we say that a parameter semantic conflict "ac_g" exists in "O_k/I_k" operation. When the conflict is related to one aspect "ac_g" we call it *simple conflict* otherwise we call it *complex conflict*. For instance, in our motivating example the conflicts are respectively denoted as: 1) *System Code Conflict*: BPR code is expressed according to "SNOMED" or "LOINC"; 2) *Measurement Unit Conflict*: BPR value has a measurement unit "mm/HG" or "cm/HG" 3) *Parameter Structure Conflict:* BPR value is represented by one value "MPA" or by two values "BPR.D" and "BPR.S"; 4) *Data Precision Conflict:* state value expressed according to the new or the old classification. In our motivating example, we have one simple conflict (System Code

Conflict) and one complex conflict (Parameter Structure Conflict, Measurement Unit Conflict) occurring in two O/I operations between "s_2" and "s_3" in "cs".

4.4 Mediation service

Mediation Services (MSs) consist of a rule assuring the transformation or the mapping in the case where some "O/I" operation causes a conflict. We deem appropriate to follow the model proposed in [2, 8] to represent the rule as a SPARQL "construct" statement. In this context, the rule having the form $\{ant \Longrightarrow cons\}$ in logic programming approach, where the symbol "$\Longrightarrow$" means the implication relation, becomes $\{CONSTRUCT\ cons\ WHERE\ ant\}$. Both the antecedent (ant) and the consequence (cons) are conjunctions of predicates represented by a set of RDF triples. An RDF triple may be represented, in standards first order logic notation, as a predicate $Predicate(Subject, Objet)$. The "ant" can either be existential predicate or built-in predicate representing arithmetic or aggregate function.

Mediation service model is a *SPARQL parametrized query* having the form $MS_j(\$I_J, ?O_J) : G_I \rightarrow G_O$, where : "$\$I_J$" and "$?O_J$" are respectively the sets of input and output variables of MS_J; G_I and G_O are the set of RDF triples representing Input and output contextualized parameters. For instance, the mediation service MS_1 assuring the mapping of BPR code value from "LOINC" code to "SNOMED" code is presented in figure 7.(a). For each conflicting aspect,

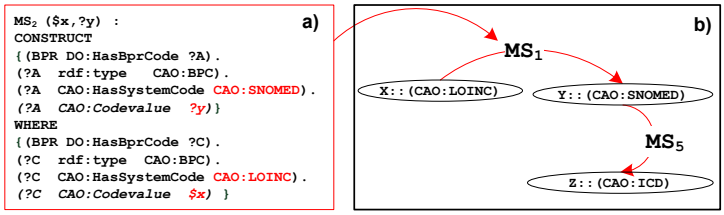


Fig. 7. a) SPARQL query representing MS_1. b) Graph of MSs for Conflicting Aspect "Codification system"

we define the set of atomic mapping rules allowing to describe the transformation between two contextualized parameters as following : 1) *System Code Conflict* and *Data Precision Conflict:* are associated to mapping rule one-to-one; 2) *Measurement Unit Conflict* is associated to mapping rule one-to-one with a conversion function in the antecedent of the mapping rule; 3) *Parameter Structure Conflict:* is associated to mapping rule many-to-one or one-to-many.

The Repository model of MSs ($\mathcal{RMS}$) represents the set of mediation services available in mediation service repository for each conflicting aspect "AC_g". This set is modeled as graph $G_{AC_g} = (V, E)$ where "V" is the set of vertices, representing the set of "ac_i", and "E" is the set of edges, representing the

mediation services. For instance, the graph depicted in figure 7.(b) represents the RMS for the conflicting aspect "Codification System" assuring the transformation from DaaS parameters expressed in $LOINC$ to $SNOMED$ or from DaaS parameters expressed in $SNOMED$ to ICD. The benefit of $\mathcal{RMS}$ lies in its capability allowing the derivation of composition or alternative mediation service from atomic mediation service.

5 Conflict detection and resolution processing

In the following, we present the details of our Conflict Detection and Resolution Algorithm (CDR) which proceeds in two stages: Detection and Resolution. The first stage described in Algorithm 1 depicts how to detect conflicts in a given composition "cs". The second stage, described in Algorithm 2 depicts how to insert automatically the mediation services in "cs" ensuring conflict resolution.

5.1 Conflict Detection

The conflict detection algorithm is presented in Algorithm 1. The inputs of the Algorithm 1 are "cs" and "O/I" operations while its output is the Conflict Object Set $\mathcal{COS}$ where the information related to detected conflicts are stored. The $\mathcal{COS}$ is 5 tuple $< id-conflict, ac_g, Ct_O.ac_i, Ct_I.ac_{i'}, Position, MService >$ where: "$id-conflict$" is the identifier of the detected conflict; "ac_g" represents the conflicting aspect related to the two parameters O_i and $I_{i'}$; "ac_i" is the instantiable class belonging to Ct_O, "$ac_{i'}$" is the instantiable class belonging to Ct_I; "$position$" indicates the position of the "O/I" operation in the "cs" where the conflict is detected; "$MService$" is the mediation service returned by Algorithm 2 to resolve the conflict "$id-conflict$".

Algorithm 1 proceeds as follow, for each "O_k/I_k"[12] operation in "cs", if there is a conflict in "O_k/I_k" and the conflict of "O_k/I_k" is not in the conflict set "$\mathcal{COS}$" then add the conflict to the conflict set "$\mathcal{COS}$", as well as its position in "cs" (line 4); otherwise, i.e., the conflict is in the conflict set "$\mathcal{COS}$", add only the position of conflict in "$cos.position$" (line 6). However, if there is not a conflict in "O_k/I_k" then "O_k/I_k" does not cause conflict in "cs'" (line 9). The detected conflicts in "cs" will be resolved by using the process described in algorithm 2.

For instance, the "$\mathcal{COS}$" set generated in basis of our motivating example is represented in table 1 where cos_1 and cos_3 are simple conflicts and cos_2 is a complex conflict.

5.2 Conflict Resolution

The details of Algorithm 2 are as follows. The inputs of the algorithm 2 are: "cs" with conflicts, the Conflict Object Set $\mathcal{COS}$, the Mediation Service set $\mathcal{MS}$ and the graph of the Repository of mediation services $\mathcal{RMS}$. The output of the

[12] K represents the K^{ieth} Output/Input operation in given "cs".

Algorithm 1 Conflict Detection Algorithm

Require: cs a DaaS composition; O/I operation set in cs.
Ensure: COS Conflict Object Set,

```
1: for each O_k/I_k in O/I do
2:   if the context of O_k and I_k have the same ac_g and different ac_i then
3:     if (O_k/I_k) ∉ {(cos.Ct_O.ac_i, cos.Ct_I.ac_i')} then
4:       cos_t.add (ac_g, Ct_O.ac_i, Ct_I.ac_i', Position)
5:     else
6:       add new position in cos_t.position;
7:     end if
8:   else
9:     no conflict is detected in O_k/I_k from cs;
10:  end if
11: end for
12: return COS
```

Table 1. Conflict Object Set (COS), (a) The non-colored part of table is filled during the detection stage; (b) The gray part is filled during the stage phase.

Conflcit-id	**position**	ac_g	$Ct_O.ac_i$	$Ct_I.ac_{i'}$	$MService$
cos_1	$\{(s_2, s_3)\}$	$SystemCode$	$LOINC$	$SNOMED$	$MS1$
cos_2	$\{(s_2, s_3)\}$	$Meas-Unit$	mm/HG	cm/HG	$MS2-MS3$
		$Structure$	BPD, BPS	MPA	
cos_3	$\{(s_3, CQx)\}$	$Classificat$	$NewCla$	$OldClasA$	$MS4$

algorithm 2 is a conflict-free composition (cs_{cf}), i.e."cs" augmented with mediation services. The algorithm processes simple conflicts (lines 2-9) and complex conflicts (lines 10-18) separately. For each simple conflict "cos_t" in "$\mathcal{COS}$", the algorithm searches the required mediation service having "$O :: Ct_O.ac_i$" as input and "$I :: Ct_I.ac_{i'}$" as output where "ac_i" and "$ac_{i'}$" belong to the same aspect "ac_g". This action is achieved by searching for the mapping variable [1] of "$O :: Ct_O.ac_i$" and "$I :: Ct_I.ac_{i'}$" respectively to the corresponding variables in "G_I" and "G_O" of each SPARQL query describing mediation services in "$\mathcal{MS}$" (line 3). Then, if a mapping is found, the algorithm selects the first returned mediation service. In case the conflict does not find the required meditation service, the algorithm searches in "$\mathcal{RMS}$" for an alternative sequence of atomic mediation services (line 5). This search computes the Shortest Path from the node "$O :: Ct_O.ac_i$" to the node "$I :: Ct_I.ac_{i'}$" in "$\mathcal{RMS}$". Otherwise, the algorithm returns an information that the composition "cs" is not executable. In case the conflict is complex the algorithm considers its resolution as a sequence of simple conflicts. For that purpose, for each conflicting aspect "cos_t" the algorithm applies the same steps from line 2 to 9 of the Algorithm 2. When all the conflicting aspects find an appropriate mediation service for their resolution the final solution is the composition of the mediation services that have been found (lines 11-14).

Algorithm 2 *Resolution* ($\mathcal{COS}$, cs, $\mathcal{MS}$, $\mathcal{RMS}$)

Require: cs composition with conflict, $\mathcal{COS}$ a set of Conflict Object, $\mathcal{MS}$ mediation service set, $\mathcal{RMS}$ a graph of mediation services,
Ensure: cs_{cf} composition conflict free.

```
1: for each cos_t ∈ COS do
2:   if cos_t is a simple conflict then
3:     if ∃MS_i ∈ MS, map(O :: Ct_O.ac_i) = input(MS_i) ∧ map(I :: Ct_I.ac_i') =
       output(MS_i) then
4:       COS.add (cos_t.MService, MS_i)
5:     else if ∃MS_i ∈ MS, ShortPath(RMS, O :: Ct_O.ac_i, I :: Ct.I.ac_i') then
6:       COS.add (cos_t.MService, MS_i)
7:     else
8:       return null
9:     end if
10:  else
11:    for each ac_g ∈ cos_t.ac_g do
12:      Simple conflict resolution processing
13:    end for
14:    Composition of the returned mediation services
15:  end if
16: end for
17: for each cos_t in COS do
18:   cs.add(cos_t.position, MS_i)
19: end for
20: cs_cf ⟵ cs
21: return cs_cf
```

In both cases, simple or complex conflict, the "$cos_t.MService$" of "$\mathcal{COS}$" is updated with the simple or composed mediation services $\mathcal{MS}_i$ returned by the algorithm (lines 4,6,12). For each position "$cos_t.position$", the insertion of the corresponding $\mathcal{MS}_i$ in "cs" is performed (line 18). Note that, when several mediation services allow to resolve the same conflict, our algorithm randomly returns one of them as they achieve the same functionality. The algorithm 2 returns a conflict-free DaaS composition as depicted in Figure 1.(b). This conflict-free composition will be added to the set of compositions that are returned to the DCS for query plan execution as explained in section 2. In the case where conflicts remain unresolved, the "cs" is added to the set of not executable DaaS composition which will not be returned to DCS for query plan execution.

6 Implementation

The architecture of a CDR Prototype System is depicted in figure 8. We have implemented and tested a Java based application with multiple examples, including the motivating example[13]. Each Web service is deployed on top of a

[13] Implementation available at http://sites.google.com/site/ehrdaa

GlassFish Web server. The *DO* and *CAO* ontologies respectively are created by Protege 4.1[14] in RDFS. We exploit the extensibility feature of WSDL to hook operations elements in a WSDL file to their corresponding contextualized *PRV*(DaaS) and SPARQL query (mediation service) to the *DO* and *CAO* ontology. Atomic mediation services are created and stored in mediation service repository. The atomic *MSs* ensure the transformation between contextualized parameters having the same conflicting aspect "ac_g" and different instantiable classes "ac_i". Jena-2.6.4[15] is used as the reasoning engine for RDFS. In the eval-

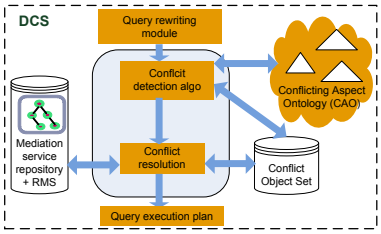
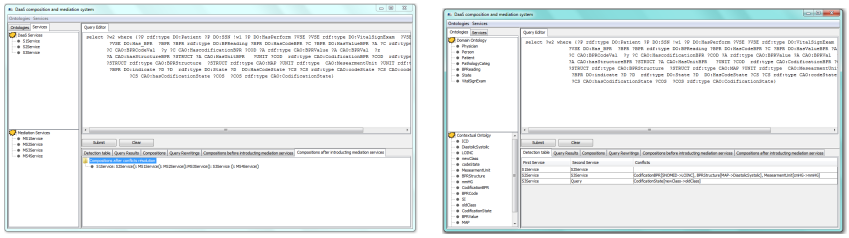


Fig. 8. CDRM architecture

uation phase we have considered a set of queries through which we identify the following: During the detection phase, we can detect the set of conflict aspects identified in "AC_g" based on our Conflict Detection Algorithm as depicted in figure 9(b)(e.g., 2 simple conflicts and one complex conflict). During the reso-



(a) DaaS composition conflcit free (b) Conflicting Object Set

Fig. 9. Screen captures of our application

lution phase, the Conflict Resolution Algorithm inserts the required mediation services and incorporates them in the right position to reconcile the conflicts as depicted in figure 9(a). The execution results indicate that the DaaS composition process normally completed; all the conflicts were successfully reconciled, that

[14] http://protege.stanford.edu/
[15] Jena Homepage http://jena.sourceforge.net

is, appropriate mediation services were properly called to convert BPR measures (value and measure units), BPR code and BPR value state; and the composition produced the expected output, namely, the state of BPR value.

7 Related work

During the last years, the DaaS composition problem has received a lot of attention through the approaches proposed in [1, 14, 12]. The DaaS composition systems proposed in these works do not have dealt with data heterogeneities at the semantic level. In our work, we propose a service-oriented approach to resolve data level conflicts with the contextualization of the PRV-based DaaS model. Further, few approaches have been developed to handle semantic heterogeneity in Web service composition. [11], [3], [9] and [7] investigate data-level heterogeneity between Web services through mapping relations to establish direct correspondences between the messages of two services. However, these works require Input-Output service parameters to be annotated with classes from the Domain Ontology as semantics, and do not take context into account. Their models do not allow expressions with variables for describing inputs and output of DaaSs. Our model is based on contextualized PRV using graph pattern, which can be considered as logical expressions with variables. Also, the approaches discussed in [5] and [6], have used the context representation for semantic mediation in Web service composition. In fact, they propose an extension of *DO* by a lightweight ontology which needs a small set of generic concepts to capture the context. However, these representations are challenged by their limited context model assuring only simple mapping between semantically equivalent context parameter (price, unit, etc.). Further, the low-level transformation code ensuring the conversion from one context to another makes the maintainability of semantic mediation between service composition components difficult. On the contrary, transformations are expressed as parametrized SPARQL queries in our context model, behaving as mapping rules, allowing to model complex mediation services. Also, the PRV-based DaaS model enriched with the notion of context allows to cover the full cycle of automatic DaaS composition unlike existing works which are restricted to handle the semantic heterogeneity at design time and to automatize DaaS composition separately.

8 Conclusion and future work

In this paper, we propose an extension to the PRV-based DaaS model based on the notion of context. The proposed context model expressed over Conflicting Aspect Ontology, which is an extension of Domain Ontology, aims to handle semantic conflicts in DaaS composition. Our model allows to specify the mediation service as aÂăparametrized SPARQL query, behaving as a mapping rule and performing simple or complex transformation of DaaS parameters values from one context to another. Our implementation demonstrates the applicability of our proposal. Our future perspectives will deal with performance and scalability

issues of our algorithm and the application of aspect-orientation to the PRVs, thus allowing to provide an organized approach to ensure the trustworthiness and the provenance of data that are returned by DaaSs.

9 Acknowledgment

The authors thank Sabrina Sahli for his valuable participation in the implementation.

References

1. Barhamgi, M., Benslimane, D., Medjahed, B. : A Query Rewriting Approach for Web Service Composition. IEEE Transactions Services Computing. 3, 206–222 (2010)
2. Euzenat, J., Polleres, A., Scharffe, F. : Processing Ontology Alignments with SPARQL. International Conference on Complex, Intelligent and Software Intensive Systems. 913–917 (2008)
3. Gagne, D., Sabbouh, M., Bennett, S., Powers, S. : Using Data Semantics to Enable Automatic Composition of Web Services. IEEE International Conference on Services Computing SCC '06. 438–444 (2006)
4. Goh, C.H, Bressan, S., Madnick, S.E, Siegel, M : Context Interchange: New Features and Formalisms for the Intelligent Integration of Information. ACM Trans. Inf. Syst. 270–293 (1999)
5. Li, X., Madnick, S., Zhu, H., Fan, Y. : Reconciling Semantic Heterogeneity in Web Services Composition. ICIS 2009 Proceedings. 20 (2009)
6. Mrissa, M., Ghedira, C., Benslimane, D., Maamar, Z. : A Context Model for Semantic Mediation in Web Services Composition. ER. 12–25 (2006)
7. Nagarajan, M., Verma, K., Sheth, A.P., Miller,J.A. : Ontology Driven Data Mediation in Web Services. Int. J. Web Service Res. 104–126 (2007)
8. Polleres, A., Scharffe, F., Schindlauer, R. : SPARQL++ for mapping between RDF vocabularies. Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part I. 878–896 (2007)
9. Sabbouh, M., Higginson, J.L., Wan, C., Bennett, S.R. : Using Mapping Relations to Semi Automatically Compose Web Services. IEEE Congress on Services. 211–218 (2008)
10. Sciore, E. : Using semantic values to facilitate interoperability among heterogeneous information systems. ACM Transactions on Database Systems. 254–290 (1994)
11. Spencer, B., Liu, S. : Inferring Data Transformation Rules to Integrate Semantic Web Services. International Semantic Web Conference. 456–470 (2004)
12. Vaculín, R., Chen, H., Neruda, R., Sycara, K. : Modeling and Discovery of Data Providing Services. ICWS. 54–61 (2008)
13. Wiederhold, G. : Mediators in the Architecture of Future Information Systems. Computer. 25, 38–49 (1992)
14. Zhou, L., Chen, H., Wang, H., Zhang,Y. : Semantic Web-Based Data Service Discovery and Composition. SKG. 213–219 (2008)