SCIENCE CHINA
Information Sciences

• **RESEARCH PAPER** •

November 2012 Vol. 55 No. 11: 2428–2444
doi: 10.1007/s11432-011-4341-5

Automatic composition of information-providing web services based on query rewriting

ZHAO WenFeng*, LIU ChuanChang & CHEN JunLiang

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876*, China*

Received May 31, 2010; accepted July 5, 2010; published online August 2, 2011

Abstract Compared with normal web services, information-providing services have unique features that have seldom been considered in existing research on the automatic composition of web services. In this paper, we present a simple, yet well-formed, semantic-based capability model for information-providing web services, which can express such features as not modifying the world state and not requiring all input/output parameters to be supplemented with class information as semantics. We then present a corresponding automatic composition method derived from a query rewriting algorithm, MiniCon, used in the data integration field. This method adequately utilizes previous features, enables primitive semantic reasoning, and can generate executable BPEL scripts automatically. Performance of the method is complementary to traditional search-based ones. Experiments on a typical type of problem show that the method is usable in practice.

Keywords web services, automatic composition, query rewriting, data integration, ontology, semantic web

Citation Zhao W F, Liu C C, Chen J L. Automatic composition of information-providing web services based on query rewriting. Sci China Inf Sci, 2012, 55: 2428–2444, doi: 10.1007/s11432-011-4341-5

1 Introduction

The widespread adoption of SOA (service-oriented architecture) concepts has facilitated the integration of heterogeneous systems, and made the problem of automatic composition of web services a more worthwhile research area. As commonly agreed, web services fall into two categories depending on their functionality: world-altering services and information-providing ones [1–4]. Services of the latter type, such as quotation searching and weather forecasting, do not modify the state of the world and are simpler than those of the former type. This means that any research on the automatic composition of these services could become the foundation for similar research on the former type. Moreover, from a safety point of view, runtime faults in this type of service tend not to lead to instant side effects, thereby contributing to their suitability for automatic composition. Furthermore, there are many more instances of this type of service, both on the Internet and in many specific fields such as life science research, than that of the former type. Thus the automatic composition of information-providing services is a particularly important research area.

In the literature, the capability of a web service is normally represented by inputs, outputs, preconditions, and effects (IOPE) [2,3]. In the implementation layer, some services require multiple message

*Corresponding author (email: zhaowenfeng@gmail.com) interchanges with the user, making them stateful, while others only require a single exchange of messages, making them stateless single operations. In stateful services, composition could be performed on the message patterns [5,6], or first on the IOPE and then, given detailed decisions, on the message patterns [7]. In stateless ones, the composition is mainly based on IOPE. This paper considers only IOPE-based composition of stateless services. Current solutions to this problem can be categorized in three groups: 1) searching directly in the state space of the problem [8–13], i.e., searching for paths in a directed bipartite graph when the starting and end nodes are given, or in simplified variants thereof, with services and their IOPEs regarded as two node types; 2) reducing the problem to a planning problem in artificial intelligence (AI) [1,2,14,15]; 3) solving the problem using automatic theorem proving in logic [16,17], reachability decisions in Petri nets [18], or other methods with relatively mature solutions. Most of these approaches do not differentiate the different features of information-providing services and world-altering ones, making them versatile, but too inefficient for practical use. In fact, a commonly agreed capability model for information-providing services is lacking at present.

This paper focuses wholly on the information-providing type of services, analyzing their features, setting up a capability model for them, which is simple but has well-formed semantics, and proposing an automatic composition method that utilizes these features. This method regards services of this type as special database views, that is, views with binding patterns, and accordingly reduces the service composition problem to a query rewriting problem in the data integration field and solves it through an efficient query rewriting algorithm MiniCon. Furthermore, executable BPEL scripts can also be generated automatically if needed. Similar to [19] and [20], this method is based on an “enumerate–combine” mechanism, which is very different to the search procedure that most of the previous approaches have relied on, resulting in different characteristics in performance. Experiments show that the performance on a typical type of problem at real-world scale could satisfy the requirements of various practical applications.

2 Problem description

In general, the automatic composition of web services is defined as, given a service repository and a requested service R, ascertaining a set of solution plans $\mathcal{P}$, in which each plan p corresponds to a composite service in the form (G, Ψ, CMP), where

1) G is a flow chart consisting of instances of some existing services in the repository (i.e., component services of p) organized by control structures such as sequence, parallel, select, and so on;

2) Ψ is a function describing the data flow in G mapping outputs of R and inputs of component services to constants, outputs of preceding component services, or inputs of R;

3) CMP is a set of arithmetic comparison expressions used by the executing engine to filter the data flow while p is executed.

In addition, p should satisfy the requirement of R. At the same time the process of obtaining $\mathcal{P}$ should be realized completely by the program without human intervention. Sometimes an executable script is further required to be derived from p. In reality the requirement for G is often relaxed to a directed acyclic graph (DAG) by disabling select, iteration and other complex structures [12,13,15]. The nodes in the DAG represent the component services, while the directed edges represent constraints on the execution order thereof. In the following we abbreviate the requested service to request and the existing service in the repository to advertisement for clarity.

How to decide whether “p satisfies the requirement of R” in the above description relies on the precise definition of the service capability. Influenced by the description of actions in the AI planning problem, current capability models for web services are usually targeted at world-altering services. But we consider that it is worth setting up separately a capability model for information-providing services because of two key differences between the two types of services as outlined below.

The first difference is that an information-providing service is simpler than a world-altering one. A service of the former type could be regarded as a query within a definite world state, and after the service has been executed its precondition remains valid, which is not the case in a service of the latter type. For example, a login service might have $\neg$ LoggedIn(x) as its precondition and LoggedIn(x) as the effect.

Second, as we pointed out in [21], the inputs and outputs (IO) of many information-providing services are more suited to be regarded as data-type properties of various objects, rather than the objects themselves. For example, in an OWL ontology the book name is typically not expressed as a class, but as a datatypeproperty of class book. Now suppose there is a book-query service S1 that takes the book name as its input. If the semantics of the input are required to be annotated with a Class, Book will probably be the most appropriate one. However, suppose because of the same reason, another requested service R1 expecting a similar capability but taking the ISBN number as its input, also annotates its input with Book. As a result, although S1 might match R1, S1 cannot be used where R1 is expected! Although similar situations exist for world-altering services, as in AI planning problems, in which the comparatively few objects usually can be categorized distinctly (e.g., as Robot, Table, or Block), these are not very common. Accordingly, many studies explicitly require IO to be annotated with classes [8–11].

2.1 Capability model of information-providing services

Intuitively, the capability of information-providing services could be expressed with inputs, outputs, and constraints on their values, with some of the constraints required to be satisfied by the caller. We use the normal IOPE structure, but with a slight simplification and a different interpretation.

Definition 2.1 (Capability description)**.** The capability description, $\mathcal{D}_S$, of an information-providing web service S is the quadruple $\langle \boldsymbol{i}, \boldsymbol{o}, P(\boldsymbol{i}, \boldsymbol{l}), E(\boldsymbol{i}, \boldsymbol{l}, \boldsymbol{o})\rangle$, where $\boldsymbol{i}$ and $\boldsymbol{o}$ are, respectively, name lists of inputs and outputs of S, $P(\boldsymbol{i}, \boldsymbol{l})$ and $E(\boldsymbol{i}, \boldsymbol{l}, \boldsymbol{o})$ are, respectively, the precondition and effect of S, both being well-formed formulas of first-order logic without quantifiers, and $\boldsymbol{l}$ is the list of local variables therein. Predicates in the formulas are either from the domain ontology or arithmetic comparison operators $=, <, \leqslant, >$, and $\geqslant$; functions in the formulas are either from the domain ontology1) or arithmetic operators $+$, $-$, $\times$, $\div$ and so on; the arguments in the formulas are constants or variables in $\boldsymbol{i}$, $\boldsymbol{l}$, and $\boldsymbol{o}$. $P(\boldsymbol{i}, \boldsymbol{l})$ could be empty, but $E(\boldsymbol{i}, \boldsymbol{l}, \boldsymbol{o})$ must not be empty. Every variable in $\boldsymbol{i}$ or $\boldsymbol{o}$ should appear in $P(\boldsymbol{i}, \boldsymbol{l})$ or $E(\boldsymbol{i}, \boldsymbol{l}, \boldsymbol{o})$.

Definition 2.1 only gives the syntactic structure for $\mathcal{D}_S$. To depict the service capability precisely, semantics need to be specified for the structure. We opt for an indirect approach and map components of the structure onto first-order logic (FOL) that has a specified well-formed semantics. Thus, we first set up a predicate with the same name as S, $S(\boldsymbol{i}, \boldsymbol{o})$. It is true iff when invoked with $\boldsymbol{i}$ as inputs, S outputs a dataset that includes $\boldsymbol{o}$ (in the following we refer to this case as: S can provide data $\langle \boldsymbol{i}, \boldsymbol{o}\rangle$). Then $\mathcal{D}_S = \langle \boldsymbol{i}, \boldsymbol{o}, P(\boldsymbol{i}, \boldsymbol{l}), E(\boldsymbol{i}, \boldsymbol{l}, \boldsymbol{o})\rangle$ is interpreted as an FOL formula:

$$\forall\, \boldsymbol{i} \forall\, \boldsymbol{o}(S(\boldsymbol{i}, \boldsymbol{o}) \;\leftrightarrow\; \exists\, \boldsymbol{l}(P(\boldsymbol{i}, \boldsymbol{l}) \rightarrow E(\boldsymbol{i}, \boldsymbol{l}, \boldsymbol{o}))). \tag{1}$$

If $P(\boldsymbol{i}, \boldsymbol{l})$ in $\mathcal{D}_S$ is empty, it is replaced by TRUE in (1). Formula (1) indicates that: a) any datum provided by S satisfies the right hand side, i.e., if it implies $P(\boldsymbol{i}, \boldsymbol{l})$ it must also imply $E(\boldsymbol{i}, \boldsymbol{l}, \boldsymbol{o})$; b) it does so in the reverse direction, i.e., any datum satisfying the right hand side can be provided by S. However, in reality, not every service has such strong semantics. For example, the advertisement services in a composition problem are usually regarded as only satisfying a) and not b), i.e.,

$$\forall\, \boldsymbol{i} \forall\, \boldsymbol{o}(S(\boldsymbol{i}, \boldsymbol{o}) \;\rightarrow\; \exists\, \boldsymbol{l}(P(\boldsymbol{i}, \boldsymbol{l}) \rightarrow E(\boldsymbol{i}, \boldsymbol{l}, \boldsymbol{o}))). \tag{2}$$

We call the semantics expressed by formula (1) the sufficient and necessary semantics and those of formula (2) the emphnecessary semantics. As to the intention of request R in the composition problem, it is usually correct to be referred to as sufficient semantics expressed by formula (3) (although perhaps in certain situations, sufficient and necessary semantics is more appropriate):

$$\forall\, \boldsymbol{i} \forall\, \boldsymbol{o}(\; S(\boldsymbol{i}, \boldsymbol{o}) \;\leftarrow\; \exists\, \boldsymbol{l}(P(\boldsymbol{i}, \boldsymbol{l}) \rightarrow E(\boldsymbol{i}, \boldsymbol{l}, \boldsymbol{o}))). \tag{3}$$

The fact that the precondition and effect (PE) are linked by an implication operator means that if the real inputs provided by the caller do not satisfy the precondition, S guarantees nothing as the output. But in reality, many services return an empty set rather than an indeterminate dataset in this case. If

1) such as OWL FunctionalProperty.

Table 1 Capability description of various services in the motivating example.

Service name	$\boldsymbol{i}$	$\boldsymbol{o}$	$P(\boldsymbol{i}, \boldsymbol{l})$	$E(\boldsymbol{i}, \boldsymbol{l}, \boldsymbol{o})$	Semantics
AmazonBook Search (Ad.)	title	author, rank	EMPTY	Book(bk) $\wedge$ hasTitle(bk, title) $\wedge$ hasAuthor(bk, author) $\wedge$ hasAmazonRank(bk, rank)	Necessary
WriterInfoSer-vice (Ad.)	name	country, birthYear, intro	name>= "A" $\wedge$ name<="zzz"	Writer(wtr) $\wedge$ Country(country) $\wedge$ comesFrom(wtr, country) $\wedge$ hasName(wtr, name) $\wedge$ bornIn(wtr, birthYear) $\wedge$ hasIntro(wtr, intro)	Necessary
CountryInfoSer-vice (Ad.)	country	area, population	Country(country)	hasArea(country, area) $\wedge$ hasPopulation(country, population)	Necessary
SyntheticBook Service (Req.)	book	saleRank, author, birthYear, population	EMPTY	Book(bk) $\wedge$ Person(wtr) $\wedge$ Country(c) $\wedge$ hasTitle(bk, book) $\wedge$ hasAmazonRank(bk, saleRank) $\wedge$ asAuthor(bk, wtr) $\wedge$ hasName(wtr, author) $\wedge$ bornIn(wtr, birthYear) $\wedge$ comesFrom(wtr, c) $\wedge$ hasPopulation(c, population)	Sufficient

this behavior could be ensured, PE could be linked instead with a conjunction operator to simplify the description and computation. The domain ontology in the definition could be either a normal semantic web ontology such as those in OWL and RDFS, or a traditional relational data schema. In practical use, it is usually necessary to restrict the expressive power of the logic formulas in the PE for the sake of computational complexity.

The above capability model embodies well the two aforementioned features of information-providing services. First, as seen in the example of the login service, the PE of a world-altering service cannot be linked directly with an implication operator. Instead, a more complex mechanism is required to specify the semantics, such as 1) situation calculation in which PE formulas are added in additional arguments representing states, so that $P \rightarrow E$ can be interpreted as an FOL formula [1], and 2) linear logic which itself is designed to express dynamic knowledge [16]. Second, Definition 2.1 does not require IO to be annotated with classes; but if such information exists, it can also be expressed by adding into the PE corresponding unary atomic formulas that behave in the same way as other atoms in the PE.

As a motivating example, Table 1 lists the capability descriptions of three advertisements and one request in a service composition problem. The three advertisements respectively provide information about books, writers, and countries, while the request is expected, given the name of a book, to find the sale rank of the book in the Amazon online store, as well as information of its author and the population of his/her country. The URI prefix of OWL classes and properties is omitted in the table.

2.2 Service composition problem

Based on the capability model in subsection 2.1, we can define the requirement "p satisfies the requirement of R" as having the following two conditions satisfied simultaneously:

CR1. p is sound w.r.t. R, i.e., all the data provided by p are in what R wants to provide;

CR2. p itself is feasible, i.e., as long as the real inputs of p at runtime satisfy the precondition of R, the precondition of each component service of p will also be satisfied.

Given that the composite service corresponding to p is notated as Serv_p, condition CR1 could be expressed as

$$\forall\, \boldsymbol{i} \forall\, \boldsymbol{o}(\mathrm{Serv}_p(\boldsymbol{i}, \boldsymbol{o}) \ \rightarrow \ R(\boldsymbol{i}, \boldsymbol{o})).$$

Because information-providing services do not modify the world state, the execution of any service cannot revoke the effects of preceding services. Thus, the constraints needed to be satisfied by the data provided by the composite service are just the conjunction of constraints of the component services. Applying formula (1) to R and the component services leads to an equivalent condition of the previous formula:

$$\forall\, \boldsymbol{i} \forall\, \boldsymbol{o}_1 \cdots \forall\, \boldsymbol{o}_n \left(\bigwedge_{i=1,\ldots,n} \exists\, \boldsymbol{l}_i (P_i(\Psi(\boldsymbol{i}_i), \boldsymbol{l}_i) \rightarrow E_i(\Psi(\boldsymbol{i}_i), \boldsymbol{l}_i, \boldsymbol{o}_i)) \wedge \bigwedge_{j=1,\ldots,m} \mathrm{cmp}_j \right.$$
$$\left. \rightarrow\ \exists\, \boldsymbol{l}(P(\boldsymbol{i}, \boldsymbol{l}) \rightarrow E(\boldsymbol{i}, \boldsymbol{l}, \Psi(\boldsymbol{o}))) \right). \tag{4}$$

Here, n is the number of nodes in G; P and E are the PE of R, while P_i and E_i are those of the service related to node V_i; Ψ maps inputs of V_i to constants, variables in $\boldsymbol{i}$, or variables in $\boldsymbol{o}_i (i = 1, \ldots, n)$ produced before V_i and maps outputs of R to constants or variables in $\boldsymbol{i}$ and $\boldsymbol{o}_i (i = 1, \ldots, n)$; and cmp_j are arithmetic comparisons. If the advertisements and request, however, do not adopt the sufficient and necessary semantics as shown in (1), but only adopt necessary semantics and sufficient semantics respectively, (4) is a sufficient, but not necessary condition for CR1.

Because determining whether the precondition of a component service S_i is satisfied only relies on the outputs of the preceding component services and inputs of p, condition CR2 could be depicted as

$$\forall\, \boldsymbol{i} \forall\, \boldsymbol{o}_{k_1} \cdots \forall\, \boldsymbol{o}_{k_T} \left(\exists\, \boldsymbol{l} P(\boldsymbol{i}, \boldsymbol{l}) \wedge \bigwedge_{V_j \in \mathrm{Prenodes}(V_i)} \exists\, \boldsymbol{l}_j (P_j(\Psi(\boldsymbol{i}_j), \boldsymbol{l}_j) \rightarrow E_j(\Psi(\boldsymbol{i}_j), \boldsymbol{l}_j, \boldsymbol{o}_j)) \wedge \bigwedge_{h=1,\ldots,m} \mathrm{cmp}_h \right.$$
$$\left. \rightarrow\ \exists\, \boldsymbol{l}_i P_i(\Psi(\boldsymbol{i}_i), \boldsymbol{l}_i) \right) \qquad (i = 1, \ldots, n), \tag{5}$$

where Prenodes(v) denotes the set of preceding nodes of node v and T=|Prenodes(V_i)|, $k_1, \ldots, k_T$ is the sequence number of nodes in Prenodes(V_i).

In addition, as an optional condition, some applications might require $\mathcal{P}$ as a whole to satisfy the condition:

CR3. $\mathcal{P}$ is complete, i.e., any piece of data that R wants to provide could be provided by some services in $\mathcal{P}$, as shown in

$$\forall\, \boldsymbol{i} \forall\, \boldsymbol{o} \left(\bigvee_{p \in \mathcal{P}} \mathrm{Serv}_p(\boldsymbol{i}, \boldsymbol{o})\ \leftarrow\ R(\boldsymbol{i}, \boldsymbol{o}) \right).$$

3 Background of proposed method

The proposed composition method in this paper is based on a query rewriting algorithm used in data integration. This section discusses briefly the related theory and algorithm; see [22–25] for further details.

In database research, data integration implies, in an environment comprising a global data schema $\mathcal{G}$, multiple data sources each with definition $\mathcal{S}$, and a mapping $\mathcal{M}$ between $\mathcal{G}$ and $\mathcal{S}$ to answer users' data queries formed in $\mathcal{G}$. If $\mathcal{M}$ defines $\mathcal{S}$s as views of $\mathcal{G}$, i.e., "local as view" (LAV), a basic problem is query rewriting. This involves rewriting the query Q by replacing all relations from $\mathcal{G}$ with views so that Q can be answered using only data stored in materialized views (i.e., data sources).

Definition 3.1 (Conjunctive query). A conjunctive query (CQ) is a data query in the following form:

$$q(\boldsymbol{X})\ :-\ e_1(\boldsymbol{X}_1), \ldots, e_n(\boldsymbol{X}_n),$$

where $\boldsymbol{X}, \boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ are vectors consisting of variables or constants, and $e_1, \ldots, e_n$ refer to database relations. If $e_1, \ldots, e_n$ could also be arithmetic comparison operators $=, <, \leqslant, >$, and $\geqslant$, the query is called a conjunctive query with arithmetic comparisons (CQAC). $q(\boldsymbol{X})$ is the *head* and represents the result relation, while $e_1(\boldsymbol{X}_1), \ldots, e_n(\boldsymbol{X}_n)$ represent the body, each clause $e_i(\boldsymbol{X}_i)$ of which is a subgoal; either an ordinary subgoal if e_i refers to a database relation, or an arithmetic comparison subgoal if e_i is a comparison predicate. The query is required to be safe, i.e., $\boldsymbol{X} \subseteq \boldsymbol{X}_1 \cup, \ldots, \cup \boldsymbol{X}_n$ and any variable that appears in an arithmetic comparison subgoal must also appear in an ordinary subgoal. Variables appearing in $q(\boldsymbol{X})$ are head variables or distinguished variables, whereas the others are existential variables.

Similar to the three types of semantics given in subsection 2.1, there are also three cases when defining data sources using Definition 3.1: exact, sound, and complete views. When not specified, the sources are usually considered to be sound [22]. This assumption , also called the open world assumption, means that an extension of a view, i.e., data instances in a materialized view, is only a subset of the data complying with its definition in the whole system. This assumption incorporates Definition 3.2.

Definition 3.2 (Maximally-contained rewriting). Let Q be a query in data schema $\mathcal{G}$, $\mathcal{V} = V_1, \ldots, V_n$ be a set of views defined in $\mathcal{G}$, and $\mathcal{L}$ be a query definition language. Then a query Q' in $\mathcal{L}$ is a maximally-contained rewriting of Q using $\mathcal{V}$ w.r.t. $\mathcal{L}$ if

1. Any ordinary subgoal in the body of Q' is formed by a view in $\mathcal{V}$;

2. For any database D, and extensions $v_1, \ldots, v_n$ of the views such that $v_i \subseteq V_i(D)$, for $(1 \leqslant i \leqslant n)$, $Q'(v_1, \ldots, v_n) \subseteq Q(D)$ for all i;

3. There is no other query Q_1 in $\mathcal{L}$, such that for every database D and extensions $v_1, \ldots, v_n$ as explained above (1) $Q'(v_1, \ldots, v_n) \subseteq Q_1(v_1, \ldots, v_n)$, and (2) $Q_1(v_1, \ldots, v_n) \subseteq Q(D)$, and there exists at least one database for which (1) is a strict set inclusion.

Note that maximally-contained rewritings are related to certain query languages. One important such language is the Union of CQACs, in which a query Q contains several CQAC queries with the same head predicate indicating that the disjunction of their bodies forms the definition of Q.

The MiniCon algorithm is an efficient query rewriting algorithm. Originally proposed by Pottinger et al. [24], it was extended and further analyzed by Afrati et al. [25]. The principle of the algorithm is that, for each ordinary subgoal in Q, first enumerate all views containing a subgoal with the same predicate to form a "bucket", and then combine views from different buckets and take them as subgoals after variable substitution embodying the join operation among them. As long as the value range, defined by arithmetic comparison subgoals, of unified variables is consistent, the combination, after inserting various new comparison subgoals to filter out data unwanted by Q, will be a valid rewriting. In some situations, the fields that the join operation operates on do not appear in the view heads, making the join impossible. The MiniCon algorithm excludes such views as soon as possible by determining adaptively the size of the join unit (called the MCD) and only needs to consider MCD combinations that form a partition of subgoals of Q. When Q and $\mathcal{V}$ are in CQ, the basic MiniCon algorithm is sound and complete in finding a maximally-contained rewriting w.r.t. Union of CQs [24]. Moreover, when Q and $\mathcal{V}$ are in CQAC and satisfy the homomorphism property, MiniCon, combined with the reasoning in the partial relation among variable values, is sound and complete w.r.t.Unions of CQACs [25].

4 Query rewriting-based automatic composition method

Because of the high similarity between information-providing services and database queries or views, the service composition problem, as described in section 2, can be transformed into the query rewriting problem and solved using the MiniCon algorithm. If we regard request R as the query Q, advertisement S as the view V and the service repository as the view set $\mathcal{V}$, then a composition plan p could be regarded as a rewriting. Every ordinary subgoal in the rewriting corresponds to a component service of p, the open world assumption of the database corresponds to the necessary semantics of web services, and the maximally-contained rewriting in this assumption is similar to a composition plan set $\mathcal{P}$ that complies with condition CR3.

Nevertheless, when delving into the details and comparing formulas (1)–(3) with Definition 3.1 and its three types of views, we can identify two differences between services and queries.

1) Services have inputs, yet no such concept exists in normal queries2).

2) In [23,26], views with access pattern limitations or binding patterns are discussed that require certain fields to be bound with determinate values when queried. These fields act the same role as inputs in services. However, the MiniCon algorithm does not take this into consideration.

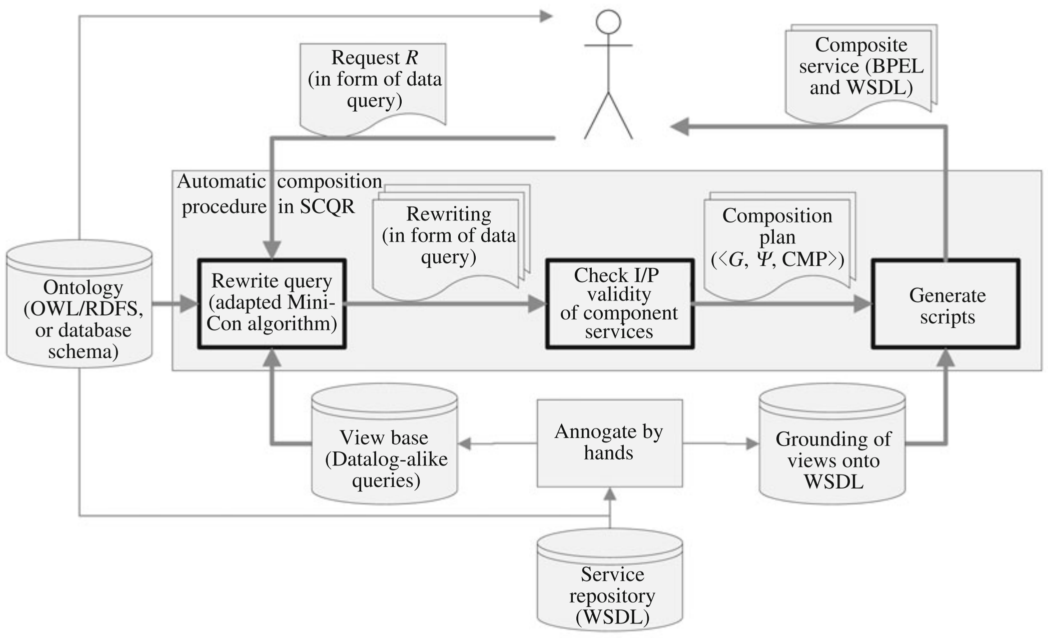
Request *R* (in form of data query)
Composite service (BPEL and WSDL)
Automatic composition procedure in SCQR
Rewriting (in form of data query)
Composition plan (⟨*G*, *Ψ*, CMP⟩)
Ontology (OWL/RDFS, or database schema)
Rewrite query (adapted Mini-Con algorithm)
Check I/P validity of component services
Generate scripts
View base (Datalog-alike queries)
Annogate by hands
Grounding of views onto WSDL
Service repository (WSDL)

Figure 1 Automatic service composition procedure based on query rewriting.

2) Subgoals in the query definition are divided into two groups, namely, preconditions and effects.

Thus, the MiniCon algorithm needs to be adapted for use in service composition.

As shown in Figure 1, the entire composition method consists of three steps. First, perform query rewriting and obtain rewritings that are valid without considering inputs and preconditions. Next, check each rewriting to see whether the inputs of the component services are covered by Ψ and their preconditions can be satisfied (i.e., CR2); if true, a valid composition plan in the form of a DAG can be derived from the rewriting. Finally, generate a BPEL process if needed.

4.1 Overview of the method

The first two steps in Figure 1, i.e., the procedure up to obtaining composition plans, are given as Algorithm 1. Phase 1 in the algorithm corresponds to the first half of the first step in the figure, i.e., the MCD forming phase of query rewriting, while phase 2 includes the MCD combining phase in the first step and the second one.

At the same time, the formulas in the PE in Definition 2.1 should be restricted as follows to apply Algorithm 1.

L1. Of the logic operators, only conjunction is allowed; implication, disjunction, not and function are disabled.

L2. No local variables may appear in the precondition.

L3. (optional) No arithmetic comparison clause may take two variables as arguments.

L4. (optional) Non-arithmetic-comparison predicates in the precondition must all be unary ones.

Of these, L1 restricts the form of the PE to a conjunction of atomic formulas since the MiniCon algorithm only supports queries and views in CQAC. L3 and L4 merely simplify the implementation by restricting the atoms in the precondition to only two kinds, that is, a description of semantic types of inputs and comparisons between inputs and constants. Services complying with L1 and L3 correspond to CQSI queries in [25]. In addition, L2 makes it possible to treat the PE in formulas (1)–(3) separately so that, in conjunction with the equivalence of the FOL formula $p \to (q \to r)$ and $p \wedge q \to r$, formulas (4) and (5) in subsection 2.2 can be transformed, respectively, to

$$\forall\, \boldsymbol{i} \forall\, \boldsymbol{o}_1 \cdots \forall\, \boldsymbol{o}_n \left(P(\boldsymbol{i}) \wedge \bigwedge_{i=1,\ldots,n} (P_i(\Psi(\boldsymbol{i}_i)) \to \exists \boldsymbol{l}_i E_i(\Psi(\boldsymbol{i}_i), \boldsymbol{l}_i, \boldsymbol{o}_i)) \wedge \bigwedge_{j=1,\ldots,m} \mathrm{cmp}_j \;\to\; \exists \boldsymbol{l} E(\boldsymbol{i}, \boldsymbol{l}, \Psi(\boldsymbol{o})) \right) \quad (6)$$

Algorithm 1. Automatic service composition based on query rewriting

```
01 set⟨Plan⟩ autoCompose($Req:Ds, $Svcs:set⟨Ds⟩, $Ont:RDFSontology)
02     $answer = EMPTY;
03     $mcds = form MCDs from $Req.E and $Svcs.E utilizing Ont; //Phase 1. form MCDs
04     //Phase 2. combine MCD and get rewritings and composition plans
05     for-each ($comb : MCD combinations from $mcds, that exactly partitions subgoals of $Req.E)
06         if (value ranges of variables in $comb are not consistent)
07             continue;
08         insert into $comb new comparison subgoals if needed to satisfy Req.E;
09         $rwrt = derive rewriting from $comb;
10         for-each ($svc : $rwrt.ordinarySubgoals)
11             if (some input of $svc could not be given data sources in $rwrt)
12                 goto 05;
13         for-each ($svc: $rwrt.services)
14             if ($svc.P in $rwrt could not be satisfied)
15                 goto 05;
16             insert into $rwrt new comparisons if needed to satisfy $svc.P;
17         $p = derive composition plan from $rwrt;
18         add $p to $answer;
19     return $answer;
```

$$\forall\, \boldsymbol{i} \forall\, \boldsymbol{o}_{k_1} \cdots \forall\, \boldsymbol{o}_{k_T} \left(P(\boldsymbol{i}) \wedge \bigwedge_{V_j \in \mathrm{Prenodes}(V_i)} (P_j(\Psi(\boldsymbol{i}_j)) \rightarrow \exists \boldsymbol{l}_j E_j(\Psi(\boldsymbol{i}_j), \boldsymbol{l}_j, \boldsymbol{o}_j)) \wedge \bigwedge_{h=1,\ldots,m} \mathrm{cmp}_h \right.$$
$$\left. \rightarrow\; P_i(\Psi(\boldsymbol{i}_i)) \right) \qquad (i = 1, \ldots, n). \tag{7}$$

Applying (7) to (6) and itself removes P_i/P_j on the left hand side. Then (6) and (7) are, respectively, simplified as (these must hold simultaneously):

$$\forall\, \boldsymbol{i} \forall\, \boldsymbol{o}_1 \cdots \forall\, \boldsymbol{o}_n \left(P(\boldsymbol{i}) \wedge \bigwedge_{i=1,\ldots,n} \exists\, \boldsymbol{l}_i E_i(\Psi(\boldsymbol{i}_i), \boldsymbol{l}_i, \boldsymbol{o}_i) \wedge \bigwedge_{j=1,\ldots,m} \mathrm{cmp}_j \;\rightarrow\; \exists \boldsymbol{l} E(\boldsymbol{i}, \boldsymbol{l}, \Psi(\boldsymbol{o})) \right) \tag{8}$$

$$\forall\, \boldsymbol{i} \forall\, \boldsymbol{o}_{k_1} \cdots \forall\, \boldsymbol{o}_{k_T} \left(P(\boldsymbol{i}) \wedge \bigwedge_{V_j \in \mathrm{Prenodes}(V_i)} \exists\, \boldsymbol{l}_j E_j(\Psi(\boldsymbol{i}_j), \boldsymbol{l}_j, \boldsymbol{o}_j) \wedge \bigwedge_{h=1,\ldots,m} \mathrm{cmp}_h \right.$$
$$\left. \rightarrow\; P_i(\Psi(\boldsymbol{i}_i)) \right) \quad (i = 1, \ldots, n). \tag{9}$$

Formula (8), without $P(\boldsymbol{i})$ on the left hand side, is exactly in the form that MiniCon can work on.

So, the actual manner in which MiniCon is applied in Algorithm 1 is as follows.

D1. While rewriting lines 1–9, regard the effects of the advertisements and requests as the bodies of their corresponding queries, ignoring $P(\boldsymbol{i})$ in (8).

D2. In lines 13–16, check the rewriting against CR2 according to formula (9).

In addition, the following restriction is applied to PE formulas:

L5. No atom may appear in both the PE of a service.

L6. Any variable in $\boldsymbol{i}$ or $\boldsymbol{o}$ should also appear in $E(\boldsymbol{i}, \boldsymbol{l}, \boldsymbol{o})$.

Ignoring $P(\boldsymbol{i})$ in D1 will lead to some very special composition plans being overlooked. However, because MiniCon itself only considers one-to-one implications among ordinary subgoals, this omission is usually acceptable.

Table 2 MCDs obtained in the motivating example

No.	Related view V	F subgoals(Q) $\rightarrow$ subgoals(V)	corresponding F' $2^{\text{Vars}(Q)} \rightarrow 2^{\text{Vars}(V)}$
i	AmazonBookSearch	$1 \rightarrow 1,\ 4 \rightarrow 2,\ 6 \rightarrow 3,\ 5 \rightarrow 4$	{bk} $\rightarrow$ {bk}, {book} $\rightarrow$ {title}, {author}$\rightarrow$ {author}, {saleRank} $\rightarrow$ {rank}
ii	WriterInfoService	$2 \rightarrow 1,\ 9 \rightarrow 3,\ 7 \rightarrow 4,\ 8 \rightarrow 5$	{wtr} $\rightarrow$ {wtr}, {c} $\rightarrow$ {country}, {author} $\rightarrow$ {name}, {birthYear} $\rightarrow$ {birthYear}
iii	WriterInfoService	$3 \rightarrow 2$	{c} $\rightarrow$ {country}
iv	CountryInfoService	$3 \rightarrow 3$	{c} $\rightarrow$ {country}
v	CountryInfoService	$10 \rightarrow 2$	{c} $\rightarrow$ {country}, {population} $\rightarrow$ {population}

Note: Q and V are Datalog-like queries corresponding, respectively, to R and S (without preconditions); the numbers in column F denote the sequence numbers of ordinary subgoals of Q or V.

Because formulas (8) and (9) are assured, the resulting plans are certain to satisfy CR1 and CR2 irrespective of whether the advertisements adopt necessary semantics or sufficient and necessary semantics. Adopting necessary semantics means that other plans may exist that satisfy CR1 and CR2, but not (8) and (9), which will therefore, be overlooked by Algorithm 1.

4.2 Rewriting the query

In previous applications of the MiniCon algorithm, the following two adaptations to the algorithm itself were necessary.

A1. While forming and combining MCDs, for both requests and advertisements , the algorithm only considered ordinary subgoals in the effects, ignoring those in the preconditions.

A2. During the insertion of arithmetic comparisons in the rewriting, the insertion criteria were modified so that the inserted comparisons would, after proper variable substitution and in conjunction with comparisons in the precondition of R and the PE of the component services, be consistent and imply comparisons in the effect of R.

In addition, to utilize the reasoning on the RDFS/OWL ontology as much as possible during the composition of services in a semantic web environment, the following adaptation is applied.

A3. During the formation of MCDs, the criteria for predicates is modified so that in determining whether a query subgoal g can be mapped to a view subgoal v, the predicates of the subgoals should be the same or the corresponding relation (e.g., OWL Class and ObjectProperty/DatatypeProperty) of the predicate of v in the domain ontology should be a descendant of that of g.

The resulting rewriting procedure is divided into two phases similar to MiniCon itself, that is, forming MCDs (line 3 in Algorithm 1) and combining MCDs (lines 4–9), where the MCD is the key concept. As shown in Table 2, an MCD represents a partial mapping F from some ordinary subgoals of Q to ordinary subgoals of a view V, where

1) The predicate of the inverse image is either the same or implied by that of the image (i.e., the latter is a descendant of the former);

2) There is a partial mapping F' between variable equivalence classes of Q and V, so that (constants can be removed beforehand by introducing '=' subgoals and new variables that are regarded as distinguished variables):

A) It maps distinguished variables of Q to distinguished variables of V;

B) If it maps an existential variable t of Q to an existential variable of V, all ordinary subgoals of Q that contain t should appear in the domain of F;

C) No equivalence class greater than 1 in the domain or the range contains a existential variable;

D) The value ranges of member variables of each equivalence class, determined by comparison subgoals in the precondition and effect, are consistent and the common range of variables in a equivalence class in the domain of F' is consistent with that of its image;

3) The domain of F is minimal while satisfying Clause 2B.

Most of the above clauses are required by MiniCon itself, with the exception of 1), which is the result of adaptation A3. The MCDs obtained in the motivating example are listed in Table 2, of which MCD (ii) is only possible after applying this adaptation, given that there is an axiom Writer⊑Person in the domain ontology.

In the second phase of the rewriting (lines 5–9 in Algorithm 1), the MiniCon algorithm ensures that we need only consider such MCD combinations that member domains of each of them forms a partition of ordinary subgoals of Q. For each such combination c, the algorithm combines equivalence classes in the domain of F' for all MCDs to obtain the set, EC, of equivalence classes of Q variables. As long as the variables in each member of EC are consistent, c will be a valid rewriting. In the motivating example, two rewritings are obtained. The first is derived from MCD combination{i, ii, iii, v} (where variables with a leading underscore represent fresh variables (i.e., not from Q), while 'b' and 'f' symbols after each service indicate whether the corresponding argument is input or output):

SyntheticBookService(book, saleRank, author, birthYear, population)bffff :–
AmazonBookSearch(book,author,saleRank)bff,
WriterInfoService(author,c,birthYear,_1)bfff,
WriterInfoService(_2,c,_3,_4)bfff,
CountryInfoService(c,_5,population)bff. (Rewriting 1)

The second rewriting is derived from MCD combination {i,ii,iv,v}:

SyntheticBookService(book, saleRank, author, birthYear, population)bffff :–
AmazonBookSearch(book,author,saleRank)bff,
WriterInfoService(author,c,birthYear,_1)bfff,
CountryInfoService(c,_2,_3)bff,
CountryInfoService(c,_4,population)bff. (Rewriting 2)

There may be redundant subgoals in the rewritings obtained directly using the MiniCon algorithm. Some of these can be removed by applying the proper rules. For example, Rewritings 1 and 2 each contain one subgoal that appears twice. After removing these, we obtain two equivalent rewritings, notated as

SyntheticBookService(book, saleRank, author, birthYear, population)bffff :–
AmazonBookSearch(book,author,saleRank)bff,
WriterInfoService(author,c,birthYear,_1)bfff,
CountryInfoService(c,_2,population)bff. (Rewriting 3)

4.3 From rewritings to composition plans

The second step in Figure 1 that checks the validity of the rewriting and generates a composition plan (G, Ψ, CMP) can further be divided into two steps. The first step (lines 10–12 in Algorithm 1) checks the inputs of component services and can be realized by a topological sort of the subgoals of the rewriting based on the data flow. Moreover, from the resulting DAG G, the function Ψ can be derived. The second step (lines 13–16) checks whether the precondition of each component service, after inserting arithmetic comparison subgoals where necessary, is satisfied. The comparison subgoals in the rewriting comprise CMP directly.

In more detail, while creating the DAG in the first step, we take ordinary subgoals in the rewriting as nodes, and each shared variable between two nodes, which is either an input or output in the two services related to the nodes, as a directed edge representing the mapping of an input of the target service to the output of the source service by Ψ. In addition, what is ignored in most of the previous research on service composition is that some outputs may have been rewritten to constants and the shared variable may be an output in both services. In Datalog, these two cases mean selection and join operations, respectively. Therefore, comparisons between these variables and constants should be included in CMP.

The second step checking preconditions occurs in two places. The first occurrence, which checks non-arithmetic comparison atoms, is in the topological sort. When a directed edge is to be added , if the precondition of the target service contains atom $C(x)$, then the effect of the source service should contain the atom $C'(y)$ where y has been mapped from x; otherwise this edge cannot be added. In the motivating example, under the mapping given by Rewriting 3, the precondition atom Country(country) of CountryInfoService can be satisfied by the effect Country(country) of WriterInfoService, so the directed edge from WriterInfoService to CountryInfoService is included.

The second occurrence, which checks against arithmetic comparisons, is after the topological sort. Each component service S is checked according to formula (9) whether comparison atoms in the precondition are satisfied and new comparisons are inserted if needed. In the motivating example, the precondition of WriterInfoService requires author in [“A”, “zzz”], but in its preceding node the corresponding variable can take values from a universal set; so author$\geqslant$ “A” and author$\leqslant$ “zzz” need to be added to CMP.

4.4 From composition plan to the BPEL script

To generate a BPEL process from the plan $(G, \varPsi$, CMP) created in the previous section, we first need to generate corresponding service invocation codes for each node in G. This requires grounding information from the semantic description of the service to the syntax thereof. We use a separate XML file to record this. This file, besides specifying which WSDL operation is associated with each capability description, maps each variable in the IO lists in Definition 2.1 to a leaf node in the WSDL messages. The leaf node is located using the XPath syntax, together with a new XPath predicate “*” that marks the location of the array in the message tree if the service outputs may contain multiple records. In addition, real web services may contain certain inputs, such as user-ID, that do not appear in the semantic layer, but should be given a fixed value during invocation. Our grounding file also records these inputs and their values so that the composition result can be executed directly.

Next we need to generate BPEL control structures for edges in G. Simulating the query evaluation procedure in relational database [27], we first set up a physical operation tree to record the query evaluation plan and then traverse it to generate the BPEL process. In this binary tree each node depicts a relation upwards as the data interface and the source of its data varies across different node types. For leaf nodes it is the invocation of a related service; for inner nodes it is the result of the related relation operation: selection, Cartesian product, join, or special “sequential join”. The last operation is a new operation type introduced by us to represent the sequential execution of two services where some of the inputs of one service are mapped to outputs of the other. Projection is implicitly expressed by marking the fields unused by ancestor nodes.

A straightforward post-order traversal of the physical operation tree generates the BPEL process.

4.5 Measures to improve performance

As the two checking steps in subsection 4.3 are time-consuming, we propose two lossless pruning methods to filter out some MCD combinations or rewritings before checking them.

P1. If an MCD combination comb contains an MCD c, the related view of which contains an argument uncovered by c and comb does not contain another MCD with the same view and covering this argument, then discard comb.

P2. If rewriting rwrt contains an output of the head of rwrt or an input of a subgoal that does not appear in the outputs of its subgoals or the inputs of the head, then discard rwrt.

Besides, if the ontology used is small and the service repository is large, there may be a large number of MCD combinations, because each subgoal in the effect of the request may have many MCDs, although only a few of these will pass the inputs/precondition check. This means that the problem state space is huge, yet solutions are sparse, although they may be comparatively dense in some places. Thus checking all MCD combinations one by one is very inefficient. In most scenarios that do not require condition CR3, i.e., that require only some valid solutions rather than all of them, by adopting a random local search strategy [28] we may be able to move quickly out of a local space without solutions, thus finding some solutions in a much shorter time. In other words, whenever we check a few consecutive MCD combinations without obtaining a solution, we randomly select an MCD from each subgoal’s bucket to form an MCD combination comb, find successively the nearest combination comb$'$ to comb that forms a partition, and then begin a new round of consecutive checks from comb$'$.

5 Performance and experiments

5.1 Performance analysis

Using the proposed method, the time to find all solutions is $O((mM/k)^n(1 + n^2(\alpha p + \beta q)))$, where $(mM/k)^n$ is the number of possible MCD combinations that determines the cost of the rewriting procedure and $n^2(\alpha p + \beta q)$ reflects the time complexity for the inputs/precondition check for each rewriting. Furthermore, n and m denote the (average) number of non-arithmetic comparison clauses in the effect of the request and advertisements, respectively, M is the number of advertisements in the repository, k is the number of relations (including OWL Class and Properties) in the ontology, p is the average number of variables shared between two component services in a composition plan, q is the average number of variables involved in the arithmetic comparisons in the precondition of a service, and α and β are constant factors. Similar to the MiniCon algorithm, what makes the method seemingly inefficient is mainly that the service composition problem itself is NP-hard, where the number of solutions might be as many as $O((mM/k)^n)$.

In the procedure to check the rewritings and create a composition plan, the topological sort is $O(n^2p)$, and the precondition check thereafter is $O(n^2q)$. But pruning according to P1 and P2 may substantially reduce the cost in some situations. The cost of generating BPEL scripts, $O(n^2)$, is omitted in the above discussion, because it only occurs for composition plans selected by the user. The subClassOf/subPropertyOf reasoning costs are negligible because typically description logic reasoners would calculate the derived axioms beforehand and cache them for use at runtime, making the runtime cost only that of a few string comparisons.

Similar to other methods that attempt to solve NP-hard problems, performance of this method varies greatly depending on the characteristics of the actual problems. For example, if MCDs with a size greater than 1 are prevalent, this method will be more efficient than that described above. Therefore, it is necessary to investigate the true performance on typical problems.

5.2 Implementation and experiments

We have implemented the proposed method as a prototype, called SCQR (Service Composer based on Query Rewriting) in Java 1.5, which can generate BPEL processes (and the related WSDL and configuration files) that can be directly executed in the open source BPEL engine Apache ODE 1.3.2. Jena-2.5.4 is used as the reasoning engine for RDFS. The data types supported include string, char, floating point, and Boolean. The source code and test data used in the following sections are available at http://semwebcentral.org /projects/scqr/.

We have tested the performance of SCQR on typical problems. The desired metric is the time to find the first 20 composition plans (subsection 4.5 gives the reason for this) for each request, or to check all the rewritings if there are fewer than 20 solutions. The requests and advertisements are randomly generated by a special program and are similar to the typical scenario given in the motivating example, where each advertisement provides certain properties of a type of object and a request expects to obtain the properties of several related objects. The generated test cases are divided into two categories; one with MCDs of size 4 and the other with size 1. In each case we tested several configurations with different values of n, m, and M. In addition, the effect and precondition of an advertisement each consist of only one arithmetic comparison clause. For each configuration we generated 100 different requests to obtain an average time. The ontology contains 5 classes, 10 ObjectProperties, and 20 DataProperties, with each predicate having 0.4–0.7 descendants on average. To test the influence of the existence of the precondition in the service capability model, in most configurations we moved clauses from the precondition of advertisements into the effect and conducted the test again. In addition, the effect of pruning according to P1 and P2, and the random strategy were also tested. The testing environment consisted of a notebook with an Intel Celeron 550 CPU (2.0 GHz), 1 GB RAM, Windows XP, and Sun JVM with both the initial and maximal heap sizes set to 256 MB. The test results are shown in Figure 2, based on which the following observations are noted.

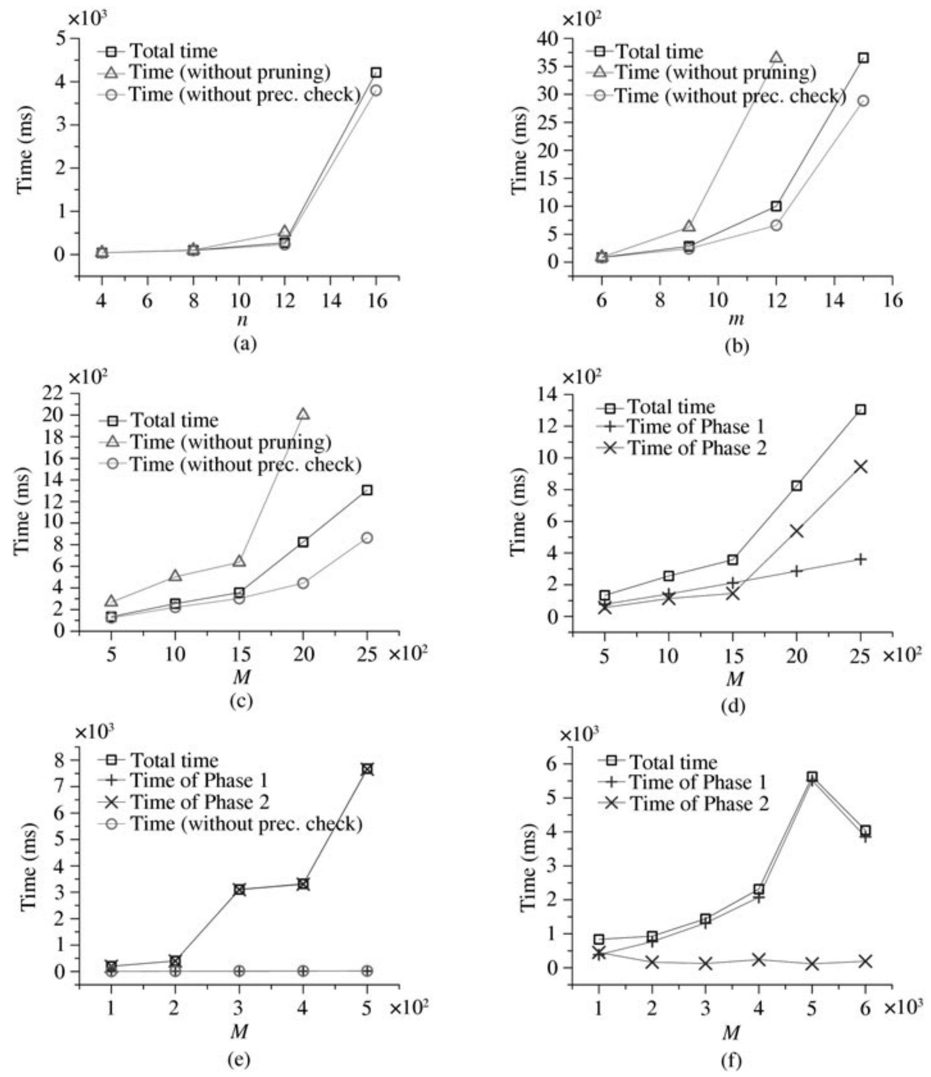

Figure 2 Performance results for SCQR. time: time to obtain first 20 solutions (or check all rewritings) for each request; n: number of non-arithmetic comparison clauses in the effect of the request; m: average number of non-arithmetic comparison clauses in the effect of each advertisement; M: number of advertisements in the repository. (a) time-n ($m = 9$, $M = 1000$; MCDSize= 4); (b) time-m ($n = 12$, $M = 1000$; MCDSize= 4); (c) time-M ($n = 12$, $m = 9$; MCDSize= 4); (d) time-M ($n = 12$, $m = 9$; MCDSize= 4); (e) time-M ($n = 3$, $m = 9$; MCDSize= 1); (f) time-M ($n = 12$, $m = 9$; MCDSize= 4; Random).

1. On problems of a real world scale, a request can be processed within 10 s, which is usually acceptable for a real development process, see Figure 2(a)–(e). In addition, Figure 2(a)–(c) show that pruning according to P1 and P2 is useful, especially for large scale problems.

2. The performance is better with larger sizes for the MCD, see Figure 2(a)–(e). Moreover, the time increases more quickly as n increases, than as m or M does, which corresponds with the theoretical analysis even though the analysis is targeted at the time to find all solutions.

3. Without incorporating the random strategy, the precondition check in phase 2 of Algorithm 1 becomes very costly when the number of MCD combinations reaches some large value and increases more quickly than that in phase 1, making it the bottleneck, see Figure 2(d)(e).

4. The performance is very sensitive to the actual features of the particular problem, such as the order of the advertisements in the repository and the predicates used in the capability model, where the higher the predicates lie in the inheritance tree, the larger the number of related MCDs. Despite each time value being the average of 100 different requests, the time increases irregularly, see Figure 2(a)–(e).

Furthermore, what is not shown in the figure is that the ratio of maximum time to average time in each configuration is usually very high; a typical value is around 10, with the highest being greater than 30. Simply changing the order of advertisements may cause a drastic change in the time actually cost .

5. The use of the random strategy in this method is effective and makes it more efficient for solving larger scale problems. In Figure 2(f), the time for phase 2 is negligible. This means that if a threshold is set for the bucket size in phase 1, the performance can be further improved. In Figure 2(f), the search step threshold is 50000 (i.e., the check is carried out on at most this many MCD combinations), while the consecutive search step threshold for each randomly generated MCD combination is 45. That the time at M =1000 is comparatively large is because there are two requests here, each of which has fewer than 20 solutions (0 and 16, respectively). As a result, the system needs to continue searching until the threshold is reached. This reflects a feature of the random strategy that it is inefficient when solutions are too sparse.

6 Discussion

The query rewriting-based service composition method differs in principle significantly from most traditional ones. Most traditional methods, including most AI planning systems and resolution principle-based theorem proving systems, determine component services by searching directly in the problem state space either forwards or backwards. In a composition plan, the selection of the next component service is affected by previous ones. In other words, they first set up a variable mapping according to the types of inputs/outputs and then check whether predicates in the PE match. In contrast, the query rewriting-based method first matches predicates and then sets up the variable mapping, which leads to the component services of a plan being selected simultaneously. What makes this feasible is the unique feature of information-providing service of never modifying the world state, which ensures that the PE of a service still hold after execution of subsequent services.

Accordingly, this method also has a different performance characteristic to traditional search-based services, which makes it particularly suited to the needs of information-providing services. As mentioned previously, it is common for information-providing types of services to have no ontology classes as semantic types for IO parameters. This has little influence on the proposed method; for search-based methods, however, the number of expanded nodes in each step increases drastically because of the loss of an important filter, which results in a significant increase in the time complexity. On the other hand, as the number of clauses in the service precondition increases, search-based methods may become faster since the precondition acts as a filter here, whereas the query rewriting-based method, which takes preconditions into consideration in the last step, will become slower. In addition, because of the characteristic of the MiniCon algorithm, the proposed method is more efficient with larger MCDs, which means that the request and advertisements both contain comparatively more local variables (as is the case in the request and first two advertisements in the motivating example).

One drawback of our method is that it is not complete even for problems complying with the restriction in subsection 4.1. Besides the reasons mentioned in subsection 4.1, there are two other reasons for this. First, the MiniCon algorithm itself is not complete in CQAC for problems where homomorphism does not hold [25]. Fortunately this kind of problem seldom appears in reality, because there are at least two variables with "similar status" in the capability description of an advertisement (see [25] for details). Second, MiniCon is limited when used in views with binding patterns. The number of subgoals of the rewriting obtained by this algorithm is never greater than that of the query. When a binding pattern exists, even though the query and views are both in CQ, the size of the rewriting may exceed that of the query, and may even be infinitely large [23,29]. However, in this case although search-based methods can obtain more solutions than our method, they still cannot find all the solutions in a finite time.

Another drawback of our method is that the reasoning capability it supports is very limited. Because MiniCon is based on a one-to-one comparison among ordinary subgoals, this method can only support the subClassOf/subPropertyOf reasoning of predefined Classes/Property names and cannot support various reasonings among complex Class/Property descriptions.

7 Related research

[4] presents a capability model for stateless information-providing services that emphasize using the relations among the IO parameters expressed in OWL formulas as part of the model. It also reduces the matching problem between two services to a subsumption problem of conjunctive queries w.r.t. description logic (DL) TBox. This model does not, however, support arithmetic comparisons and does not discriminate preconditions from effects. It requires IO parameters to have classes from the ontology as semantics, which is too rigid. It defines the service composition problem, but the definition is too simple and no service composition method is presented.

Of the previously mentioned automatic composition approaches, [12,13,19,20] are closest to ours since all of these are targeted at information-providing services and do not require IO parameters to be annotated with classes. In [12] the constraints on inputs and relations among IOs in RDF graph patterns are expressed in, respectively, the input message pattern and the output message pattern, corresponding to our precondition and effect. In [13], the relations among parameters are expressed in a DL language Power Loom obtained by expanding Loom, "slim services" are inserted when parameters do not match only because of the unit and type, and join operations are supported as in our model. Both these two approaches are based on direct searching and the data schemas they use only support unary and binary predicates, which is a limitation that does not exist in our method. All of the methods in [19, 20, 30–32] explicitly regard services as queries and transform the service composition problem into a query rewriting problem. In [19], an "enumerate-combine" procedure is employed as we do, but the manner in which a query is divided into blocks is less flexible than the MiniCon algorithm, which means that if some data sources store different properties of the same type of objects, certain rewritings, that need some of these data sources to be joined to satisfy a block, will be overlooked. [20] uses another rewriting algorithm, Inverse-Rule, and is able to insert "tuple level filters" in the composition plans according to arithmetic comparisons in data sources to avoid unnecessary querying at runtime. Our method, utilizing the IOPE-based capability model, can also realize most functions of these filters as early as at composition time. The rewriting procedure in [20] does not consider semantic reasoning and the Inverse-Rule algorithm is less efficient than MiniCon [24]. In [30,32], rewriting is carried out in the context of the Semantic Web as in our method. [30] supports basic DL reasonings but the query definition it uses to express a service contains only Class predicates without Property predicates, which simplifies the realization of the rewriting algorithm, but severely limits its expressive power. As a result, the description of the service capability is poor and not precise. In [32], services are also regarded as RDF views on an ontology, while [31] discusses how to implement such services on the DB2 database platform. Neither, however, presents details of the composition method, especially the treatment of inputs. Finally, all these studies, except for [12], do not differentiate the precondition from the effect, and thus they differ significantly from the prevailing capability model of Semantic web services.

In addition, various database research targeted at data integration w.r.t. views with binding patterns closely resemble our work. [26] designs a "query planning system" Occam that performs a forward search in the state space based on argument types. In [29], it is proved that when views have binding patterns, methods exist to find the maximally-contained rewriting if Datalog-style recursion is allowed in the result plan. [33] implements such a system, Razor, in which the view definition is in conjunctive queries and the arithmetic comparisons are regarded as the views' guarantee about local completeness of the data it stores. With this assumption it introduces the union operation in the result plan and achieves data completeness as defined in CR3. Despite having appeared prior to web services, these studies are valuable as a reference for automatic composition research on information-providing services. However, none of these methods differentiate the precondition from the effect or support semantic reasoning.

8 Conclusion

The capability model proposed in this paper can flexibly express the semantics of IO. It is simple yet able to depict rigidly the have-no-negative-effect feature of this type of service and, is therefore useful while deciding the correctness of a composition plan. The proposed MiniCon query rewriting algorithm-based automatic composition method for information-providing services utilizes adequately the features of this type of service. It uses an "enumerate-combine" procedure that first matches predicates and then maps variables, and is complementary in performance to traditional search-based methods. The performance test on a typical kind of problem shows that this method satisfies the needs of various practical applications. Future work includes supporting arithmetic comparisons between variables, introducing union nodes to get as much data as possible through a single composition plan (as in [33]) and introducing conditional selection nodes (as in [11]).

Acknowledgements

This work was supported by National Natural Science Foundation of China (Grant No. 60432010), National Grand Fundamental Research Program of China (Grant Nos. 2011CB302506, 2007CB307103). We wish to thank Dr. Rachel Pottinger at the University of British Columbia for her email correspondence that enabled us to quickly implement the MiniCon algorithm.

References

1 McIlraith S, Son T C, Zeng H. Semantic web services. IEEE Intell Syst, 2001, 16: 46–53

2 Wu D, Parsia B, Sirin E, et al. Automating DAML-S web services composition using SHOP2. In: Fensel D, Sycara K, Mylopoulos J, eds. Proceedings of the 2nd International Semantic Web Conference (ISWC 2003). LNCS, 2870. Berlin/Heidelberg: Springer-Verlag, 2003. 195–210

3 Martin D, Paolucci M, McIlraith S, et al. Bringing semantics to web services: the OWL-S approach. In: Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004). LNCS, 3387. Berlin/Heidelberg: Springer-Verlag, 2004. 26–42

4 Hull D, Zolin E, Bovykin A, et al. Deciding semantic matching of stateless services. In: Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06). California: The AAAI Press, 2006. 1319–1324

5 Berardi D, Calvanese D, Giacomo G D, et al. Automatic service composition based on behavioral descriptions. J Coop Inf Syst, 2005, 14: 333–376

6 Bultan T, Fu X, Hull R, et al. Conversation specification: a new approach to design and analysis of e-service composition. In: Proceedings of the 12th International Conference on World Wide Web (WWW 2003). New York: ACM Press, 2003. 403–410

7 Bertoli P, Hoffmann J, Lecue F, et al. Integrating discovery and automated composition: from semantic requirements to executable code. In: Proceedings of 2007 IEEE International Conference on Web Services (ICWS 2007). Los Alamitos: IEEE Computer Society, 2007. 815–822

8 Zhang R, Arpinar I B, Aleman-Meza B. Automatic composition of semantic web services. In: Proceedings of 2003 IEEE International Conference on Web Services (ICWS 2003). Los Alamitos: IEEE Computer Society, 2003. 38–41

9 Gu Z F, Li J Z, Xu B. Automatic service composition based on enhanced service dependency graph. In: Proceedings of 2008 IEEE International Conference on Web Services (ICWS 2008). Los Alamitos: IEEE Computer Society, 2008. 246–253

10 Lecue F, Leger A. A formal model for semantic web service composition. In: Cruz I, Decker S, Allemang D, et al., eds. Proceedings of the 5th International Semantic Web Conference (ISWC 2006). LNCS, 4273. Berlin/Heidelberg: Springer-Verlag, 2006. 385–398

11 Kona S, Bansal A, Blake M B, et al. Generalized semantics-based service composition. In: Proceedings of 2008 IEEE International Conference on Web Services (ICWS 2008). Los Alamitos: IEEE Computer Society, 2008. 219–227

12 Liu Z, Ranganathan A, Riabov A. Modeling web services using semantic graph transformations to aid automatic composition. In: Proceedings of 2007 IEEE International Conference on Web Services (ICWS 2007). Los Alamitos: IEEE Computer Society, 2007. 78–85

13 Ambite J L, Kapoor D. Automatically composing data workflows with relational descriptions and shim services. In: Proceedings of the 6th International Semantic Web Conference (ISWC 2007). LNCS, 4825. Berlin/Heidelberg: Springer-Verlag, 2007. 15–29

14 McDermott D V. Estimated-regression planning for interactions with web services. In: Ghallab M, Hertzberg J, Traverso P, eds. Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems (AIPS'02). Menlo Park: AAAI Press, 2002. 204–211

15 Klusch M, Gerber A. Semantic web service composition planning with OWLS-XPlan. In: Proceedings of AAAI-05 Fall

Symposium on Agents and the Semantic Web. Menlo Park: AAAI Press, 2005. 55–62

16 Rao J. Semantic web service composition via logic-based program synthesis. Dissertation for the Doctoral Degree. Trondheim: Norwegian University of Science and Technology, 2004. 32–39

17 Dong J, Sun Y T, Yang S, et al. Dynamic web service composition based on OWL-S. Sci China Ser F-Inf Sci, 2009, 49: 843–863

18 Narayanan S, McIlraith S A. Simulation, verification and automated composition of web services. In: Hencsey G, White B, eds. Proceedings of the 12th International Conference on World Wide Web (WWW 2003). New York: ACM Press, 2003. 77–88

19 Zhou L H, Chen H J, Mao Y X. Data service composition approach based on query rewriting. Comp Integ Manuf Sys (in Chinese), 2009, 15: 823–832

20 Thakkar S, Ambite J L, Knoblock C A. A data integration approach to automatically composing and optimizing web services. In: Proceedings of 2004 ICAPS Workshop on Planning and Scheduling for Web and Grid Services. Menlo Park: AAAI Press, 2004. 86–93

21 Zhao W F, Meng X W, Chen J L, et al. Integrating information-providing web services into the data integration system. In: Proceedings of the 2008 IEEE International Conference on Web Services (ICWS 2008). Los Alamitos: IEEE Computer Society, 2008. 801–802

22 Lenzerini M. Data integration: a theoretical perspective. In: Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. New York: ACM Press, 2002. 233–246

23 Halevy A Y. Answering queries using views: a survey. Int J Very Large Data Bases, 2001, 10: 270–294

24 Pottinger R, Halevy A. MiniCon: a scalable algorithm for answering queries using views. Int J Very Large Data Bases, 2001, 10: 182–198

25 Afrati F, Li C, Mitra P. Rewriting queries using views in the presence of arithmetic comparisons. Theor Comput Sci, 2006, 368: 88–123

26 Kwok C T, Weld D S. Planning to gather information. In: Proceedings of AAAI 13th National Conference on Artificial Intelligence. Menlo Park: AAAI Press, 1996. 32–39

27 Garcia-Molina H, Ullman J D, Widom J. Database System Implementation. New Jersey: Prentice Hall, 1999. 329–422

28 Russell S J, Norvig P. Artificial Intelligence: a Modern Approach. 2nd ed. New Jersey: Prentice Hall, 2002. 110–119

29 Duschka O M, Levy A Y. Recursive plans for information gathering. In: Pollack M, ed. Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97). San Francisco: Morgan Kauffman Publishers, 1997. 778–784

30 Bao S H, Zhang L, Lin C X, et al. A semantic rewriting approach to automatic information providing web service composition. In: Proceedings of 1st Asian Semantic Web Conference (ASWC 2006). LNCS, 4185. Berlin/Heidelberg: Springer-Verlag, 2006. 488–500

31 Lu J, Yu Y, Mylopoulos J. A lightweight approach to semantic web service synthesis. In: International Workshop on Challenges in Web Information Retrieval and Integration at ICDE. Washington DC: IEEE Computer Society, 2005. 240–247

32 Barhamgi M, Benslimane D, Ouksel A M. Composing and optimizing data providing web services. In: Proceedings of the 17th International Conference on World Wide Web (WWW 2008). New York: ACM Press, 2008. 1141–1142

33 Friedman M, Weld D. Efficiently executing information-gathering plans. In: Pollack M, ed. Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97). San Francisco: Morgan Kauffman Publishers, 1997. 785–791