Data-Driven Service Composition in Enterprise SOA Solutions: A Petri Net Approach
Wei Tan, Yushun Fan, MengChu Zhou, Fellow, IEEE, and Zhong Tian

Abstract—Under Service Oriented Architecture (SOA), service composition is used to integrate service components together to meet new business needs. In this paper, we propose a novel data-driven method to provide service composition guidance to implement given requirements. Based on the relations between business domain data and service domain data, we generate additional data mediations according to three composition rules. With these data relations and composition rules, we propose a Petri-net based approach to the composition of services. In our approach, all the in/output messages of the service operations are modeled as colored places, and service operations themselves are modeled as transitions with input/output places. We first generate a Service Net (SN) that contains all operations in a given service portfolio, and then use Petri-net decomposition techniques to derive a subnet of SN, and this subnet meets the need of the business requirement. Our work can be seen as an effort to bridge the gap between business and service domains.

Note to Practitioners—Web services composition is an emerging area for business process automation. This work presents a novel framework to compose web services from the perspective of data. It is based on colored Petri nets and a newly proposed concept called Service Net. The proposed method represents both data relations and service composition rules with colored Petri nets. If a business requirement is given with input/output data, we convert the Service Net into a reduced one, and decompose it into subnets that can be candidate composition solutions. A real-life case is used to illustrate the feasibility of the proposed concepts and method. Our method can be readily used in industrial web service composition for business automation.

Index Terms—Colored Petri nets, data-driven, service net (SN), web service composition.

I. INTRODUCTION

Web service composition is considered to be a key factor in stimulating the web evolving from an information-delivering platform to better support automated use. It is of great importance in the emerging paradigm of Service Oriented Architecture (SOA) [1], because reusability is a key issue in SOA to ensure scalability and productivity. When new business requirements emerge, solution designers should devise a composite process that makes the best use of existing services, and SOA provides a way to glue all components together with least augmentation or modification. Service composition techniques provide ways to devise a composition of services, which fulfills the requirement. There are a number of studies on automatic web service composition, but there is not much on the linkage between requirements and services.

In this paper, we propose a data-based approach to provide guidance for service (or process) composition with a given service portfolio in mind. The data relations between business domain and service domain are explored and data mediation constructs are added to bridge the existing artifacts. We use colored Petri nets as a formalism to represent the data relations. Based on this formalism, three composition rules, i.e., sequential, parallel, and choice composition rules, are proposed. Then, we present a formal method to derive service compositions that satisfy the requirement with respect to the input/output data types. This method is based on a Petri net decomposition technique. A prototype system is developed to test the validity of the proposed approach. To the best of our knowledge, we for the first time present a Petri-net decomposition based approach and integrated system that utilize the data relation in business/service portfolio to derive composite services.

II. RELATED WORK

Current research on service composition methods can be divided into two categories, i.e., top-down and bottom-up. The former focuses on building new solutions from scratch [2], [3]. It addresses well the issue of how to reflect the requirements and to refine them into finer grain of abstraction such that programmers can take over to implement. It ignores the fact that enterprise solutions are built on top of an existing IT infrastructure. Designers need extra help in putting the solution design in the context of current IT infrastructure, including the current service portfolio. Bottom up approaches mainly focus on the automatic service composition methods. They can further be classified into logic [4]–[14], optimization [15]–[19], and automata-based approaches [20]–[22].

A logic-based approach assumes that service providers give semantic descriptions of the service, e.g., input, output, precondition, and effect. The survey in [23] describes the problem of automatic service composition as a five tuple \((S, s_0, G, a, \Gamma)\) in which \(S\) is the set of all possible system states, \(s_0\) and \(G\) are the initiate and goal state regarding the service composition. \(a\) is the set of all possible actions (i.e., all the service operations), \(\Gamma = S \times A \times S\) defines the precondition and effect for each action. Then, if we start from the initiate (or goal) state, with all the possible actions in \(\Gamma\), various AI-planning methods can be used to derive a feasible plan to the goal (or initiate) state. Researchers usually use ontology modeling language like OWL-S [24] to add semantic annotations on web service description. Then, situation calculus [4], [9], PDDL [5], rule-based approach like SWORD [10] and HTN [6] are used to derive feasible solution(s). A Petri net-based planning framework is proposed in [9]. This framework is based on state-space analysis.

Optimization-based approaches are used to derive composite services that achieve the best quality-of-service (QoS). Usually, the QoS constraints include local constraints (constraints imposed on a single service) and global ones (constraints imposed on multiple services), and therefore two selection approaches are used to solve these two kinds of constraints, respectively [19]. Optimization methods like linear programming [15], [17] and genetic algorithm [16] are used to solve the QoS constraints and yield feasible and optimal solution(s).

Manuscript received February 04, 2009; revised June 04, 2009; accepted September 01, 2009. Date of publication November 24, 2009; date of current version July 02, 2010. This paper was recommended for publication by the reviewers’ comments. This work was supported in part by an IBM Ph.D. Fellowship, in part by the National Science Foundation of China (60674080), in part by the National 863 Program of China under Contract 2007AA04Z150 and Contract 2008AA04Z109, in part by Ministry of Education of China under Changjiang Scholars Program, and in part by the National Basic Research Program of China (2006CB705407).
An automata-based approach uses automata to describe the behavior of target composite service as well as participant services. The key of this approach is computing a delegator that coordinates the activities from those participant services such that the overall behavior corresponds to the composite service. If a delegator exists, the behavior of the target composite service can be achieved by synthesizing the participant services and delegating their individual behaviors [20]–[22].

However, current solutions have limitations when they are used in real business scenarios. First, most of them assume a coherent representation of requirement (target service) and participant services. Few consider the gap between business and service domains. As we can see in Section III, business service domains usually have different domain models with different aspects of concern. This gap hampers the effective use of the logic, optimization and automata-based approaches. Second, many approaches use ontology to describe the behavior of services and related data. However, nowadays in most enterprises, semantic information of their web services is still missing. The logic-based approach needs much additional effort in adding semantic tags on top of the current web service standard stack.

On the one hand, current solutions need a more ideal usage scenario than the one in reality. On the other hand, service portfolio contains abundant data to be investigated and used. We believe that from these data, much guidance can be derived to help service composition. We also believe that our data-driven solution is a lightweight approach that can be used in conjunction with other approaches.

III. PROBLEM STATEMENT

In software tools such as IBM WebSphere Business Modeler [25] and WebSphere Integration Developer [26], composite services can be directly derived from business processes. It seems that business requirements represented by business processes can be directly transformed to refined processes realized by available services. In practice, however, some gaps remain.

First, the model elements of a business process and a service composition are not identical. In practice, there are various kinds of specifications, languages and notations to model a business process and a service composition [17], [18], [21], [23], [27], [33], [34]. Second, the data model in business and service domains are heterogeneous. Therefore, direct or indirect mappings between them are required to give an integrated and coherent view of the two domains. The third reason is more critical. Without proper guidance from the service domain, there is no guarantee that the refined process model can be implemented by available services. Hence, it would be the best if we can generate certain operational guidance from the existing service portfolio to help refine a business domain process into a customized process that can be implemented by existing services to a greater extent.

On the one hand, the difficulties mentioned above hamper the effective and efficient utilization of available services. On the other hand, service portfolio contains abundant information to be further investigated. For example, WSDL (Web Services Description Language) [28] files contain the input/output data type of operations, and WSDL files with associated data definition schemas contain the relations among these data types, such as aggregation and generalization. From these data relations, guidance can be derived to help service composition.

A. Domains and Data Relations

Various languages exist for modeling, and we need a common one to represent the artifacts. Colored Petri nets [29] capture both control and data aspects of a process. In this paper, we use them as a common model for both business processes as well as services. Due to space limitation, we omit the introduction of colored Petri nets. One can refer to [29], [35] for more information.

There are two domains involved in the scenario of service composition, i.e., business and service. Business domain is also known as the requirement specification domain in which requirements are represented by business processes consisting of abstract activities with input/output data. Service domain is also known as the implementation domain in which a service is modeled by a set of operations with input/output data modeled as messages in WSDL.

We borrow the ideas from UML class diagram and type definition in XML schema; here, we concern two kinds of relations, i.e., in-domain relation and cross-domain relation. From now on, the data types in business domain are denoted with uppercase strings, while those in service domain with lowercase strings.

Before we start to define data relations, we would like to emphasize the difference between data and data type. A data type defines a set of values that a data element can take. Since this paper is about service composition at build-time (instead of at runtime), we focus on “data type” instead of “data.” Therefore, except those otherwise noted, the term data also refers to data type throughout this paper. For example, when we talk about “data relations” we mean the relations between two data types.

1) In-Domain Relations:
   1) Aggregation: has-a relation.
   2) Generalization: is-a relation.
   3) Generation: the relation between input and output data types of a business activity or service operation is defined as generation.

Let us explain with a real life example, i.e., an ADSL order processing service (AOPS) provided by a telecommunication company. This example is taken from a real customer case in IBM SOA solution in telecom industry. In Fig. 1, the upper part is a segment extracted from WSDL file. Operation generate_worksheet receives ADSL application order (data type order) and generates a worksheet (data type worksheet). Data type worksheet represents the work-items to fulfill the order, and it aggregates inhouse_wss that represents the work-items undertaken in a customer’s house. Data type order is the generalization of nl_order that represents new-telephone-line-plus-ADSL business.

With this information the data structure in service domain is derived and illustrated in the lower part of Fig. 1. In a business domain, there are two data types ORDER and WORKSHEET, a business requirement is expressed as ORDER generates WORKSHEET (ORDER → WORKSHEET).

2) Cross-Domain Relation: Since we intend to make the best of existing services to meet the needs of business requirements, ideally data types in service domain should be the implementation of those in business domain. We note that realization is defined as a primitive in UML.
2.0 specification, and it signifies the specification-implementation relation between two model elements. Thus here we use realization as a cross-domain relation to signify that one data type in a service domain directly implements one data type in a business domain. For example, in Fig. 1, data type order in the service domain is the realization of data type ORDER in the business domain. In the following part of this paper, a pair of data types, one’s name in lower case and the other in upper case, have a realization relation between them.

Given that the granularity of business data is usually coarser than that of service data, a complete realization relation between them is not guaranteed. In order to model an incomplete one, we use two stereotypes to extend the realization relation, i.e., partial and specialized.

1) Partial realization describes the case that service data realizes a part of business data. For example, in Fig. 1, inhouse_ws is a part of worksheet, which realizes WORKSHEET. Thus, we define inhouse_ws as the partial realization of WORKSHEET.

2) Specialized one describes the case that service data realizes a special kind of business data. For example, in Fig. 1, nl_order is a special class of order, which realizes ORDER. Thus, we define nl_order as the specialized realization of ORDER.

Fig. 1 illustrates all the data relations we have discussed and gives the notations to model these relations. For simplicity, we only present the UML model while actual data definition in XML is omitted.

B. Problem Formulation

In Section III-A, we use UML notation to express data relations. Due to space limit, we show only the graph presentation of UML class diagrams and in this section, we will show their corresponding Petri-net constructs. We believe that this formalism is sufficient for the completeness of this paper and a more formal presentation of the type definition for colored Petri nets can be found in [29].

To align with the requirement of process composition, we use colored Petri nets to represent all the data relations in service domain, as illustrated in Fig. 2. The notation of a place (for example, order) represents the color set attached to it. The color set, in turn, represents a data type associated with this place. A hollow transition represents the generation relation between two data types (order can generate worksheet); a solid one represents the generalization/aggregation relations (worksheet is the aggregation of inhouse_ws and line_ws; and order is the generalization of addl_order and nl_order). For simplicity, we omit the arc expressions that represent the token consuming relation when a connecting transition fires. In our model, each arc will either consume exactly one token of the input place’s data type or yield one token of the output place’s data type, when a connecting transition fires.

To address the challenge of service composition from the perspective of data, first, we define the ideal property that a composite service should satisfy. Since a composite service is represented by a colored Petri net, we define data coherency on a colored Petri net.

Definition 1: [Data Coherency] A colored Petri net \( N \) is data coherent w.r.t. given business/service portfolio, if every transition in \( N \) either represents an operation in service portfolio, and transforms an input data type into an output data type; or represents the aggregation or generalization relationship among data types.

Remark: \( N \) is data coherent if every transition \( t \) in \( N \) represents a kind of data relations, as shown in Fig. 2.

Based on the formalization of process and data, we have the following problem statement (Fig. 3).

Requirement is expressed as a colored Petri net with a single transition \( t \), the data type of the input and output places of \( t \) is denoted as \( I \) and \( O \), respectively. So the requirement can be simply expressed as \( I \rightarrow O \). Operations in service portfolio can also be expressed as \( i \rightarrow a \), i.e., the operation consumes one instance of data type \( i \) and yields one instance of data type \( o \).

Our goal is to find a collection of operations in a large service portfolio. These operations connect to form a colored Petri net \( N \). As shown in Fig. 3, data type attached to one place is denoted as an underlined string.

\( N \) is a valid composition of the requirement \( I \rightarrow O \) if:
1) \( N \) takes \( i \) as its only input and \( o \) as its only output.
2) \( N \) is data coherent.
3) \( N \) has some additional properties, to be stated in Section V-B.

Before addressing the problem of service composition, we introduce in Section IV data-driven composition rules used to refine processes in a business domain into processes in a service domain.

IV. DATA-DRIVEN COMPOSITION RULES

The composition and transformation rules in this paper are all expressed with graph transformation formalism. It is intuitive enough to understand, and more details on graph transformation can be found in [30]. The sequential, parallel and choice composition rules correspond to the sequential, parallel, and conditional routings given in Workflow Net [31].

1) Sequential Composition Rule: The sequential composition rule is illustrated in Fig. 4. For a business requirement \( A \rightarrow C \), if there are two operations in service portfolio, \( a \rightarrow b \) and \( b \rightarrow c \), we can refine the business requirement into a process realized by existing services, i.e., \( a \rightarrow b \rightarrow c \).

2) Parallel Composition Rule: We explain this rule through an example in Fig. 5. If \( a = a_1 \times a_2, b = b_1 \times b_2 \), and we have two operations \( \{a_1 \rightarrow b_1, a_2 \rightarrow b_2\} \), the requirement \( A \rightarrow B \) can be realized by the AND collection of \( \{a_1 \rightarrow b_1, a_2 \rightarrow b_2\} \), with two additional mediation transitions as represented by black rectangles. With mediation transitions, \( a \) is decomposed to \( a_1 \) and \( a_2 \), and \( b \) to \( b_1 \) and \( b_2 \).

3) Choice Composition Rule: This rule is related to data generalization as explained via Fig. 6. In a business domain, we have a requirement \( A \rightarrow B \), while in a service domain, we have two operations \( \{a_1 \rightarrow b, a_2 \rightarrow b\} \) \( \{a = a_1 \cup a_2, a_1 \cap a_2 = \emptyset\} \). Then, requirement \( A \rightarrow B \) can be realized by the XOR (exclusive OR) collection of \( \{a_1 \rightarrow b, a_2 \rightarrow b\} \), with two additional mediation transitions as
represented by black rectangles. With mediation transitions, data type

\( a \) is specified to either \( a_1 \) or \( a_2 \).

In the composition rules we illustrate in this section, there are direct mappings between business data and service data. Nevertheless, usually there is no existing service data that realizes some business data. Hence, we need to add newly created service data to make the data structure coherent. These newly created service data can be regarded as virtual data type to facilitate service composition. For example, in Fig. 5, if there is no service data type \( a(b) \) that directly realizes \( A(B) \), we should create virtual data type \( a \) and \( b \) from the partial realization data types of \( A \) and \( B \), s.t. \( a = a_1 \times a_2, b = b_1 \times b_2 \), then \( A \rightarrow B \) is refined to the AND collection of \( \{a_1 \rightarrow b_1, a_2 \rightarrow b_2\} \). Another circumstance is that, if \( a, b, a_1 \) and \( b_1 \) exist, but \( a_2 \) and \( b_2 \) do not exist in service portfolio, we should manually add \( a_2 \) and \( b_2 \) as virtual data types in the service domain to indicate that there is a missing operation \( a_2 \rightarrow b_2 \) in service portfolio to fulfill the requirement \( a \rightarrow b \). We believe that a virtual data type can be a powerful tool to better glue together business data and service data, and further glue together business requirement and service operations. In this paper, we concentrate on the approach and algorithm for data-driven service composition.

V. DATA-DRIVEN SERVICE COMPOSITION

Based on the data relations and composition rules, we are now ready to solve the problem formulated in Section III-B. That is, if a business requirement is represented as \( I \rightarrow O \), find a service composition from service portfolio to realize it.

A. Preliminary Definitions

Some definitions are given as follows.

**Definition 2:** [Acyclic Well Structured Process (AWSP)] An Acyclic Well Structured Process is defined as follows.

1) A transition with one single input place and one single output place is an AWSP.
2) All Petri nets obtained by using transformation rules 1–3 in Fig. 7 are AWSPs.

Based on the composition rules in Section IV, we give the definition of a Service Net and the method to derive it given a service portfolio.

**Definition 3:** [Service Net (SN)] A Service Net \( N_s \) with respect to a service portfolio and data mediation is a colored Petri net \( (P, T, A, \Sigma, C) \), where

1) \( \Sigma \) is a finite set of data types modeled as color sets;
2) \( P \) is a finite set of places;
3) \( T \) is a finite set of transitions, \( T = T_O \cup T_M \) where \( T_O \) is the set of operation transitions, and \( T_M \) is the set of mediation ones;
4) \( A \) is a finite set of arcs;
5) \( C \) is a color function defined from \( P \) into \( \Sigma \). \( C \) is injective, i.e.,

\[ C(p_1) = C(p_2) \Rightarrow p_1 = p_2. \]

An SN can be constructed in the following way.

1) Data relations and mediations are modeled with mediation transitions.
2) Service operations are modeled with operation transitions.
3) Places attached with identical data type are merged into a single one.

Note that \( N_s \) is data coherent since transitions represent data mediation or transformation. Any subnet of \( N_s \) also data coherent. An SN after any refinement process in Figs. 4–6 is still so.

**Definition 4:** [Conflict place] \( P_C \subseteq P \) in \( N_s \equiv (P, T_O \cup T_M, F) \) is a set of conflict places if \( \forall p \in P_C, |p^*| > 1 \) and \( p^* \cap T_O \neq \emptyset \).

A service net is conflict-free iff it does not contain any conflict places, i.e., \( P_C = \emptyset \).

**Definition 5:** [Connectivity between two nodes] In a Petri net \( (P, T, F) \), given \( n_1, n_j \in P \cup T \), connectivity relation \( B(n_1, n_j) = \text{true} \) iff \( \exists n_1, n_2, \ldots, n_j \in P \cup T \) such that \( \{n_i, n_{i+1}\} \in F \) for \( 1 \leq i \leq j-1, \) and \( i \neq k \Rightarrow n_i \neq n_k \) for \( 1 \leq i, k \leq j \).

B. Derive AWSP From Service Net

With the definitions given in Section V-A, the problem raised in Section III-B can be interpreted more formally.
Definition 6: [Feasible solution] Given a service portfolio and service composition requirement \( I \rightarrow O \), \( N \) is a feasible solution if:
1) \( N \) takes \( i \) as its only input and \( o \) as its only output;
2) \( N \) is data coherent w.r.t. the given service portfolio; and
3) \( N \) is an AWSP, and is conflict-free.

Remarks:
Why are these requirements imposed on a feasible solution?
1) The solution should consume data \( i \) and yield data \( o \), given the requirement \( I \rightarrow O \);
2) Data coherent: because we are addressing the service composition problem from the perspective of data, the composite process should represent the data relations in a service portfolio;
3) Well-structuredness and conflict-freeness are favorable properties of business processes. We also require the solution to be acyclic because a cyclic structure represents iterative processing of the same type of data. Therefore, for computational simplicity, we generate acyclic solutions, but in an execution phase the process can be cyclic.

We have the following problem formulation and steps to generate a feasible solution.

Input: \( I \rightarrow O \).

Output: An AWSP, data coherent \( N \), which is a sub graph of service net \( N_S \), with \( i \) as its input and \( o \) as its output, and contains no conflict place.

Step 1. Construct a service net \( N_S \) given a service portfolio and data mediations in business/service domain, as given in Definition 3;

Step 2. Derive a Reduced Service Net \( N_R \) from \( N_S \);

Step 3. Decompose \( N_R \) into subnets, each of which contains no conflict place; and

Step 4. Check each of the decomposed nets derived from Step 3 to decide whether it is a feasible solution.

These steps are further explained as follows.

Definition 7: [Reduced Service Net (RSN)] A reduced service net \( N_R = (P_R, T_R, F_R) \) of service net \( N_S = (P, T, F) \) with respect to input data type \( i \) and output data type \( o \), or \( N_R(N_S, i, o) \), is a sub graph of \( N_S \), s.t., \( \forall n \in P \cup T, n \in P_R \cup T_R \) if in \( N_S, B(i, n) = B(n, o) = true \).

Remark: when \( N_S \) is reduced to \( N_R \), any operations, data mediations and data types that are not on a path from data type \( i \) to data type \( o \) are eliminated. We reduce an \( N \) in order to remove the unrelated parts, and the reduction process can be done by slightly modifying the graph traverse algorithm.

The branches after each conflict place represent options from which we can choose for data processing. A snippet of RSN is shown in Fig. 8. Place \( P_1 \) is a conflict place. Hence, data type \( a \) attached to \( P_1 \) can be processed by operation \( f_1 \), or \( f_2 \), or the XOR-join of operations \( \{ f_{31}, f_{32} \} \).

An RSN must be decomposed into a set of subnets that do not contain any conflict places. Simply speaking, each time a conflict place is encountered, one branch is selected as the active one, and the other branches are removed. As Fig. 8 illustrates, three subnets are generated because \( P_1 \) has three succeeding transitions \( (t_{31}, t_{32}) \) and treated as one single branch).

When there are more than one conflict place in an RSN, the decomposition is more complicated. If there are \( n \) conflict places each with \( m \) branches, possibly we have \( m^n \) distinctive subnets. We use selection value to decide which branch to choose for each conflict place. One selection value corresponds to one situation of decomposition, and we use decomposition algorithm to get one subnet for each valuation. Now, we give function definitions used in a decomposition algorithm.

Given a reduced service net
\[ N_R(N_S, i, o) = (P_R, T_R, F_R) \]

1. \( \forall \ p \in P_R, p \) is a source place iff \( p \not\in P_R \).

2. \( \forall \ p \in P_R, \) function \( S(p) \) is a Boolean function.

3. \( \forall \ p \in P_R, \) function \( D_i(p) \) is a Boolean function.

4. \( \forall \ p \in P_R, \) returns true iff \( p \) is a source place and is explicitly marked to be dead.

Given a net \( N = (P, T, F), p \in P \), for \( t \in T, p \in T \)

Function \( M(p, t, p', N) \) modifies the structure of \( N \) by redirecting \( (p, t) \) to \((p', t)\), that is:
\[ P = P \cup \{ p' \}; \quad F = F - \{(p, t)\} \cup \{(p', t)\} \]

Given a net \( N = (P, T, F), n \in P \cup T \)

Function \( D(n, N) \) modifies the structure of \( N \) by deleting \( n \) and the arcs leading and ending at \( n \), that is,
\[ P = P - \{ n \}; \quad F = F - \{(n, x) \in T \land (n, x) \in F \}
\]

\[ \cup \{(y, n) \in T \land (y, n) \in F \} \]

If \( n \in T \)
\[ F = F - \{(n, x) \in P \land (n, x) \in F \}
\]

\[ \cup \{(n, y) \in P \land (n, y) \in F \} \]

If \( N' \subseteq P \cup T \), \( D(N', N) \) modifies the structure of \( N \) by invoking \( D(n, N) \) sequentially for all \( n \in N' \).

Definition 8: [Selection Value] Given a reduced service net \( N_R = (P, T, F) \) in which \( P_C = \{ p_1, p_2, \ldots p_n \} \subseteq P \), and \( p_i^* = \{ t_{1i}, t_{2i}, \ldots t_{ki} \}, 1 \leq i \leq n \) \( \psi(p_i) \) is defined as the set of all transitions in \( p_i^* \).

\[ \psi(p_i) = \{ t_{mi} \mid m_i \in [1, k_i] \} \]

\( \Psi(P_C) \) is a finite set of \( n \)-tuples. Each tuple \( h \) in \( \Psi(P_C) \) is an ordered list of transitions, and each transition is an output transition of a conflict place, that is

\[ \Psi(P_C) = \{(t_{1m_1}, t_{2m_2}, \ldots, t_{nm_n}) \mid m_i \in [1, k_i], i \in [1, n] \} \]

Each \( h = (t_{1m_1}, t_{2m_2}, \ldots, t_{nm_n}) \) in \( \Psi(P_C) \) is a selection value (or \( h \) value) w.r.t. conflict place set \( P_C \).

For \( h = (t_{1m_1}, t_{2m_2}, \ldots, t_{nm_n}) \in \Psi(P_C) \), we define:
\[ \prod_h (p_i) = t_{i_1} \]
For example, in Fig. 8, \( P_C = \{p_1\} \), then

\[
\psi(p_1) = \{t_1, t_2, t_{31}, t_{32}\},
\]

\[
\Psi(P_C) = \{(t_1), (t_2), (t_{31}), (t_{32})\}.
\]

If we define \( h_1 = (t_{31}), h_2 = (t_{32}) \in \Psi(P_C) \), then

\[
\prod_{h_1}(p_1) = t_{31}, \prod_{h_2}(p_1) = t_{32}.
\]

We also define the \( \approx \) relation among transitions, \( t_i \approx t_j \) iff \( i = j \), or \( t_i \) and \( t_j \) belongs to the same group of a mediation transition. For example, in Fig. 8, \( t_{31} \approx t_{32} \). We use it later in our decomposition algorithm such that mediation transitions are treated as one group.

Algorithm 1: [Decomposition Algorithm] The decomposition algorithm is intuitively explained as follows. For each conflict place, it removes all the unwanted branches according to a given selection value. First, for each conflict place, the branches not selected are isolated (the outmost ForEach procedure). Afterwards all the source places in the net are examined; and they are deleted with their succeeding transitions or marked as dead, till no new node can be removed/marked (the outmost While procedure).

```plaintext
Input:
\( N_S = (P, T, F); P_C = \{p_1, p_2, ..., p_n\} \subseteq P \\
R = (T, F, \Sigma)

Output:
\( R' = (T', F', \Sigma') \)
```

```plaintext
\[
R_1 \triangleq N_S
\]

ForEach \( p_i \in P_C \)

\[
\text{ForEach } t_s \text{ s.t. } t_s \in p^*_s \Rightarrow \neg \prod(t_s) \approx t_i
\]

\[
M(p_i, t_s, p_{pa_i}, R)
\]

EndFor

EndFor

While \( (\exists \in P \text{ s.t. } S(\in) \land \neg D(\in)) \)

\[
t = \in
\]

If \( t = \in \)

\[
D(t, R)
\]

ElseIf \( t \Rightarrow \in \land \exists \begin{array}{l} \in' \text{ s.t. } \neg D(\in') \\
\end{array} \)

\[
M(p_i, t_s, p_{pa_i}, R)
\]

Mark \( s \) as dead

EndIf

EndWhile

\[\prod_{h_1}(p_1) = t_{31}, \prod_{h_2}(p_1) = t_{32}\]

\[\prod_{h_1}(p_1) = t_{31}, \prod_{h_2}(p_1) = t_{32}\]

\[\prod_{h_1}(p_1) = t_{31}, \prod_{h_2}(p_1) = t_{32}\]

\[\prod_{h_1}(p_1) = t_{31}, \prod_{h_2}(p_1) = t_{32}\]

```

**Proof:** Suppose that there is a feasible solution \( \chi \). Then, \( \chi \) must be a subnet of \( N_S \), because \( N_S \) contains all the data relations in this service portfolio. Our decomposition algorithm finds all ASWP and conflict-free subnets of \( N_S \) by iterating on all possible selection values, and we denote the derived subnets set as \( \Omega(N_S) \). Because \( \chi \) is also ASWP and conflict-free, therefore, \( \chi \in \Omega(N_S) \) and that is, once there is a feasible solution, our decomposition algorithm can find it.

**B. Complexity Analysis**

The complexity of each run of the decomposition algorithm equals to the complexity of traversing the reduced service net, i.e., \( O(|V| + |E|) \) where \( V \) and \( E \) are the vertex and edge sets of the reduced service net, respectively.

If \( RSN \) contains \( n \) OR-split places, each with \( m \) branches, we have \( m^n \) selection values. In the worst case, from each value we run the decomposition algorithm and obtain one distinctive subnet. Thus, we can have \( m^n \) subnets altogether. In the worst case, the computational complexity is \( O((|V| + |E|) \times m^n) \), i.e., \( O(m^n) \).

In practice \( m \) and \( n \) are usually not large. Moreover, it is observed that some selection values may derive nonfeasible subnets, and some different values derive identical subnets. By detecting these circumstances, many selection values can be simply ignored.

One circumstance is death path elimination, i.e., to detect whether different selection values lead to the same decomposition.

As shown in Fig. 9(a), if \( h = (t_1, +, +) \) (+ stands for arbitrary value), when we apply the decomposition algorithm on the net, \( p_{4,5} \) is removed. We can conclude that once \( p_{4,5} \) selects \( t_1 \), the selection value of \( p_{4,5} \), i.e., \( \prod(p_{4,5}) \) does not make a difference on the result. So many different selection values will lead to the same decomposition result, and by using this fact we do not have to run the decomposition algorithm for each selection value.

Another circumstance is lack-of-synchronization detection, i.e., to detect whether one valuation will derive nonfeasible subnets. When we run the decomposition algorithm, if \( \exists t, |t| > 1 \land (\exists p \in p^*_t, p \) is a dead source place) \land (\exists p \in p^*_t \text{ s.t. any path from } i \text{ to } p \text{ does not contain any conflict place}), then this subnet must be the one with lack-of-synchronization. At this point, the algorithm can stop and we conclude that no feasible solution will be derived from this selection value. In Fig. 9(b), \( |t_4| > 1, p_1 \in p^*_t \text{ and } p_1 \text{ is a dead source place; } p_2 \in p^*_t, \text{ and all paths from } i \text{ to } p_2 \text{ do not contain any conflict place. Therefore, lack-of-synchronization is detected and we can conclude this } h \text{ value does not yield any feasible solution.} \]

**VI. Effectiveness and Efficiency Analysis**

**A. Solution Effectiveness**

**Theorem 1:** [Solution effectiveness] Given a service net \( N_S \) and a requirement \( I \rightarrow O \), if there is a feasible solution w.r.t requirement \( I \rightarrow O \), the decomposition algorithm can find it.
VII. AN EXAMPLE

Based on the proposed concepts and algorithms, we have developed a prototype system called Data-driven Service Composition System (DSCS). The implementation details of this system can be found from our previously published paper [32].

We take the AOPS example mentioned in Section I to illustrate the validity of the proposed approach. The service portfolio and data relations in AOPS are shown in Fig. 10. For simplicity, we omit the unrelated data/relation so that in the next step RSN equals SN. The operation and mediation transitions are explained as follows:

\[ mt_{11}, mt_{12} \] Mediation transition, to classify order into two subtypes, i.e., ADSL order (\( ads_{-}order \)) and new-telephone-line order (\( nl_{-}order \)).

\[ t_{2} \] Operation transition, to search a corresponding worksheet according to an order. The worksheet has to be already generated before the search can return a meaningful result.

\[ t_{3}/t_{4} \] Operation transition, to generate a worksheet based on an ADSL/new-telephone-line order.

\[ mt_{21} \] Mediation transition, to decompose worksheet into two parts, i.e., line worksheet (\( line_{-}ws \)) and in-house worksheet (\( inhouse_{-}ws \)).

\[ mt_{22} \] Mediation transition, to compose line and in-house worksheet confirmation (\( line_{-}con/inhouse_{-}con \)) into worksheet confirmation (\( ws_{-}con \)).

\[ t_{5}/t_{7} \] Operation transition, to carry through line/in-house construction work based on the given worksheet, and return a corresponding confirmation form.

\[ t_{6} \] Operation transition, to search an in-house worksheet according to its corresponding line one.

\[ t_{8} \] Operation transition, to generate a customer receipt (\( cus_{-}rec \)) based on a worksheet confirmation (\( ws_{-}con \)).

The requirement is ORDER—\( cus_{-}rec \) (given an ADSL business order, do all the necessary operations and return a customer receipt).

As shown in Fig. 10, given this portfolio in which service operations are modeled as operation transitions and data relations modeled with mediation transitions, an SN is easily obtained by merging the places attached with identical data type into a single one. We use two eclipses to group places labeled with order and worksheet, respectively, to demonstrate the merge of places. Fig. 11 illustrates the RSN (same as SN, in this case) and decomposed nets displayed in Petri-Net Kernel. The uppermost net is the RSN.

From top to bottom, the second to fourth nets in Fig. 11 illustrate three subnets decomposed from RSN. They are solution candidates. The second net corresponds to the \( h \) value \((mt_{11}, t_{5})\) or \((mt_{12}, t_{5})\). The third net corresponds to \( h \) value \((t_{2}, t_{5})\). They are both feasible service composition candidates. The fourth net corresponds to the \( h \) value \((t_{2}, t_{5})\). In the fourth net, lack-of-synchronization is encountered, and therefore it is not a feasible solution. Thus we obtain two feasible ones based on the proposed data-driven approach.

In the mean time, we find that the second net is a satisfactory solution. The result for service composition is made up of three steps, i.e., Apply, Construct and Offering. In step Apply, AOPS receives customer’s ADSL order and generates a worksheet according to the type of business; in step Construct, AOPS undertakes required construction work according to the worksheets’ content and generates confirmation forms; and in step Offering, it delivers a receipt to the customer.

Later on, if we manually check the third net, we find that it does not fulfill the requirement since operation \( t_{2} \) does a search job instead of a worksheet generation job. Therefore, it is not a satisfactory solution although it is a feasible one from the perspective of data flow.

Through this example we can see that data-driven approach is well suited for service composition as long as the business/service data and their relations are available.

VIII. DISCUSSION

In the proposed data-driven approach, we use a subnet decomposition technique to generate all conflict-free solutions. Note that our definition of conflict-freeness (Definition 4) allows choices over mediation transitions such that a generic data type can be classified into all its child types. We believe that the decomposition-based approach is an efficient way to find feasible solutions. If a desired solution is required to have conflicts, users can easily change the definition of selection value (see Definition 8) from one to multiple branches for each conflict place. By this means, more subnets are obtained and can contain cyclic nets if choices are allowed.

Although our algorithm can be easily adapted to this scenario that allows choice places, when we introduce our approach we still adopt the conflict-free option because of:

1) Less computation complexity. Allowing arbitrary combination of choices among branches will increase the computation complexity from \( O(m^{n}) \) to \( O(2^{mn}) \), i.e., \( O(2^{mn}) \);

2) Compactness and intuitiveness to users. When each choice place selects multiple branches, many meaningless solutions will result.

For example, in the AOPS example in Section VII, if we allow choice places, then the RSN (see the upmost net in Fig. 11) itself is a candidate solution but obviously this net is an undesired one. Although the candidate solutions generated by the conflict-freeness constraint do not contain loop and choice (again, it may contain choice
over sub types of data), they offer the “backbones” that can be further refined (e.g., by adding choice branches or loop) by users. This assumption is realistic since: 1) the choice branches occur in different candidate solutions we give and it is easy to combine them; 2) more inputs from process designers are needed to decide, which branches to select and when to start and terminate a loop; and 3) in our practice of business process engineering, every solution generated by automatic algorithms must be examined and refined before it is put into real execution.

Therefore, our solution approach can be extended to the scenario that allows choices and loops. However, in this paper, we focus on a conflict-free approach to quickly identify the relevant part inside the service portfolio and generate solution backbones. These backbones can be combined and merged to form needed solutions that can contain choices and loops when required.

IX. CONCLUSION

It is necessary to have a way that enhances the reusability of service portfolio by generating composition guidance when fulfilling a business requirement. This research can be seen as an important step towards the effort to bridge the gap between business and service domains in building enterprise SOA solutions. We utilize data relations in both domains, and add data mediation constructs to make the data model in these domains complete and coherent. We devise three composition rules, i.e., sequential, parallel and choice, based on the augmented data model. Based on the data relations and composition rules we propose a formal method to derive all the possible composition candidates given a service portfolio. First, we obtain a connected service net from the service portfolio; then we reduce it with respect to the given requirement (i.e., the input/output signature); next we decompose the reduced one into subnets, each of which represents a composition candidate. A prototype system is developed and an example is given to validate our methodology.

The contributions of this paper are summarized as follows.

1) A lightweight approach making the best use of the existing service portfolio in enterprise SOA solutions is presented. It does not need additional semantic information. What we need is data type and data relation definition in business/service domain. Such information can be obtained by parsing WSDL files, XML schemas, and the UML class definitions in a business domain. Minor effort is needed to set up the cross-domain data relations, and add necessary data mediations.

2) No existing work tackles the problem of service composition using Petri-net decomposition. This idea can be seen as a combination of the bottom-up and top-down approaches in service composition domain. That is, the generation of a service net is a bottom-up way to give a compact representation of the service portfolio. The decomposition of a service net is a top-down way to derive solution candidates with a desired structure. The introduction of service nets has two advantages: a) a service net regarding a service portfolio can be reused for multiple service composition requests and b) when new services are added or...
services retire, it can be modified incrementally and does not need to be built from the scratch.

3) Our approach does not preclude the integration with other approaches, e.g., semantic and rule-based ones. The solution obtained by our approach provides hints to find a satisfactory one. Even when no feasible one is derived, in the decomposed nets we can find fragments that may be relevant to a final solution.

Future work includes building a more comprehensive data model to derive better solutions, and validating our approach via service libraries such as ITIL (Information Technology Infrastructure Library) and SCOR (Supply-Chain Operations Reference-model), or artificially generated large-scale service networks [12]. Our work does not address how the generated Petri nets are to be executed in a workflow engine. One feasible solution is to transform the resultant Petri nets into BPEL processes such that they can be executed in a compliant engine. Well-established methods, for example, the one proposed in [17] and [33] can be used to serve this purpose. We will also seek the applications of the proposed method to other web services in transportation, e-commerce, workflow, and manufacturing [36]–[41].

ACKNOWLEDGMENT

The first author thanks F. Rao, R. Fang, and L. Wang, all from IBM China Research Laboratory, for the constructive discussions with them.

REFERENCES


[34] Proc. AI Planning Conf. Syst. (AIPS 02), Toulouse, France, 2002, pp. 204–211.


