Process Guided Service Composition in Building SoA Solutions: A Data Driven Approach

Wei Tan[1,3], Zhong Tian[2], Fangyan Rao[1], Li Wang[1], Ru Fang[1]
IBM China Research Lab[1], IBM China Software Development Lab[2], Beijing 100094, China
Dept. Automation, Tsinghua University[3], Beijing 100084, China
E-mail: {weitan, tianz, raofy, wanglcrl, fangru} @cn.ibm.com

Abstract

Solution design has been more of an art than an engineering discipline. Lots of researchers and practitioners have proposed and exercised different kinds of approaches with varied success. Most of these methods seem to have focused on building new solutions from scratch. However, enterprise solutions today are mostly built on top of an existing IT infrastructure. The notion of SoA is trying to pave a way to integrate heterogeneous components together to meet new business needs. When a new requirement is given to a system developer in the form of business processes, it would be ideal if s/he can make the best of existing services for many reasons. In this paper we propose a data driven approach to provide service composition guidance to implement the given requirement. Based on the relations among business domain data and service domain data, we can generate additional data mediations according to three composition rules. With these data relations and composition rules, we give a formal approach to devise choreography of services from current service portfolio, plus additional data mediation artifacts to realize a given requirement. Our work can be seen as an effort to bridge the gap between business and service domain.

1. Introduction

The web is evolving from an information-delivering center to a function offering platform to better support automated use. Service Oriented Architecture (SoA) is the architecture for the publication, discovery, binding, composition, deployment and control of service-based applications [1].

Reusability is a key issue in SoA to ensure scalability and proficiency. When new business requirements emerge, solution designers should devise a design that makes the best use of existing services, and SoA provides a way to glue all components together with least augmentation or modification. Current practices seem to have focused on building new solutions from scratch in mind [2]. They address well the issue of how to reflect the requirements and to refine them into finer grain of abstraction so that programmer can take over to implement. They seem to ignore 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. There are a number of good research works and development efforts on web service modeling and composition [3-5], but there is not much on the linkage between requirements and services. Academia focuses on automatic service composition (through the method of semantic web, AI planning, constraint solving, etc) and verification of composed service [3-5]. In the meantime, industry community concentrates more on service composition languages [5, 6] and the software tools to support them.

In this paper we propose a data based approach to provide guidance for process (or service) 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 net as the formalism to represent three composition rules, which is based on the data relations we get. Then we propose a formal approach to derive service compositions from service portfolio, and these service compositions satisfy the requirement in respect of the input/output data type. A prototype system is developed to test the validity of the formal approach we proposed. To the best of our knowledge, we present the first formal approach and integrated system that utilize the data relation in business/service portfolio to derive composite services.

2. Motivation

IEEE International Conference on Web Services (ICWS'06)
0-7695-2669-1/06 $20.00 © 2006 IEEE

In software tools such as IBM WBI modeler and WSAD-IE [7], service composition/process 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, there are some gaps remained.

First, the model elements of business process and service composition are not identical. In practice, there are various kinds of specifications, languages and notations to model business process and service composition [5, 6]. Second, the data model in business domain and service domain 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 realized by available services. So it would be 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 realized by existing services to a greater extend.

On one hand, the difficulties mentioned above hampers the effective and efficient utilization of available services. On the other hand, service portfolio contains abundant information. For example, WSDL (Web Services Description Language) files contain the input/output data type of operations, and WSDL files with associated data definition schema contain the relations among these data types, such as data aggregation and generalization. We believe that from these data relations, much guidance can be derived to help service composition.

3. Problem statement

In this section we first present how to drive data structures from WSDL files and how to model data structures in two domains, then we give the formulation of service composition.

3.1 Domains and data relations

Various languages exist for modeling, and we need a common language to represent the artifacts we are working on. Colored Petri net [8] captures both control and data aspects of process. In this paper we use colored Petri net as the common model for both business processes as well as services, as most other models can have an equivalent representation in it. For space limitations we omit the introduction of colored Petri nets, one can refer to [8] for more information.

There are two domains involved in service composition scenario, i.e. the business domain and the service domain. Business domain is also known as the requirement specification domain, in which requirements are represented by business processes consisting abstract activities with input/output data. Service domain is also known as the implementation domain, in which 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 data elements in service domain are denoted with lowercase strings.

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's explain with a real life example, i.e., an ADSL Order Processing Service (AOPS) provided by a telecommunication company. See Fig. 1, the upper part is a segment extracted form WSDL file. Operation *generate_worksheet* receives ADSL application order (data type *order*) and **generate** a worksheet (data type *worksheet*). Data type *worksheet* represents the work-items to fullfill the order, and it **aggregates** *inhouse_ws* which represents the work-items undertaken in customer's house. Data type *order* is the **generalization** of *nl_order* which 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 business domain, there are two data types *ORDER* and *WORKSHEET*, a business requirement is expressed as *ORDER* generates *WORKSHEET* (*ORDER*→ *WORKSHEET*).

Cross-domain relation

Since we're trying 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. So here we use *realization* as a cross-domain relation to signify that one data type in service domain directly implements one data type in business domain. For example, in Fig. 1, data type *order* in service domain is the realization of data type *ORDER* in 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, has realization relation between them.)

Given that the granularity of business data is usually coarser than that of service data, complete realization relation between business data and service data are not guaranteed to be explored. In order to model incomplete realization relation between data types, we use two stereotypes to extend realization relation, i.e., *partial* and *specialized* realization.

1) Partial realization. Partial realization describes the situation that a service data realizes one part of a business data. For example, in Fig. 1, data type *inhouse_ws* is one part of *worksheet*, which realizes *WORKSHHET*, so we define *inhouse_ws* as the *partial realization* of *WORKSHEET*.
2) Specialized realization. Specialized realization describes the situation that a service data realizes one special kind of a business data. For example, in Fig. 1, data type *nl_order* is one special class of data type *order* which realizes *ORDER*. So we define *nl_order* as the *specialized realizatio*n of *ORDER*.

Fig. 1 illustrates all the data relations we discussed and gives the notations to model these relations.

```
<xs:element name="order" type="order"/>
<xs:complexType name="nl_order">
  <xs:complexContent>
    <xs:extension base="order">
      ...
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="worksheet">
  <xs:sequence>
    ...
    <xs:element name="inhouse_ws"/>
  </xs:sequence>
</xs:complexType>
<operation name="generate_worksheet">
<input message="order"/>
<output message="worksheet"/>
</operation>
```

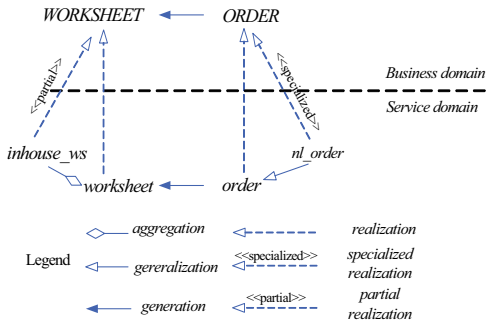

Fig. 1 Domains and data relations

3.2 Problem formulation

Based on the formalization of process and data, we have the following problem statement. (See Fig. 2)

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 $di \rightarrow do$, i.e., the operation consumes one instance of data type *di* and yields one instance of data type *do*.

Our goal is to find a collection of operations in a large service portfolio. These operations connect to form a colored Petri net *Proc*. (See Fig. 2, data type attached to one place is denoted as underlined string).

We assert that *Proc* is a valid composition of the requirement $I \rightarrow O$ iff:

1. *Proc* takes *i* as its only input and *o* as its only output.
2. *Proc* is data coherent, i.e., each place in *Proc* can receive correct data type from its preceding transitions and yield correct data type to its succeeding transitions.

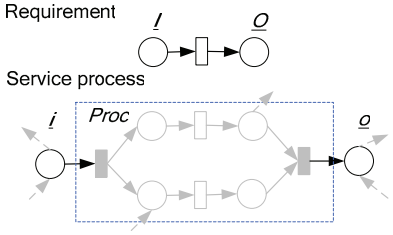

Fig. 2 Service composition problem formulation

Before addressing the problem of service composition, we introduce in the next section data driven composition rules used to refine processes in business domain into processes in service domain.

4. Data driven composition rules

The composition rules and other transformation rules in this paper are all expressed with graph transformation formalism. We believe this formalism is intuitive enough for understanding, and more details on graph transformation can be found at [10].

Sequential composition rule

The sequential composition rule is illustrated in Fig. 3. 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., $a \rightarrow b \rightarrow c$.

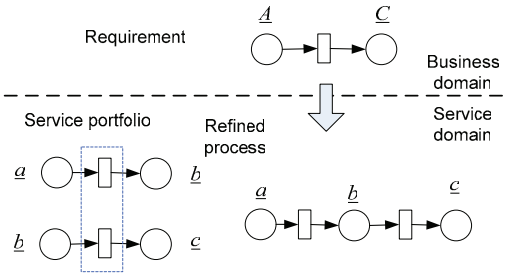

Fig. 3 Sequential composition rule

Parallel composition rule

Here we explain parallel composition rule through an example in Fig. 4. 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 (which are represented black rectangles). With mediation transitions, a is decomposed to a_1 and a_2, and b is decomposed to b_1, b_2.

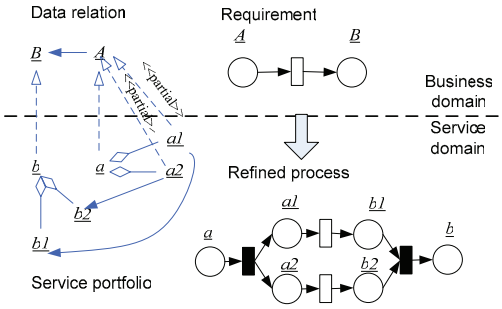

Fig. 4 Parallel composition rule

Choice composition rule

Choice composition rule is related to data generalization, and this rule is explained in Fig. 5.

In business domain, we have a requirement $A \rightarrow B$, in service domain, we have two operations $\{a_1 \rightarrow b, a_2 \rightarrow b\}$ ($a = a_1 \cup a_2$, $a_1 \cap a_2 = \varnothing$). 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 (which are represented by black rectangles). With mediation transitions, data type a is specified to either a_1 or a_2.

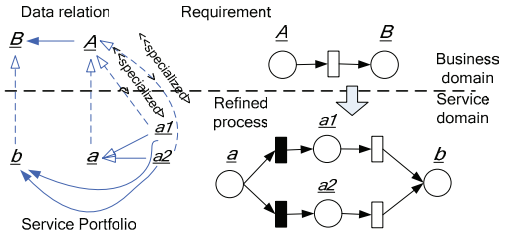

Fig. 5 Choice composition rule

In the composition rules we illustrate in this section, there is direct mapping between business data and service data. Nevertheless, usually there is no existing service data that realizes some business data, so we may 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, See Fig. 4, if there is no service data type $a(b)$ which 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 don't exist in service portfolio, we should manually add a_2 and b_2 as virtual data types in 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 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 process composition, and we'll pay more attention to the issue of virtual data in our future work.

5. Data driven process composition

Based on the data relations and composition rules, we are now ready to solve the problem formulated in Section 3.2. That is, if a business requirement is represented as $I \rightarrow O$, find a service composition from service portfolio to realize this requirement.

5.1 Preliminary Definitions

In this section some definitions which will be used in the following sections are given.

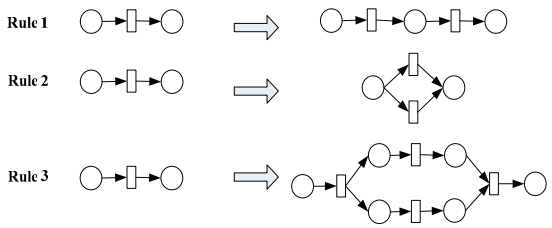

Fig.6 Generation rules for AWSP

Definition 1. (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. 6 are AWSPs.

Based on the composition rules we give in Section 4, we give the definition of Service Net and the method to derive a service net with given service portfolio.

Definition 2. (Service Net, SN) A Service Net with respect to a service portfolio and data mediation is a colored Petri net $(P, T, A, \sum, C)$, where

1) $\sum$ 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. $T = OT \cup MT$, OT is the set of operation transitions, and MT is the set of mediation transitions.
4) A is a finite set of arcs.
5) C is a color function defined from P into $\sum$. C is injective, i.e., $C(p_1) = C(p_2) \Rightarrow p_1 = p_2$.

A Service Net is 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.

Remarks:

1) A service net SN is data coherent since transitions represents data mediation or transformation.
2) The refined process in Fig. 3, 4 and 5 are all Service Nets.

Definition 3. (Conflict place) Given a service net $SN = (P, T, F)$, $T = OT \cup MT$, $Pc \subset P$ is the set of conflict places, $p \in Pc$ iff $p^\bullet > 1 \wedge (\exists t \in p^\bullet$ s.t. $t \in OT)$

Definition 4. (Reachability) In a Petri net (P, T, A), we define relation **R** as the reachability relation between two nodes. $n_1, n_j \in P \cup T$, $R(n_1, n_j)$ is true iff there is a path C from n_1 to $n_j < n_1, n_2, \ldots, n_j >$ such that $(n_i, n_{i+1}) \in A$ for $1 \le i \le j\text{-}1$, and for any two nodes n_p and n_q in C, $p \ne q \Rightarrow n_p \ne n_q$. We define $R(n_i, n_i) = true$.

5.2 Derive AWSP from Service Net

With the definitions in Section 5.1, the problem raised in Section 3.2 is interpreted more formally:

Input: $I \rightarrow O$.

Output: An AWSP *Proc*, which is a sub graph of SN, with i as its input and o as its output, and contains no conflict place.

The solution approach is made up of four steps.

Step 1. Construct a service net SN from given service portfolio and data mediations in business/service domain. The method is given in Section 5.1.

Step 2. Derive a Reduced Service Net RSN from SN.

Let's first define Reduced Service Net.

Definition 5. (Reduced Service Net) A reduced service net RSN of $SN = (P, T, F)$ with respect to input data type i and output data type o, or $RSN(SN, i, o)$, is a sub graph of SN, s.t., $\forall n \in P_{SN} \cup T_{SN}$, $n \in P_{RSN} \cup T_{RSN}$ iff in SN, $R(i,n) = R(n,o) = true$.

Remark: when a SN is reduced to a RSN, any operations, data mediations and data types which are not on a path from data type i to data type o are eliminated. We reduce a Service Net in order to remove the un-related parts, and the reduction process can be done by slightly modifying the graph traverse algorithm.

Step 3: Decompose RSN into sub nets, each of which contains no conflict place.

The branches after each conflict places represent options from which we can choose for data processing. A snippet of a RSN is shown in Fig. 7. Place p_1 is a conflict place, so data type a attached to p_1 can be processed by operation t_1 or t_2, or the XOR-join of operations $\{t_{31}, t_{32}\}$.

A RSN must be decomposed into a set of subnets which 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. 7 illustrates, three subnets are generated because p_1 has three succeeding transitions (t_{31} and t_{32} is grouped and treated as one single branch).

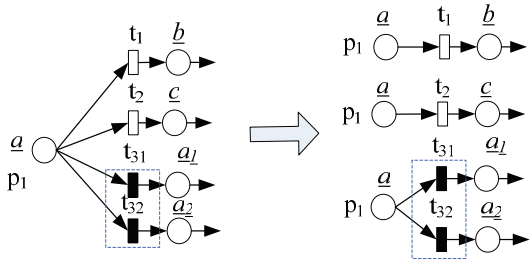

Fig. 7 Illustration of the decomposition algorithm

When there are more than one conflict places in a RSN, things get more complicated. If there are n conflict places each with m branches, possibly we have m^n distinctive sub nets. We use *selection function* to decide which branch to choose for each conflict place. One valuation of selection function corresponds to one situation of decomposition, and we use *decomposition algorithm* to get one subnet for each valuation. Now we'll give formal definitions of the functions we mentioned above.

Given a reduced service net
$RSN(pi, po) = (P, T, F)$:

1. $\forall p \in P$, p is a *source place* iff $p \neq pi \wedge {}^\bullet p = \varnothing$.
2. $\forall p \in P$, function SP(p) is a Boolean function. SP(p) returns true iff p is a source place, all source place is initially marked to be *non-dead*.
3. $\forall p \in P$, function DSP(p) is a Boolean function. DSP(p) returns true iff p is a source place and is explicitly marked to be *dead*.

Given a net $N = \{P, T, F\}$, $p \in P$, $t \in T$, $p \in {}^\bullet t$
Function Redirect (p, t, p', N) modifies the structure of N by redirect (p, t) to (p', t), that is:

$$P = P \cup \{p'\};\ F = (F - \{(p, t)\}) \cup \{(p', t)\}$$

Given a net $N = \{P, T, F\}$, $n \in P \cup T$
Function Remove(n, N) modifies the structure of N by deleting n and the arcs leading and ending at n, that is,

If $n \in P$

$$P = P - \{n\}$$
$$F = F - \{(n, x) | \ x \in T \wedge (n, x) \in F\} \cup \{(y, n) | \ y \in T \wedge (y, n) \in F\}$$

If $n \in T$

$$T = T - \{n\}$$
$$F = F - \{(n, x) | \ x \in P \wedge (n, x) \in F\} \cup \{(y, n) | \ y \in P \wedge (y, n) \in F\}$$

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

Definition 6. (Selection Function) Given a reduced service net $RSN = (P, T, F)$, $T = OT \cup MT$, $Pc = \{pc_1, pc_2, \ldots, pc_n\} \subset P$. Selection function Sel maps each p in Pc to one of its succeeding transitions, i.e.,

$$\text{Sel}(pc_1, pc_2, \ldots, pc_n) = (t_1, t_2, \ldots, t_n)$$
$$t_i \in pc_i^\bullet,\ 1 \le i \le n$$

For example, in Fig. 7, the value of Sel(p_1) can be t_1 or t_2 or $\{t_{31}, t_{32}\}$. Each valuation of selection function corresponds to a sub net decomposed from RSN.

Decomposition Algorithm

The decomposition algorithm is intuitively explained as follows. For each conflict place, the decomposition algorithm removes the inactive branch according to the valuation of selection function. Consider the procedure in the outmost While loop. First, the branches not selected are isolated (the first **For** procedure). Next, the choice places which are also source places are taken out since they're to be removed too (the first **While** procedure). Then all the source places in the net are examined; they are deleted with their succeeding transitions or marked dead (the second **While** procedure). These three steps are iterated until no new node is to be removed.

Input:
$RSN = (P, T, F)$; $Pc = \{pc_1, pc_2, \ldots, pc_n\} \subset P$
One valuation of Sel($pc_1, pc_2, \ldots, pc_n$)
Output:
$R_i = \{P_i, T_i, F_i\}$

$R_i = RSN(pi, po)$
While $Pc \neq \varnothing$
  Select $pc_i \in Pc \wedge \text{SP}(pc_i)$
  For every t_k s.t. $t_k \in pc_i^\bullet \wedge \text{Sel}(pc_i) \neq t_k$
    Redirect(pc_i , t_k, pc_{ik}, R_i)
  EndFor
  $Pc = Pc - \{ pc_i \}$

  While ($\exists s \in R_i$ s.t. $s \in Pc \wedge \text{SP}(s)$)
    $Pc = Pc - \{s\}$
    For all $t_k \in s^\bullet$
      Redirect(s, t_k, s_k, R_i)
    End for
  EndWhile

  While ($\exists s \in R_i$ s.t. $\text{SP}(s) \wedge \neg\text{DSP}(s)$)
    $t = s^\bullet$
    If ${}^\bullet t = \{s\}$
      Remove(s , R_i)
      Remove(t , R_i);
    ElseIf ${}^\bullet t \supset \{s\} \wedge \exists s' \in {}^\bullet t - \{s\}$ s.t. $\neg\text{DSP}(s')$
      Denote s as dead.
    ElseIf ${}^\bullet t \supset \{s\} \wedge \forall s' \in {}^\bullet t - \{s\} \Rightarrow \text{DSP}(s')$
      Remove (${}^\bullet t$, R_i)
      Remove (t, R_i)
    EndIf
  EndWhile

End while

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

5.3 Complexity analysis

The complexity of each run of decomposition algorithm equals to the complexity of traverse the reduced service net, i.e., O($|V| + |E|$). (V and E are the

vertex and edge set of reduced service net respectively.)

If a reduced service net *RSN* contains n OR-split places, each with m branches, we have m^n valuations of selection function. In the worst case, from each valuation we run the decomposition algorithm and get one distinctive sub net, thus we'll get m^n sub nets altogether. So in worst case, the computation complexity is $O((|V|+|E|)\times m^n)$.

However, it's observed that some valuations might derive non-feasible subnets, and some different valuations will derive identical subnets. By detecting these circumstances, many valuations can be simply ignored.

One circumstance is *death path elimination*, i.e., to detect whether different valuations of select function lead to the same decomposition.

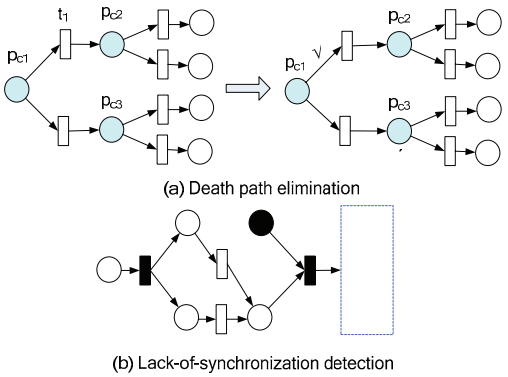

Fig. 8 Methods to reduce computation complexity

See Fig. 8(a), if Sel(p_{c1}, p_{c2}, p_{c3},) = (t_1, *, *) (* stands for arbitrary value), when we impose the decomposition algorithm on the net, p_{c3} is removed. We can conclude that once p_{c1} selects t_1, the selection of p_{c3} do not make a difference on the result. So many different valuations of selection function will lead to the same decomposition result, and by detecting this fact we do not have to run the decomposition algorithm for each valuation of selection function..

The other circumstance is *lack-of-synchronization detection*, i.e., to detect whether one valuation will derive non-feasible subnets. When we run the decomposition algorithm, if $\exists t$, $|{}^\bullet t|>1 \wedge (\exists p \in {}^\bullet t$, p is a dead source place) $\wedge$ ($\exists p \in {}^\bullet t$ s.t. any path from i to p do not contain places in set Pc), then this subnet must be lack-of-synchronization. At this point, decomposition algorithm can stop and we conclude no feasible solution will be derived from this valuation. (See Fig. 8(b) for an example.)

6. System Implementation and an Example

Based on the concepts and algorithms we proposed in this paper, we've developed a prototype system *Ddscs* (Data driven service composition system).

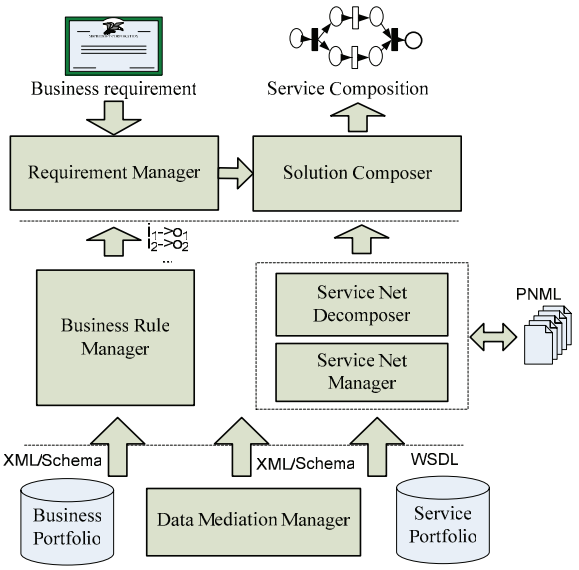

Fig. 9 System architecture of Ddscs

The system architecture is presented in Fig. 9, and is briefly explained as follows:

1) A new business requirement is fed into Requirement Manager, where it's normalized based on the business rules stored in Business Rule Manager. After normalization, business requirement is expressed in the form of $\{i_1 \rightarrow o_1, i_2 \rightarrow o_2, \ldots\}$ and is put into Solution Composer.
2) Business Portfolio (BP) and Service Portfolio (SP) maintain business and service data/relations, respectively. Data Mediation Manager (DMM) maintains the cross-domain data relations. These data/relations are expressed in XML format for transformation convenience.
3) Based on BP, SP and DM, service net can be derived and managed by Service Net Manager. Service net is stored in PNML [11] format, which is an XML-based interchange format for Petri nets. Given the normalized requirement, a service net can be decomposed by Service Net Decomposer to derive candidate solutions which are presented to users in Solution Composer. In Solution Composer we use Petri Net Kernel [12], a Petri net modeling and analyses tool, to display the decomposed net to users. Here we use Petri Net Kernel because it support PNML format input and this tool is easy for further extension.

We take the AOPS example mentioned in Section 1 to illustrate the validity of the proposed approach. The service net is derived from one WSDL file. For simplicity, here we omit the unrelated parts and only give the RSN.

The requirement is *ORDER→CUS_REC* (given an ADSL business order, do all the necessary operations and return a customer receipt).

Fig. 10 illustrates the reduced service net and decomposed net displayed in Petri Net Kernel. The uppermost Petri net is the reduced service net. The operation and mediation transitions are explained as follows:

mt_{11}, mt_{12}: mediation transition, to classify *order* into two sub-types, i.e., ADSL order (*adsl_order*) and new- telephone-line order (*nl_order*).

t_2: operation transition, to search a corresponding worksheet according to an order.

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.

t_6: operation transition, to search an in-house worksheet according to its corresponding line worksheet.

t_8: operation transition, to generate a customer receipt (*cus_rec*) based on a worksheet confirmation (*ws_con*).

The 2nd to 4th Petri nets in Fig. 10 illustrate three subnets decomposed from RSN. The 2nd net corresponds to the valuation ($\{mt_{11}, mt_{12}\}$, t_5). The 3rd net corresponds to the valuation (t_2, t_5). They're both feasible service composition candidates. The 4th net corresponds to the valuation (*, t_6). In this case lack-of-synchronization is encountered, and the decomposed net is not a feasible solution. Till now we get two solution candidates based on the data driven approach we proposed. If we manually check the 3rd net we'll found it does not fulfill the requirement since operation t_2 does a search job instead of a worksheet generation job.

In the mean time, we'll find the 2nd 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 generate a worksheet according to the type of business; in step *Construct*, AOPS undertakes required construction work according to the worksheet content and generates confirmation form; in step *Offering*, AOPS delivers a receipt to the customer.

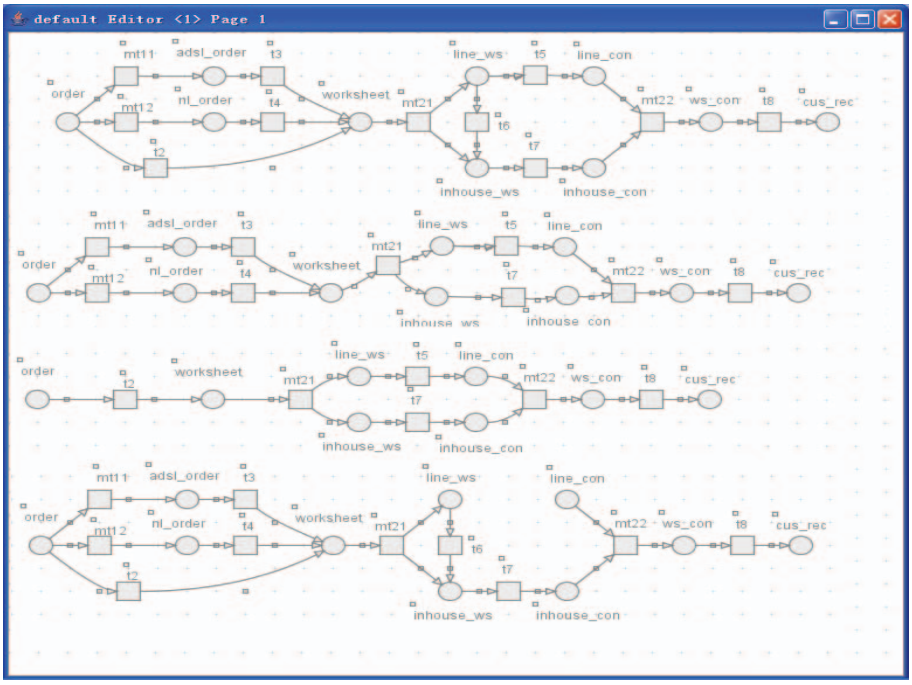

Fig. 10 The reduced service net and decomposition results of AOPS (in Petri net kernel)

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.

7. Conclusion

In this paper we try to bridge a gap between business domain and service domain in building SoA solutions from data perspective.

It is very important to have a way that enhances the reusability of service portfolio by generating service composition guidance on implementing a business process. We utilize data relations in business and service domain, and we 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 composition rule based on the augmented data model. Based on the data relations and composition rules we propose a formal approach to derive all the possible composition candidates from given service portfolio. First we get a connected service net from service portfolio; then we reduce the service net with respect to the given requirement (i.e. the input/output signature); later we decompose the reduced service net into subnets, each of which represents a composition candidate. A prototype system is developed and an example is given to validate our approach as well as the algorithm.

Briefly speaking, through our approach we can quickly pick out operations related to the business requirement, and chain them together as a process. Our work can be seen as the first step towards the effort to bridge the gap between business domain and service domain. Future work includes the formal definition and derivation of virtual data types, the validation of our approach in real industry scenario, and extending this idea to other issues in SoA, for example, service lifecycle management.

References

[1] N. H. Michael, P. S. Munindar, "Service-Oriented Computing: Key Concepts and Principles", *IEEE Internet Computing*, 2005(9), pp. 75-81.

[2] Z. Olaf, D. Vadim, G. Jonas and H. Kerard, "Service-Oriented Architecture and Business Process Choreography in an Order Management Scenario: Rationale, Concepts, Lessons Learned", *Proceedings of Conference on Object Oriented Programming Systems Languages and Applications*, ACM Press, New York, 2005, pp. 301-312.

[3] J. Cardoso, A. Sheth, "Introduction to Semantic Web Services and Web Process Composition", *Lecture Notes In Computer Science 3387*, Springer-Verlag, Berlin, 2005, pp. 1-13.

[4] H. Richard, S. Jianwen, "Tools for composite web services: a short overview", *SIGMOD Record*, 2005, 34(2), pp. 86-95.

[5] N. Milanovic, M. Malek, "Current Solutions for Web Service Composition", *IEEE Internet Computing*, 2004, 8(6), pp. 51-59.

[6] W.M.P van der Aalst, M. Dumas and A.H.M. ter Hofstede, "Web Service Composition Languages: Old Wine in New Bottles?", *Proceedings of 29th Euromicro Conference*, IEEE Press, 2003, pp. 298 – 305.

[7] M. Tilak, "From business modeling to Web services implementation: Part 1: Modeling a business process", http://www-128.ibm.com/developerworks/websphere/library/techarticles/0502_mitra1/0502_mitra1.html, 2005.

[8] Web Services Description Language. http://www.w3.org/TR/wsdls, 2001.

[9] Jensen K., *Coloured Petri nets: basic concepts, analysis methods, and practical use*, Springer-Verlag, Berlin, 1992.

[10] Grzegorz R., *Handbook of graph grammars and computing by graph transformation: volume I. foundations*, World Scientific Publishing, River Edge, NJ USA, 1997.

[11] J. Billington, S. Christensen, K. van Hee, et al, "The Petri Net Markup Language: Concepts, Technology, and Tools", Proceedings of Applications and Theory of Petri Nets 2003, *Lecture Notes In Computer Science 2679*, Springer-Verlag, Berlin, 2003 pp. 483-505

[12] K. Ekkart, W. Michael, "The Petri Net Kernel An infrastructure for building Petri net tools", *International Journal on Software Tools for Technology Transfer (STTT)*, Sept. 2001, pp. 486-497.