Vol.2009-SE-166 No.3
2009/11/5

入出力データ構造の対応による Web サービスの自動合成

王秋時[†]　　紫合治[††]

既存の Web サービスを合成して，よりアプリケーションに適した新たな Web サービスを生成することは，Web アプリケーション開発効率化にとって重要になる．ここでは，サービスの出力がリスト構造を含むデータ構造をもつ既存サービスを合成して，ネストされたデータ構造を出力とする新しいサービスを生成する方式を提案する．この方式は，フローチャートのようなワークフローを記述することなく，既存サービスのデータ構造と合成サービスのデータ構造の対応関係を規定することで合成する方式である．

Web Service Composition by I/O Data Structure Correspondences

Qiushi Wang[†]　and Osamu Shigo[††]

The Web service composition from existing web services to obtain the more suitable service for application still is a highly complex task. We propose a method to combine existing web services with list of results as their output into more functional Web service with nested list of results, which seems to be synthesis of the existing web service results, as its output. The method requires no workflow description, like flowchart, but only data correspondences among data structures of existing and combined Web services, as composition specification..

1. はじめに

近年，Web サービスを呼出して Web アプリケーションを作成する SOA[1](サービス指向アーキテクチャ)が注目されている．既に多くの Web サービスが公開され，さらに多くの企業内 Web サービスが SOA のために利用されている．Web アプリケーション開発では，単一の Web サービスが要求機能を満たさない場合は，いくつかのサービスを合成する必要がある．しかし，Web サービスの合成はアプリケーション開発者にとっては複雑な作業を伴うことが多い．

複雑な Web サービス合成をサポートするツールとしては，BPEL[2]が有名で広く使われている．しかし，プログラム経験に乏しいビジネスデザイナーにとっては，BPEL 利用は簡単ではない．ここでは，Web サービスの仕様である WSDL (Web Services Description Language)から入出力のデータ構造を取得し，そのデータ構造の対応関係を定めることにより，Web サービスを合成する方式を提案する．本方式は，ワークフローのようなプログラム的な処理記述でなく，入出力データ構造の対応関係の規定を使う．システムは既存の Web サービスの WSDL からデータ構造を取得し，C#による合成プログラムを自動生成する．

以下に，２節で現在の Web サービスの合成方式の問題点を述べる．３節で Web サービス合成の記述について説明し，４節でその記述からの合成プログラムの生成について述べる．　５節では，３つの Web サービスから，合成サービスを合成する例を述べ，最後に６節でまとめと今後の課題について議論する．

2. Web サービス合成の従来技術

これまで様々な Web サービス自動合成方式が提案されている[3]が，それらは大きくデータフロー型（Golog[4], SHOP2 planner[5]等）とコントロールフロー型（BPEL[2], EFlow[6]等）の２つのタイプに分けることができる．

テータフロー型では，ある Web サービスの出力から他の Web サービスの入力への対応を記述するだけで簡単に合成を行うことができる．しかし，Web サービスの出力がリスト構造を持つ場合，単純な入出力対応は規定できず，データフロー型は使えない．

コントロールフロー型として，BPEL はビジネスシステム開発の分野でＷｅｂサービスを合成することに広く適用されている．BPEL には，Web サービスの呼び出し，データの操作，障害通知，例外処理，プロセスの終了を含むプロセスフローを記述するためのワークフロー記述言語を持つ．コントロールフロー方式は，出力がリスト構

[†]Qiushi Wang, Graduate School of Information Environment,　Tokyo Denki University
[††]Osamu Shigo, School of Information Environment, Tokyo Denki　University

造を持つ Web サービスの合成にも適用できる．しかし，その合成では普通のプログラミングと同様に繰り返しのコントロール構造からなる比較的複雑な記述が必要になる．

リスト構造がある Web サービスの合成では，データフローは簡単だがうまく適用できないし，コントロールフローは適用できるが簡単ではない． 次章では，リスト構造の出力を持つ Web サービスの合成をデータフロー型と同様に簡単にできる方式を紹介する．

3. Web サービス合成

我々の Web サービス合成方式では，既存の Web サービスと合成 Web サービスの間のデータ構造の関係を規定する．既存 Web サービスの入出力データ構造は WSDL（Web Services Description Language）[7]記述から取得できる．図 1 に我々の Web サービス合成プロセスの概要を示す．まず WSDL から既存の Wed サービスのデータ構造を取得し，それらと合成 Web サービスのデータ構造の関係を規定し，そこから合成サービスのプログラムを自動生成する．

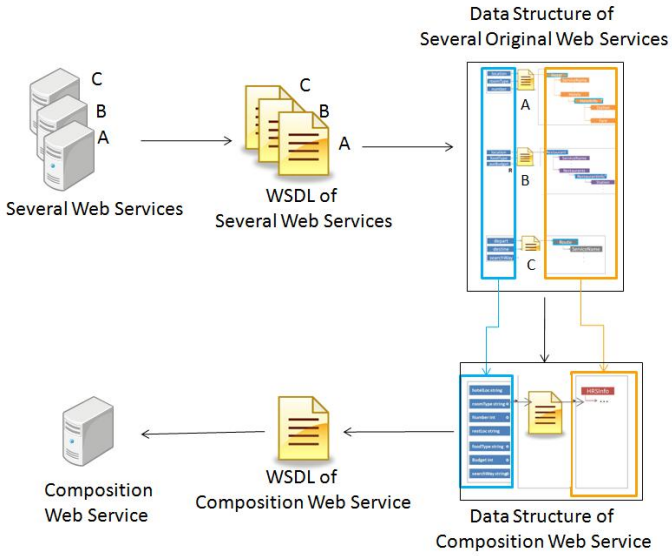

図1 Web サービス合成プロセスの概要
Figure 1 Outline of Web service composition processes.

3.1 Web サービスのデータ構造

Web サービスのデータ構造は入力と出力のデータ構造から構成される．各データはデータ名，フィールド名，データ型，個数情報を含む．データ型は原子型か,ユーザ定義型であり，個数情報は要素の最大値と最小値を表し，0-1，1-1，0-unbounded 3 つの標準パターンがある．図 2 に WSDL 記述例，図 3 にその WSDL 記述から生成されたデータ構造図を示す．この構造図では，ジャクソン木[8]のように，“○”で省略可（0-1），“*”で繰返し(0-unbounded)を表す．

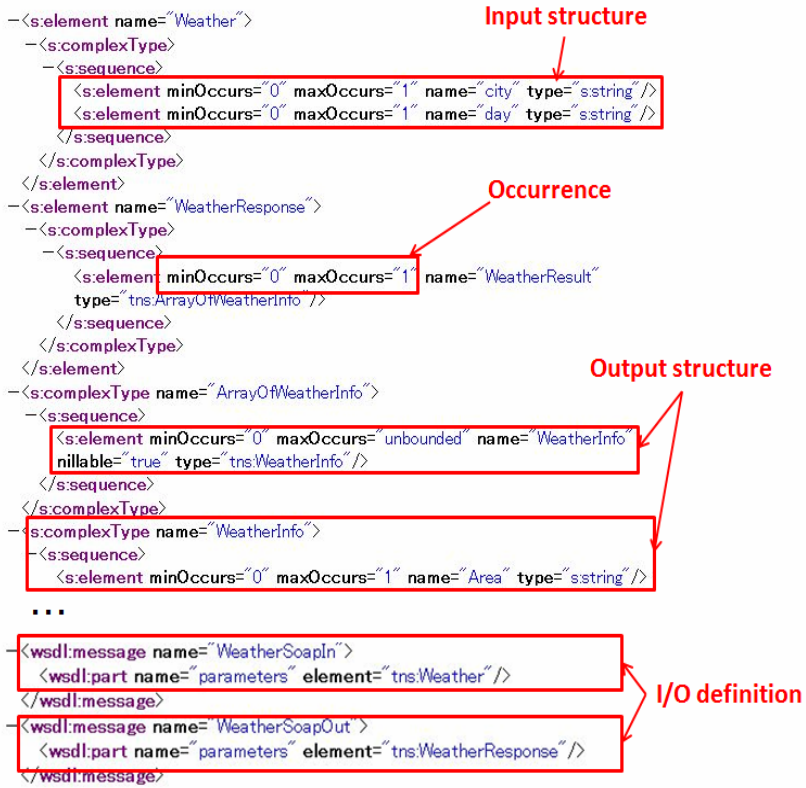

図2 WSDL 記述例
Figure 2 WSDL description.

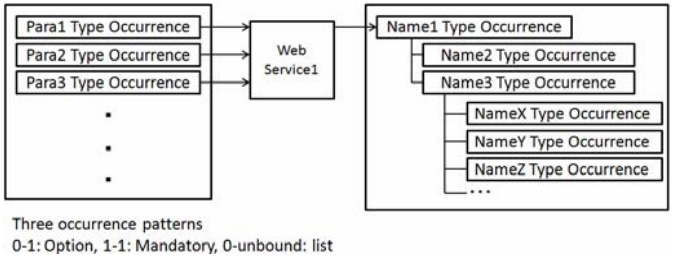

図3　Web サービスデータ構造図
Figure 3 Data structure diagram.

3.2 Web サービス合成記述

既存の Web サービスのデータ構造を定めた後，データ構造の対応関係により，合成 Web サービスを規定する．

(1) **全体の合成図**

Web サービスの全体合成図を図 4 に示す．図で，既存の Web サービスは合成サービスの中にあり，入力と出力は合成サービスの外にある．入力の対応関係により，既存の Web サービスのそれぞれの入力が，どこから（合成 Web サービスの入力，または他の既存 Web サービスの出力等）得られるかを規定する．また，出力の対応関係で，合成 Web サービスの出力の部分が，どの既存の Web サービスから得られるかを規定する．これらの対応関係から，システムは自動的にプログラムを生成する．

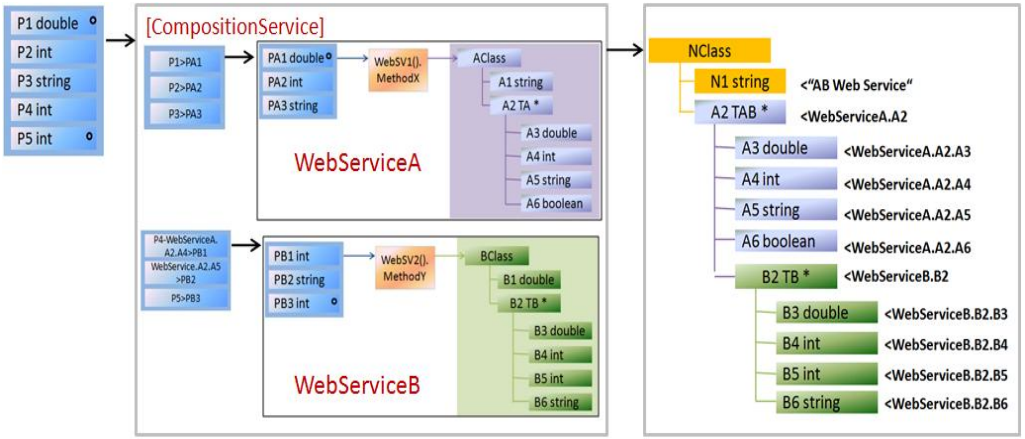

図4　Web サービス合成図
Figure 4 Whole composition Generation of input correspondence diagram.

(2) **入力の対応関係**

入力側の対応関係は，既存 Web サービスの入力がどこから得らるのかを規定する．既存の Web サービスの入力データは，定数，合成 Web サービスの入力，他の既存 Web サービスの出力，またはそれらの計算の結果から得られる．図 5 に入力対応関係の例を示す．図で，合成 Web サービスの入力と既存 Web サービス入力の間にあるボックスは，入力の対応関係を規定したものである．ここで，“A>B”は既存 Web サービスの入力 B のデータは，合成サービスの A から得られることを示す．入力の対応関係を決めることで，合成 Web サービスの入力が定まる．

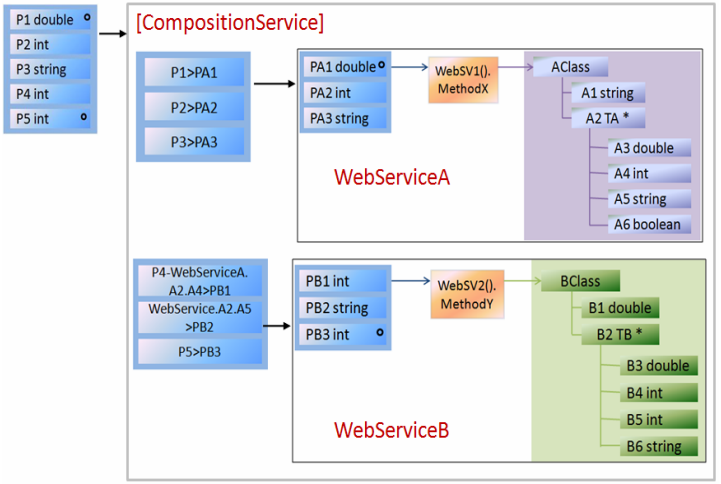

図5 入力の対応関係
Figure 5 Generation of input correspondence.

(3) **出力の対応関係**

合成 Web サービスの出力データ構造は，いくつかの既存 Web サービスの出力データ構造の合成から得られる．まず，他の既存 Web サービスの出力を使わない，メイン Web サービスを決める．図 4 では，WebServiceA がメイン Web サービスになる．他の既存 Web サービスの出力データがメイン Web サービスの出力データ構造に追加されることで，データ構造の合成が規定される．このようにデータ構造の合成を繰り返し，最後に不必要なデータフィールドを削除したり，新たなデータフィールドを追加して，求める合成 Web サービスの出力を得る．例えば，図 4 では，合成 Web サービスの名前のために String 型の N1 が追加され，WebServiceA の A1 と WebServiceB の B1 が削除されている．図 6 に，このようにして得られた階層的なデータ構造を示す．

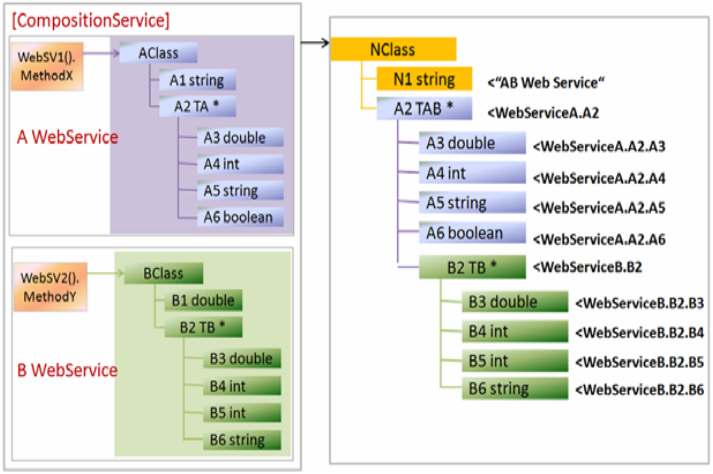

図6　出力の対応関係
Figure 6 Generation of output correspondence.

階層のトップレベルには合成サービスの戻り値型のクラス名が書かれる．出力データ構造の各フィールドの元になるデータは，図 6 のように“<”で示される．ここでは，定数と既存 Web サービスの一部の例を示している．なお，これらの関係は，現在開発中のグラフィックツールではデータフィールド間の矢印によって示される．

4. プログラム生成

最終的な合成 Web サービスのプログラムはデータの対応関係から自動的に生成される．プログラムは合成 Web サービスで使うデータ構造を表すクラスと，合成 Web サービスの処理を行うスタティックメソッドからなる．なお，既存の Web サービスを呼出すプログラム部分は，WSDL 記述から生成する．

4.1 合成 Web サービスで使うデータ構造のクラス

合成 Web サービスで使うデータ構造に対して，C#のプロパティだけからなるクラスが生成される．図 4 の例では，合成 Web サービスの出力で使われる 3 つの型，NClass，TAB，TB に対するクラスが生成される．NClass は string 型の N1 と TAB 型の A2 を，TAB は 4 つの primitive 型のフィールドと TB 型の B2 を，TB は primitive 型のフィールドのみを持つ．

4.2 合成 Web サービス処理のクラス

合成 Web サービスのプログラムは，その出力データ構造に沿って生成する．まず，クラスヘッダーを生成し，出力データ構造のトップ要素に対応したメソッドヘッダーを生成する．その後，出力のそれぞれの要素毎にプログラムコードを生成する．

(1) **クラスの骨組み**

図 4 の合成 Web サービスに現れるクラス名とメソッド名を使って，以下のような合成サービスのクラスとメソッドの骨組みを生成する．ここで，メソッドの引数は合成 Web サービスの入力，戻り値型は出力のトップ要素により決める．メソッドの中身のコード生成の前に，変数名の一部に使われるレベル数を 0 に初期化しておく．

```
public class ClassName
{
 public static OutputDataClass MethodName
 ( InputParameters )
 {
 OutputDataClass x0 = new OutputDataClass();
 - - -
 return x0;
 }
}
```

(2) **プリミティブ型のフィールドの処理**

出力データ要素がプリミティブ型の場合は，簡単な代入文を生成する．もしそのデータソース（値の元になるデータ）が定数なら，

xi.フィールド名=定数;

を生成する（i はレベル数）．例えば，図 4 で N1 のデータソースは"ABWebService"という定数で，レベルは 0（初期値のまま）であるので，N1 に対して，

x0.N1 = "ABWebService";

を生成する．

もしデータソースが既存の Web サービスの要素である場合，

xi.フィールド名=yi.データソースフィールド名;

を生成する．ここで，yi はプリミティブでないデータの処理に出てくるデータソース変数名である．

(3) **プリミティブでない型のフィールドの処理**

プリミティブでない出力フィールドのデータソースは，既存 Web サービスの出力の一部でなければならない．よって，最初にデータソースに対応する既存 Web サービスの呼出しを生成する．プリミティブでない型の処理の初めに，レベル数を 1 増加し，最後に 1 減少する．

① Web **サービス呼び出し**

データソースが既存 Web サービスの出力の場合，Web サービス呼出を生成する．これは最初に既存 Web サービスの出力が現れた時に行われる．例えば，図 4 では"A2 TAB *"のフィールド（データソースは WebServiceA.A2）に対して，WebServiceA の呼出しを生成する．呼出パラメータは 3.2(2)で述べた入力の対応を見て生成する．例えば，図 4 の WebServiceA の呼出は，

(new WebSV1()).MethodX(P1, P2, P3);

となる．また，WebServiceB の呼出は，

(new WebSV2()).MethodY(P4 - y1.A4, y1.A5, P5);

となる．ここで，呼出のパラメータに出てくるべき WebServiceA.A2 は，以下に述べる WebServiceA.A2 リストの繰返し処理で使われる繰返し変数 y1 に置き換える．

② 繰返し処理

プリミティブでないフィールドが要素の繰返し（"*"マーク付き）をもつ場合，以下のような繰返しのコードを生成する．

```
List<OutputClass> zi = new List<OutputClass>();
foreach( InputClass yi in WebServiceInvocation.part )
{
  OutputClass xi = new OutputClass ();
  - - -
  zi.add( xi );
}
xi-1.fieldName = zi;
```

繰返し本体には，"xi.フィールド名=yi.フィールド名;"のような代入文を生成することになる．例えば，図 4 の"A2 TAB * < WebServiceA.A2"の部分に対して以下のコードを生成する．

```
List<TAB> z1 = new List<TAB>();
foreach( TA y1 in(new WebSV1()).MethodX(P1, P2, P3) . A2 )
{
  TAB x1 = new TAB();
  - - -
  z1.add(x1);
}
x0.A2 = z1;
```

③ 繰返しのない処理

プリミティブでないフィールドが繰返しを持たない場合は，List<>は使われないで，以下のようなコードを生成する．

```
InputClass yi = WebServiceInvocation . part ;
OutputClass xi = new OutputClass ();
- - -
xi-1.fieldName = xi;
```

繰返し処理と同様，上のコードは"xi.フィールド名=yi.フィールド名;"のような代入文を含む．

図 7 に図 4 に対応して生成されるプログラムコードを示す．

```
public class CompositionService
{
  public static NClass ABWebService( double P1, int P2, String P3, int P4, int P5 )
  {
    NClass x0 = new NClass();
    x0.N1 = "ABWebService";
    List<TAB> z1 = new List<TAB>();
    foreach( TA y1 in(new WebSV1()).MethodX(P1, P2, P3).A2 )
    {
      TAB x1 = new TAB();
      x1.A3 = y1.A3;
      x1.A4 = y1.A4;
      x1.A5 = y1.A5;
      x1.A6 = y1.A6;
      List<TB> z2 = new List<TB>();
      foreach ( TB y2 in(new WebSV2()).MethodY (P4-y1.A4, y1.A5, P5).B2)
      {
        TB x2 = new TB();
        x2.B3 = y2.B3
        x2.B4 = y2.B4;
        x2.B5 = y2.B5;
        x2.B6 = y2.B6;
        z2.add(x2);
      }
      x1.B2 = z2;
      z1.add(x1);
    }
    x0.A2 = z1;
    return x0;
  }
}
```

図7　生成されたプログラムコード
Figure 7 Whole generated program codes.

5. 例

提案方式の評価のため，比較的大きな Web サービスを本方式に従って生成した．まず，既存の Web サービスとして，ホテル検索（HotelHS），レストラン検索（RestHS），経路検索（RouteHS）を使う．ホテル検索サービスは，場所，部屋タイプ，宿泊人数からホテル情報のリストを，レストラン検索サービスは，場所，食事タイプ，値段の平均からレストラン情報のリストを，経路検索サービスは出発点，到着点，交通手段により可能な経路のリストを出力する．目的の合成 Web サービスはこれらのサービスを合成し，近くのレストランとそこへの経路情報を含むホテル情報のリストを出力する．

5.1 入力の対応関係

もとの Web サービスのそれぞれにつき，入力の対応関係を定める．ホテル検索サービスの 3 つの入力パラメータは location, roomType，number であり，それぞれ，合成サービスの hotelLoc, roomType，number に対応させる．レストラン検索サービスの 3 つの入力パラメータのうち location と foodType は合成サービスの restLoc, foodType に対応させ，aveBuget は，合成サービスの予算(budget)からホテル料金（HotelWS. Hotels.Fare）を引いた値を対応させる．また，経路検索サービスは depart, destine, searchWay の 3 つの入力パラメータがあるが，それぞれホテルの近くの駅（HotelWS. Hotels.Station），レストランの近くの駅（RestWS.Rests.Station）と，合成サービスの入力の交通手段（searchWay）を対応させる．ここで searchWay は，電車，地下鉄，バス，タクシー等の交通手段を記述する．

表 1 入力の対応関係
Table 1. Relation of input

Hotel.location←hotelLoc
Hotel.roomType←roomType
Hotel.number←number
Rest.location←restLoc
Rest.foodType←foodType
Rest.aveBudget←budget-HotelWS.Hotels.Fare
Route.depart←HotelWS.Hotels.Station
Route.Destine←RestWS.Rests.Station
Route.searchcWay←searchWay

ここでは，合成サービスの要求者は主にホテルの情報を知りたいので，ホテル検索サービスをメインサービスとして呼び出した．もし，要求者が，主に食べ物に興味があって，その後，適切なホテルを探したいなら，レストラン検索サービスを最初に呼び出すことになる．この場合，レストラン費用の残りの予算をホテルの料金データとして，ホテル検索サービスを呼び出す．

5.2 出力の対応関係

出力の対抗関係は，合成サービスの出力の各項目がどこから得られるかについて規定する．このデータソースとしては，定数か，既存の Web サービスの出力の一部が使われる．表 2 に出力の対応関係を示す．

表 2 出力の対応関係
Table 2. Relation of Output

ServiceName←”HRSWebSerivce”
SearchTime←DateTime.Now
HotelsRests←HotelWS.Hotels
HotelsRests.Name←HotelWS.Hotels.Name
HotelsRests.State←HotelWS.Hotels.State
・・・
RestsRoutes←RestWS.Rests
RestsRoutes.Name←RestWS.Rests.Name
・・・

出力のトップレベルの名前（HRSInfo）は，合成サービスの戻り値型に使われるクラス名を示す．ServiceName フィールドには定数"HRSWebService"を設定し，SearchTime フィールドには C#の標準機能による現在時間（DateTime.Now）を設定する．HotelRests はプリミティブでないフィールドであり，ホテル検索サービス（HotelWS）の出力の一部からデータを得る．そこで，ホテル検索サービスの呼び出しが必要になる．HotelRests の各要素フィールドに対して，ホテル検索サービスの出力である Hotels の関連した要素フィールドがデータソースとして対応する．その後に，プリミティブでないフィールド RestsRoutes が来る．そのデータソースはレストラン検索サービス（RestWS）を呼出した結果の出力である Rests 部分が対応する．

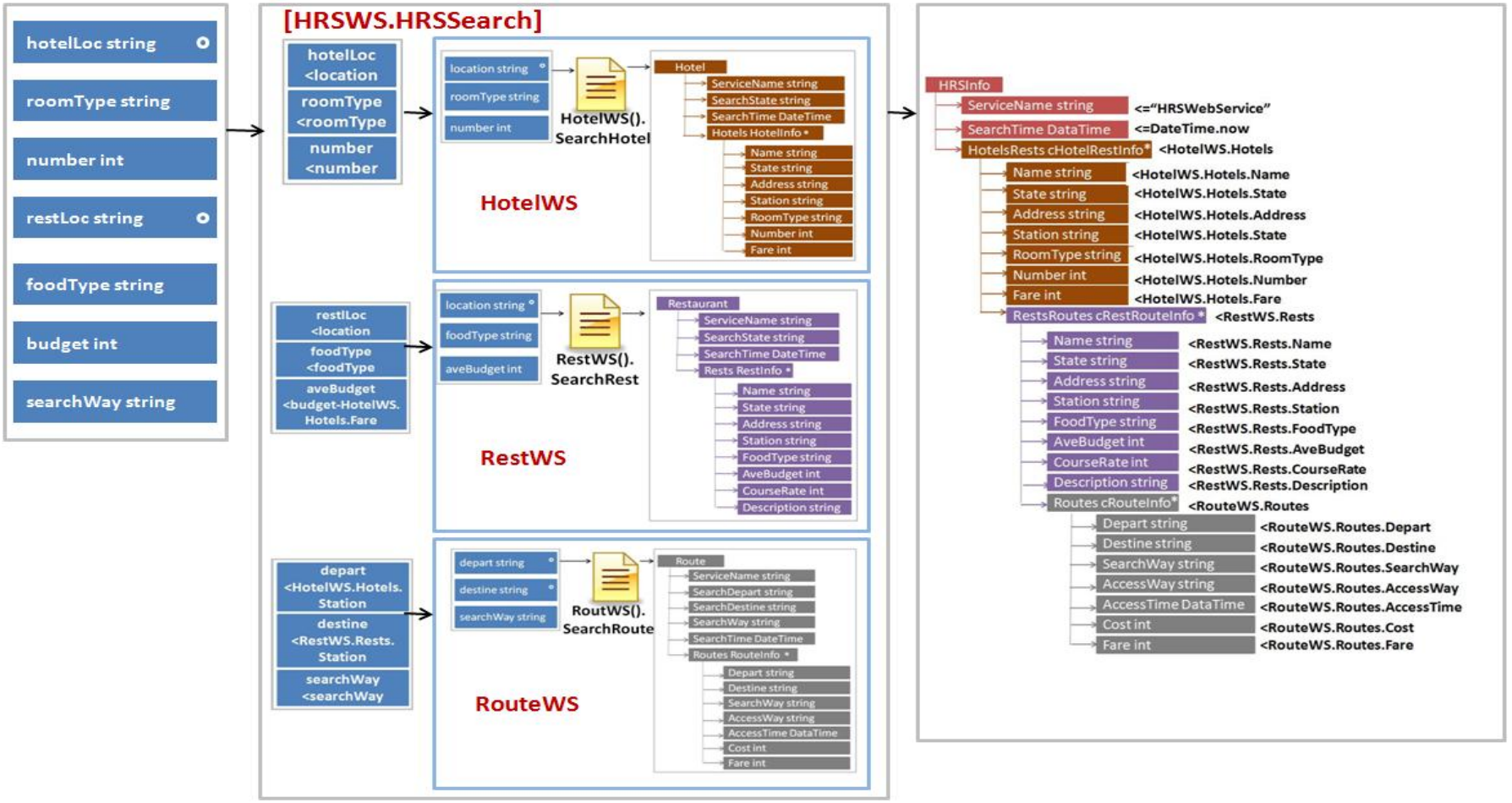

図8 例の全体合成図
Figure 8 The whole composition diagram for the example

5.3 合成図

全体の合成図の結果を図 8 に示す．図は，左に合成サービスの入力データ，中央に入力の対応関係を持った既存 Web サービス，そして右に出力の対応関係を含む合成サービスの出力データを表示する．

5.4 プログラム生成

図 8 に対応して生成されるプログラムは，HRSInfo, cHotelRestInfo, cRestRouteInfo, cRouteInfo の 4 つのデータ構造に対応するクラスと，1 つのサービス実行クラス HRSWS からなる．HRSWS クラスは HRSInfo を戻り値型とし，図 8 の 7 つの入力を引

数とする HRSSearch メソッドを含む．このメソッドはホテル検索サービスの Hotels，レストラン検索サービスの Rests，経路検索サービスの Routes に対するネストした 3 つの foreach ループを持つ．図 9 に，図 8 の合成仕様から生成されたプログラムコードの部分を示す．

```
public class HRSWS
{
    public static HRSInfo HRSSearch
        (string hotelLoc, string roomType, int number,
     string foodType, int budget, string searchWay )
    {
        HRSInfo x0 = new HRSInfo ();
        x0.ServiceName = "HRSWebService";
        x0.SearchTime = DateTime.Now;
        List<cHotelRestInfo> z1 = new List<cHotelRestInfo>();
        foreach (HotelInfo y1 in
        (new HotelWS()).SearchHotel
        (hotelLocal, roomType, number).Hotels)
        {
            cHotelRestInfo x1 = new cHotelRestInfo();
            x1.Name = y1.Name;
            x1.State = y1.State;
      x1.Address = y1.Address;
            x1.Station = y1.Station;
            x1.RoomType = y1.RoomType;
            x1.Number = y1.Number;
            x1.Fare = y1.Fare;

            List<cRestRouteInfo> z2 = new List< cRestRouteInfo >();
            foreach (RestInfo y2 in
           (new RestWS()).RestSearch
           (restLocal, foodType, Budget – y1.Fare).Rests)
              {
                cHotelRestInfo x2 = new cHotelRestInfo();
                x2.Name = y2.Name;
                - - -
                - - -
                - - -
              }
              x1.Restaurants = z2;
              z1.add(x1);
    }
    x0.HotelsRests = z1;
    return x0;
    }
}
```

図9　例により生成されたプログラムコードの一部分
Figure 9 A part of generated program codes of the sample

6. おわりに

データのリストを出力する既存 Web サービスを合成する方式について述べた．そこでは，Web サービスの出力リスト中の１つのデータが別の Web サービスの入力となって，サービスを合成する．このため，2 つめの Web サービスをリストの要素分だけ繰返し呼出す制御が必要になる．BPEL のワークフローを使えば，このような繰返しを含む Web サービス合成を記述することができるが，それはプログラミングと同様の複雑な作業になる．我々の方式では，ワークフローのような記述
ではなく，既存サービスと合成サービスの間のデータ構造間の対応関係によって合成を規定する．本方式の有効性を示すため，ホテル検索，レストラン検索，経路検索の 3 つのサービスを合成する例を掲げた．

現在開発中の本方式による自動合成システムは，既存 Web サービスの WSDL 記述からデータ構造図を生成する WSDL 解析部，既存 Web サービスの出力データから合成 Web サービスの出力データを合成し，さらにデータ間の関連を示す矢印線を描くための合成図エディタ部，そして合成図から C#のプログラムや構成ファイル等を生成するプログラム生成部からなる．

今後の課題として，提案方式と開発中のシステムが，より実際的で複雑な Web サービス合成に対する実行可能性や有効性を評価する所存である．

参考文献

1) Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall PTR, 2005/8/12.
2) Juric, M. B.: Bussiness Process Execution Language for Web Service, Packt Publishing, 2004/10/31.
3) Rao, J., and Su, X.: A Survey of Automated Web Service Composition Methods, *SWSWPC*, Springer Berlin / Heidelberg, 2004, pp. 43-54.
4) Narayanan, S. and McIlraith, S.: Simulation, verification and automated composition of Web service. In Proceedings of the 11th International World Wide Web Conference, Honolulu, Hawaii, USA, May 2002. ACM. presentation available at http://www2002.org/presentations/narayanan.pdf.
5) Wu, D., Sirin, E., Hendler, J., Nau, D. and Parsia. B.: Automatic Web services composition using SHOP2. In Workshop on Planning for Web Services, Trento, Italy, June 2003.
6) Casati, F., Ilnicki, S. and Jin. L.: Adaptive and dynamic service composition in EFlow. In Proceedings of 12th International Conference on Advanced Information Systems Engineering(CAiSE), Springer Verlag, Stockholm, Sweden, June 2000.
7) Booth, D.: et al., Web Services Description Language (WSDL), Version 2.0 Part 0: Primer, June. 2007, http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626.
8) Jackson, M.: Principles of Program Design, Academic Press, 1975.