
Service adaptation through trace inspection

Antonio Brogi and Razvan Popescu*

Department of Computer Science,
University of Pisa,
Pisa, 56127, Italy
E-mail: popescu@di.unipi.it
*Corresponding author

Abstract: Service-oriented computing strongly supports the development of future distributed business applications through the use of (web) services. Due to the heterogeneous and evolving nature of business processes, *service adaptation* is necessary in order to overcome mismatches between the interacting parties. Our long-term objective is to develop a general methodology for service adaptation capable of suitably overcoming semantic and behaviour mismatches in view of business process integration within and across organisational boundaries. In this paper, we show how to adapt a service in order to fulfil a client query requesting a service with certain inputs and outputs. The proposed technique relies on inspecting service execution traces and it generates a service contract tailored to the client needs. Service contracts include a description of the service behaviour (expressed by a YAWL workflow) as well as an (ontology-annotated) signature.

Keywords: web services; service discovery; service adaptation; YAWL.

Reference to this paper should be made as follows: Brogi, A. and Popescu, R. (2007) 'Service adaptation through trace inspection', *Int. J. Business Process Integration and Management*, Vol. 2, No. 1, pp.9–16.

Biographical notes: Antonio Brogi is a Professor at the University of Pisa, Italy. His research interests include coordination and adaptation of software services.

Razvan Popescu is pursuing PhD at the University of Pisa, working on aggregation and adaptation of (web) services.

1 Introduction

Service-oriented computing (Papazoglou and Georgakopoulos, 2003) is emerging as a new promising computing paradigm that centres on the notion of *service* as the fundamental element for developing future business applications. The platform-neutral nature of services creates the opportunity for building composite business processes by integrating existing elementary or complex services, possibly offered by different service providers (Yang, 2003). In this scenario, two prominent issues involved in the development of next generation heterogeneously distributed software applications can be roughly synthesised as discovering available services that fulfil a given request and suitably adapting and aggregating such services to build a needed business application.

Currently, WSDL (2001) and UDDI (2000) are the universally adopted standards for web service description and discovery, respectively. Providers publish (purely syntactic) WSDL advertisements to UDDI registries (constructed in the style of yellow pages) which in turn provide clients with keyword- or taxonomy-based service discovery capabilities. On the one hand, WSDL descriptions do not include any *semantic information* and hence they are not 'self-described' in a machine-interpretable way. This severely limits the quality of the discovery results as the matched services may

not necessarily offer the requested functionality, and hence fully -automated service discovery becomes unfeasible. On the other hand, WSDL descriptions lack *behaviour information*. A direct consequence of this is that service compositions may lock during execution. Stated differently, without any protocol information (e.g. order of messages sent/received) no guarantee on the behaviour of service compositions can be ensured.

Various proposals have been put forward in order to enhance service descriptions. WSDL-S (Akkiraju et al., 2005), OWL-S (2004), SWSO (2005), WSMO (2005) or METEOR-S (Rajasekaran et al., 2004) annotate services with semantic information. (BPEL4WS Coalition., 2003), WSCDL (2005), METEOR-S (Aggarwal et al., 2004), OWL-S Coalition (2004), SWSO Coalition (2005) or recently YAWL (van der Aalst and ter Hofstede, 2005) add protocol information to service descriptions. All the above proposals can be in principle exploited for improving the accuracy of service matching, for extending the properties of service compositions as well as for automatising both processes.

On the other hand, adding semantics and/or behaviour information to service descriptions leads to distinguishing *more* service descriptions and hence to matching *less* services or service compositions with a given query. In this perspective, owing to the heterogeneous and evolving

nature of business processes, *service adaptation* is hence needed to overcome the (unavoidable) mismatches between the interacting parties.

Our long-term objective is to develop a general methodology for service adaptation capable of suitably overcoming both semantic and behaviour mismatches. In this paper, we tackle the problem of discovering services that can be adapted to fulfil a client request. We consider a registry R of *service contracts* where each contract includes a description of the service behaviour (expressed by a YAWL workflow (van der Aalst and ter Hofstede, 2005)), as well as an (ontology-annotated) signature. We also consider a query Q requesting a service with certain inputs and outputs.

The adaptation process roughly consists of:

- 1 matching a candidate service S in R by checking the compatibility of the execution traces of S with respect to the given query Q and
- 2 generating the contract of the service S adapted so as to fulfil the query Q .

The rest of this paper is organised as follows. We first introduce service contracts (Section 2.1) and give an overview of the phases of the adaptation methodology (Section 2.2), followed by a motivating example (Section 2.3). We then describe the service matching phase (Section 2.4) and then the contract generation phase (Section 2.5). In Section 3, we briefly overview related work. Finally, Section 4 presents some concluding remarks.

2 Service adaptation

2.1 Service contracts

We consider services that are described by *contracts* (Meredith and Bjorg, 2003), and we argue that contracts should in general include different types of information:

- 1 ontology-annotated signatures
- 2 behaviour and
- 3 extra-functional properties.

Following (OWL-S Coalition, 2004), we argue that WSDL signatures should be enriched with ontological information (e.g. expressed with OWL (McGuinness and van Harmelen, 2004) or WSDL-S (Akkiraju et al., 2005)) to describe the semantics of services, necessary to automate the process of overcoming signature mismatches as well as service discovery, adaptation and composition. Still, the information provided by ontology-annotated signatures is necessary but *not* sufficient to ensure a correct interoperation of services (e.g. absence of locks). Following (Meredith and Bjorg, 2003), we argue that contracts should also expose a (possibly partial) description of the interaction protocols of services. We argue that YAWL (van der Aalst and ter Hofstede, 2005) (see below) is a good candidate to express service behaviour as it has a well-defined formal semantics and it supports a number of workflow patterns. Finally, we argue that service contracts should expose, besides annotated signatures and behaviour, also so-called extra-functional properties, such as performance, reliability or security (we will not, however,

consider these properties in this work and leave their inclusion into the adaptation framework to future work).

We intend to build an adaptation framework capable of translating the behaviour of a service described using existing process/workflow modelling languages (e.g. BPEL (2003), OWL-S (2004), etc.) into equivalent descriptions expressed through an abstract language with a well-defined formal semantics and vice-versa. Two immediate advantages of using such an abstract language are the possibility of developing formal analyses and transformations, independently of the different languages used by providers to describe the behaviour of their services. We consider that YAWL (van der Aalst and ter Hofstede, 2005) is a promising candidate to be used as an abstract workflow language for describing service behaviour. YAWL is a new proposal of a workflow/business processing system, which supports a concise and powerful workflow language and handles complex data, transformations and web service integration. YAWL defines 20 most used workflow patterns gathered by a thorough analysis of a number of languages supported by workflow management systems. These workflow patterns are divided in six groups (basic control-flow, advanced branching and synchronisation, structural, multiple instances, state-based and cancellation).¹ YAWL extends Petri Nets by introducing some workflow patterns (for multiple instances, complex synchronisations and cancellation) that are not easy to express using (high-level) Petri Nets. Being built on Petri Nets, YAWL is an easy to understand and to use formalism. With respect to process algebras, YAWL features an intuitive (graphical) representation of services through workflow patterns. Furthermore, as illustrated in van der Aalst (2004), it is likely that a simple workflow which is troublesome to model for instance in π -calculus may be instead straightforwardly modelled with YAWL. A thorough comparison of workflow modelling with Petri Nets versus π -calculus may be found in van der Aalst (2004). With respect to the other workflow languages (mainly proposed by industry), YAWL relies on a well-defined formal semantics. Moreover, not being a commercial language, YAWL supporting tools (editor, engine) are freely available.

2.2 Bird's eye-view of the adaptation methodology

The service adaptation methodology we propose consists of four main phases:

- 1 *Service translation*: this preliminary phase deals with translating real-world service descriptions (e.g. BPEL + semantics, OWL-S, etc.) into equivalent service contracts using YAWL as an abstract workflow language for expressing behaviour and OWL, for example, for expressing semantic information. One may note that such a translation may be done off-line and hence it is not a burden for the adaptation process (a thorough analysis of how to transform BPEL specifications into workflow patterns can be found in Wohed et al. (2003)).
- 2 *Service matching*: the first step of this phase – to be done off-line – derives from each service workflow a *trace table* which associates each service process with preconditions, inputs and outputs. The compatibility

between the trace table of the query and the trace table of each service is then checked to determine the services (if any) that can be adapted to fulfil the query. It is worth noting that the trace compatibility check is ontology-aware in that it copes with exact/plug in/subsumes semantic matches Paolucci et al. (2002).

- 3 *Contract generation*: the workflow of a candidate service found by the previous phase (if any) is then modified in order to generate the contract of the service adapted to fulfil the client query. Informally, the service workflow is modified so as to specify a refined behaviour of the initial service that enforces the needed adaptation.
- 4 *Service deployment*: finally, the generated contract can be deployed as a real-world web service (i.e. described using OWL-S, BPEL + semantics, etc.). The client will hence view the requested functionality as a new web service that can now be discovered and further adapted to other requests. This operation is intuitively the inverse of the operation done during the service translation phase.

As mentioned in Section 1, we will focus on phases (1) and (2) in the rest of this paper.

2.3 Motivating example

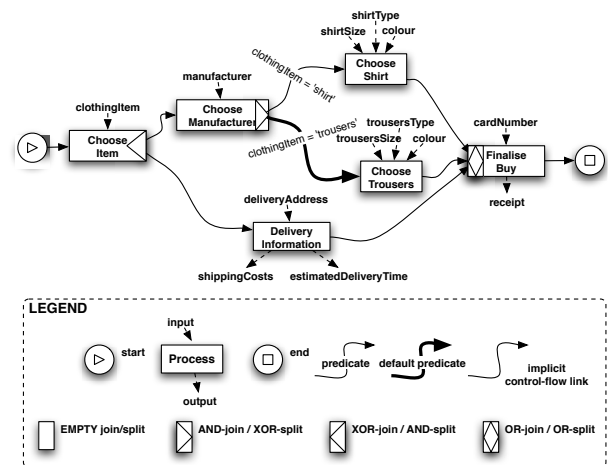
Due to the particular usage of YAWL to model web services/business processes, we shall use the term ‘process’ to denote YAWL tasks as well as ‘service’ to denote a workflow specification.

Consider the example in Figure 1 which will be used as a basis for describing the proposed methodology. As one may note, a service consists of processes and control-flow links among them. There are two specific processes (which we call *start* and *end*) corresponding to the YAWL *input* and *output condition*, respectively. A process Q is to be executed after another process P if there is a directed link from P to Q . Processes employ one *join* and one *split* construct. A join or split control construct may be one of the following: AND, OR, XOR or EMPTY. Intuitively, the join specifies ‘how many’ processes *before* P are to be terminated in order to execute P , while the split construct specifies ‘how many’ processes *after* P are to be executed. The EMPTY-join (split) is used when *only one* process execution precedes (follows, respectively) the execution of P . We graphically represent parameters as well. YAWL uses predicates in the form of logical expressions to decide the control-flow in the case of XOR- and OR-splits.

Clothing Shop is a service that sells shirts and trousers. When invoked, the service firstly executes the Choose Item process, for which the client has to provide the item she is interested in (either ‘shirt’ or ‘trousers’). The execution continues with both Choose Manufacturer and Delivery Information the AND-split of Choose Item. Choose Manufacturer inputs the desired designer and based on the item to be bought, it enables *only one* of the following two processes: Choose Shirt if the requested item is a shirt or Choose Trousers otherwise. Please note its XOR-split as well as the predicates annotating the respective control-flow links. Both processes input the type (e.g. ‘T-shirt’ or ‘jeans’), size (e.g. ‘M’ or ‘33’) and colour

(e.g. ‘black’) of the requested item. Delivery Information is a process which inputs the address intended for delivery. Such information may be submitted by the client at any moment of the purchase (after choosing the item yet prior to the payment). Delivery Information outputs the shipping costs as well as the estimated delivery time. Last but not least, *Finalise Buy* employs an OR-join in order to wait for an item to be chosen as well as for the delivery information to be available. It inputs a credit card number and generates the purchase receipt. Please note that the example is not supposed to present a software masterpiece and in order to keep it manageable we omit scenarios in which, for example, a desired item is not available, payment issues and so on.

Figure 1 Clothing Shop service selling shirts and trousers



Let us assume now that a client desires a service which sells trousers only. She might search one by issuing the following query:

- *requested inputs*: {jeans, designer, address, size, dark-blue, cardNumber}
- *requested outputs*: {shippingCosts, receipt}.

For simplicity we shall assume that both query and services use the same parameter ontology and that there is an exact/plug-in/subsumes match (Paolucci et al., 2002) between the following parameter pairs²: jeans and clothingItem, jeans and trousersType, designer and manufacturer, address and deliveryAddress, size and trousersSize, dark-blue and colour, cardNumber and cardNumber, shippingCosts and shippingCosts and finally receipt and receipt.

It is easy to see that the service does not match the given query, since the service requires more inputs than those specified in the query (viz. shirtSize and shirtType). However, the Clothing Shop service could be in principle adapted so as to inhibit its capability of selling shirts and convert it into a ‘trousers shop’ service.

In the rest of this paper, we show how the proposed adaptation methodology performs an ontology-aware matching at the level of subservices (viz. processes) and succeeds in adapting the initial service so as to fully satisfy the client query.

2.4 Service matching

The objective of this phase is to determine whether there are services in the registry that can be adapted to fully match the given query.

2.4.1 Deriving the trace table

Firstly, each service workflow is analysed in order to generate a Trace Table (TT) which associates each service process with preconditions, inputs and outputs. More precisely, each process P is associated with a set of triples of the form $\langle \text{Preconditions, Needed Inputs, Generated Outputs} \rangle$, where Preconditions represents the set of data and control constraints that must be satisfied to be able to execute P and the other processes executed so far (in that execution trace). Needed Inputs is the set of inputs that are requested by P together with the inputs of the other processes that have been executed so far. Similarly, Generated Outputs is the set of outputs generated by P together with the other processes executed so far.

We generate the TT for a service from its Reachability Graph (RG). The RG of a service can be derived from its YAWL workflow augmented with explicit conditions. An example of such workflow may be seen in Figure 2. It is worth noting that YAWL conditions can be represented as (Petri net) places holding tokens (van der Aalst and ter Hofstede, 2005). On the one hand, tokens are placed into places by firing processes depending on their split constructs and on the YAWL predicates (if present). For processes with EMPTY- (AND-) splits, YAWL considers implicit conditions and a token is generated for (all) the output place(s). In the case of XOR- or OR-splits, YAWL uses predicates to determine which output places will receive tokens. All predicates of such a split are ordered (by the workflow designer) and one is chosen as default (with lowest priority). For a XOR-split, a token is sent on the link corresponding to the predicate with the lowest order that evaluates to true. For an OR-split, a token is sent along all links whose predicates evaluate to true. For both splits, if all predicates are false then a token is sent along the default link only. On the other hand, places are used to enable processes for execution. If the process has an EMPTY-join then its input place has to contain a token. For an AND-join, all input places have to contain tokens. In the case of a XOR-join at least one input place has to have a token. Finally, according to van der Aalst and ter Hofstede (2005), if the process has an OR-join, it is enabled only when at least one of its input places contains a token and no other tokens can be placed in its remaining (empty) input places. For example in Figure 2, Finalise Buy is enabled if and only if C_7 and either C_5 or C_6 contain tokens. Hence, it cannot be enabled when only C_7 contains a token before a token will arrive at C_5 or C_6 . One should note that tokens cannot be added to both C_5 and C_6 during a workflow instance owing to the XOR-split of the Choose Manufacturer process.

Following (Wynn et al., 2005) we derive an RG having *markings* as nodes and *labelled arrows* as edges. A marking M consists of the set of all places containing tokens and it is denoted as $C_i + \dots + C_j$. An arrow states that the workflow execution state evolves from a marking M into a marking M' and it is labelled with the process that fires and – in the case of

OR- and XOR-splits – also with the places that receive tokens. Figure 3 shows the RG corresponding to the Clothing Shop workflow in Figure 2. For example, the arrow from the initial marking C_i to the marking $C_1 + C_2$ is labelled as Choose Item. This is to be read as: ‘from the initial marking containing a token in C_i only, the Choose Item process is enabled by consuming the token in C_i and by firing it produces a token in C_1 and another in C_2 ’. As one should note, in the markings circled in Figure 3 the Finalise Buy process cannot fire as it expects one more token to be placed in one of its empty input places.

Figure 2 Clothing Shop workflow with places/conditions

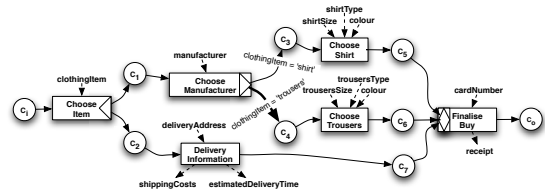
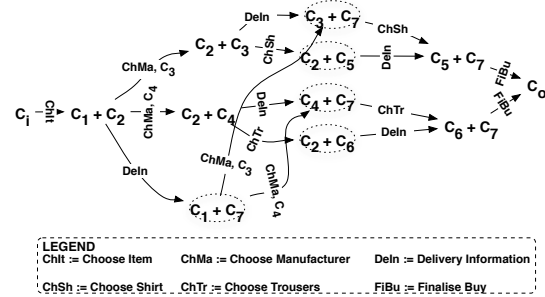


Figure 3 RG for the Clothing Shop service



RG is incrementally built by starting from the initial marking (that contains C_i only) and by looking for processes which can be enabled. Labelled arrows and new markings are successively added to the graph. One should note that checking whether a process having an OR-join can be enabled is done using the algorithm given in Wynn et al. (2005).

For example, by considering the marking C_i , we generate and add to RG the marking $C_1 + C_2$ and we label the arrow as Choose Item. Or, by assuming that the markings $C_3 + C_7$ and $C_4 + C_7$ have already been generated, and by considering the marking $C_1 + C_7$, we have noted that Choose Manufacturer can fire and it can place a token either in C_3 or in C_4 . Hence, we just add arrows from the $C_1 + C_7$ to $C_3 + C_7$ and to $C_4 + C_7$ and we label them as ‘Choose Manufacturer, C_3 ’ and ‘Choose Manufacturer, C_4 ’ respectively.

The process of generating the TT for a process P looks in the RG for all paths (i.e. traces) p originating in the initial marking and ending with a marking having an outgoing arrow labelled by P . The preconditions for the p execution of P are expressed as the set of all conditions (viz. places) in the markings of p . The set of needed inputs is obtained by taking the inputs of all processes labelling arcs of path p , together with the inputs of P . Similarly, the set of generated outputs consists of the outputs of all processes labelling arcs of path p , together with the outputs of P . For example, if we consider the arrow labelled as ‘Choose Manufacturer, C_3 ’ that originates in the marking $C_1 + C_2$, we

have to add to TT(Choose Manufacturer) the following entry: $\langle \{C_i, C_1, C_2\}, \{\text{clothingItem}, \text{manufacturer}\}, \emptyset \rangle$. If we consider the arrow labelled as Choose Shirt that originates in the marking $C_3 + C_7$ then we have to add to TT(Choose Shirt) the following entry: $\langle \{C_i, C_1, C_2, C_3, C_7\}, \{\text{clothingItem}, \text{manufacturer}, \text{deliveryAddress}, \text{shirtSize}, \text{shirtType}, \text{colour}\}, \{\text{shippingCosts}, \text{estimatedDeliveryTime}\} \rangle$ which actually synthesises the two different traces leading to the marking $C_3 + C_7$. The final TT obtained for our example is illustrated in Table 1.

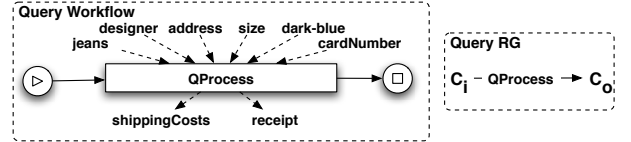
Table 1 TT of the *Clothing Shop* service

<i>Process Name:</i> $\langle \text{Preconditions, Needed Inputs, Generated Outputs} \rangle$
C_i : $\langle \emptyset, \emptyset, \emptyset \rangle$
<i>Choose Item:</i> $\langle \{C_i\}, \{\text{clothingItem}\}, \emptyset \rangle$
<i>Choose Manufacturer:</i> $\langle \{C_i, C_1, C_2\}, \{\text{clothingItem}, \text{manufacturer}\}, \emptyset \rangle$; $\langle \{C_i, C_1, C_2, C_7\}, \{\text{clothingItem}, \text{manufacturer}, \text{deliveryAddress}\}, \{\text{shippingCosts}, \text{estimatedDeliveryTime}\} \rangle$
<i>Delivery Information:</i> $\langle \{C_i, C_1, C_2\}, \{\text{clothingItem}, \text{deliveryAddress}\}, \{\text{shippingCosts}, \text{estimatedDeliveryTime}\} \rangle$ $\langle \{C_i, C_1, C_2, C_3\}, \{\text{clothingItem}, \text{manufacturer}, \text{deliveryAddress}\}, \{\text{shippingCosts}, \text{estimatedDeliveryTime}\} \rangle$ $\langle \{C_i, C_1, C_2, C_3, C_5\}, \{\text{clothingItem}, \text{manufacturer}, \text{deliveryAddress}, \text{shirtSize}, \text{shirtType}, \text{colour}\}, \{\text{shippingCosts}, \text{estimatedDeliveryTime}\} \rangle$ $\langle \{C_i, C_1, C_2, C_4\}, \{\text{clothingItem}, \text{manufacturer}, \text{deliveryAddress}\}, \{\text{shippingCosts}, \text{estimatedDeliveryTime}\} \rangle$ $\langle \{C_i, C_1, C_2, C_4, C_6\}, \{\text{clothingItem}, \text{manufacturer}, \text{deliveryAddress}, \text{trousersSize}, \text{trousersType}, \text{colour}\}, \{\text{shippingCosts}, \text{estimatedDeliveryTime}\} \rangle$
<i>Choose Shirt:</i> $\langle \{C_i, C_1, C_2, C_3\}, \{\text{clothingItem}, \text{manufacturer}, \text{shirtSize}, \text{shirtType}, \text{colour}\}, \emptyset \rangle$ $\langle \{C_i, C_1, C_2, C_3, C_7\}, \{\text{clothingItem}, \text{manufacturer}, \text{deliveryAddress}, \text{shirtSize}, \text{shirtType}, \text{colour}\}, \{\text{shippingCosts}, \text{estimatedDeliveryTime}\} \rangle$
<i>Choose Trousers:</i> $\langle \{C_i, C_1, C_2, C_4\}, \{\text{clothingItem}, \text{manufacturer}, \text{trousersSize}, \text{trousersType}, \text{colour}\}, \emptyset \rangle$ $\langle \{C_i, C_1, C_2, C_4, C_7\}, \{\text{clothingItem}, \text{manufacturer}, \text{deliveryAddress}, \text{trousersSize}, \text{trousersType}, \text{colour}\}, \{\text{shippingCosts}, \text{estimatedDeliveryTime}\} \rangle$
<i>Finalise Buy, C_o:</i> $\langle \{C_i, C_1, C_2, C_3, C_5, C_7\}, \{\text{clothingItem}, \text{manufacturer}, \text{deliveryAddress}, \text{shirtSize}, \text{shirtType}, \text{colour}, \text{cardNumber}\}, \{\text{shippingCosts}, \text{estimatedDeliveryTime}, \text{receipt}\} \rangle$ $\langle \{C_i, C_1, C_2, C_4, C_6, C_7\}, \{\text{clothingItem}, \text{manufacturer}, \text{deliveryAddress}, \text{trousersSize}, \text{trousersType}, \text{colour}, \text{cardNumber}\}, \{\text{shippingCosts}, \text{estimatedDeliveryTime}, \text{receipt}\} \rangle$

2.4.2 TT compatibility check

In order to match the requested query with the TT of a given service, we firstly express the query as a simple service contract whose workflow contains one process only (together with *start* and *end*). The inputs and the outputs of the process are as requested by the query. Figure 4 presents the workflow, the RG and the TT for the request described in Section 2.3.

Figure 4 Query workflow, RG and TT



C_i : $\langle \emptyset, \emptyset, \emptyset \rangle$.
$QProcess, C_o$: $\langle \{C_i\}, \{\text{jeans}, \text{designer}, \text{address}, \text{size}, \text{dark-blue}, \text{cardNumber}\}, \{\text{shippingCosts}, \text{receipt}\} \rangle$.

The compatibility check between the TT of the query Q and the TT of a service S consists of looking for a process P of S such that there exists a trace T in $TT(P)$ for which

- 1 the set of inputs needed by T is included in the set of inputs of Q and, dually
- 2 the set of outputs of Q is included in the set of outputs generated by T .

In other words, Q has to provide all inputs needed by T , and T has to generate all outputs desired by Q . One should note that the inclusion relation is ontology-aware. The query of our example matches the service in Figure 1 as for the second trace (top to bottom) in TT(Finalise Buy) for example, we have:

- 1 $\{\text{clothingItem}, \text{manufacturer}, \text{deliveryAddress}, \text{trousersSize}, \text{trousersType}, \text{colour}, \text{cardNumber}\} \supseteq \{\text{jeans}, \text{designer}, \text{address}, \text{size}, \text{dark-blue}, \text{cardNumber}\}$ and
- 2 $\{\text{shippingCosts}, \text{receipt}\} \supseteq \{\text{shippingCosts}, \text{estimatedDeliveryTime}, \text{receipt}\}$.

One should note that the preconditions set constraining T into fulfilling Q is $\{C_i, C_1, C_2, C_4, C_6, C_7\}$.

In order to provide a more user-friendly answer to the query we construct a logical expression from the set of preconditions of a trace. We can achieve this by firstly assigning a logical expression to each place of the workflow and then by taking the conjunction of all the conditions in the preconditions set of a trace. For instance, the above preconditions set constraining T into fulfilling Q might be expressed as '(clothingItem = 'trousers') or (not clothingItem = 'shirt')

Being inspired by the usage of YAWL predicates to enable processes (van der Aalst and ter Hofstede, 2005), we enhance the expressiveness of YAWL conditions by assigning them a logical expression as follows: for C_i , C_o or for an output place of a process having an EMPTY- or an AND-split we

consider an always ‘true’ condition (e.g. C_1 , C_2 , C_7 and so on). In the case of a XOR-split, we consider an output condition to be true provided ‘either the YAWL predicate for the corresponding link is true as well as the other lower-order predicates are false, or the corresponding predicate is the default one and all other predicates of the respective process are false’ (van der Aalst and ter Hofstede, 2005). For example, for C_3 we consider the following expression ‘(clothingItem = ‘shirt’ and not clothingItem = ‘trousers’) or (clothingItem = ‘shirt’ is default and not clothingItem = ‘trousers’)’, or simply ‘(clothingItem = ‘shirt’ and not clothingItem = ‘trousers’)’. Similarly, for C_4 we have ‘(clothingItem = ‘trousers’) or (not clothingItem = ‘shirt’)’. Hence, a token is placed into C_4 even if a client requests ‘shoes’ as item. Last, but not the least, for a process having an OR-split, we consider an output condition to be true if and only if ‘its corresponding predicate is true, or the respective predicate is the default one and all other predicates of the considered process are false’ (van der Aalst and ter Hofstede, 2005).

2.5 Contract generation

Assume that the client wishes to deploy a service that strictly satisfies queries of the type she has issued. In other words, she wants a service, say Trousers Boutique, that only sells dark-blue jeans made by a certain designer. This phase of the methodology achieves that by modifying the contract of a service to fulfil the query. This is done in two steps:

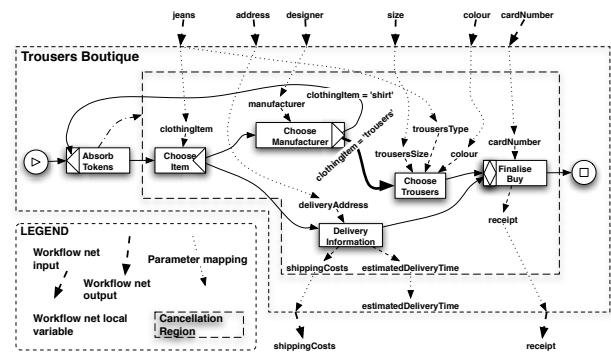
- 1 First, we choose one trace T (if any) from $TT(C_o)$ of a service that is compatible with the request. This is to ensure that we will not remove processes (see below) needed for the successful termination of the service’s execution. Then, we individuate the processes of the trace T as service processes having as input places the preconditions set of T . For instance, the preconditions set $\{C_i, C_1, C_2, C_4, C_6, C_7\}$ corresponds to the set of processes {Choose Item, Choose Manufacturer, Delivery Information, Choose Trousers, Finalise Buy}. We call *redundant* all other service processes (but *start* and *end*).
- 2 Then we copy the contract of the original service and modify it by cancelling redundant processes (if any) and suitably redirecting the workflow links. Workflow redirection is necessary to ensure that the workflow of the new service is consistent with the trace satisfying the query. This is achieved by adding to the workflow a process Absorb Tokens, and by directly connecting it as output of the *start* process. Then, all initial control-flow links that point at a redundant process are redirected to the XOR-join of Absorb Tokens. Then all redundant processes together with their outgoing links are removed. Finally, service outputs that are not requested by the query are hidden (see below).

For instance, the adaptation scenario for our example yields the workflow presented in Figure 5.

Notice that a YAWL service is a workflow specification which consists of one or more workflow nets (van der Aalst and ter Hofstede, 2005) – one of which is the starting net. Variables are defined at both workflow and task (i.e. process)

levels and the data-flow is specified by binding parameters of the workflow net and of its processes. In Figure 5 one may see that the Trousers Boutique service is made of one workflow net which contains the original processes of the Clothing Shop service except the redundant Choose Shirt process. As previously indicated, the process Absorb Tokens has been added at the beginning of the workflow and the original control-flow link leading at Choose Shirt has been redirected to Absorb Tokens. The inner dashed portion of the workflow in Figure 5 delimits the cancellation region (van der Aalst and ter Hofstede, 2005) associated with the process Absorb Tokens. Informally, this means that whenever Absorb Tokens is executed, all tokens enabling processes in its cancellation region are removed.

Figure 5 Workflow of the Trousers Boutique service



If a client of the Trousers Boutique service requests a shirt, then Choose Manufacturer will enable Absorb Tokens for execution and further Choose Item which will ask the client for another input. Moreover, Absorb Tokens will clear remaining tokens in the workflow so as to avoid (possibly) multiple executions of the other processes in the workflow (e.g. Delivery Information). Outputs that are not desired are hidden by mapping them to local net variables.³ This is the case for the estimatedDeliveryTime output of the Delivery Information process.

3 Related work

Different approaches to *service discovery* have been proposed, ranging from UDDI-based approaches to proposals which take into account semantics and/or behaviour information. Liang et al. (2004) use UDDI registries and constraints over services to semi-automatically discover (composed) services. Kawamura et al. (2004) enhance UDDI matching by extending WSDL service descriptions with semantic information in the style of DAML-S profiles. Kifer et al. (2004) employ WSMO to define a logical framework for service discovery. Some proposals use DAML-S/OWL-S and address service matching either at the service profile level (e.g. Aversano et al., 2004; Paolucci et al., 2002) or at the service model level (e.g. Bansal and Vidal 2003; Brogi et al. 2005).

Web service adaptation is in its early stages and current approaches feature only partial solutions to the issues of adaptation. Hau et al. (2003) propose a framework for semantic matchmaking and service adaptation, which

deals with signature mismatches, yet not with behavioural ones. Syu (2004) proposes an OWL-S based approach to deal with only three cases of adaptation of input parameters (permutation, modification and combination). Iyer et al. (2002) employ XML scripts and XSL to (manually) achieve the signature-level interoperability of SOAP services. Kaykova et al. (2005) show how to semantically adapt heterogeneous industrial resources. Yet their approach – as (Hau et al., 2003) – relies on black-box views of services and on semantically annotated signatures. A methodology for generating adapters to solve behavioural mismatches was presented in Brogi et al. (2004), yet ensuring the availability of an adapter specification to be manually generated.

To the best of our knowledge, our service adaptation approach is the first to take into account both semantics and behaviour information and to feature a fully automated generation of a service adapted to the client's needs.

4 Concluding remarks

Our long-term objective is to develop a general methodology for service adaptation capable of suitably overcoming both semantic and behaviour mismatches.

The main features of our approach can be summarised as follows:

- it is a *fully automatic* approach capable of generating service contracts tailored to client requests – given a registry of service contracts (containing both semantics and behaviour information) and a query
- it supports both service discovery and adaptation at the level of subservices (and not only of entire services)
- it is amenable to efficient implementations, as it relies on the inspection of execution traces that can be generated off-line
- it can be exploited to discover and adapt services written in different languages and to generate multiple deployments of the adapted contract, given that it relies on intermediate YAWL descriptions of the behaviour of services.

In this paper, we have illustrated how to automatically discover and adapt a service to match a client query requesting a service with certain inputs and outputs. We intend to devote our future work to extending the adaptation methodology so as to cope with queries specifying (part of) the behaviour of the desired service (i.e. not only its inputs and outputs) and to experimenting the deployment of adapted services with BPEL and OWL-S.

References

- Aggarwal, R., Verma, K., Miller, J.A. and Milnor, W. (2004) 'Constraint driven web service composition in METEOR-S', *IEEE SCC*, IEEE Computer Society, pp.23–30.
- Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M-T., Sheth, A. and Verma, K. (2005) 'Web service semantics – WSDL-S version 1.0', Available at: <http://lsdis.cs.uga.edu/library/download/WS-DL-S-V1.html>.
- Aversano, L., Canfora, G. and Ciampi, A. (2004) 'An algorithm for web service discovery through their composition', *IEEE International Conference on Web Services*, p.332.
- Bansal, S. and Vidal, J. (2003) 'Matchmaking of web services based on the DAML-S service model', in T. Sandholm and M. Yokoo (Eds). *Second International Joint Conference on Autonomous Agents (AAMAS'03)*, ACM Press, pp.926–927.
- BPEL4WS Coalition (2003) 'Business process execution language for web services (BPEL4WS) version 1.1', Available at: <ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf>.
- Brogi, A., Canal, C., Pimentel, E. and Vallecillo, A. (2004) 'Formalizing web service choreographies', *Proceedings of First International Workshop on Web Services and Formal Methods*, Available at: <http://www.lcc.uma.es/~av/Publicaciones/04/-ws-fm04.pdf>.
- Brogi, A., Corfini, S. and Popescu, R. (2005) 'Composition-oriented service discovery', F. Gschwind, U. Assmann and O. Nierstrasz (Eds), *Proceedings of Software Composition '05, LNCS*, Vol. 3628, pp.15–30.
- Hau, J., Lee, W. and Newhouse, S. (2003) 'The ICENI semantic service adaptation framework UK e-Science all hands meeting', Available at: <http://www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/017.pdf>.
- Iyer, A., Smith, G., Roe, P. and Pobar, J. (2002) 'An example of web service adaptation to support B2B integration', Available at: <http://ausweb.scu.edu.au/aw02/papers/refereed/smith2/paper.html>.
- Kawamura, T., Blasio, J.D., Hasegawa, T., Paolucci, M., van Harmelen, F. and Sycara, K. (2004) 'Public deployment of semantic service matchmaker with UDDI business registry', in S.A. McIlraith, D. Plexousakis and F. van Harmelen (Eds). *Proceedings of The Semantic Web ISWC'04, LNCS*, Vol. 3298, pp.752–766.
- Kaykova, O., Khriyenko, O., Kovtun, D., Naumenko, A., Terziyan, V. and Zharko, A. (2005) 'An approach to semantic adaptation of heterogeneous industrial web resources', Available at: <http://www.cs.jyu.fi/ai/papers/SJIS-2005.pdf>.
- Kifer, M., Lara, R., Polleres, A., Zhao, C., Keller, U., Lausen, H. and Fensel, D. (2004) 'A logical framework for web service discovery', *ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications*, Vol. 119, Hiroshima, Japan, *CEUR Workshop Proceedings*.
- Liang, Q., Chakarapani, L.N., Su, S.Y.W., Chikkamagalur, R.N. and Lam, H. (2004) 'A semi-automatic approach to composite web services discovery, description and invocation', *International Journal of Web Services Research*, Vol. 1, No. 4, pp.64–89.
- McGuinness, D. and van Harmelen, F. (Eds) (2004) 'OWL web ontology language overview', *Web guide*, Available at: <http://www.w3.org/TR/owl-features>.
- Meredith, L. and Bjorg, S. (2003) 'Contracts and types', *CACM*, Vol. 46, No. 10.
- OWL-S Coalition (2004) 'OWL-S: semantic markup for web services version 1.1.', Available at: <http://www.daml.org/services/owl-s/1.1/overview/>.
- Paolucci, M., Kawamura, T., Payne, T. and Sycara, K. (2002) 'Semantic matchmaking of web services capabilities', in I. Horrocks and J. Hendler (Eds). *First International Semantic Web Conference on The Semantic Web, LNCS 2342*, Springer-Verlag, pp.333–347.
- Papazoglou, M.P. and Georgakopoulos, D. (2003) 'Service-oriented computing', *Communication of the ACM*, Vol. 46, No. 10, pp.24–28.

- Rajasekaran, P., Miller, J.A., Verma, K. and Sheth, A.P. (2004) 'Enhancing web services description and discovery to facilitate composition', in J. Cardoso and A.P. Sheth (Eds). *SWSWPC*, Vol. 3387 of *Lecture Notes in Computer Science*, Springer, pp.55–68.
- SWSO Coalition (2005) 'Semantic web services ontology (SWSO) version 1.0.', Available at: <http://www.daml.org/services/swsf/1.0/swso/>.
- Syu, J.Y. (2004) *An Ontology-Based Approach to Automatic Adaptation of Web Services*, Department of Information Management National Taiwan University, Available at: <http://www.im.ntu.edu.tw/IM/Theses/r92/R91725051.pdf>.
- UDDI Coalition (2000) *The UDDI Technical White Paper*, Available at: <http://www.uddi.org/>.
- van der Aalst, W.M.P. (2004) 'Pi calculus versus Petri nets: Let us eat humble pie rather than further inflate the Pi hype', Available at: <http://tmitwww.tn.tue.nl/staff/wvdaalst/pi-hype.pdf>.
- van der Aalst, W.M.P. and ter Hofstede, A.H.M. (2005) 'YAWL: yet another workflow language', *Information System*, Vol. 30, No. 4, pp.245–275.
- van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B. and Barros, A.P. (2003) 'Workflow patterns', *Distributed Parallel Databases*, Vol. 14, No. 1, pp.5–51.
- Wohed, P., van der Aalst, W.M.P., Dumas, M. and ter Hofstede, A.H.M. (2003) 'Analysis of web services composition languages: the case of BPEL4WS', in I-Y. Song, S.W. Liddle, T.W. Ling and P. Scheuermann (Eds). *Proceedings of the 22nd International Conference on Conceptual Modeling*, Vol. 2813 of *Lecture Notes in Computer Science*, Springer, pp.200–215.
- WSCDL Coalition (2005) 'Web services choreography description language version 1.0.', Available at: <http://www.w3.org/TR/ws-cdl-10/>.
- WSDL Coalition (2001) 'Web service description language (WSDL) version 1.1', Available at: <http://www.w3.org/TR/wsdl>.
- WSMO Coalition. (2005) 'Web service modeling ontology (WSMO) D2v1.2', Available at: <http://www.wsmo.org/TR/d2/v1.2/>.
- Wynn, M.T., Edmond, D., van der Aalst, W.M.P. and ter Hofstede, A.H.M. (2005) 'Achieving a general, formal and decidable approach to the OR-join in workflow using reset nets', in G. Ciardo and P. Darondeau (Eds). *ICATPN*, Vol. 3536 of *Lecture Notes in Computer Science*, Springer, pp.423–443.
- Yang, J. (2003) 'Web service componentization', *Communications of the ACM*, Vol. 46, No. 10, pp.35–40.

Notes

- ¹Space limitations do not allow us to illustrate these patterns. A detailed description of them may be found in van der Aalst et al. (2003).
- ²The first parameter in a pair belongs to the query while the second to the service.
- ³The reason why we do not remove them is, for example, that the respective process may correspond to a YAWL task that invokes a WSDL service and then a mapping between its parameters and the ones of the WSDL service is necessary.