Matchmaking for Business Processes Based on Choreographies

Andreas Wombacher, Integrated Publication & Info. Systems Institute, Germany
Peter Fankhauser, Integrated Publication & Info. Systems Institute, Germany
Bendick Mahleko, Integrated Publication & Info. Systems Institute, Germany
Erich Neuhold, Integrated Publication & Info. Systems Institute, Germany

ABSTRACT

Web services have a potential to enhance B2B e-commerce over the Internet by allowing companies and organizations to publish their business processes on service directories where potential trading partners can find them. This can give rise to new business paradigms based on ad-hoc trading relations as companies, particularly small to medium scale, can cheaply and flexibly enter into fruitful contracts, for example through subcontracting from big companies by simply publishing their business processes and the services they offer. More business process support by the Web service infrastructure is however needed before such a paradigm change can materialize. A service for searching and matchmaking of business processes does not yet exist in the current infrastructure. We believe that such a service is needed and will enable companies and organizations to be able to establish ad-hoc business relations without relying on manually negotiated interorganizational workflows. This article gives a formal semantics to business process matchmaking based on finite state automata extended by logical expressions associated to states.

Keywords: PLEASE PROVIDE

INTRODUCTION

Web services have a potential to enhance B2B e-commerce over the Internet by allowing companies and organizations to publish their business processes on service directories where potential trading partners can discover them. This can give rise to new business paradigms based on dynamic trading relations as companies, particularly small to medium scale, can cheaply and flexibly enter into fruitful contracts, for example, through subcontracting from big companies by simply publishing their business processes and the services they offer.

To date, loosely coupled business processes are quite rare. Either simple (stateless) Web services are used or the binding of Web services is done statically. While stateless services are not sufficient for implementing business processes, static

binding of services does not use the full potential of loosely coupled systems also known as service oriented architectures. Existing standards supporting brokering of Web services are UDDI (Ariba, 2000) and WS-Inspection (Ballinger et al., 2001). Both approaches are based on string comparisons, which are used for searching in classification schemes or t-models (like WSDL). This is not sufficient for business processes, especially if there does not exist any pre-negotiated and uniquely named frame contracts published by standardization organizations as, for example, RosettaNet.

Other service based infrastructures face the same issue. In particular, within the ebXML framework business partners can express their business capabilities (including their business processes) using trading partner profiles (CPPs) without providing any means to match these.

This article presents an approach to more precise service discovery using business process descriptions rather than individual messages. The next section illustrates limitations of existing approaches to service discovery by way of simple examples of compatible and incompatible business processes. The Approach section formalizes a more precise notion of business processes matching based on annotated finite state automata. The next section discusses at which level the introduced techniques can be deployed in existing service description frameworks, followed by a description of the implementation details including the complexity of the algorithms. The article concludes with a discussion of related work, conclusions and an outline of future work.

EXAMPLE

Figure 1 depicts two business processes involving two trading parties: a vendor *v* and a customer *c*. Nodes represent the states of a business process; the end states are identified by a double circle. Edges represent state transitions, which are labeled with messages denoted as *from#to#message_name*, where *from* represents the message sender, *to* represents the message recipient, and *message_name* is the name of the message.

Figure 1(a) shows the vendor business process, where the vendor expects to receive a purchase order (*c#v#PO*) message, followed by a credit card payment (*c#v#ccPay*) and finally sends back a de-

Figure 1: (a) Vendor Message Sequence (b) Customer Message Sequence

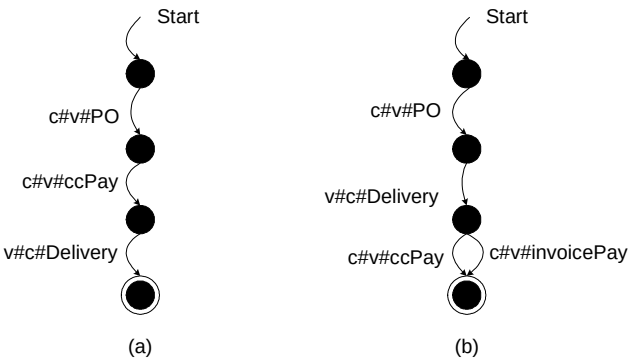
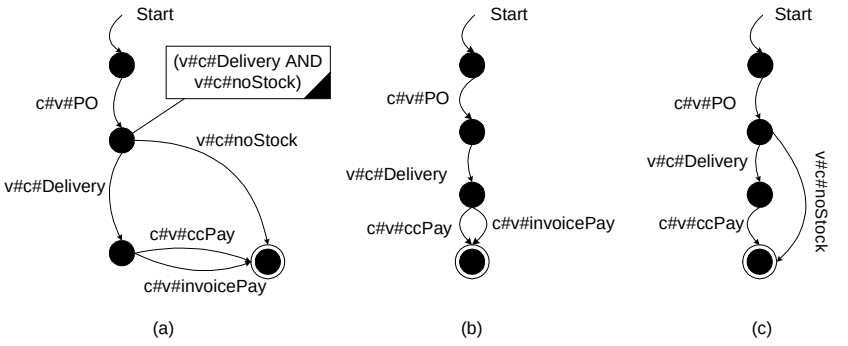


Figure 2: (a) Vendor Message Sequence Insisting on v#c#noStock and v#c#Delivery Messages (b) Customer Message Sequence (c) Customer Message Sequence with Optional v#c#noStock Message



livery (*v#c#Delivery*) message to the respective customer. The customer process depicted in Figure 1(b) also initiates the process with a purchase order request (*c#v#PO*). But then it insists on delivery (*v#c#Delivery*) before payment by credit card (*c#v#ccPay*) or by invoice (*c#v#invoicePay*).

At the level of individual messages these two business processes match. However, because they require a different order of payment and delivery, they are incompatible, that is, they can not successfully interact. In order to avoid matching incompatible business processes we thus need to take into account message sequences rather than individual messages.

Figure 2(a) shows a purchase order business process provided by a vendor. The process starts with a purchase order (*c#v#PO*) message, followed by a delivery (*v#c#Delivery*) message, and either a credit card payment (*c#v#ccPay*) or an invoice payment (*c#v#invoicePay*) message. In case the ordered product is not on stock, the vendor may reject a purchase order by sending a no stock available (*v#c#noStock*) message.

The vendor process involves two kinds of messages: mandatory and optional ones. On the one hand, it insists on the availability of both, the *v#c#noStock* and the *v#c#Delivery* message (two mandatory messages) represented as an annotated logical expression. On the other hand, it supports the two payment options as genuine alternatives (optional messages) not requiring any further annotation, because this is intended to be standard automata semantics.

Figure 2(b) (this process is equivalent to the one depicted in Figure 1(b) and described above) depicts a customer business process. While this process matches the vendor process with respect to the delivery payment order, it cannot handle the required *v#c#noStock* message. Therefore, the two business processes cannot reach the end state, if an ordered product is not on stock.

Conversely, the business process in Figure 2(c) supports *v#c#noStock* and *v#c#Delivery* messages, whereas it supports only one payment option. This process now satisfies all conjunctive (mandatory messages) and disjunctive (optional messages) choices of the vendor process.

Thus, the vendor process and the customer process are compatible that is guaranteeing a successful business interaction.

In summary, the two examples in Figures 1 and 2 illustrate that (1) message sequence and (2) conjunctive choices need to be taken into account to determine the compatibility of business processes.

APPROACH

Finite state automata (Hopcroft et al., 2001) constitute a suitable starting point to model business processes for the purpose of matchmaking, where matchmaking is based on non-empty intersection of finite state automata. They can represent (possibly infinite) sets of message sequences without considering branching conditions and parallel execution capabilities as provided by more expressive approaches such as Petri Nets. While Petri Nets are also closed under intersection (Peterson, 1981), they require a much higher computational and space complexity compared to finite state automata. In particular, Petri Nets allowing concurrent execution are non-polynomial for reachability and liveness problems (Esparza & Nielsen, 1994). If the Petri Net class is limited to bounded (a net is bounded if it is has a finite set of possible markings) nets, several polynomial results exist. In case of bounded nets the reachability graph can be represented as a finite state automaton with high complexity, but finite. Thus it does not exceed the expressiveness of finite state automata. There exist approaches (Molina-Jimenez et al., 2003) addressing matchmaking using some sort of state machines having the same expressiveness as finite state automata (Peterson, 1981). An in-depth investigation of the necessary expressiveness of the language required by real business processes and the implication on the general computability will be addressed in future work.

Finite State Automaton

Finite State Automata (FSA) are well studied. Formally, finite state automata can be represented as follows:

Definition 1
(Finite State Automata (FSA))

A finite state automaton A is represented as a tuple:

$A = (Q, \Sigma, \Delta, q_0, F)$ where:

- Q is a finite set of states,
- $\Sigma \subseteq P \times P \times M$ is a finite set of messages in M sent by a sender in P to a receiver in P, where P represents the parties being involved,
- $\Delta : Q \times \Sigma \times Q$ represents labeled transitions,
- q_0 a start state with $q_0 \in Q$, and
- $F \subseteq Q$ a set of final states.

The only difference to the standard definition of FSAs (Hopcroft et al., 2001) is that the alphabet Σ consists of triples rather than of atomic tokens. However, for the purpose of matchmaking business processes, these triples can be treated like atomic tokens: Two message triples are equal, if their sender, their receiver, and the message (with its parameters) are equal.

A FSA A generates a language $L(A)$ that enumerates the (possibly infinite) set of all message sequences supported by a business process. Two FSAs match if their languages have a non-empty intersection. The intersection of two FSAs is again a FSA, which can be determined with the

Equation A

$$Empt(curP, q_i) := \neg\ Reach(curP, q_i)$$

$$Reach(curP, q_i) := \begin{cases} true & \text{if } q_i \in F \\ \cdot_{\{q\_l \mid \delta(q_i,l)=q\_l\}} . Reach(curP.q_i, q\_l) & \text{if } q_i \notin F \wedge q\_l \notin curP \\ false & \text{otherwise} \end{cases}$$

usual cross product construction (Hopcroft et al., 2001):

Definition 2
(Intersection of Two FSAs)

The intersection $A_1 \cap A_2$ of two automata $A_1 = (Q_1, \Sigma_1, \Delta_1, q_{10}, F_1)$, and $A_2 = (Q_2, \Sigma_2, \Delta_2, q_{20}, F_2)$ is $A = (Q, \Sigma, \Delta, q_0, F)$, with

- $Q = Q_1 \times Q_2$,
- $\Sigma = \Sigma_1 \cap \Sigma_2$,
- $\Delta\ ((q_{11}, q_{21}), \alpha, (q_{12}, q_{22}))$ *with* $\Delta_1\ (q_{11}, \alpha, q_{12}) \wedge \alpha_2\ (q_{21}, \alpha, q_{22})$,
- $q_0 = (q_{10}, q_{20})$, and
- $F = F_1 \times F_2$.

If the resulting automaton does not contain at least one path (possibly of zero length) between the start state and an end state, its language is the empty language $\varnothing$. In this case, the business processes modeled by the FSAs are incompatible, because they do not share a common message sequence.

An emptiness algorithm like in (Hopcroft et al., 2001) is based on the reachability of states within an automaton starting from the start state q_0. The automaton accepts an empty language, if and only if no final state is within the set of reachable states.

A functional definition of an emptiness test is based on a recursive reachability function, where *curP* represents the current path of the recursion and q_i represents the current state. The function terminates, if a final state has been reached (first line of definition) or no further non-cyclic transition is available (third line). The function traverses the automaton in a deep-first manner (second line) seeking at least one path to a final state. An automaton is empty if no final state is reachable. A formal definition is given in Equation A.

The standard automaton intersection of the vendor process in Figure 2(a) and the customer process in Figure 2(b) is equivalent to the customer process. Although this intersection is not empty, it does not contain the required transition *v#c#noStock* of the vendor process. The reason for this false match is that standard FSAs cannot distinguish between mandatory and optional messages. Usually, a message in a standard FSA is regarded as an optional one. However, the intended meaning of the vendor process requires both mandatory and optional messages. It is not possible to represent these semantics in message sequences or FSA directly. Thus, an annotation containing this additional meta-information is required and relevant only for matchmaking purposes.

Annotated FSA

Based on the above observation, annotated FSA are introduced as a standard

FSA, where each state might be assigned a propositional logical term.

Definition 3
(Annotated FSA (aFSA))

An annotated FSA A is represented as a tuple $A=(Q, \Sigma, \Delta, q_0, F, QA)$ where

- Q is a finite set of states,
- $\Sigma \subseteq P \times P \times M$ is a finite set of messages in M sent by a sender in P to a receiver in P, where P represents the parties being involved,
- $\Delta : Q \times \Sigma \times Q$ represents transitions,
- q_0 a start state with $q_0 \in Q$,
- $F \subseteq Q$ a set of final states, and
- $QA: Q \times E$ is a finite relation of states and logical terms within the set E of propositional logic terms.

The terms in E are standard Boolean formulas. Adapting the definition in Chomicki and Saake (1998):

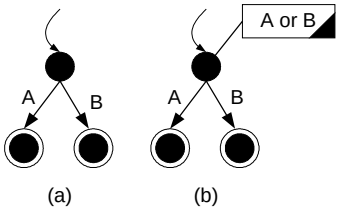
Definition 4
(Definition of Terms)

The syntax of the supported logical formulas is given as follows:

- the constants *true* and *false* are formulas,
- the variables $v \in \Sigma$ are formulas,
- if φ is a formula, so is $\neg\varphi$,
- if φ and $\grave{E}$ are formulas, so is $\varphi \wedge \psi$ and $\varphi \vee \psi$.

The standard semantics of automata is an optional execution of transitions. This is observable also in the functional emptiness test definition given above: a single path to a final state returns a *true*, causing the whole disjunction to return *true* in the reachability function. Thus, the logical mapping of automata to annotated automata is

Figure 3: (a) automata (b) annotated automata equivalent to (a)



an annotation containing a disjunctive expression including all transition labels as depicted in Figure 3. For simplicity reasons, the OR annotations are neglected in the following.

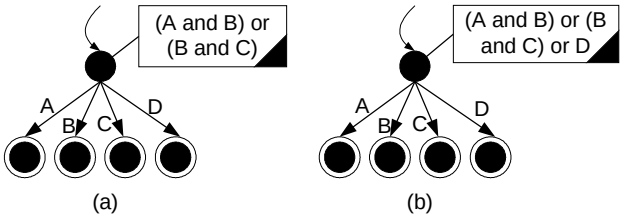
The definition of terms does not enforce a term to contain all labels of outgoing transitions of the associated state. Thus, annotations may be incomplete, that is, not containing all outgoing transition labels. Such incomplete annotations can be completed by extending them with a disjunction of all labels not contained yet. This method is best explained if the expression is in disjunctive normal form as depicted in Figure 4a. The annotation means that the matching process must support message *B* in combination with either message *A* or *C*. Message *D* is unrelated to messages *A*, *B*, and *C*, thus represents an independent alternative, which is combined with the existing term by a disjunction as depicted in Figure 4b.

Extending terms of annotated automata is quite important for defining the emptiness test later on. The set of variables X^{q_i} corresponding to state q_i is defined as the set of outgoing transition labels of state q_i. Formally expressed as:

$$X^{q\_i} := \{ x^{q\_i} \mid \exists q' \in Q.\ \Delta(q_i, x^{q\_i}, q')\}$$

Similar to standard Boolean logic definitions *Var* is the set of all variables bound

Figure 4: (a) Incomplete Annotated Automata (b) Completely Annotated Automata Equivalent to (a)



in a term t^{q_i}associated to a state q_i with $(q_i,t^{q\_i}) \in QA$. Be aware that the formula:

$$\mathrm{Var}(t^{q\_i}) \subseteq X^{q\_i}$$

is not necessarily true. There might exist variables in a term associated to a state q_i without a counterpart in outgoing transition labels. A counter-example is depicted in Figure 5a and explained in the intersection subsection later on. As stated above, a term t^{q_i} might be incomplete; that is:

$$X^{q\_i} \setminus \mathrm{Var}(t^{q\_i}) \neq \varnothing$$

and must be extended. The completed term t'^{q_i} is defined as a disjunction of the annotated term t^{q_i} associated to state q_i and all outgoing transition labels not used in the term t^{q_i} so far. A formal definition is given below:

$$t'^{q\_i} := t^{q\_i} \vee \left(\bigvee_{x \in X q\_i \setminus \mathrm{Var}(t q\_i)} x \right)$$

Intersection of aFSA

Matchmaking business processes has been defined as a non-empty intersection. The intersection automaton of two automata contains the language accepted by both automata. Therefore, the annotation of the result automaton must support the annotation of the first AND the annotation of the second automaton. The intersection definition is given:

Definition 5
(Intersection of Two aFSAs)

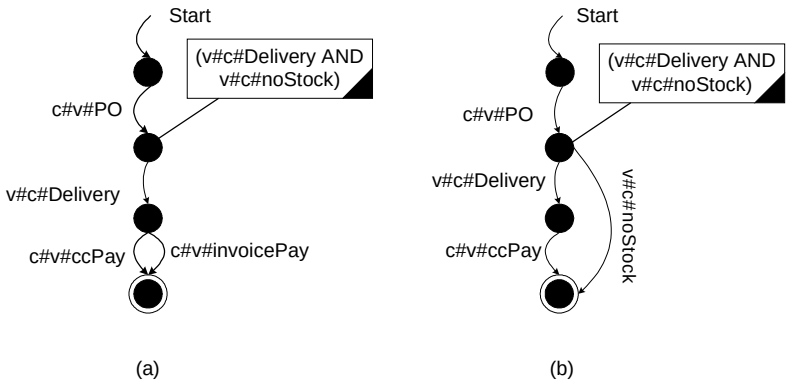
The intersection $A_1 \cap A_2$ of two annotated automata $A_1 = (Q_1, \Sigma_1, \Delta_1, q_{10}, F_1, QA_1)$, and $A_2 = (Q_2, \Sigma_2, \Delta_2, q_{20}, F_2, QA_2)$ is $A = (Q, \Sigma, \Delta, q_0, F, QA)$, with:

- $Q = Q_1 \times Q_2$,
- $\Sigma = \Sigma_1 \cap \Sigma_2$,
- $\Delta((q_{11},q_{21}), \alpha, (q_{12},q_{22})$ with $\Delta_1(q_{11}, \alpha,q_{12}) \wedge \Delta_2(q_{21},\alpha,q_{22})$,
- $q_0 = (q_{10},q_{20})$,
- $F = F_1 \times F_2$, and (see Equation B)

Equation B

$$QA = \bigcup_{\substack{q_1 \in Q_1, \\ q_2 \in Q_2}} \begin{cases} ((q_1,q_2), e_1 \wedge e_2) & if (q_1,e_1) \in QA_1, (q_2,e_2) \in QA_2 \\ ((q_1,q_2), e_1) & if (q_1,e_1) \in QA_1, q_2 \in Q_2, \ \nexists e'.(q_2,e') \in QA_2 \\ ((q_1,q_2), e_2) & if (q_2,e_2) \in QA_2, q_1 \in Q_1, \ \nexists e'.(q_1,e') \in QA_1 \\ \emptyset \quad otherwise & \end{cases}$$

Figure 5: (a) Intersection of Vendor and Customer Process with Missing no_Stock Message (b) Intersection of Vendor and Customer Process with no_Stock Message



The intersection definition above is a slight extension of standard automaton intersection definition. In particular, the annotations are maintained independently of the automaton structure itself. The evaluation of the resulting annotated automaton with regard to matchmaking is done in the emptiness test.

To illustrate this definition the example of the Example section is reconsidered. The minimized intersection automaton of the vendor and customer process in Figure 2(a) and (b) is depicted in Figure 5(a). The resulting automaton is the standard automaton intersection plus the corresponding annotation. The annotation requires a *no_Stock* message, although the automaton structure does not provide this transition. Figure 5(b) depicts the intersection automaton of the vendor and the customer process given in Figure 2(a) and (c). The resulting automaton contains both required messages: *Delivery* and *no_Stock*.

Emptiness Test of Annotated FSA

So far, state annotations have been maintained, but not yet been evaluated. Within the emptiness test the annotated terms are now evaluated. The evaluation of annotated terms is done in accordance to standard logical interpretation, as for example defined in Chomicki and Saake (1998), where an interpretation is based on a valuation of variables. A variable is evaluated as *true* if and only if the target state of the transition labeled with the variable name can reach a final state. Thus, the word associated with the current state concatenated with the variable name is a prefix of at least one word accepted by the language of the automaton.

Based on this definition of truth of the annotated terms it is required to first determine whether the target state of outgoing transitions of a state can reach a final state before evaluating the annotated term. This may result in cyclic dependencies, like for example observable in a self-loop. The truth value of a state cannot be determined, because the result depends on its own (not yet defined) truth value. This issue can be resolved by using a three-valued logic providing the standard truth values *true* and *false*, and in addition a value *indeterminate* used in case of recursion. This issue is well known from primitive recursive function theory. The formal definition of the

Table 1

	$\neg_3$		$\vee_3$	**f**	**t**	**i**		$\wedge_3$	**f**	**t**	**i**
f	t		**f**	f	t	i		**f**	f	f	f
t	f		**t**	t	t	t		**t**	f	t	i
i	i		**i**	i	t	i		**i**	f	i	i

emptiness test is based on the Kleene's system of "strong connectives" (Panti, 1995). (The special definition of implications of this system is not required in the presented approach.) The corresponding operations of the three valued logic are negation $\neg_3$, disjunction $\vee_3$, and conjunction $\vee_3$. The corresponding truth tables are given below for completeness. (See Table 1).

The standard interpretation | .| of the logic is based on the operations defined above, but must consider the current path *curP* of the evaluation to enable circle detection. The characters t,t_1,t_2, and c, and x represent terms, constant, and variable symbols respectively.

$$| \neg t|^{v}_{curP} := \neg_3 |t|^{v}_{curP}$$
$$|t_1 \vee t_2|^{v}_{curP} := |t_1|^{v}_{curP} \vee_3 |t_2|^{v}_{curP}$$
$$|t_1 \wedge t_2|\_\{curP\}^{v} := |t_1|^{v}_{curP}{}^{v} \wedge_3 |t_2|^{v}_{curP}$$
$$| true |^{v}_{curP} := t; |false|^{v}_{curP} := f$$
$$|x|^{v}_{curP} := v_I \text{ with } v_I \in \{t, f, i\}; x \in \Sigma$$

As stated above, the truth value of variables are derived by checking whether there exists a path to a final state starting from the current path *curP* extended by the current state q_i and following the transition labeled with the name of the variable $x\_j^{q\_i}$. The value *intermediate i* is returned if the transition labeled $x\_j^{q\_i}$ has a target state contained in the current path *curP* concatenated with the current state q_i. This is because the evaluation of the variable $x\_j^{q\_i}$ depends on its own evaluation. In case the target state of the transition labeled $x\_j^{q\_i}$ is not contained in the current path nor in the current state q_i, the evaluation of $x\_j^{q\_i}$ is done by a function called reach *R()* checking the reachability of a final state. The function is quite similar to the *Reach()* function given above and is defined in more detail later on. In case no transition labeled with $x\_j^{q\_i}$ exists the evaluation is *false f*. The formal definition of the valuation of variables is given in Equation C.

Based on this valuation definition emptiness in annotated automata denoted as *Empt'()* is *false* if the modified reachability function *R()* returns truth value *t*. Emptiness is defined by a comparison to ensure a Boolean result rather than a three-value logical result. (See Equation D).

The reachability function *Reach'()* terminates with *true t* if the current state q_i is a final state. If the current state q_i is not a final state the completed annotation must be valuated.

Determination of Annotated FSA

A non-deterministic FSA (NFA) is characterized by containing two transitions with the same source state and the same transition labels. The determination usually is constructed by representing the set of states of the NFA as a single state in the corresponding deterministic FSA (DFA). A DFA definition differs from the NFA definition such that the transitions are represented by a function δ: $Q \times \Sigma \rightarrow Q$ rather than a relation ".

Equation C

$$|x\_j^{q\_i}|^{v}_{curP} := \begin{cases} i & \text{if } \Delta(q_i, x\_j^{q\_i}, q') \wedge q' \in curP.q_i \\ R(curP.q_i, q') & \text{if } \Delta(q_i, x\_j^{q\_i}, q') \wedge q' \in curP.q_i \\ f & \text{otherwise} \end{cases}$$

Equation D

$$Empt'(curP, q_i) := R(curP, q_i) \neq t$$

$$R(curP, q_i) := \begin{cases} t & \text{if } q_i \in F \\ | t'^{q\_i}|^{v}_{curP} & \text{otherwise} \end{cases}$$

Equation F

$$\Phi(q^P) := \begin{cases} \varnothing & \text{if } \varphi(q^P) = \varnothing \\ e & \text{if } \varphi(q^P) = \{e\} \\ \cdot_{e \in \varphi(qP)} e & \text{otherwise} \end{cases}$$

Definition 6
(Determination of FSA)

Let $A=(Q, \Sigma, \Delta, q_0, F)$ be an NFA and let $\delta: 2^Q \times \Sigma \rightarrow 2^Q$ be based on Δ by $(\{q_1,\ldots,q\_n\},\alpha) \cup^{n}_{i=1} \{q' \mid (q_i,\alpha,q') \in \Delta\}$. The powerset automaton $P(A)$ of A is the DFA $P(A)=(Q^P, \Sigma, \delta^P, q_0^P, F^P)$, where

- $Q^P := \{\delta(\{q_0\},\omega) \mid \omega \in \Sigma^*\}$
- $\delta^P : Q^P \times \Sigma \rightarrow Q^P, (Q', \omega) \rightarrow \delta(Q',\omega)$
- $q_0^P := \{q_0\}$
- $F^P := \{Q' \in Q^P \mid Q' \cap F \neq \varnothing\}$

The annotation is maintained by combining the annotations of the NFA states clustered in a DFA state by disjunction.

Definition 7
(Determination of Annotated FSA)

Let $A=(Q, \Sigma, \Delta, q_0, F, QA)$ be an annotated NFA. The powerset automaton $P(A)$ of A is the annotated DFA $P(A)=(Q^P, \Sigma, \delta^P, q_0^P, F^P, QA^P)$, where the above definition applies for Q^P, q_0^P, δ^P, F^P and QA^P is constructed with

- $QA^P := \cup q^P \in Q^P \cdot \Phi(q^P)$ collecting the transformations of each set of states q^P
- *See Equation F*
- $\varphi(q^P) := \{e \mid q \in q^P \wedge (q,e) \in QA\}$ collecting all annotations of the states contained in the set

Equation E

```
algorithm:
input: NFA (Q,, Σ, Δ, q0, F, QA)
  Q^P:={{q0}}; rest := {{q0}}; q0^P := {q0}; QA^P:=∅
  if q0 ∉ F then F^P:={{q0}} else F^P=∅;
  while rest ≠ ∅ do
        select and delete any Q1 ∈ rest;
        for all α ∈ Σ
              Q2 := ∅;
              for all q1 ∈ Q1
                    Q2 := Q2 ∪ {q' | (q1,α,q') ∈ Δ}
              end for
              if Q2 ∉ Q^P then
                    rest := rest ∪ {Q2};
                    Q^P := Q^P ∪ {Q2};
                    if F ∩ Q2 ≠ ∅ then F^P := F^P ∪ {Q2};
              end if
               δ^P(Q1,α) := Q2;
        end for
     end while
     for all Q1 ∈ Q^P
        e= null;
        for all q' ∈ Q1
              (q',e') ∈ QA
              if e' ≠ null then
                    if e=null then e:= e' else e:=e ∨ e';
              end if
```

Equation D is a corresponding algorithm with a complexity of $O(2^n*n^2*|\Sigma|)$.

Epsilon Removal of Annotated FSA

Epsilon ε transitions represent transitions which might be followed by traversing the automaton, but not contributing to the resulting word accepted by this traversal.

The standard algorithm is based on representing new states by sets, where each set contains all states being reachable by epsilon transitions only. The adaptation to annotated FSA is a bit tricky. In particular, the concept of epsilon has to be extended a bit. In particular, the single epsilon in standard FSA is replaced by a set of epsilon values, which have in common not to contribute to the word, but can be differentiated against each other. This differentiation is required to distinguish variables in the aFSA annotations.

Similar to the determinization, the epsilon removal relies on the standard epsilon algorithm introduced previously. In addition, annotations are substituting the epsilon contained in an annotation by the annotation assigned to the state reachable by this epsilon transition.

Figure 6: (a) aFSA with Epsilon; (b) Equivalent aFSA without Epsilon

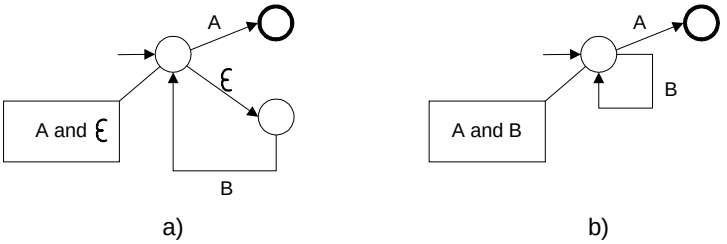
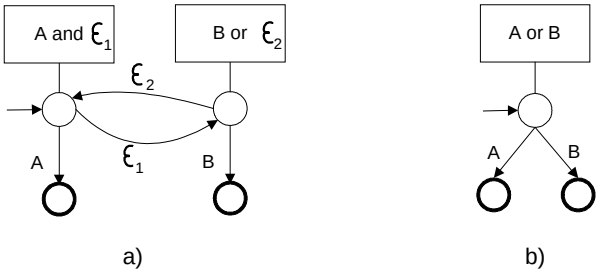


Figure 7: (a) aFSA with Epsilon; (b) Equivalent aFSA without Epsilon



This approach has to be changed in case of cyclic epsilon transitions, where the replacement does not recurse, but simply replace epsilon as long as the cycle does not close. Afterwards, relate the derived expressions by disjunction.

In this example, $A \wedge \varepsilon_1$ is annotated to a state and ε_1 has to be replaced by $\varepsilon_1 \rightarrow B \vee \varepsilon_2$. Because ε_2 closes the loop and thus could be ignored, the real substitution of ε_1 is $\varepsilon_1 \rightarrow B$. Correspondingly, the annotation $B \vee \varepsilon_2$ uses the substitution $\varepsilon_2 \rightarrow A$. The resulting annotation after removing the epsilon transitions is $(A \wedge B) \vee (B \vee A)$, which can be reduced to $B \vee A$.

APPLICATION TO SERVICE DEFINITION LANGUAGES

This section illustrates how the presented approach to matchmaking for business processes is related and can be applied to existing service definition languages. Services are typically described at three major levels: messages, abstract processes, and execution processes.

1. Message descriptions such as WSDL and EDIFACT describe the syntax and structure of messages. The Web Service Definition Language (WSDL) (Christensen et al., 2001) uses XML Schema to describe the input and output of operations supported by a service. These operations can be associated to roles, which correspond to sender and receiver of message descriptions used in this article. Thus, WSDL descriptions may be used as one concrete form to encode and match individual message descriptions. Alternatively, Web based EDI like EDIFACT can be used for this purpose. Such syntactic message descriptions can be matched by component-wised comparison. A more ambi-

tious approach is addressed by DAML-S profiles (DAML-S Coalition, 2002): these profiles describe messages by means of ontological concepts such that semantic reasoning can be used to more flexibly match messages.

2. Abstract processes describe the sequences in which messages may be exchanged. There are several proposals for specifying abstract processes, including WSCL, cpXML, the abstract part of BPEL, and ebXML BPSS.
   WSCL (Banerji et al., 2002) uses finite state automata to model abstract business processes. Conversation Policy XML (cpXML) (Hanson et al., 2002) extends finite state automata with hierarchical states, which encapsulate again a finite state automaton. BPEL (Andrews et al., 2003) (synthesized from XLANG and WSFL) and ebXML BPSS (Levine et al., 2001) also allow for the specification of branching conditions as well as parallel recursive business processes, which cannot be described by finite state automata.
   Therefore, for the purpose of matchmaking, parallelism needs to be abstracted away, which may introduce false matches.
3. Execution process description extends abstract process description with information necessary to execute a business process. This includes the binding of the abstract process to internal processes, constraints on message parameters and on time. This additional information is usually confidential and therefore not advertised publicly (Casati & Discenza, 2000; Georgakopoulos et al., 1999). Nevertheless, especially constraints may be deployed for improving the precision of matchmaking.

IMPLEMENTATION

The described annotated automata aFSA has been implemented in Java in the deterministic version for evaluation purposes. The package can freely be downloaded for non-commercial usage at *http://www.ipsi.fhg.de/oasys/ipsi-pf*. It contains a simple demo application parsing two aDFA from XML files, calculating the intersection automaton, and checking emptiness of the result automaton. The main parts of this implementation are sketched below due to the limitation of space.

aDFA XML Schema

The aDFA XML Schema definition is strongly related to the definition in the Approach section. Thus, the messages, states and transitions are enumerated, while IDs are added for referencing for example, a source state in a transition to a corresponding state element in the state enumeration. To guarantee syntactical consistency of these references, *key* and *keyref* rules are added to the XML Schema. In addition, uniqueness rules are added for message and state names to guarantee the set property within the automaton definition respectively.

aDFA Algorithms

The implementation of aDFA algorithms is strongly related to the one of standard automata. The intersection algorithm is a slight extension of the standard one without influencing the overall complexity of the algorithm. The emptiness test of aDFA differs from standard automata algorithm. An algorithm is presented in pseudo code in addition to the functional definition of the emptiness test described.

The symbols used in pseudo code are taken from the definition of aFSA. The algorithm contains a function *isEmpty* returning *TRUE* if the accepted language is empty, *FALSE* otherwise. This function calls the recursive function *reachable*, returning

- *TRUE* if from the current state *c* a final state can be reached,
- *FALSE* if starting from the current state *c* no path to a final state exists,
- *INTERMEDIATE* otherwise.

To reduce the number of recursions intermediate results are collected in an array *Val* containing a truth value associated to a state. The set of already visited states is maintained in the variable *Vis*. Thus, *Vis* contains all index values that can be applied to the array *Val*. The current path is maintained in variable *P*, while the current state is represented in variable *c*. The interpretation of an expression in three value logic is a standard traversal of the operator tree of the expression and, thus, not presented here. Instead, the formalism introduced above is used accordingly, while $v = v \cup \{(x \rightarrow v_3)\}$ means extending the valuation by the assignment of a variable name x to a truth value v_3 of three valued logic. (See Equation F.)

The function *reachable* terminates, returning TRUE if the current state is a final state (line 6). Otherwise, the transitions with source state being the current state *c* are iterated (line 8) to determine the truth value of all states targeted by transitions starting from the current state and store it in the valuation v (line 18). The corresponding truth value is INTERMEDIATE in case the target state is either the current state *c* or it is in the path *P* (lines 9,10). Or, the truth value can be taken from the intermediate results stored in variable *Val* if the target state has already been visited (lines 11,12). Or, finally, the truth value has to be calculated by calling the function *reachable* recursively and storing the intermediate results afterwards in variable *Val* (lines 14-16). After the valuation has been defined, the truth value associated to the current state can be calculated by evaluating the expression (lines 21-28). The return value of the function is the disjunction (lines 29-33) of the expression evaluation and the remaining transitions not considered in the expression (lines 27,28).

The complexity of this algorithm is the one of standard emptiness algorithm plus the complexity for evaluating expressions. The complexity of expression evaluation is linear to the number of elements in the expression operation tree. Let's assume as a worst case scenario the expression is provided in conjunctive normal form. So, the number of leaf nodes of the operation tree is $2^{|\Sigma|}$, and thus the number of total elements in the operation tree is $2^{|\Sigma|+1}-1$. Because expressions are assigned to states, the overall worst case complexity of evaluating expressions is $|Q|*(2^{|\Sigma|+1}-1)$.

We believe that real business processes have quite limited annotations, although no empirical data regarding structure and complexity of realistic expressions exists to give a more appropriate average estimation of computational cost. Future work will address this issue.

RELATED WORK

Handling processes in inter-organizational cooperation usually is based on the existence of a global predefined workflow split into different parts to be executed locally (Grefen et al., 2000; Klingemann et al., 1999; van der Aalst & Weske, 2001). Because these approaches are based on a global workflow definition, the local

Equation F

```
algorithm
 1:  isEmpty() {
 2:              return Ø (reachable({},{},q0,{}) = );
 3:  }
 4:
 5:   reachable(Val, P, c, Vis){
 6:        if (c ∈ F) return TRUE;
 7:        else {
 8:             for all δ(c,label)=target {
 9:                  if (target ∈ P ∪ {c})
10:                       ret = INTERMEDIATE;
11:                  else if (target ∈ Vis)
12:                       ret = Val [target] ;
13:                  else {
14:                       ret = reachable(Val, P ∪ c, target, Vis);
15:                       Vis = Vis ∪ c;
16:                       Val [target] = ret;
17:                  }
18:                   v = v ∪ {(label→ret)}
19:             }// end of for
20:             // evaluation of the partial results
21:             ret = FALSE;
22:             if ( (c,expr) ∈ QA)
23:                  ret = \|expr\|_P^v;
24:                  if (ret = TRUE)
25:                       return TRUE;
26:                  else
27:                       for (l ∈ Var(expr))
28:                            v = v \ { (l→TRUE),
                                        (l→INTERMEDIATE),
                                        (l→FALSE) } ;
29:             if (∃ l. (l→TRUE) ∈ v)
30:                  return TRUE;
31:             else if (ret = INTERMEDIATE ∨
                                   ∃ l. (l→INTERMEDIATE) Î í)
32:                       return INTERMEDIATE;
33:                  else return FALSE;
34:        }
35:  }
```

workflows are also known and, thus, they do not require service discovery considering message sequences. If a collaboration is established without a global pre-defined workflow (Finin et al., 1994; Kafeza et al., 2001; Kimbrough & Moore, 1997; van der Aalst, 1999), ,service discovery based on matchmaking message sequences comes into place.

Typically, matchmaking must get along with incomplete information (Casati & Discenza, 2000; Georgakopoulos et al., 1999), because trading partners will not publish their business critical information like the highest price a customer is willing to pay within a negotiation or auction process. The creation of consistent global workflow from local ones considering this limitation is an open research issue.

Other matchmaking approaches are based on semantic information (Berbers-Lee et al., 2001, McIlraith et al., 2001). In Sycara et al. (1999) a language is introduced describing the functional aspects as well as the messages and their parameters based on a domain specific ontology. DAML-S (DAML-S COALITION, 2002) uses workflow aspects as well as the functional semantic description of the service within the matchmaking. In contrast to model the complete service description using semantic Web technology, the Web service offering language (WSOL) provides additional semantic meta-information to increase precision of the service discovery. In particular, classes of services are modeled by specifying functional constraints, QoS, simple access rights, price, and other constraints in addition to a WSDL description (Tosic et al., 2002). An even more simplified approach (Bernstein & Klein, 2002; Klein et al., 2001) uses a process ontology to improve precision of key word based querying. The main drawback of semantic annotation is the necessity of a common ontology used for annotating and querying services. Unfortunately, no such ontology currently is in place.

Logic based approaches addressing service discovery are Web Service Request Language (WSRL) and Product Lifecycle Management PLM_*flow*. WSRL (Aiello et al., 2002; Papazoglou et al., 2002) addresses planning of an orchestration and composition of services to fulfill user requirements. While WSRL performs service discovery on behalf of temporal and linear constraints, PLM_*flow* (Zeng et al., 2002) is based on rule inferencing using the specified business rules rather than a fixed workflow. Thus, PLM_*{flow}* is characterized as a rule-based non-deterministic workflow engine aiming to establish cooperation based on local decidability of the trading partners of their involvement. These approaches are based on the fact that the local workflow model/rules are provided to the trading partners and do not consider the need of hiding business critical information.

The approach presented in the article is based on modeling message sequences derived from a local workflow model. As stated above, (Molina-Jimenez et al., 2003) presents a similar approach, but does not distinguish between optional and required transitions. In Mecella et al. (2001), two e-services are compatible if every possible trace in one service has got a compatible one in the second one. Unfortunately, the description of the compatibility check of the traces is not described at all.

SUMMARY & FUTURE WORK

This article has introduced an approach to match business process descriptions. By explicating message sequence and required messages, such descriptions allow

for more precise matches than current approaches limited to matching only individual messages. Thereby, an implementation of the extended finite state automata has been presented that can be deployed as a service description language for more precise service discovery.

Currently, the approach is used to implement a business process repository supporting matchmaking and discovery based on business process descriptions. Next steps are the evaluation of the expressiveness of the approach by investigating the mapping from process models like BPEL to annotated deterministic finite state automata. Because the expressiveness differs the correctness of the matchmaking will be analyzed. In particular, we will investigate the percentage of introduced false matches. Finally, future work will investigate the applicability of bilateral matching for business processes to multi-lateral process matchmaking.

REFERENCES

Aiello, M., Papazoglou, M., Yang, J., Pistore, M., Carman, M., Serafini, L., & Traverso, P. (2002). A request language for Web services based on planning and constraint satisfaction. *Proceedings of the Third International Workshop, Technologies for E-Services (TES)* (pp. 76-85).

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., &Weerawarana, S. (2003). *Business process execution language for Web services, Version 1.1.*

Ariba. (2000). Inc. Ariba, IBM Corporation, and Microsoft Corporation. Universal description, discovery and integration. *http://www.uddi.org/*

Ballinger, K., Brittenham, P., Malhotra, A., Nagy, W.A., & Pharies, S. (2001) Web services inspection language (ws-inspection) 1.0. *http://www.ibm.com/developerworks/library/ws-wsilspec.html*

Banerji, A., Bartolini, C., Beringer, D., Chopella, V., Govindarajan, K., Karp, A., Kuno, H., Lemon, M., Pogossiants, G., Sharma, S., & Williams, S. (2002). Web services conversation language (WSCL) 1.0 W3C note. *http://www.w3.org/TR/wscl10/.*

Berbers-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic Web. *Scientific America, 284*(5), 34-43.

Bernstein, A., & Klein, K. (2002). Discovering services: Towards high-precision service retrieval. *Proceedings of CAiSE International Workshop, Web Services, E-Business, and the Semantic Web (WES)* (pp. 260-275).

Casati, F., & Discenza, A. (2000). *Modeling and managing interactions among business processes.* Technical Report HPL-2000-159, Hewlett Packard Laboratories.

Chomicki, J., & Saake, G. (1998). *Logics for database and information systems.* Kluwer.

Christensen, E., Curbera, F., Meredith, G., & Weerawarana, S. (2001). Web services description language (WSDL) 1.1 W3C note. *http://www.w3.org/TR/wsdl*

DAML-S Coalition. (2002). *DAML-S: Web service description for the semantic Web.*

Esparza, J., & Nielsen, M. (1994). Decibility issues for Petri nets - a survey. *Journal of Informatik Processing and Cybernetics, 30*(3), 143-160.

Finin, T.W., Fritzson, R., McKay, D., & McEntire, R. (1994). KQML as an agent communication language. *CIKM* (pp. 456-463).

Georgakopoulos, D., Schuster, H., Cichocki, A., & Baker, D. (1999). Managing process and service fusion in virtual enterprises. *Information Systems, 24*(6), 429-456.

Grefen, P., Aberer, K., Hoffner, Y., & Ludwig, H. (2000). Crossflow: Cross-organizational workflow management in dynamic virtual enterprises. *International Journal of Computer Systems Science & Engineering, 15*(5), 277-290.

Hanson, J.E., Nandi, P., & Levine, D.W. (2002). Conversation-enabled Web services for agents and e-business. *Proc. of the International Conference on Internet Computing (IC-02)* (pp. 791-796).

Hopcroft, J.E., Motwani, R., & Ullman, J.D. (2001). *Introduction to automata theory, languages, and computation.* Addison Wesley.

Kafeza, E., Chiu, D.K.W., & Kafeza, I. (2001). View-based contracts in an e-service cross-organizational workflow environment. *TES* (pp. 74-88).

Kimbrough, S.O., & Moore, S.A. (1997) On automated message processing in electronic commerce and work support systems: Speech act theory and expressive felicity. ACM *Transactions on Information Systems, 15*(4), 321-367.

Klein, M., & Bernstein, A. (2001). *Searching for services on the semantic Web using process ontologies.* Proceedings of the First Semantic Web Working Symposium (SWWS-1).

Klingemann, J., Wäsch, J., & Aberer, K. (1999). *Adaptive outsourcing in cross-organizational workflows.* Proceedings of the 11th Conference on Advanced Information Systems Engineering (Caise).

Levine, P., Clark, J., Casanave, C., Kanaskie, K., Harvey, B., Clark, J., Smith, N., Yunker, J., & Riemer, K. (2001). Business process specification schema. *www.ebxml.org/specs/ebBPSS.pdf*

McIlraith, S., Son, T., & Zeng, H. (2001). Semantic Web services. *IEEE Intelligent Systems* (Special Issue on the Semantic Web).

Mecella, M., Pernici, B., & Craca, P. (2001). Compatibility of e-services in a cooperative multi-platform environment. *TES* (pp. 44-57).

Molina-Jimenez, C., Shrivastava, S., Solaiman, E., & Warne, J. (2003). *Contract representation for run-time monitoring and enforcement.* Proc. of Conf. on Electronic Commerce (CEC).

Panti, G. (1995). Multi-valued logics. In D. Gabbay & P. Smets (Eds.) *Handbook of defeasible reasoning and uncertainty management systems* (vol. 1, Quantified representation of uncertainty and imprecision, ch. 2, pp. 25-74). Kluwer.

Papazoglou, M., Aiello, M., Pistore, M., & Yang, J. (2002). Planning for requests against Web services. *Bulletin of the Technical Commitee on Data Engineering, 25*(4), 41-46.

Peterson, J.P. (1981). *Petri net theory and the modeling of systems.* Prentice-Hall.

Sycara, K.P., Klusch, M., Widoff, S., & Lu, J. (1999). Dynamic service matchmaking among agents in open information environments. *SIGMOD Record, 28*(1), 47-53.

Tosic, V., Patel, K., & Pgurek, B. (2002). WSOL - Web service offering language.

Proceedings of CAiSE International Workshop, Web Services, E-Business, and the Semantic Web (WES) (pp. 5-67).

van der Aalst, W.M.P. (1999). Interorganizational workflows: An approach based on message sequence charts and petri nets. *Systems Analysis - Modelling - Simulation, 34*(3), 335-367.

van der Aalst, W.M.P., & Weske, M. (2001). *The P2P approach to interorganizational workflows.* Proc. of 13. Int. Conf. on Advanced Information Systems Engineering (CAISE'01).

Zeng, L., Flaxer, D., Chang, H., & Jeng, J.J. (2002). PLM_flow - dynamic business process composition and execution by rule inference. Proceedings of the Third International Workshop, Technologies for E-Services (TES) (pp. 141-150).

Please provide bios