Transforming BPEL into annotated Deterministic Finite State Automata for Service Discovery

Andreas Wombacher Peter Fankhauser
Erich Neuhold
Fraunhofer Gesellschaft, Integrated Publication and Information Systems Institute,
64293 Darmstadt, Germany
firstname.lastname@ipsi.fhg.de

Abstract

Web services advocate loosely coupled systems, although current loosely coupled applications are limited to stateless services. The reason for this limitation is the lack of a method supporting matchmaking of state dependent services exemplarily specified in BPEL. In particular, the sender's requirement that the receiver must support all possible messages sent at a certain state are not captured by models currently used for service discovery. Annotated deterministic finite state automata provide this expressiveness. In this paper the transformation of a local process specification given in BPEL to annotated deterministic finite state automata is presented.

1. Introduction

Web services and related technologies promise to facilitate efficient execution of B2B e-commerce by integrating business applications across networks like the Internet. A lot of effort has been expended to define standards, e.g., to model business processes and describing workflows and interfaces (BPEL4WS, WSDL, etc.) as well as specifying the technical infrastructure for carrying out business transactions (e.g., SOAP, UDDI). Conceptually, web services are advertised as a technology for implementing loosely coupled business processes, that is a dynamic and flexible binding of services. Nowadays, web services are mainly deployed as stateless components accessible via a single request-response remote procedure call (RPC). One reason for this limited deployment is the missing support for searching and finding state-maintaining/ complex web services. UDDI and the corresponding distributed approach WSIL support searching of name-value pairs, which does not suffice for service discovery of complex services. In particular, it can not generally be guaranteed by matching name-value pairs that the service results in a deadlock-free and bounded business transaction. Thus, several extensions of UDDI have been proposed such as addressing Service Level Agreement [8], semantic [4], or logical [1] extensions of UDDI. An extension of UDDI taking care of internal states (process specification) of a service is discussed in [19].

Service discovery for the Business Process Execution Language (BPEL) which explicate internal states of processes requires an adequate formal model of BPEL [2]. This paper presents a transformation of major parts of BPEL to such a formal model providing a matchmaking definition.

The two major requirements for the matchmaking is to guarantee that i) all messages sent by a party are supported by the receiving party, while ii) messages offered by a receiving party, but not required by a sending party might be neglected. A formal model fulfilling these requirements is annotated deterministic finite state automata (aDFA) [20] modeling matchmaking by means of automata intersection. This paper presents the mapping of BPEL to aDFAs in a constructive way.

The paper is organized as follows: Section 2 describes a sample application, Section 3 sketches the overall approach, Section 4 briefly introduces the used formal models, followed by a presentation of the transformation in Section 5. The paper concludes with related work in Section 6 and a summary and future work in Section 7.

2. Example

The example scenario used for further discussion is a simple procurement workflow within a virtual enterprise comprising a buyer, an accounting department, and a logistics department. The accounting department approves an order (*order* message) sent by a buyer and forwards the order to the logistics department (*deliver* message) to deliver the requested goods. The logistics department confirms the re-

Proceedings of the IEEE International Conference on Web Services (ICWS'04)
0-7695-2167-3/04 $ 20.00 IEEE

ceipt (*deliver_conf* message) and forwards it to the buyer extended by the expected deliver date and the parcel tracking number using the *delivery* message. Further, the buyer may perform parcel tracking (*get_status* and *status* messages) of the shipped goods, which is forwarded by the accounting department to the logistics department. The overall scenario is depicted in Figure 1.

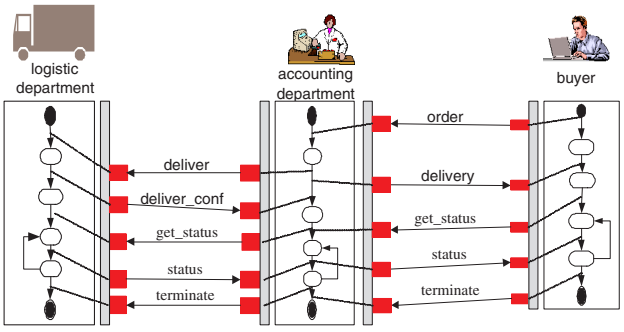

Figure 1. Global Procurement Scenario

The scenario sketched above represents the global workflow, while the local workflows can be derived from the global one, e.g., by using [17]. In the following the local workflow of the accounting department denoted according to the BPEL specification is described in more detail. To keep the example simple, the detailed structure of messages is neglected and the names of messages are simplified. Concrete message structures could be, e.g., taken from the RosettaNet Partner Interface Processes (PIPs) 3A4 (Request Purchase Order), 3A7 (Notify of Purchase Order Update), 3B2 (Notify of Advanced Shipment) [13].

Within web service specifications one or more messages specify an operation representing a potential message exchange. If an operation contains only a single input message, then the operation is asynchronous, otherwise it is synchronous. A porttype contains a set of operations provided by a service provider, which is specified in the corresponding WSDL file. Figure 2 depicts the operations and the related messages used in the exemplary accounting department BPEL description, where the messages are labeled in accordance to messages in the global workflow depicted in Figure 1. The *buyer* and the *logistics* porttypes represent the operations provided by the corresponding department service, that is contain messages that are received by the buyer and logistics department respectively. Consequently, the *accBuyer* and *accLogistics* porttypes contain operations received by the accounting department and sent by the buyer and the logistics department respectively. All operations are asynchronous except the synchronous *getStatusOP* operation provided in the *logistics* porttype.

```
<portType name="accBuyer">
    <operation name="orderOp">
        <input message="tns:order"/>
    </operation>
    <operation name="getStatusOp">
        <input message="tns:getStatus"/>
    </operation>
    <operation name="terminateOp">
        <input message="tns:terminate"/>
    </operation>
</portType>
<portType name="buyer">
    <operation name="deliveryOp">
        <input message="tns:delivery"/>
    </operation>
    <operation name="statusOp">
        <input message="tns:status"/>
    </operation>
</portType>
```

```
<portType name="accLogistics">
    <operation name="outOfStockOp">
        <input message="tns:delivery"/>
    </operation>
    <operation name="deliver_confOp">
        <input message="tns:deliver_conf"/>
    </operation>
</portType>
<portType name="logistics">
    <operation name="getStatusOp">
        <input message="tns:getStatus"/>
        <output message="tns:status"/>
    </operation>
    <operation name="deliverOp">
        <input message="tns:deliver"/>
    </operation>
    <operation name="terminateOp">
        <input message="tns:terminate"/>
    </operation>
</portType>
```

Figure 2. WSDL porttype definition of the exemplary scenario

The description of the local workflow is based on these porttype definitions by directly referencing them. Local workflows are denoted in BPEL [3], where a workflow is specified in terms of tasks (named *activities* in BPEL terminology) representing basic pieces of work to be performed by potentially nested services. The control flow of a BPEL process constrains the performance of tasks by selective (*switch* and *pick* activities), sequence (*sequence* activity), and parallel (*flow* activity) execution. In addition, a BPEL process also defines the data flow (variable handling and *assign* activity) of the business process regardless of concrete implementation of tasks. Based on this understanding, a workflow model includes activities realizing the interaction with partners represented by exchanging messages (*receive*, *reply*, *invoke*, and *pick* activities).

The BPEL specification of the accounting department local workflow is depicted in Figure 3. The partnerLink definition associates a partner name to a bilateral interaction between two roles. The association of roles to porttypes is done in the partnerLinkType definition contained in the WSDL file, where each role is related to porttypes and operations received by itself. Thus, buyer and logistics department are mapped to the corresponding porttypes, while accounting department is related to accBuyer and accLogistics porttpyes.

The process starts by receiving an *order* message sent by the buyer, which is forwarded to the logistics department via a *deliver* message. The logistics department answers asynchronously with a *deliver_conf* message, which is forwarded by the accounting process to the buyer via a *delivery* message. Due to the fact that the buyer is allowed to do parcel tracking an undetermined number of times, the parcel tracking must be contained within a non-terminating loop. To enable a termination of the accounting and logistics

```
<process>
....
    <sequence name="accounting department process">
        <receive partnerLink="buyer" portType="tns:accounting"
                 operation="orderOp" variable="order"/>
        <invoke partnerLink="logistic" portType="tns:logisticCallback"
                operation="deliverOp" inputVariable="order"/>
        <receive partnerLink="logistic" portType="tns:logistic"
                 operation="deliver_confOp" inputVariable="order"/>
        <invoke partnerLink="buyer" portType="tns:accountingCallback"
                operation="deliveryOp" inputVariable="order"/>
        <while name="parcel tracking" condition="1=1">
            <pick>
                <onMessage partnerLink="buyer" portType="tns:accounting"
                           operation="getStatusOp" variable="getStatus">
                  <sequence>
                    <invoke partnerLink="logistic" portType="tns:logisticCallback"
                            operation="getStatusOp" .../>
                    <invoke partnerLink="buyer" portType="tns:accountingCallback"
                            operation="statusOp" inputVariable="status"/>
                  </sequence>
                </onMessage>
                <onMessage partnerLink="buyer" portType="tns:accounting"
                           operation="terminateOp" variable="terminate">
                  <sequence>
                    <invoke partnerLink="logistic" portType="tns:logisticCallback"
                            operation="terminateOp" inputVariable="terminate"/>
                    <terminate/>
                  </sequence>
                </onMessage>
            </pick>
        </while>
    </sequence>
</process>
```

Figure 3. BPEL notation of the accounting local workflow

department processes a *termination* message initiated by the buyer and forwarded by the accounting to the logistics department terminates the corresponding processes. Alternatively, the accounting department may receive a *get_status* message sent by the buyer, which is forwarded by a synchronous invocation of the logistics process and reporting the status via a *status* message back to the buyer.

3. Overall Approach

Given the local workflow for the accounting department the task of matchmaking is to find the complementary workflows for the logistics and the buyer process that guarantee a successful business interaction.

Figure 4 contains potential trading partners of the accounting workflow. In particular, the buyer workflow is quite similar to the one described above, but requires an additional *status* message not supported by the accounting. As a consequence, the depicted buyer process does not match the accounting process due to different requirements of the trading partners on message ordering. The logistics process depicted in Figure 4 does not support parcel tracking. In particular, the get_status and status messages are not supported, but they are mandatory from the point of view of the accounting process. Thus, the logistics process does not match due to not providing messages sent by the accounting, so considered to be mandatory for a trading partner.

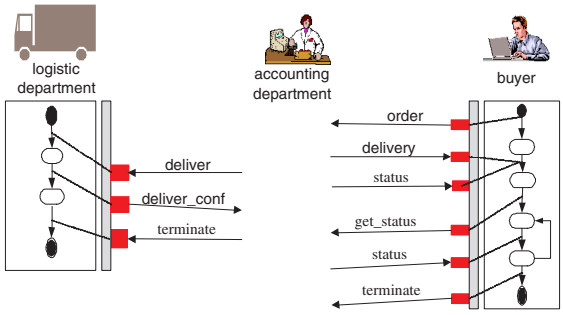

Figure 4. potential trading partners

Opposed to the above negative examples, the logistics and the buyer depicted in Figure 1 match the accounting process.

The definition of matchmaking stateful web services specified in BPEL requires a formal model, which is not provided by BPEL itself [18]. The model used in this paper is annotated Deterministic Finite State Automata (aDFA) [20] being an extension of deterministic finite state automata. A finite state automata approach has been selected, because the BPEL subset used in this paper can be interpreted as message sequences representing a regular language. Thus, an approach based on standard automata suffices and no more complex models like Petri Nets [12], Workflow Nets [16], or Statecharts [6] are required. Based on this formal model matchmaking is defined as an intersection of aDFA automata [20].

Since, we do not expect people to provide descriptions of processes in aDFA notation, the aim of this paper is to provide a transformation from BPEL to aDFA notation. The transformation is quite similar to transformations, e.g., from regular expressions to finite state automata (for example Berry-Sethi Algorithm).

The approach performs the structural traversal of the BPEL XML document and recursively transforms an activity by interrelating the already transformed child activities. Thus, an activity is transformed by representing it in a partial structure combined by the partial structures of the child activities. These partial structures must maintain an input state q_{in} representing the state to enter the partial structure and an output state q_{out} representing the state to leave the partial structure finally. This partial structure is called partial aDFA and is formally defined in the next section.

The recursion has to start with the first activity within the BPEL document being a child element of the <*process*> element. So, the transformation of the <*process*> is different from the one of the activity elements, because here the recursion is initiated by creating a start and final state passed to the partial structure used in the recursion as input and

output places. Finally, the mapping from the partial aDFA to the aDFA structure is performed.

4. Formal Model

In the following the required formal definitions of aDFA and partial aDFA is introduced. A more detailed discussion can be found in [20].

Definition 1 ***(annotated Deterministic Finite State Automata (aDFA))***
An annotated deterministic finite state automaton A is represented as a tuple
$A = (Q, \Sigma, \delta, q_0, F, QA)$ *where Q is a finite set of states, Σ is a finite set of messages, $\delta : Q \times \Sigma \rightarrow Q$ 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 [5]:

Definition 2 ***(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 $\neg\phi$,*
- *if ϕ and ψ are formulas, so is $\phi \wedge \psi$ and $\phi \vee \psi$.*

Based on the aDFA definition, an intersection and emptiness operation has been defined in [20], which is quite similar to the one of standard automata. In particular, intersection combines annotation of states by conjunction, while the emptiness test checks reachability of final states not only via a single transition, but via all transitions contained in a conjunction of an annotation. A detailed discussion of the sketched operations and their applicability to matchmaking business processes is given in [19].

Based on the aDFA definition above, the partial aDFA is defined. As stated above, this structure is used by the recursive transformation of the BPEL description to aDFA notation, where the input state and the output state represent the states to enter / leave the partial structure used within the recursion step.

Definition 3 ***(partial aDFA)***
A partial annotated deterministic finite state automaton PA *is an aDFA extended by an output state $q_{out} \in Q$ and an input state q_{in} by not passing by the start state q_0. The resulting signature is $PA = (Q, \Sigma, \delta, q_{in}, q_{out}, F, QA)$.*

When constructing a partial aDFA by combining lower level partial aDFAs, the corresponding input and output states must be interrelated to exemplarily form a sequence. In particular, states of different partial aDFAs are considered to be equivalent, thus one of them need to be renamed by another one. So, a renaming function
$\tilde{\sigma} : (Q \cup \{\varepsilon\}) \times (Q \times Q) \rightarrow (Q \cup \{\varepsilon\})$ of states on partial aDFAs is defined, where ε represents a non-existing state .

Definition 4 ***(renaming of states)***
A state q is renamed with q' by function $\tilde{\sigma}$ defined for a state $\tilde{q}$ with

$$\tilde{\sigma}(\tilde{q}, q \rightarrow q') := \begin{cases} q' & if\ \tilde{q} = q \\ \tilde{q} & otherwise \end{cases}$$

Extending the above definition to partial automata results in a renaming function $\sigma : PA \times (Q \times Q) \rightarrow PA$ which renames a state of the input annotated automaton PA resulting in a new automaton PA' by adding the new state to the set of states of PA', renaming all source and target states in transitions of PA', and renaming the corresponding variables within annotations of PA'.

Definition 5 ***(renaming of partial aDFA)***
Let $PA = (Q, \Sigma, \delta, q_{in}, q_{out}, F, QA)$ and
$PA' = (Q', \Sigma, \delta', q'_{in}, q'_{out}, F', QA')$ *be partial automata. Then, $PA' := \sigma(PA, q \rightarrow q')$ with $Q' := \{q'\} \cup (Q \setminus \{q\})$, $\delta' := \{(\tilde{\sigma}(q_1, q \rightarrow q'), l, \tilde{\sigma}(q_2, q \rightarrow q')) \mid (q_1, l, q_2) \in \delta\}$, $q'_{in} := \tilde{\sigma}(q_{in}, q \rightarrow q')$, $q'_{out} := \tilde{\sigma}(q_{out}, q \rightarrow q')$,*

$$F' := \bigcup_{\tilde{q} \in F} \tilde{\sigma}(\tilde{q}, q \rightarrow q')$$

$$QA' := \bigcup_{(\tilde{q}, \tilde{e}) \in QA \setminus \{(q', e')\}} \begin{cases} (q', e' \wedge \tilde{e}) & if\ \tilde{q} = q \\ \emptyset & if\ \tilde{q} = \varepsilon \\ (\tilde{q}, \tilde{e}) & otherwise \end{cases}$$

5. Transformation

The transformation translates BPEL syntax to annotated DFA. In particular, the transformation represents messages that might be sent by a party at a particular state as messages that *must* be supported by the corresponding receiving party. This is because the sender has the choice to select a particular message to be sent, while the receiving party must be able to handle all possible choices of the sender. This is modeled by the sender workflow annotating each choice of sending messages as mandatory transitions, that is a conjunction of message labels. In contrast, a receiving party represents all supported options as genuine alternatives via a disjunction of message labels. This explicit modeling of mandatory transitions of a sender and optional transitions of a receiver is the main contribution of the approach.

5.1. Example

The query process of the accounting department as depicted in Figure 3 can be translated into an aDFA by the following mapping of BPEL activities:

- represent *send*, *receive*, *pick*, and *invoke* activities as transitions
- *switch* and *pick* activities represent choices, that is modeled as several transitions each connected with the current place by an input arc
- a *flow* activity represents a parallel execution, that is modeled by enumerating all possible execution sequences of the parallel execution
- data management operations like *assign* are neglected

The aDFA model derived by this transformation is depicted in Figure 5, where the annotations of the transitions are partnerlink *pl*, porttype *pt*, operation *op*, and direction *dir* depicted as $pl\#pt\#op\#dir$.

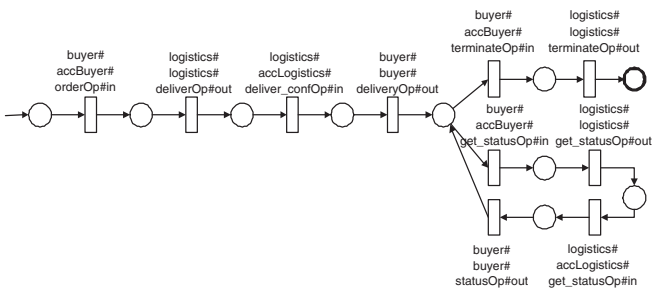

Figure 5. aDFA notation of the accounting local workflow

The process is started by the buyer sending an *order* message to the accounting department, which forwards the order to the logistics department via a *deliver* message. The logistics department confirms this request (*deliver_conf* message extending the provided information with the planned delivery date and the parcel tracking number) to the accounting department, which forwards the delivery details of the order (*delivery* message) to the buyer. Afterwards, the buyer is allowed to do parcel tracking with the logistics department, where the accounting department acts as a proxy to the buyer.

A detailed formal description of the transformation is introduced next.

5.2. Messages

The representation of interactions in BPEL and aDFA differ, because BPEL is based on communication activities while aDFA is based on exchanged messages between trading partners. Communication activities are either synchronous or asynchronous, and the specification is done in terms of partnerlink, porttype and operation. The latter one are specified in a corresponding WSDL file (as already discussed in section 2). The partnerlink is defined in the BPEL document and implicitly assigns roles to porttypes. While asynchronous communication can be characterized by porttype and operation executed via a partnerlink, synchronous communication requires a differentiation between outgoing and incoming message. This is because a receiving trading partner models synchronous communication by two seperate activities (*receive* and *reply*). Thus, an additional parameter '*in*' or '*out*' expressing the direction of the sender of the synchronous exchange is added to the corresponding message. Thus, an exchange within a bilateral interaction is characterized by: portType, operation, direction, and partnerLink (latter one to enable a generalization of the approach from bilateral to multi-lateral matchmaking). A definition of a message equivalence relation used for the intersection and emptiness test is as follows:

$$(pl, pt, op, dir) \approx \quad (pl', pt', op', dir') \iff pt = pt' \wedge op = op' \wedge dir = dir'$$

Note that *pl* and *pl'* are not taken into account, because message equivalence does not depend on the local role model.

The underlying assumption here is that the two process to be compared reference the same WSDL document, thus guaranteeing that portType and operation are referencing to commonly agreed messages by the bilateral trading parties.

5.3. Process Element

The recursive transformation starts at the top level element <*process*> of BPEL resulting in a partial aDFA representing the child activity.

<*process*> $activity$ </*process*>

The recursion starts by transforming the $activity$ by the rules below resulting in a partial aDFA PA, which can be represented as an aDFA A by adding the output state q_{out} to the set of final states and assigning the input state q_{in} as the start state q_0 resulting in an aDFA
$A = (Q, \Sigma, \delta, q_{in}, F \cup \{q_{out}\}, QA)$
with $PA := (Q, \Sigma, \delta, q_{in}, q_{out}, F, QA)$.

5.4. Internal Activities

Internal activities do not need to be represented in a description of the bilateral interaction. Such activities are: *scope*, *assign*, or *wait*. To provide a full composability of the transformation, internal activities are represented by an *empty* activity.

5.5. Simple Activities

Simple activities are related to a single state without having a transition, thus, the corresponding annotation to the

state is $true$. Simple activities are *empty* and *termination* activity.

5.5.1. empty activity denoted in BPEL as <*empty/*> is represented as a partial aDFA by a single state only:

$$PA = (\{s_0\}, \emptyset, \emptyset, s_0, s_0, \emptyset, \{(s_0, true)\})$$

5.5.2. terminate activity denoted in BPEL as <*terminate/*> is modeled by a single state, where no further activity can be appended, thus, the output state is an empty state. Further, the input state is marked final.

$$PA = (\{s_0\}, \emptyset, \emptyset, s_0, \varepsilon, \{s_0\}, \{(s_0, true)\})$$

5.6. Communication Activities

Communication activities exchange messages with trading parties. They are represented as partial aDFAs with a single transition per exchanged message, while the corresponding state is annotated with the transition label in case of sending activities, or with true in case of receiving activities. The output state is the state reached after the last message has been exchanged.

5.6.1. reply and asynchronous invocation activities denoted in BPEL as

<reply partnerLink="pl" portType="pt"
operation="op" variable="var"/>
<invoke partnerLink="pl" portType="pt"
operation="op" inputVariable="var"/>

are modeled as a single transition with an message label at the transition source state represented as

$$\begin{aligned} PA = (\quad & \{s_0, s_1\}, \{(pl, pt, op, in)\}, \\ & \{(s_0, (pl, pt, op, in), s_1)\}, \\ & s_0, s_1, \emptyset, \{(s_0, (pl, pt, op, out)), (s_1, true)\}) \end{aligned}$$

5.6.2. receive activity denoted in BPEL as

<receive partnerLink="pl" portType="pt"
operation="op" variable="var"/>

is modeled as a single transition annotated with true represented as

$$\begin{aligned} PA = (\quad & \{s_0, s_1\}, \{(pl, pt, op, out)\}, \\ & \{(s_0, (pl, pt, op, out), s_1)\}, \\ & s_0, s_1, \emptyset, \{(s_0, true), (s_1, true)\}) \end{aligned}$$

5.6.3. synchronous invoke activity denoted in BPEL as

<invoke partnerLink="pl" portType="pt" operation="op"
inputVariable="var" outputVariable="var2"/>

is modeled by two transitions, while the first one is sending transition annotated with the message name, the second one is a receiving transition annotated with true. The resulting partial aDFA is represented as

$$\begin{aligned} PA = (&\{s_0, s_1, s_2\}, \{(pl, pt, op, in), (pl, pt, op, out)\}, \\ &\{(s_0, (pl, pt, op, in), s_1), (s_1, (pl, pt, op, out), s_2)\}, \\ &s_0, s_2, \emptyset, \{(s_0, (pl, pt, op, in)), (s_1, true), (s_2, true)\}) \end{aligned}$$

The remaining communication activity *pick* will be discussed later, because it is a mix of structural and communication activities.

5.7. Structural Activities

Structural activities are *sequence*, *while*, *switch*, and *flow*. They take some partial automata $PA_0, \ldots, PA_n$ and combine them to a new partial automaton PA with $PA_i := (Q_i, \Sigma_i, \delta_i, q_{in,i}, q_{out,i}, F_i, QA_i)$ for $i = 0, \ldots, n$ and $PA = (Q, \Sigma, \delta, q_{in}, q_{out}, F, QA)$.

5.7.1. while activity denoted in BPEL by

<while condition="cond"> PA_1 </while>

allows a single activity inside the *while* activity only. The loop is created in the partial aDFA by replacing the output state with the input state, formally denoted as

$$PA = \sigma(PA_1, q_{out,1} \rightarrow q_{in,1})$$

The previous output state $q_{out,1}$ is disabled.

5.7.2. sequence activity denoted in BPEL by

<sequence> $PA_1\ PA_2\ \ldots\ PA_n$ *</sequence>*

connects the independent partial aDFAs by renaming the input state of PA_{i+1} with the output state of PA_i of all partial aDFAs except the last one, that is PA_n which remains unchanged formally denoted as

$$PA = PA_n \cup \Big(\bigcup_{i=1}^{n-1} \sigma(PA_i, q_{out,i} \rightarrow q_{in,i+1}) \Big)$$

5.7.3. flow activity denoted in BPEL by

<flow> $PA_1\ PA_2\ \ldots\ PA_n$ *</flow>*

specifies parallel execution of the partial automata $PA_1, \ldots, PA_n$. Automata do not provide means to model parallel execution, thus, the resulting execution sequences must be enumerated. A well known operation to generate these enumeration is the shuffle product. In particular, the shuffle product keeps the message order within each message sequence, but combines two message sequences in all possible combinations. A standard algorithm of the shuffle product can be found in [9]. Applied to partial aDFA the shuffle product forms a conjunction of the annotations of the combined states.

Definition 6 *(shuffle annotated automata definition)[9] The shuffle product* $PA := PA_1 \& PA_2$ *of* PA_1 *and* PA_2 *is defined as* $Q := Q_1 \times Q_2$, $\Sigma := \Sigma_1 \cup \Sigma_2$, $q_{in} := q_{in,1} \times q_{in,2}$, $q_{out} := q_{out,1} \times q_{out,2}$, $F := F_1 \times F_2$,

$$\begin{aligned}\Delta := \quad & \{((p,q_1),\alpha,(p,q_2)) \in \\ & (Q_1 \times Q_2) \times \Sigma_2 \times (Q_1 \times Q_2) \mid (q_1,\alpha,q_2) \in \Delta_2\} \\ \cup \quad & \{((p_1,q),\alpha,(p_2,q)) \in \\ & (Q_1 \times Q_2) \times \Sigma_1 \times (Q_1 \times Q_2) \mid (p_1,\alpha,p_2) \in \Delta_1\}\end{aligned}$$

$$QA = \bigcup_{(q_1,e_1)\in QA_1,(q_2,e_2)\in QA_2} ((q_1,q_2), e_1 \wedge e_2)$$

Based on the shuffle product definition the *flow* activity can easily be transformed into partial aDFA by shuffling all partial automata and finally rename the combination of the input states of all automata by a new input state, and the combination of all output states with a new output state respectively. The formal definition is given below

$$PA = \sigma\Big(\quad \sigma\big(\&_{i=1}^{n} PA_i, (q_{in,1},\ldots,q_{in,n}) \rightarrow q_{in}\big), \\ (q_{out,1},\ldots,q_{out,n}) \rightarrow q_{out}\Big)$$

5.7.4. switch activity denoted in BPEL by

<switch>
<case condition="$cond_1$"> PA_1 </case>
<case condition="$cond_2$"> PA_2 </case>
$\vdots$
<case condition="$cond_n$"> PA_n </case>
<otherwise> PA_0 </otherwise>
</switch>

specifies an internal choice performed by evaluating the condition statements being XPath 1.0 boolean expressions. This choice is represented in partial aDFAs by introducing a new input state q_{in} and an output state q_{out}, and renaming input and output states of $PA_0,\ldots,PA_n$ by q_{in} and q_{out} respectively. The formal definition is

$$PA = \bigcup_{i=0}^{n} \sigma\big(\sigma(PA_i, q_{in,i} \rightarrow q_{in}), q_{out,i} \rightarrow q_{out}\big)$$

The presented approach is based on the assumption that all conditions are pairwise disjoint.

5.7.5. pick activity denoted in BPEL as

<pick>
<onMessage partnerLink="pl_1" portType="pt_1"
operation="op_1" variable="var_1"> PA_1 </onMessage>
$\vdots$
<onMessage partnerLink="pl_n" portType="pt_n"
operation="op_n" variable="var_n"> PA_n </onMessage>
<onAlarm> PA''_0 </onAlarm>
</pick>

is a combination of a *switch* activity applied to several sequences of a *receive* activity and a partial automaton PA_i. In the current modeling the time constraints which might be expressible in *onAlarm*, that is *for* and *until* , are not considered.

Each *onMessage* element is modeled as a single receive transition formally described as

$$\begin{aligned}PA'_j := \quad & (\{s_0,s_1\}, \{(pl_j, pt_j, op_j, in)\}, \\ & \{(s_{0,j}, (pl_j, pt_j, op_j, in), s_{1,j})\}, s_{0,j}, s_{1,j}, \emptyset, \\ & \{(s_{0,j}, true), (s_{1,j}, true)\})\end{aligned}$$

Each of these receive transitions is sequentially combined with the corresponding PA_j in accordance with the *sequence* activity formally described as

$$PA''_j := \sigma(PA'_j \cup PA_j, q'_{out,j} \rightarrow q_{in,j})$$

Finally, the above constructed sequences are combined by a choice resulting in the final partial automaton formally denoted as

$$PA := \bigcup_{i=0}^{n} \sigma\big(\sigma(PA''_i, q''_{in,i} \rightarrow q_{in}), q''_{out,i} \rightarrow q_{out}\big)$$

5.8. Limitations

The transformation defined above is partial, in particular, the attributes *joinCondition* and *suppressJoinFailure*, as well as the elements *link* and *throw* have not been considered. The first ones are relevant to process execution only thus do not effect the matchmaking. The latter ones are introducing additional dependencies between the different activities, which can not be resolved in a recursive traversal of the BPEL, but require a post-processing to reflect these additional constraints.

6. Related Work

Service discovery is a hot research issue. In particular, a lot of work is dedicated to define service descriptions addressing specific aspects required for service discovery.

There exist logic based approaches like for example the Web Service Request Language (WSRL) [1], where the corresponding matchmaking is based on temporal and linear constraint satisfaction. Further, DAML-S [4] aims to describe services in semantic web communities. The main draw back of semantic annotation is the necessity of a common ontology used for annotating and querying services. Unfortunately, no such ontology currently is in place. While process annotations as discussed in this paper are addressed within several papers [10, 15], unfortunately no concrete approach is mentioned. In [11] an approach based on state machines is presented providing the same expressiveness of finite state automata, but the approach has not been related to concrete process specification languages like BPEL, and further does not address the issue of a receiver guaranteeing the support of the senders potential choice.

All the above mentioned approaches extend the currently established service repository being based on UDDI [7], although several extensions already exists, like for example [14, 8, 21], none of them has addressed the issue of process annotation as presented in this paper.

7. Summary and Future Work

This paper presented the concept of matching business processes in loosely coupled architectures. In particular, a transformation from BPEL to annotated deterministic finite state automata has been defined. Future work will investigate in more detail the effect of the BPEL *link* language concept on the transformation process.

The transformation presented in this paper has been used for implementing the **IPSI Process Finder (IPSI-PF)** process matchmaking engine. Currently, this implementation is evaluated on a set of process definitions derived from the Internet Open Trading Protocol (IOTP). First results are convincing with regard to the precision, while the performance is an issue of ongoing work. In particular, the application of IPSI-PF to dynamic service discovery scenarios require the development of an index structure supporting the specific queries.

References

[1] M. Aiello, M. Papazoglou, J. Yang, M. Pistore, M. Carman, L. Serafini, and P. Traverso. A request language for web services based on planning and constraint satisfaction. In *Proc. of 3rd Int. Workshop, Technologies for E-Services (TES)*, LNCS 2444, pages 76–85. Springer, 2002.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business process execution language for web services, version 1.1, March 2003.

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business process execution language for web services. version 1.1. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbiz2k2/html/BPEL1-1.asp, May 2003.

[4] D.-S. C. A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. McDermott, D. Martin, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. Daml-s: Web service description for the semantic web. In *Proc. 1st Int'l Semantic Web Conf. (ISWC 02)*, volume 2342 of *LNCS*, pages 348–363. Springer, 2002. http://citeseer.nj.nec.com/ankolekar02damls.html.

[5] J. Chomicki and G. Saake, editors. *Logics for Database and Information Systems*. Kluwer, 1998.

[6] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[7] IBM, Microsoft, HP, Oracle, Intel, and SAP. Universal description, discovery and integration, July 2002. http://www.uddi.org/.

[8] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck. A service level agreement language for dynamic electronic services. *Electronic Commerce Research*, 3(1-2):43–59, 2003.

[9] O. Matz, A. Miller, A. Podtthoff, W. Thomas, and E. Valkema. Report on the program AMoRE. Technical Report 9507, Christian-Albrechts Universtaet, 1995.

[10] M. Mecella, B. Pernici, and P. Craca. Compatibility of e-services in a cooperative multi-platform environment. In F. Casati, D. Georgakopoulos, and M. Shan, editors, *TES 2001 LNCS 2193*, pages 44–57. Springer, 2001. Lecture notes in Computer Science 2193.

[11] C. Molina-Jimenez, S. Shrivastava, E. Solaiman, and J. Warne. Contract representation for run-time monitoring and enforcement. In *Proc. of Int. Conf. on Electronic Commerce (CEC)*, pages 103–110. IEEE, 2003.

[12] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, 1981.

[13] RosettaNet. Rosettanet homepage. http://www.rosettanet.org.

[14] A. ShaiklAli, O. F. Rana, R. Al-Ali, and D. W. Walker. Uddie: An extended registry for web services. In *Proceedings of the 2003 Symposium on Applications and the Internet Workshops (SAINT-w03)*. IEEE Computer Society, 2003.

[15] K. Sycara, J. Lu, M. Klusch, and S. Widoff. Matchmaking among heterogeneous agents on the internet. In *Proceedings AAAI Spring Symposium on Intelligent Agents in Cyberspace, Stanford, USA*, 1999.

[16] W. van der Aalst and K. van Hee. *Workflow Management - Models, Methods, and Systems*. MIT Press, 2002.

[17] W. van der Aalst and M. Weske. The P2P approach to interorganizational workflows. In *Proc. of 13. Int. Conf. on Advanced Information Systems Engeneering (CAISE'01)*, Interlaken, Switzerland, 2001.

[18] W. van der Aalst. Dont go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, pages 72–76, Jan/Feb 2003.

[19] A. Wombacher, P. Fankhauser, B. Mahleko, and E. Neuhold. IPSI-PF: A business process matchmaking engine. In *submitted to CEC 04*, 2004.

[20] A. Wombacher, P. Fankhauser, B. Mahleko, and E. Neuhold. Matchmaking for business processes based on choreographies. In *Proc. of International Conference on e-Technology, e-Commerce and e-Service (EEE-04)*. IEEE Computer Society Press, 2004. to appear.

[21] L. Zeng, B. Benatallah, P. Nguyen, and A. H. H. Ngu. Agflow: Agent-based cross-enterprise workflow management system. In P. M. G. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. T. Snodgrass, editors, *Proc. of 27 international conference. on VLDB*, pages 697–699, 2001.