Finding Trading Partners to Establish Ad-hoc Business Processes[*]

Andreas Wombacher and Bendick Mahleko

Fraunhofer Gesellschaft, Integrated Publication and Information Systems Institute,
64293 Darmstadt, Germany
{Andreas.Wombacher,Bendick.Mahleko}@ipsi.fhg.de
http://ipsi.fhg.de/oasys/

Abstract. Enabling technology for realizing ad-hoc business processes currently is becoming more and more popular, like for example web services. Ad-hoc business processes are semantically characterized by a description of the exchanged messages and by the potential message sequences. These semantic meta-data are required to enable efficient and precise searching and finding of potential trading partners. Based on the experience of current web technology, manual maintenance of meta-data is neglected by most editors, therefore the meta-data must be generated automatically. Within this paper, we propose a method to derive a specification of potential message sequences based on a private workflow model. Further, we describe an algorithm for matchmaking of message sequence specifications as part of a search engine for ad-hoc business processes.

1 Introduction

Electronic data interchange is nearly explicitly used in long lasting B-2-B relationships, because high investments are needed for setting up cross-organizational processes by using for example EDIFACT. More flexible business relationships and especially B-2-C process coupling is either done without EDI or proprietarily implemented. Due to the fact of increasing speed of innovation resulting in shortened product life cycles there is a need for optimization of business processes and cost reduction resulting in establishing EDI on a more flexible or ad-hoc basis.

Ad-hoc business processes provide the flexibility of selecting appropriate business partners and establishing cross-organizational business processes dynamically. Exemplary arising technologies are ebXML mainly applied in the domain of B-2-B relationships or Web services targeting all types of business relationships. One of the main problems of ad-hoc business processes is to find appropriate trading partners, which are using the same messages in a similar way; that is to be able to process the same potential message sequences.

[*] This work has been partially funded by the European Comission under contract IST-1999-11060 (eBroker) as part of the Information Technology Society program.

1.1 Motivation

Finding appropriate trading partners to establish an ad-hoc business process requires that both parties are using the same messages and support the same business scenarios instantiated in the same potential message sequences. In the following, we name the potential message sequences supported by a trading partner an *interaction pattern*.

Potential trading parties must support the *same messages*, if they want to interact. This requirement can easily be understood, because two persons not talking the same language will not understand each other and therefore can not interact.

Potential trading parties must support the *same interaction patterns*, if they want to set up an ad-hoc business process. We assume, the trading partners support the same messages (language). In addition, it is required that they have a common understanding how to interact. If, for example, two human beings want to do a phone call, both parties must be familiar on how to do it. In particular, the communication is initiated by the caller dialing the number, while the called person says hello, continued by the caller telling his name and the matter of his call. After this initialisation, they start a discussion and follow their specific common protocol how to finish the phone call. If one of the two persons is not familiar with the protocol or interaction pattern of a phone call, communication becomes difficult and will result in several misunderstandings. While human beings may be able to resolve the misunderstandings automized ad-hoc business processes typically will not. Therefore, trading partners in an ad-hoc business process must support similar interaction patterns.

Thus, to be able to find a potential trading partner, the matchmaking must take both dimensions into account.

– comparing the semantics of the messages used by the business processes, and
– comparing the process interaction patterns specified by the business processes, i.e. the containment of all potential message sequences required by the user and supported by the provider.

While the first type of comparison is based on flat key word comparison the second one is dealing with structured specifications, which is not supported by current search engines. Using both types of comparison within the matchmaking increases the quality of the searching result, that is precision and re-call improve.

The above mentioned type of matchmaking requires the availability of meta-data specifying the interaction patterns. Generation of meta-data is an additional time-consuming task, which has to be automized to increase usability and to avoid meta-data inconsistency.

In accordance with this approach we propose within this paper a concept for automatic generation of interaction patterns for bilateral ad-hoc business processes and their usage within the searching and matchmaking process.

1.2 Example

The example depicted in Fig. 1 describes two different business models provided by the vendor resulting in different interaction patterns. The first business model (depicted on the left hand side of the Figure) involves a customer A and the vendor himself. It specifies

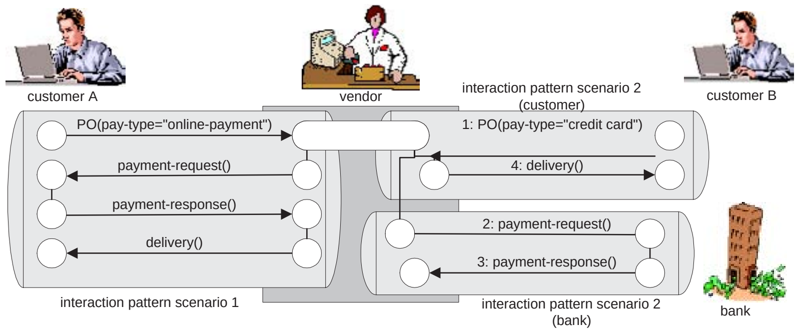

customer A
vendor
interaction pattern scenario 2 (customer)
customer B
PO(pay-type="online-payment")
1: PO(pay-type="credit card")
payment-request()
4: delivery()
payment-response()
2: payment-request()
delivery()
3: payment-response()
interaction pattern scenario 1
interaction pattern scenario 2 (bank)
bank

Fig. 1. Scenario

a purchase using an online-payment, where the customer sends a purchase order (PO) containing the payment type to the vendor, who requests and receives the payment from the customer using an online-payment and afterwards delivering the content. The second business model (depicted on the right hand side of the Figure) involves a customer B and a bank. It specifies a credit card purchase, where the customer sends a purchase order (PO) containing the payment type and the credit card details to the vendor, who requests and receives the payment from a bank and finally delivers the content to the customer.

In both cases, the vendor receives a PO first and continues afterwards with different sequences depending on a parameter represented in the PO. Observing the interaction of the vendor with the customer, it turns out that the required message types as well as the sequencing differs within the scenarios.

Within the first scenario the vendor communicates exclusively with customer A, so if a customer wants to purchase a good from the vendor using online payment, he must be able to handle the specified process, that is must support this interaction pattern. The second scenario contains a bank and a customer in addition to the vendor. The depicted communication results in two interaction patterns, because the bank is not involved in interactions between the vendor and the customer and vice versa.

We assume the vendor has specified the provided interaction patterns and made these accessible by a search engine. We now consider a customer willing to do a purchase. Because of the limitation of his internal workflow he is capable to perform credit card payment only. To specify the query for searching an appropriate vendor the customer must specify a description of supported messages and interaction patterns. Based on this description the search engine is able to determine a qualified list of vendors, which might be used to establish an ad-hoc business process.

1.3 Application Domain: Web Services

Currently, the application domain of web services is facing the problem described above. There are language proposals available to describe complex, hierarchical workflows (like Web Service Flow Language (WSFL) [1] or XLANG [2]) used to represent and enact internal processes of a single party as well as representing global views for interactions among parties. Further, the Web Service Definition Language (WSDL) [3] proposes a specification language of message types providing extension capabilities, like for example the specification of Quality of Service aspects using Web Service Endpoint Language (WSEL) [1]. While WSFL and XLANG specify the internal process of a party (interactions among parties as well), WSDL describes the messages and collections of messages exported, respectively imported, by an external service. The Web Service Communication Language (WSCL [4]) specifies interaction patterns, which might be added as an extension to the WSDL description of a service offering. The above described problems still remain:

- There are tools available for specifying an internal workflow, providing workflow and WSDL descriptions, but not supporting automated generation of interaction pattern descriptions.
- Within web service architecture there is no matchmaking of interaction patterns specified.

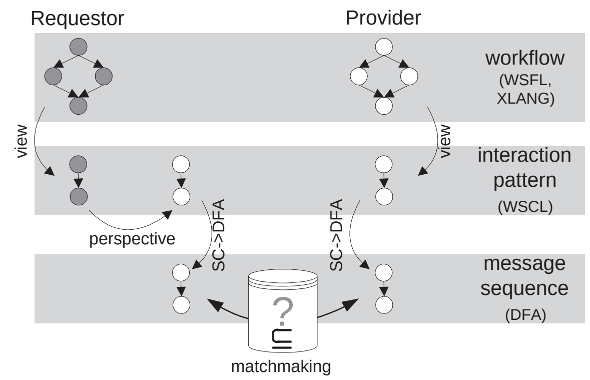
This paper aims at addressing the problems highlighted above. Section 2 presents two things, which are (i) a way how to generate views of a business process to allow only parts relevant to intended collaboration to be made public and (ii) a way how to find if a business process of a service requestor is compatible with the stored process of a service provider (i.e., business process matching). Section 3 is on related work and Section 4 is summary and future work.

2 Approach

In this paper we propose a novel approach for generating business process views and matching of business processes for bilateral collaborations. We will build on the example given in the above section, where two parties play complementary roles i.e., that of a service provider and a service requestor. We will assume in our model that parties involved in collaborations use standard vocabulary, e.g., EDI standard terms.

Fig. 2 shows a typical application scenario. We imagine a situation were a service provider (e.g. the seller of the example above) has a complex workflow depicting the various types of interactions that are needed for his business. Since business processes are regarded as assets by most organizations, the service provider will publish only those parts of his business process that are needed for interaction with outside organizations. This implies that the service provider must generate a view of his business process that is relevant to potential business partners. Multiple views of the same business process can be generated, depending on who the service provider intends to interact with. There might be a view for interacting with buyers, shipping companies, financial service providers etc. An automated way to perform such an operation is desirable, but unfortunately, traditional Web Service technologies do not offer it.

We imagine also that a service requestor (e.g. customer of the example above) wants to search for a service provider whose business process is compatible with his, i.e., a service requestor already has a business process, and he wants to find a provider with a matching business process, e.g., so that he does not have to alter any of his processes. The service requestor will generate a service provider's view of his business process to



Legend
DFA : Deterministic Finite Automata
SC : Statechart

Fig. 2. Overview

remain with only those parts of his business process that are relevant to the interaction with a service provider. So again, like the service provider, the service requestor will generate a provider's view of his business process.

To be able to check if the business process of the service requestor matches with that of the service provider stored in the repository, we compute the perspective of the provider on the requestors process. By computing the perspective, we mean we derive a flow model from an existing business process by observing the pattern of incoming and outgoing messages and the changes in state that take place. The derived model represents a business process that is compatible with the stored service provider process model. Not all potential partners will utilize all the interaction functionalities offered by the generated model. This is because the generated process model must capture a wide range of interaction possibilities, but for some potential partners, not all of them might be necessary. Therefore, it is enough if the generated process is a subset of the process of a service provider. So, to check if the process of a service requestor matches with that of a provider we have to prove that his process is a subset of the generated process. Fig. 2 gives a high-level view of the needed steps.

In this paper, statecharts are the used business modeling technique for a number of reasons [5]. We agree with the findings of Benyoucef and Keller that statecharts are adequate to model e-negotiations [5]. Statecharts are also used in SELF-SERV to model services that are to be composed in a peer-to-peer environment [6]. The simplicity of the formalism and the availability of tool-support makes them appealing for business process modeling [7]. The UML standard also supports statecharts, making them (statecharts) readily available in commercial tools that are based on UML. In addition, off-the-shelf simulation and analysis tools e.g., STATEMATE [8], are readily available which can be used to validate statechart models. We also believe that conversion to other commercial tools will not be a problem due to the wide popularity of statecharts (e.g., it is supported by UML). The next section gives a basic introduction to statecharts.

2.1 Statecharts

Statecharts can be regarded as an extension of finite state automata with constructs for hierarchy of states, broadcast communication and parallelism of execution [7,9]. We extend the definition of statecharts given in [10], as follows:

Definition 1 (Statechart) *A statechart is a structure consisting of* $\langle \Pi, S, T, r, V, F, R \rangle$*, where*

– Π *is a set of primitive events.*
– S *is a non-empty finite set of states.*
– $T \subseteq S \times \Pi \times \Pi \times S$*, is a non-empty finite set of transitions.*
– $r \in S$ *is the root state.*
– V *is a set of variables.*
– $F \subseteq S$ *is a finite set of terminal states.*
– R *is a finite set of roles.*

The extension of statecharts definition made in this paper makes extensive use of the concept of a *role*. A role refers to the *type* of a trading partner without referring to concrete instances. The set of roles might comprise *buyer* and *seller* roles which means the types of partners needed to fulfill a given business activity.

There are three functions which define structural relations between states:

– The function $children : S \longrightarrow 2^S$ defines the immediate substates subsumed by each state. A state s is called *basic* if $children(s) = \emptyset$; otherwise it is *composite*. If a state $s_2 \in chilren(s_1)$, s_1 is a *parent* of s_2, and s_2 is a *child* of s_1. There exists a unique state $r \in S$ which has no parent, i.e., $\forall s \in S, r \notin children(s)$. This state is called the *root* of the statechart.
– The function $type : S \longrightarrow \{and, or\}$ is a partial function that assigns to each *composite* state its type, and identifies it as either an *or-state* or an *and-state*.
– The function $default : S \longrightarrow S$ identifies for each *or-state* s, one of its children $default(s) \in children(s)$ as its initial state. This has the same meaning as the *start state* in finite state automata.

Several functions that identify parameters of transitions have been defined in [10]. We briefly introduce them below:

– $source(t), t \in T$: identifies a non-empty set of source states from which a transition departs.
– $target(t), t \in T$: identifies the set of target states.
– $arena(t), t \in T$: identifies an *or-state* which contains both $source(t)$ and $target(t)$.
– $trigger(t), t \in T$: consists of a set of literals $\ell_1 \dots \ell_k, k \geq 0$, each of which is a primitive event $e \in \Pi$ that trigger a transition t. In this paper, we will deviate from the semantics of [10] by disallowing negation of events.

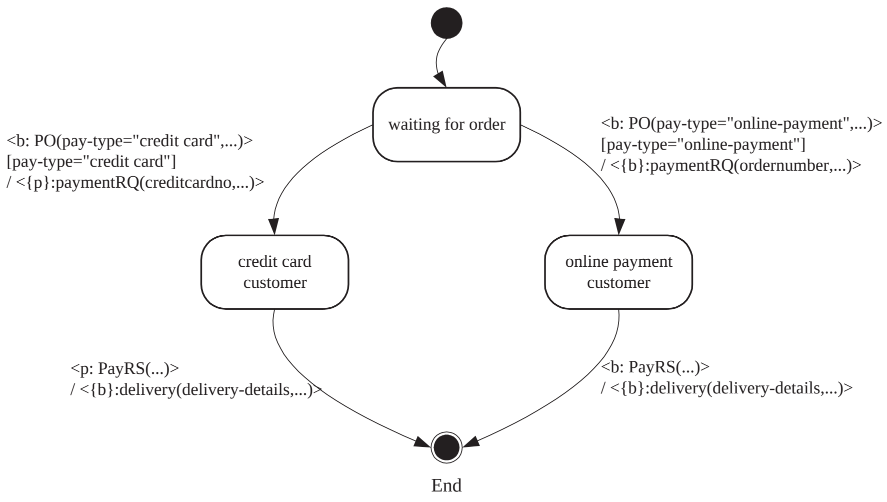

waiting for order
<b: PO(pay-type="credit card",...)>
[pay-type="credit card"]
/ <{p}:paymentRQ(creditcardno,...)>
<b: PO(pay-type="online-payment",...)>
[pay-type="online-payment"]
/ <{b}:paymentRQ(ordernumber,...)>
credit card customer
online payment customer
<p: PayRS(...)>
/ <{b}:delivery(delivery-details,...)>
<b: PayRS(...)>
/ <{b}:delivery(delivery-details,...)>
End

Fig. 3. Vendor Business Process as a Statechart, b is a customer, p is a payment processor

- $triggered(E), E \subseteq \Pi$: a transition t is triggered by E if $\forall e, e \in E, e \in trigger(t)$.
- $actions(t), t \in T$: is a set of events $g_1, \ldots, g_m$, where $g_i \in \Pi$, for $i = 1, \ldots, m$ that is generated when transition t is taken.
- $generated(T) = \cup_{t \in T} actions(t)$, the set of events generated when the set of events, T was taken.

Statecharts are a powerful modeling tool for concurrently communicating processes [7]. In this paper, we restrict the communication notion of statecharts to directed channel communication, where an *event* is prefixed with the sender identity and the *actions* are prefixed with the recipient identity rather than broadcast to everyone who is listening, i.e., every event e and action a takes the form $e \in R \times \Pi$ and $a \in 2^R \times \Pi$ respectively. This restricts communication to known entities, allowing us to study the behavior of such entities by examining the messages they receive (events) and send outside (actions). We will use the following notation:

Definition 2 (Event) *An event $e \in R \times \Pi$ is of the form $\langle p : \ell(x_1, \ldots, x_n)\rangle$, where $p \in R$, the sender of the event and $\ell(x_1, \ldots, x_n) \in \Pi$, the primitive event, and $x_1, \ldots, x_n$ are parameters.*

Definition 3 (Action) *An action $a \in 2^R \times \Pi$ is of the form $\langle P : g(x_1, \ldots, x_n)\rangle$ where $P \subseteq R$, the set of recipients of the action and $g(x_1, \ldots, x_n) \in \Pi$, the primitive event, and $x_1, \ldots, x_n$ are parameters.*

The business process for the vendor as depicted in Fig. 1 can be represented by a statechart in Fig. 3. The figure shows that the vendor business process stays in a *default* state, the *wait for order state* until he receives a ⟨*b: PO(pay-type="credit card",...)*⟩ event or ⟨*b:*

PO(pay-type="online-payment",...)$\rangle$ event. On receiving one of these events, the vendor uses his internal business rules to determine how to process it. If the requestor wants to pay by credit card, the internal logic of the vendor will take him to the *credit card customer state*; if the requestor wants to pay via online-payment, the internal logic of the vendor will take him to the *online payment customer state*. If the requestor is paying by credit card, the vendor system will automatically generate an action event that will be sent to a payment processor to check the credit card credentials of the requestor. If the credit card details are verified to be okay, the payment processor sends a message to okay the transaction; on receiving the okay event from the payment processor, the vendor sends a message to the requestor informing him that his order has been processed along with delivery information. Different processing steps are taken if the requestor was paying by online-payment as shown in Fig. 3. The next section discusses view generation of a statechart to derive interaction patterns.

2.2 Generating Views

In this section we give a step by step description explaining how views are generated from a statechart that represents a business process. A view of a statechart is a subset of a more general statechart that pertains to a specific role, e.g., we can generate the view of a service requestor on a service provider statechart. As explained in previous sections, views are generated by observing the behavior of a business model which is described in terms of *received events* and *sent actions* of the relevant statechart. We begin by giving some definitions.

Definition 4 (Participant's View on Events) *From the statechart of a role p, if events $e \in \Pi$, with form $\langle p_1 : \ell_1(x_{1,1}, \cdots, x_{1,m})\rangle \wedge \ldots \wedge \langle p_n : \ell_n(x_{n,1}, \cdots, x_{n,m})\rangle$, are given, the view of a complementary role p_c on the events is,*

$$view(p_c, e) := view(p_c, \langle p_1 : \ell_1(x_{1,1}, \cdots, x_{1,m})\rangle) \wedge \ldots \wedge view(p_c, \langle p_n : \ell_n(x_{n,1}, \cdots, x_{n,m})\rangle)$$

with $view(p_c, \langle p_i : \ell_i(x_{i,1}, \cdots, x_{i,m})\rangle) := \begin{cases} \langle p_c : \ell_i(x_{i,1}, \cdots, x_{i,m})\rangle, \ if \ p_i == p_c, \\ \epsilon, \ else. \end{cases}$

The definition implies that a role's p_c view on events is obtained by retaining those events that were sent by him, and discarding those that were not sent by him.

Definition 5 (Participant's View on Actions) *From the statechart of a role p, if actions $a \in \Pi$, with form $\langle P_1 : g_1(x_{1,1}, \cdots, x_{1,m})), \ldots, (P_n : g_n(x_{n,1}, \cdots, x_{n,m})\rangle$, are given, the view of a complementary role p_c on the actions is,*

$$view(p_c, a) := view(p_c, \langle P_1 : g_1(x_{1,1}, \cdots, x_{1,m})\rangle), \ldots, view(p_c, \langle P_n : g_n(x_{n,1}, \cdots, x_{n,m})\rangle)$$

with $view(p_c, \langle P_i : g_i(x_{i,1}, \cdots, x_{i,m})\rangle) := \begin{cases} \langle \{p_c\} : g_i(x_{i,1}, \cdots, x_{i,m})\rangle, \ if \ p_c \in P_i, \\ \epsilon, \ else. \end{cases}$

This definition implies that the view of a role p on a set of actions is obtained by retaining all actions of which the recipient p_c is among the recipients, and discarding those of which he is not among the recipients.

Definition 6 (Participant's View on Transitions) *From the statechart of a role p, if a transition $t : (source(t), trigger(t), actions(t), target(t))$ is given, the view of a complementary role p_c on t is,*
$view(p_c, t) := (source(t), view(p_c, trigger(t)), view(p_c, actions(t)), target(t))$.

The definition uses earlier definitions for views on events and actions to derive transitions, where a transition's label is changed such that it comprises only events and actions whose views are of this role.

Definition 7 (Participant's View on a Statechart) *From the statechart $S = \langle \mathit{\Pi}, S, T, r, V, F, R \rangle$ of a role p, a complementary role p_c's view on S is a statechart $S_c := \langle \mathit{\Pi}, S, T_c, r, V, F, R \rangle$, where*
$T_c := \{view(p_c, t) | t \in T\}$.

This definition says the view of a role p_c on a statechart is obtained by computing the views of the role p_c in question on each transition of the statechart. This makes use of the view on transitions definition.

To make things more clearer, we will use the statechart in Fig. 3 to compute the view of a requestor who is a customer in our example. *Definition 7* implies that to get the view of the given statechart, we must compute the participant's view on each transition of the given statechart. There are four transitions in the original statechart, and for convenience, we label them as shown below:

- t_1: $\big($waiting for order, $\langle b$: PO(pay-type="online-payment",...)$\rangle$,
  $\langle\{b\}$:paymentRQ(ordernumber,...)$\rangle$, online payment customer$\big)$,
- t_2: $\big($online payment customer, $\langle b$: PayRS(...)$\rangle$,
  $\langle\{b\}$:delivery(delivery-details,...)$\rangle$, End$\big)$,
- t_3: $\big($waiting for order, $\langle b$: PO(pay-type="credit card",...)$\rangle$,
  $\langle\{p\}$:paymentRQ(creditcardno,...)$\rangle$, credit card customer$\big)$,
- t_4: $\big($credit card customer, $\langle p$: PayRS(...)$\rangle$, $\langle\{b\}$:delivery(delivery-details,...)$\rangle$, End$\big)$.

By applying the view definition (which also relies on action and event definitions) on transitions, t_1, t_2, t_3, t_4 with $t'_i := view(b, t_i)$ calculates to

- transition t_1 and t_2 remain unchanged, thus $t'_1 = t_1$ and $t'_2 = t_2$
- t'_3: $\big($waiting for order, $\langle b$: PO(pay-type="credit card",...)$\rangle$,ϵ, credit card customer$\big)$,
- t'_4: $\big($credit card customer, ϵ, $\langle\{b\}$:delivery(delivery-details,...)$\rangle$, End$\big)$.

According to the definition we gave, the *default state* remains the same. Graphically, the requestor's view on the statechart of Fig. 3 is as depicted in Fig. 4).

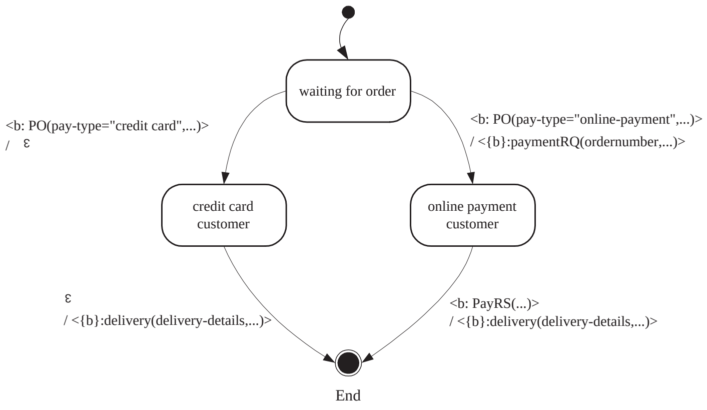

waiting for order
<b: PO(pay-type="credit card",...)>
/ ε
<b: PO(pay-type="online-payment",...)>
/ <{b}:paymentRQ(ordernumber,...)>
credit card customer
online payment customer
ε
/ <{b}:delivery(delivery-details,...)>
<b: PayRS(...)>
/ <{b}:delivery(delivery-details,...)>
End

Fig. 4. Requestor's View on the Provider's Statechart

2.3 Generating Perspectives

In this section we give descriptions how to generate a participant's perspective from the statechart of another participant. The approach is to transform all events that are in the view of the involved participant and convert them to actions, with the role changing to the one that receives the action. Actions are treated in a similar way, i.e., all actions that are in the view of the participant are converted to events, with the role part changing to the one that sends the event. We give the used definitions below, and illustrate this concept with an example.

Definition 8 (Participant's Perspective on Events) *Given a statechart $S_c = \left\langle \Pi, S, T_c, r, V, F, R \right\rangle$ of a role p representing the view of role p_c; p's perspective on events e of form $\langle p_1 : \ell_1(x_{1,1}, \ldots, x_{1,m})\rangle \wedge \ldots \wedge \langle p_n : \ell_n(x_{n,1}, \ldots, x_{n,m})\rangle$ is,*

$$\begin{aligned} perspective(p_c, e) := \; & perspective(p_c, \langle p_1 : \ell_1(x_{1,1}, \ldots, x_{1,n})\rangle), \ldots, \\ & perspective(p_c, \langle p_n : \ell_n(x_{n,1}, \ldots, x_{n,m})\rangle) \end{aligned}$$

with
$perspective(p_c, \langle p_i : \ell_i(x_{i,1}, \ldots, x_{i,m})\rangle) := \langle \{p\} : \ell_i(x_{i,1}, \ldots, x_{i,m})\rangle$ *and*
$perspective(p_c, \epsilon) := \epsilon$

What this definition says is that an event is in the perspective of a role p_c if that role is a sender of that event, hence the role part changes to that of the source statechart. The rational for the change in role part is that this event is perceived as an action by the role whose perspective we are generating. In the definition, a set with a single element is returned because an event is changed to an action and actions are defined based on the powerset of roles (see definition of statechart).

Definition 9 (Participant's Perspective on Actions) *Given a statechart $S_c = \left\langle \Pi, S, T_c, r, V, F, R \right\rangle$ of a role p representing the view of role p_c; p's perspective on actions a of form $\langle \{p_c\} : g_1(x_{1,1}, \ldots, x_{1,m}), \ldots, \langle \{p_c\} : g_n(x_{n,1}, \ldots, x_{n,m})\rangle$ is,*

$$perspective(p_c, a) := perspective(p_c, \langle \{p_c\} : g_1(x_{1,1}, \ldots, x_{1,n})\rangle) \wedge \ldots \wedge$$
$$perspective(p_c, \langle \{p_c\} : g_n(x_{n,1}, \ldots, x_{n,m})\rangle)$$

with
$perspective(p_c, \langle \{p_c\} : g_i(x_{i,1}, \ldots, x_{i,m})\rangle) := \langle p : g_i(x_{i,1}, \ldots, x_{i,m})\rangle$ *and*
$perspective(p_c, \epsilon) := \epsilon$

The meaning of this definition is that an action is in the perspective of a role if this role is among the recipients of the action, hence the role part of the action changes to that of the source statechart. The change of role is due to the fact that the action will be perceived as an event by the role whose perspective we are generating.

Definition 10 (Participant's Perspective on Transitions) *From the statechart* $S_c = \langle \Pi, S, T_c, r, V, F, R\rangle$ *of a role* p *representing the view of role* p_c*, if a transition* $t : (source(t), trigger(t), actions(t), target(t))$ *is given, we want to compute the perspective of a complementary role* p *on* t*. This is computed as follows,*
$$perspective(p_c, t) := (source(t), perspective(p_c, actions(t)), perspective(p_c, trigger(t)), target(t)).$$

The definition means that the perspective of a role on a transition is obtained by first computing perspectives on events and actions for the role (as described in *definition 8 and 9* respectively), and swapping them, i.e., events that are in the perspective of the role become actions, and actions that were in the perspective of the role become events. The rational is that if in a source statechart, an event was sent by a given role, then in his statechart, this event must have been an action directed at the statechart we are using as source. The same explanation goes for actions, i.e., if an action is send to a role, this action will be perceived as an event coming from the statechart which is acting as the source.

Definition 11 (Participant's Perspective on a Statechart) *From the statechart* $S = \langle \Pi, S, T, r, V, F, R\rangle$ *of a role* p *representing the view of role* p_c*, a complementary role* p*'s perspective on* S *is a statechart*
$S_{c'} := \langle \Pi, S, T_{c'}, r, V, F, R\rangle$*, where*
$T_{c'} := \{perspective(p_c, t) | t \in T_c\}$.

This definition allows us to transform all transitions in a source statechart by recursively calling the perspective function on each transition of the source statechart.

Before we give an example to illustrate the definitions given above, we will describe the pre-processing needed on the source statechart before perspective rules can be applied. This is the separation of events from actions.

Definition 12 (How to Separate Events from Actions) *Given a statechart* $S = \langle \Pi, S, T, r, V, F, R\rangle$*, separate events and actions that lie on the same transitions* T *as follows:*
For each transition $t \in T$*, create an intermediary state* s_t *such that the new set of states* S' *is* $S' = S \cup \{s_t \mid t \in T\}$*), and the new transition set* T' *is*
$T' = \{(source(t), trigger(t), \epsilon, s_t), (s_t, \epsilon, actions(t), target(t)) \mid t \in T\}$.

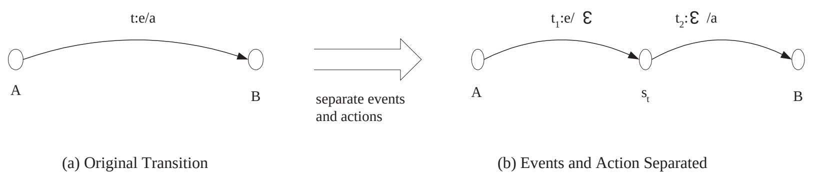


Fig. 5. Separating Events and Actions

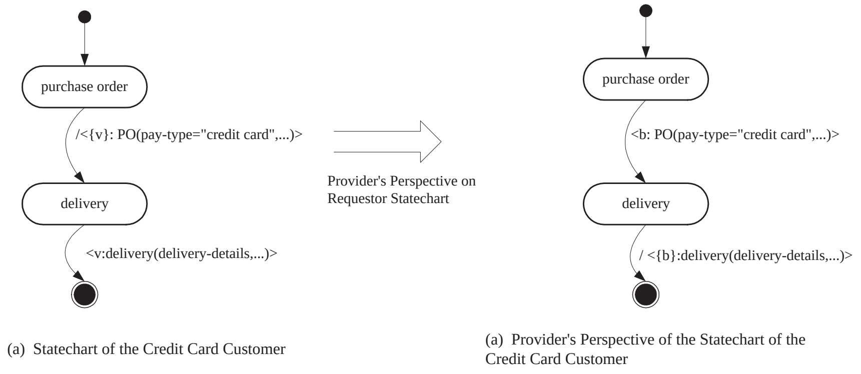

purchase order
/<{v}: PO(pay-type="credit card",...)>
delivery
<v:delivery(delivery-details,...)>
Provider's Perspective on Requestor Statechart
purchase order
<b: PO(pay-type="credit card",...)>
delivery
/ <{b}:delivery(delivery-details,...)>
(a) Statechart of the Credit Card Customer
(a) Provider's Perspective of the Statechart of the Credit Card Customer

Fig. 6. Provider Perspective on Credit Card Customer's Statechart, v represents the vendor, b the customer

Fig. 5 illustrates how to separate events and actions. This step is needed so that when we do the transformation, we preserve the causative relationship between actions and events that lie on the same transition, i.e., events trigger actions during a transition. The statechart of the requestor (see Fig. 6(a)), however does not comprise actions and events on the transitions, so trying separate actions and events will yield the same statechart. In more complicated statecharts however, we need to.

We want to generate the perspective of the provider on the statechart of the requestor. *Definition 11* for perspective generation implies that we must compute the perspective of a given role on each transition of the source statechart. By taking each transition of Fig. 6(a), and using *definition 10*, we get Fig. 6(b). which represents the perspective of the provider on the statechart of the requestor (perspective of vendor on the statechart of the credit card customer).

2.4 Process Matching

Earlier on in this section we described the steps needed to match business processes (see Fig. 2). Here we show how each step is realized.

The first step is to generate views for both service provider and requestor. The view generated from the service provider's statechart is stored in a repository. In Fig. 4 we

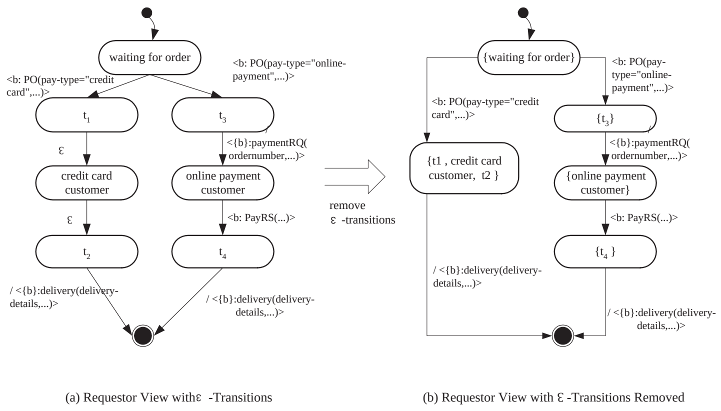

waiting for order
<b: PO(pay-type="credit card",...)>
<b: PO(pay-type="online-payment",...)>
t_1
t_3
ε
<{b}:paymentRQ(ordernumber,...)>
credit card customer
online payment customer
ε
<b: PayRS(...)>
t_2
t_4
/ <{b}:delivery(delivery-details,...)>
/ <{b}:delivery(delivery-details,...)>
remove ε -transitions
{waiting for order}
<b: PO(pay-type="online-payment",...)>
<b: PO(pay-type="credit card",...)>
{t_3}
<{b}:paymentRQ(ordernumber,...)>
{t1 , credit card customer, t2 }
{online payment customer}
<b: PayRS(...)>
{t_4 }
/ <{b}:delivery(delivery-details,...)>
/ <{b}:delivery(delivery-details,...)>
(a) Requestor View with ε -Transitions
(b) Requestor View with ε -Transitions Removed

Fig. 7. Requestor's View on the Provider's Statechart with Actions and Events Separated

shown how to generate the view of a requestor on the statechart of a provider. This view represents a subset of the service provider's business process that he wants to make public. Before storing this view on a repository, we must separate events and actions using *Algorithm 12*, and then remove all ϵ-transitions (see [11] for algorithm to remove ϵ-transitions) to make the statechart simple and ease to understand. Fig. 7 shows a requestor's view on the provider's statechart with events and actions removed (Fig. 7(a)), and ϵ-transitions removed (Fig. 7(b)). The statechart depicted in Fig. 7(b) is stored on the repository and will be used for matching purposes.

In the next step, we compute the provider perspective on the requestor. We have already shown how to compute the perspective of a participant in the previous section. Fig. 6 shows a statechart representing a requestor that uses a credit card for buying, and the provider's perspective of it. We now give detailed descriptions of the steps to be taken.

1. Convert the provider's statechart stored in the repository to a deterministic finite state automata (DFA) S_p (see algorithm 1).
2. Convert the requestor's statechart (in Fig. 6(b)) to a deterministic finite state automata (DFA) S_r (see algorithm 1).
3. Check if $S_r \subseteq S_p$ as follows,
   if ($S_r \cap \overline{S_p} = \emptyset$)
      then the requestor process is compatible with the provider process,
   else
      the requestor process is not compatible with the provider process. [12]

The above steps need some explaining. Before applying the above steps, we eliminated those parts of the process models that are not relevant to the collaboration between the requestor and provider. This has been achieved by generating mutual views based on the respective statecharts of requestor and provider. Next, we computed the provider's perspective on the requestor's statechart to get Fig. 6. Like we said before, Fig. 6 must be a subset of the stored provider process if at all the requestor's process can match with the provider process. To do that we check if the generated statechart is a subset of the stored provider statechart.

Statecharts are not deterministic finite state automata (DFA), but rather an extension of them [9] so we cannot directly use a DFA subsumption algorithm without first converting the statecharts to DFA. The algorithm for converting a statechart to its equivalent DFA can be described as follows:

Given a statechart $S = \langle \Pi, S, T, r, V, F, R \rangle$, we can change S to an equivalent statechart S_a without hierarchical states as follows,

Algorithm 1 (Changing Statechart to Equivalent DFA) *The algorithm is,*

```
1. For each composite state $s \in S$,
       For each transition $t \in T$,
           if (target(t) == {s})
               then target(t) = {default(s)}.
           if (source(t) == {s})
               then source(t) = children(s).
2. remove all superstates.
3. make the resulting statechart deterministic using standard finite state automata algorithms, e.g., [11].
```

The final step is checking for subsumption. To check if the process of the requestor is compatible with that of the provider, we must prove that the DFA of the generated process (which is compatible with the process of the provider) is a subset of the DFA of the stored provider process. Below we give references how the subsumption problem is solved.

Definition 13 (DFA Subsumption) *Given two DFA, D_1 and D_2, to check if D_2 is subsumed by D_1, i.e., $D_2 \subseteq D_1$ we must show that $D_2 \cap \overline{D_1} = \emptyset$ [12].*

Computing the complement of a DFA can be done using standard algorithms [12] and takes linear time. A liner time algorithm to test for emptiness of intersection of two DFA is also available in [12].

3 Related Work

A lot of research has been carried out to allow organizations to establish ad hoc business relationships. In the Web Service domain, a number of standards have been developed with this goal in mind. WSDL [3] which is now a W3C Note allows organizations to publish descriptions of their applications in such a way that they can be invoked over the Internet. The WSDL standard however does not give a way to describe business processes, i.e., specify the choreograph in which processes are called. Standards like WSFL [1], XLANG [2] and WSCL [4] address this problem by describing ways to model business process interactions. UDDI [13] describes an XML Schema to publish and discover services. The UDDI repository stores information that is necessary to find a service and a way how to dynamically invoke it. In a way, these Web Service technologies play complementary roles with the goal of allowing business organizations to seamlessly interoperate on a global scale irrespective of the platforms they use. What seems to be missing from this picture is how to match business processes dynamically. It is possible for example to search for a business partner in a certain category based on meta-data, but so far there is no way to search for a partner with a compatible business process. Our work is a contribution in this regard.

ebXML is an initiative by OASIS and UN/CEFACT to develop standards that allow organizations to make their business processes interoperable to facilitate trade on a global scale [14]. A number of standards have been developed, among them the *Business Process Specification Schema (BPSC)* [15] and *Collaboration-Protocol Profile and Agreement Specification* [16]. [15] provides a standard schema which organizations should use in order to facilitate interoperability among potential business partner processes. [16] on the other hand uses the schema described by [15] to create business contracts in terms of exchanged messages. The standards however do not address the problem highlighted in this paper, i.e., how to automatically find matching business processes.

Additional approaches dealing with view generation and matchmaking of services are supported by the workflow community. In [17] a method for generating local workflows based on a global one is described. The approach is based on petri-nets and addresses the problem of setting up inter-organisational workflows based on a global workflow, while our approach is based on local workflows.
Another approach is described in [18] where views are defined on behalf of a local workflow. The views are then compiled to contracts by using additional knowledge of a global workflow. It seems that the major advantage of this view is hiding mission critical information from the outside world and transformation of the interface provided by a local workflow to fit the requirements of the global one.
Within [19] the expressivnes of interaction patterns with regard to establish inter-organizational workflows is compared to global workflow defintions and the differences are stated.

The issue of matchmaking is addressed in [20], where M. Mecella, B. Pernici and P. Craca describe an algorithm for checking if two e-Services are compatible. They do this by checking if every possible trace in one flow has got a compatible one in another without describing how they perform this checking in case the traces are compatible.

In addition to the workflow community also the semantic web community [21,22] is dealing with matchmaking of services. In [23] a language is introduced describing the functional aspects as well as the messages and their parameters based on a domain specific ontology. DAML-S described in [24] uses workflow aspects as well as the functional semantic description of the service within the matchmaking. The generation of these descriptions is not elaborated in the publications and seems to be done manually.

4 Summary and Future Work

This paper has presented two things which are (i) a way how to generate views of a business process to allow only parts relevant to intended collaboration to be made public and (ii) a way how to find if a business process of a service requestor is compatible with the stored process of a service provider (i.e., business process matching). We illustrated with an example how business process view generation and matching concepts can be incorporated into the Web Service infrastructure (see Figure 2). This will extend meta-data search with search based on a compatible business processes thereby making ad-hoc business relationships to be more easily established. These ideas are novel and can greatly facilitate inter-organizational business collaboration on an ad-hoc basis. Rather than companies having to rely on long-term established contracts to trade, more flexible business relationships can be established on an as needed basis, i.e., companies can quickly find partners with matching business processes and also break away from such relationships more easily.

In this paper we dealt only with business process view generation and matching for bilateral collaborations. In future work we will extend these concepts for multi-party collaborations. This is an important extension in that most of today's trading relationships are based on multi-party interactions, e.g., a long-running transaction might comprise interaction with a buyer, a supplier, a shipping company, an insurance company and a bank. Automatic checking of business process matching with all concerned parties is a non-trivial endeavor. We are also interested in investigating how to automatically enforce contracts in multi-party business process collaborations that have been deemed compatible.

References

1. F. Leymann. Web services flow language (WSFL 1.0), May 2001. http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
2. S. Thatte. XLANG web services for business process design.
3. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) 1.1 W3C note, March 2001. http://www.w3.org/TR/wsdl.
4. A. Banerji, C. Bartolini, D. Beringer, V. Chopella, K. Govindarajan, A. Karp, H. Kuno, M. Lemon, G. Pogossiants, S. Sharma, and S. Williams. Web services conversation language (WSCL) 1.0 W3C note, March 2002. http://www.w3.org/TR/wscl10/.
5. M. Benyoucef and R. K. Keller. An evaluation of formalisms for negotiations in e-commerce. *Proceedings of the Workshop on Distributed Communities on the Web*, June 2000.
6. Q.Z. Sheng, B. Benatallah, M. Dumas, and E.O.-Y. Mak. SELF-SERV: A platform for rapid composition of web services in a peer-to-peer environment. In *Proc. of 28th VLDB Conference*, Hong Kong, China, 2002.
7. D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
8. D. Harel and A. Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.
9. V. Hilaire, A. Koukam, P. Gruer, and J. Muller. Formal specification and prototyping of multi-agent systems. pages 114–127, 2000. http://citeseer.nj.nec.com/430779.html.
10. A. Pnueli and M. Shalev. What is in a step: On the semantics of statecharts. In T. Ito and A.R. Meyer, editors, *Theoretical Aspects of Computer Software*, pages 244–264. Springer, 1991. Lecture notes in Computer Science 526.
11. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 2001.
12. I. Wegener. *Theoretische Informatik*. B. G. Teubner Stuttgart, Germany, 1993.

13. Inc. Ariba, IBM Corporation, and Microsoft Corporation. Universal description, discovery and integration, September 2000. http://www.uddi.org/.
14. ebXML. ebXML standardization. http://www.ebxml.org/.
15. Business Process Team. Business process specification schema v1.01, May 2001.
16. Trading Partners Team. Collaboration-protocol profile and agreement specification v1.0, May 2001.
17. W.M.P. 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. E. Kafeza, D. K. W. Chiu, and I. Kafeza. View-based contracts in an e-service cross-organizational workflow environment. In F. Casati, D. Georgakopoulos, and M. Shan, editors, *TES 2001 LNCS 2193,* pages 74–88. Springer, 2001.
19. W. v.d. Aalst. Interorganizational workflows: An approach based on message sequence charts and petri nets. *Systems Analysis - Modelling - Simulation*, 34(3):335–367, 1999.
20. 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.
21. T. Berbers-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific America*, 284(5):34–43, 2001.
22. S. McIlraith, T. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems (Special Issue on the Semantic Web),* April 2001.
23. K. P. Sycara, M. Klusch, S. Widoff, and J. Lu. Dynamic service matchmaking among agents in open information environments. *SIGMOD Record*, 28(1):47–53, 1999.
24. Daml-S Coalition Anupriya. DAML-S: Web serice description for the semantic web, 2002.