Privacy Preserving Business Process Matching

Dilian Gurov
KTH Royal Institute of Technology
E-mail: dilian@kth.se

Peeter Laud
Cybernetica
E-mail: peeter.laud@cyber.ee

Roberto Guanciale
KTH Royal Institute of Technology
E-mail: robertog@kth.se

***Abstract*—Business process matching is the activity of checking whether a given business process can interoperate with another one in a correct manner. In case the check fails, it is desirable to obtain information about how the first process can be corrected with as few modifications as possible to achieve interoperability. In case the two business processes belong to two separate enterprises that want to build a virtual enterprise, business process matching based on revealing the business processes poses a clear threat to privacy, as it may expose sensitive information about the inner operation of the enterprises. In this paper we propose a solution to this problem for business processes described by means of service automata. We propose a measure for similarity between service automata and use this measure to devise an algorithm that constructs the most similar automaton to the first one that can interoperate with the second one. To achieve privacy, we implement this algorithm in the programming language SECREC, executing on the SHAREMIND platform for secure multiparty computation. As a result, only the correction information is leaked to the first enterprise and no more.**

I. INTRODUCTION

Enterprises engage increasingly in distributed and loosely coupled collaborations. For instance, possibly competitive businesses can share their skills to form temporary alliances, usually called *virtual enterprises*, in order to catch new business opportunities. An effective management of such collaborations requires techniques to engineer the inter-organizational business processes.

In this context, a non-trivial process engineering task is to ensure *soundness* (i.e., interoperability) of these collaborations: can the interactions between the partners lead to a deadlock; are the involved parties guaranteed to terminate properly; can the collaboration lead to documents that are never collected by the recipient? Several formal modeling languages such as Petri Nets [8], [26], Service Automata [15] and process algebra [9] have been proposed, allowing to formally state soundness properties such as *weak termination* [16] and *deadlock freedom* [15], and to formulate algorithms to check these properties.

The value of business processes is steadily increasing, and enterprises may even use the patent mechanism to protect the investment required to optimize their work flows and to obtain the needed domain-specific knowledge. Thus, forming virtual enterprises typically raises *privacy* issues. For instance, the structure of a process and the dependencies between its activities can reflect the internal structure of the enterprise and can reveal certain weaknesses in handling business opportunities. However, the need for ensuring soundness of interorganizational processes goes across the increasing concerns about privacy.

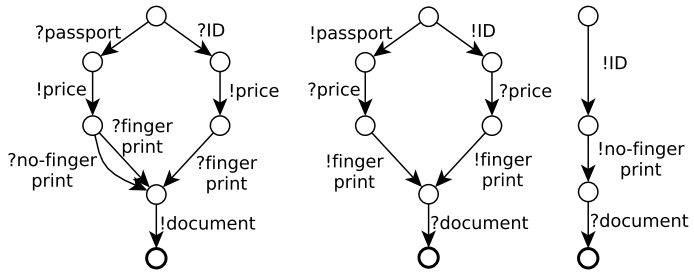


Fig. 1. A registration office and two customer processes

Whenever applicable, top-down development approaches such as [24], [9] can be used to ensure soundness of the interorganizational processes, while meeting the privacy requirements. Such approaches typically consist of three phases: (i) the participants agree upon a global (and publicly known) process that is subject to global soundness analyses, (ii) from the global process, local interfaces (one for each participant) are automatically synthesized, and (iii) each partner locally implements a private process and checks that it conforms to its public interface. This scheme allows to transfer the soundness properties of the global process to the composition of the private processes *without* publishing the latter.

If the agreement phase is not desirable, and the corresponding definition of a shared business process is not possible, a bottom-up approach must be taken instead. Each participant owns then an already existing private process, and the process engineering task becomes the one of guaranteeing that their composition is sound. Such a bottom-up approach must face two problems: (i) the global process and the public interfaces are not available, thus the soundness of the composition of "secret" processes must be checked without revealing information about the constituents, and (ii) if the soundness check fails, a Boolean result is of little use, since the participants have no information about the global process and thus can not investigate what is wrong. This prevents the participants from adapting their local procedures to form a sound collaboration.

In this paper we present a bottom-up approach to checking soundness of interorganizational business processes that addresses both these problems. We demonstrate a scenario based on an example from [24], assuming two parties p_A and p_B that own private business processes represented as service automata A and B. Figure 1a depicts the service automaton of a registration office. According with a customer request, this office can prepare passports and ID cards. In both cases, the office informs the customer about the cost and, if needed, stores the customer's fingerprints. Finally, it delivers the requested

document to the customer. Two customers (Figures 1b and 1c) check whether they can collaborate with this office. Assuming soundness means deadlock freedom, the soundness check will succeed for customer A_1 and fail for customer A_2. In fact, the second customer requests an ID card without providing the necessary fingerprints, thus leading the collaboration to a deadlock, and it never collects the delivered invoice.

To address problem (ii) above, we introduce a measure for *behavioral similarity* between two automata. We use this measure to check whether the composition of A and B is sound, and in case it is not, to allow p_A to discover, among all automata that can be soundly composed with B, the one that is most similar to A. This enables a correction loop, where one of the two participants can refine its own process (possibly accepting the suggested correction) and repeat the soundness check. This goal is achieved under strong privacy constraints: nothing more is leaked to the participants or any third party than what can be deduced from the result.

Our approach is based on the combination of three techniques: (i) we lift the check of the soundness of the automata composition to matching one of the automata against the *operating guideline* of the other (i.e., a representation of all service automata that can soundly collaborate with it, see [15]), (ii) we introduce the notion of *weighted matching* as a measure for matching degree, and show how to compute this measure and in the same time extract the behaviourally most similar service that can soundly collaborate with the other party, and (iii) we implement an algorithm to compute weighted matching while preserving privacy by means of Secure Multiparty Computation (SMC) techniques.

Our notion of weighed matching is inspired by the work of Lohmann [14] on service correction. That work uses similarity measures inspired by *edit distances*, with the aim of finding the minimal number of structural changes needed to obtain isomorphism between the input automaton and an automaton that matches the input operating guideline. We argue that, since service automata model process behavior (rather than structure), the similarity measure should focus on similar behavior rather that on similar structure. For this reason, our notion of weighted matching extracts from an operating guideline the strategy that is most similar to the input automaton in accordance with an established behavioral similarity measure [22].

The paper is organized as follows. In Section II we recall standard definitions and results on service automata, operating guidelines and behavioural similarity. In Section III we formalize "service matching with correction" and give an overview of our approach. In Section IV we introduce the notion of weighted matching and an algorithm to extract from an operating guideline the strategy most similar to an input automaton. In Section V we describe an implementation of service matching that uses Secure Multiparty Computation to satisfy the confidentiality requirements. Finally, in Sections VI and VII we present related work and concluding remarks.

II. Background

Business processes are usually expressed in high-level modeling languages such as BPMN [25] and EPC [21], to describe the inter-dependencies among activities and the events inside an enterprise. A number of formal languages have been proposed to provide unambiguous formalization of processes, the majority of which are inspired by Petri Nets [8] and Process Algebra [18]. There is also extensive literature (e.g. Global Calculus [9] and Open Nets [26], [24]) on top-down approaches to manage interorganizational processes. Here, we use *service automata* to represent processes, as they are expressive enough to handle common work flow patterns. Moreover, techniques exist to map high-level models (e.g. BPMN) to service automata [24]. In this section we recall some standard definitions and results on which our work rests. We largely follow the definitions and notation from [23].

A. Service Automata

We assume a finite set C of message channels over which services communicate asynchronously.

Definition 1 (Service Automaton): A (nondeterministic) *service automaton* is a quintuple $A = (Q, I, \delta, q_0, \Omega)$, where:

- (i) Q is a finite set of *states*;
- (ii) $I = I_{in} \cup I_{out}$ is an *interface* consisting of input and output channels so that $I_{in}, I_{out} \subseteq C$ and $I_{in} \cap I_{out} = \emptyset$;
- (iii) $\delta \subseteq Q \times L_A \times Q$ is a labelled *transition relation* over the set of labels of A, $L_A \stackrel{\text{def}}{=} \{?x \mid x \in I_{in}\} \cup \{!x \mid x \in I_{out}\} \cup \{\tau\}$;
- (iv) $q_0 \in Q$ is the *initial* state;
- (v) $\Omega \subseteq Q$ is a set of *final* states such that $q \in \Omega$ and $(q, l, q') \in \delta$ imply $l \in I_{in}$.

A service automaton is *deterministic* if it has no τ-transitions and for every state at most one outgoing transition for any given label. Most of the works in the literature (as e.g. [17]) consider *acyclic* service automata only.

We use $q_1 \xrightarrow{l} q_2$ to denote $(q_1, l, q2) \in \delta$. If q_2 is unique, we shall sometimes denote it as $\delta(q_1, l)$. For a state $q \in Q$, the set $en(q) \stackrel{\text{def}}{=} \{l \in L_A \mid \exists q' \in Q.\ q \xrightarrow{l} q'\}$ is the set of *enabled labels* at state q.

Two service automata are *composable* if the input channels of the first automaton are the output channels of the second one, and vice versa. Service automata use asynchronous communications; the *composition* of two composable service automata is a service automaton, the states of which are triples consisting of a state of the first automaton, a state of the second one, and an *internal message bag* representing the messages that have been sent by one automaton but not yet been received by the other.

In the literature, various criteria have been used to define soundness of such compositions. For example, in [15] a state of the composition is called a *deadlock* if no transition is enabled from this state, and either some of the composed automata is not in a final state or the message bag is not empty. A composition is then considered sound if it contains no deadlocks. In [15] the authors use the alternative soundness notion of *weak termination* [16], which captures the idea that the each partner should correctly terminate.

A service automaton A is called a *strategy* for service automaton B if their composition is sound w.r.t. a chosen

soundness notion. The set of strategies for automaton B is denoted by $Strat(B)$. Note that $A \in Strat(B)$ exactly when $B \in Strat(A)$.

B. Operating Guidelines

The operating guideline for a service automaton C is a finite structure characterizing all service automata that can soundly cooperate with C. That is, the operating guideline is a finite representation of $Strat(C)$. The test for this is phrased in terms of *service matching*, i.e., service automaton A can cooperate correctly with C if and only if A can be matched with the operating guideline of C.

Given service automaton $A = (Q_A, I, \delta_A, q_{0A}, \Omega_A)$, let $Atoms_A \stackrel{\text{def}}{=} L_A \cup \{\mathsf{final}\}$ be the set of atomic propositions of A, and let 2^{Atoms_A} be the set of truth assignments over $Atoms_A$. Let $Forms_A$ be the set of negation-free Boolean formulas over $Atoms_A$. For $\varphi \in Forms_A$, let $Sat_A(\varphi) \subseteq 2^{Atoms_A}$ denote the set of all truth assignments for which φ is true.

Definition 2 (Annotated Service Automaton): An *annotated service automaton* is a pair (B, Φ), where $B = (Q, I, \delta, q_0, \emptyset)$ is a deterministic service automaton without final states, and Φ is an annotation on states such that $\Phi(q) \in Forms_B$ for every $q \in Q$.

Definition 3 (Intrinsic Truth Assignment): Let $q_A \in Q_A$. The *intrinsic truth assignment* of q_A is defined as:

$$\beta_{q_A} \stackrel{\text{def}}{=} \begin{cases} en(q_A) & \text{if } q_A \notin \Omega_A \\ en(q_A) \cup \{\mathsf{final}\} & \text{otherwise.} \end{cases}$$

Then, a state $q_A \in Q_A$ is termed to *satisfy* a Boolean formula φ, denoted $q_A \models \varphi$, if $\beta_{q_A} \in Sat_A(\varphi)$.

The notion of service matching is first defined for service automata, and then adapted to matching a service automaton against an operating guideline.

Definition 4 (Complete Matching): Let $A = (Q_A, I, \delta_A, q_{0A}, \Omega_A)$ and $B = (Q_B, I, \delta_B, q_{0B}, \Omega_B)$ be service automata over the same interface. A *complete matching* between A and B is a binary relation $R \subseteq Q_A \times Q_B$ such that $(q_{0A}, q_{0B}) \in R$, and for every $(q_A, q_B) \in R$ we have:

(*i*) $q_A \xrightarrow{\tau} q'_A$ implies $(q'_A, q_B) \in R$;

(*ii*) $q_A \xrightarrow{a} q'_A$ (for $a \neq \tau$) implies $q_B \xrightarrow{a} q'_B$ and $(q'_A, q'_B) \in R$ for some $q'_B \in Q_B$.

Complete matching is thus a weak simulation relation when B is deterministic (as will be the case below).

An operating guideline for a service is a description that characterizes all services that can soundly cooperate with it.

Definition 5 (Operating Guideline): Let $A = (Q_A, I, \delta_A, q_{0A}, \Omega_A)$ be a service automaton. The annotated service automaton (B, Φ) with $B = (Q_B, I, \delta_B, q_{0B}, \emptyset)$ is an *operating guideline* for A iff for every strategy C of A there is a complete matching R between C and B such that $q_C \models \Phi(q_B)$ whenever $(q_C, q_B) \in R$. In the latter case we say that the service automaton C *matches* the operating guideline (B, Φ).

Since $\tau \in Atoms_B$, the operating guideline allows through Φ the matching of both deterministic and non-deterministic service automata C. Note also that since $\Phi(q_B)$ in effect specifies $en(q_A)$ to be among certain subsets of $en(q_B)$, the matching relation R is reminiscent of a bisimulation. This will be exploited in section IV.

Depending on the chosen definition of soundness there are different algorithms to extract the proper operating guideline from a given service automaton. If soundness is chosen as deadlock-freedom, it has been shown in [15] that every finite-state service automaton A that has at least one strategy, has an operating guideline (B, Φ) that can be constructed efficiently. The underlying service automaton B is itself a strategy for A, and is called its most permissive strategy.

C. Bisimulation Degree

Our notion of behavioural similarity between services is defined as a *bisimulation degree* between service automata, closely following the notion as developed by Sokolsky et al [22].

For service automata A and B we assume given a *discount parameter* $p \in [0, 1]$ to promote immediate similarity against delayed similarity, and a (symmetric) *label similarity* function $\lambda_{A,B} : L_A \times L_B \to [0, 1]$ to account for the relative matching of labels. To account for the relative matching of final states, we define two state similarity functions:

$$\theta_L(q_A, q_B) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } q_A \in \Omega_A \text{ implies } q_B \in \Omega_B \\ p_{\mathsf{fin}} & \text{otherwise.} \end{cases}$$

$$\theta_R(q_A, q_B) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } q_B \in \Omega_B \text{ implies } q_A \in \Omega_A \\ p_{\mathsf{fin}} & \text{otherwise.} \end{cases}$$

where $p_{\mathsf{fin}} \in [0, 1]$ is a discount parameter for final states. Our results below assume further that $0 < p < 1$ and $p_{\mathsf{fin}} < 1$, and that $\lambda_{A,B}(a, b) = 1$ if and only if $a = b$.

Definition 6 (Bisimulation Degree): Let $A = (Q_A, I, \delta_A, q_{0A}, \Omega_A)$ and $B = (Q_B, I, \delta_B, q_{0B}, \Omega_B)$ be service automata over the same interface. *Bisimulation degree* is a function $\xi : Q_A \times Q_B \to [0, 1]$ defined as follows:

$$\xi(q_A, q_B) \stackrel{\text{def}}{=} min(\xi_L(q_A, q_B), \xi_R(q_B, q_B))$$

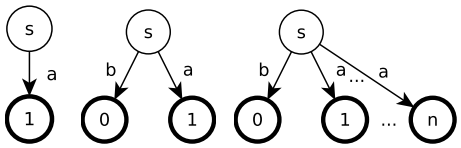
$$\xi_L(q_A, q_B) \stackrel{\text{def}}{=} \begin{cases} \theta_L(q_A, q_B) & \text{if } q_A \stackrel{a}{\not\to} \text{ for all } a \in L_A \\ (1-p) \cdot \theta_L(q_A, q_B) + p \cdot \chi_L(q_A, q_B) & \text{otherwise} \end{cases}$$

$$\xi_R(q_A, q_B) \stackrel{\text{def}}{=} \begin{cases} \theta_R(q_A, q_B) & \text{if } q_B \stackrel{b}{\not\to} \text{ for all } b \in L_B \\ (1-p) \cdot \theta_R(q_A, q_B) + p \cdot \chi_R(q_A, q_B) & \text{otherwise} \end{cases}$$

$$\chi_L(q_A, q_B) \stackrel{\text{def}}{=} \operatorname*{avg}_{q_A \xrightarrow{a} q'_A} \max_{q_B \xrightarrow{b} q'_B} \lambda_{A,B}(a, b) \cdot \xi(q'_A, q'_B)$$

$$\chi_R(q_A, q_B) \stackrel{\text{def}}{=} \operatorname*{avg}_{q_B \xrightarrow{b} q'_B} \max_{q_A \xrightarrow{a} q'_A} \lambda_{A,B}(a, b) \cdot \xi(q'_A, q'_B)$$

where max, min and avg denote the maximum, minimum and average functions over $[0, 1]$, respectively (with the convention that the maximum of the empty set is 0). The bisimulation degree between A and B is defined as $\xi(A, B) \stackrel{\text{def}}{=} \xi(q_{0A}, q_{0B})$.



(a) A (b) B_1 (c) B_n

Fig. 2. The set $\{\xi(A, B_i) \mid i \geq 1\}$ does not have a maximum

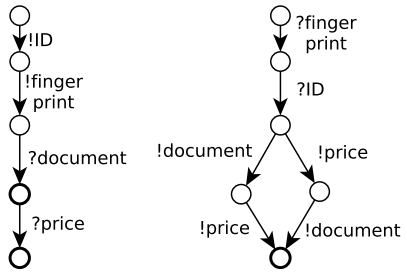
Well-definedness of bisimulation degree follows from the same arguments used to establish well-definedness of the Sokolsky's bisimulation [22]. The notion of bisimulation degree is strongly connected to the standard notion of strong bisimulation for labeled transition systems [20]. In fact, it is easy to show that $\xi(A, B) = 1$ if and only if A and B are strongly bisimilar. Moreover, since several soundness properties are preserved by strong bisimulation (e.g. weak termination and deadlock freedom), if $\xi(A, B) = 1$ then A and B have the same set of strategies. Thus the intuition is that bisimulation degree is a good candidate to measure the behavioral similarity between two automata and their corresponding sets of strategies.

III. Service Matching with Correction

In this section we give an overview of our approach to service matching with correction. We shall henceforth assume two enterprises, p_A and p_B, which own two private business processes, represented by the service automata A and B, respectively. If their composition is unsound, we are interested in correcting A by returning to p_A some other service automaton A' that can be composed soundly with B. However, there can potentially be a huge number of such automata. It is natural to require A' to be as close as possible to the original service automaton A, for two reasons: this solution would require fewer changes on A in order to achieve its transformation into A', and more importantly, it would keep most of the initial intention of the service modeled through A. Thus, the goal of *service matching with correction* is to allow participant p_A to discover, among all strategies for B, the service automaton most similar to A.

In this work we shall measure similarity behaviourally, by means of bisimulation degree ξ, and shall denote with $M_\xi(A, B)$ the strategy for B that is most similar to A. The latter notion, however, is not always well-defined: For some infinite sets of non-deterministic automata, bisimulation degree ξ does not have a maximum. This is exemplified by the automata in Figure 2, where $\xi(A, B_n) = (1-p) + p\frac{n+\lambda(a,b)}{n+1}$. We therefore aim to define $M_\xi(A, B)$ as the automaton A if $A \in \mathit{Strat}(B)$, i.e., if A and B can be composed soundly, and otherwise as some automaton A' such that $\xi(A, A'') \leq \xi(A, A')$ for any deterministic A'' such that $A'' \in \mathit{Strat}(B)$. In fact, it is easy to show that for every (possibly infinite) set of deterministic automata, the bisimulation degree ξ with respect to an input automata has a maximum.

We shall henceforth assume that soundness is defined as deadlock freedom. We illustrate our approach on the example from Figure 1. Service matching between A_1 and B yields to participant p_A the original automaton A_1, indicating that the composition of the two automata is sound.



(a) A'_2 (b) B_1

Fig. 3. Confidential matching with correction

On the other hand, A_2 cannot be soundly composed with B, since A_2 (i) requests an ID card without providing the necessary fingerprints, thus leading the collaboration to a deadlock, and (ii) never collects the delivered invoice. Assuming that for $a \neq b$ it holds that $p = p_{\mathsf{fin}} = \lambda_{A,B}(a, b) = 0.5$, service matching yields the automaton A'_2 shown in Figure 3a. Intuitively, (i) the transition "noFingerprint" is replaced with "fingerprint" and (ii) an additional step is added to the process: the collection of the invoice. This process can be directly accepted by p_A and be used as the new enterprise work flow. Otherwise, it can be the input to a refinement loop, where p_A uses A'_2 as a blueprint to correct its own process, and reiterates service matching.

In addition to the above, we want to compute $M_\xi(A, B)$ without compromising the privacy of the participants. That is, nothing more than what can be deduced from $M_\xi(A, B)$ is to be leaked to the participants or any third party. For instance, the participant p_A should not be able to distinguish between the participant p_B owning the automaton of Figure 1a and the one in Figure 3b. In fact, in both cases the most similar automaton to A_2 that can be soundly composed with the automaton of p_B is A'_2.

To implement service matching with correction (i.e., compute $M_\xi(A, B)$) and preserve participant's privacy, we use the following strategy:

1) we assume that an algorithm extracts an operating guideline (C, Φ) that represents $\mathit{Strat}(B)$,
2) we introduce a new measure, the *weighted matching* $\mu(A, C, \Phi)$ that measures "how much" an automaton A matches an operating guideline (C, Φ),
3) we define an algorithm that computes μ and extracts a correction automaton A' for which we demonstrate that:
   a) A' matches the operating guideline (C, Φ) and is thus a strategy for B,
   b) $\xi(A, A') = \mu(A, C, \Phi)$, and that
   c) no deterministic automaton matching the operating guideline is more similar to A than A', and
4) we develop an implementation of the algorithm for computing A' by means of SMC techniques.

IV. Weighted Service Matching

This section develops the notion of weighted matching, which estimates the degree of matching of service automata against operating guidelines. The intuition is that weighted matching can be used to measure the similarity between an

automaton and the set of strategies for a partner. This is indeed justified by Theorem 2, which demonstrates that the weighted matching is exactly the maximum bisimulation degree between the automaton and any deterministic strategy for the partner.

Let A be a service automaton and (B, Φ) be an operating guideline. Our formalization follows closely the one from Section II-C. However, in the present case we have a deterministic service automaton B without finals states, while τ-transitions and acceptance are implicitly prescribed by Φ. We shall therefore relativize similarity on truth assignments.

We shall again assume a *discount parameter* p, a *label similarity* function $\lambda_{A,B}$ and a *final state discount* p_{fin}. For a state $q_A \in Q_A$, state $q_B \in Q_B$ and truth assignment $\beta \in Sat_A(\Phi(q_B))$, we define the state *similarity functions* as follows:

$$\theta'_L(q_A, q_B, \beta) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } q_A \in \Omega_A \text{ implies } \mathsf{final} \in \beta \\ p_{\mathsf{fin}} & \text{otherwise.} \end{cases}$$

$$\theta'_R(q_A, q_B, \beta) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \mathsf{final} \in \beta \text{ implies } q_A \in \Omega_A \\ p_{\mathsf{fin}} & \text{otherwise.} \end{cases}$$

Notice that for the intrinsic truth assignment of q_A from Definition 3 we have $\theta'_L(q_A, q_B, \beta_{q_A}) = \theta'_R(q_A, q_B, \beta_{q_A}) = 1$.

Definition 7 (Weighted Matching): Let $A = (Q_A, I, \delta_A, q_{0A}, \Omega_A)$ be a service automaton and (B, Φ) with $B = (Q_B, I, \delta_B, q_{0B}, \emptyset)$ be an annotated service automaton over the same interface. *Weighted matching* is a function $\mu : Q_A \times Q_B \to [0, 1]$ defined as follows:

$$\mu(q_A, q_B) \stackrel{\text{def}}{=} \max_{\beta \in Sat_A(\Phi(q_B))} \min(\mu_L(q_A, q_B, \beta), \mu_R(q_B, q_B, \beta))$$

$$\mu_L(q_A, q_B, \beta) \stackrel{\text{def}}{=} \begin{cases} \theta'_L(q_A, q_B, \beta) & \text{if } q_A \stackrel{a}{\not\to} \text{ for all } a \in L_A \\ (1-p) \cdot \theta'_L(q_A, q_B, \beta) + p \cdot \nu_L(q_A, q_B, \beta) & \text{otherwise} \end{cases}$$

$$\mu_R(q_A, q_B, \beta) \stackrel{\text{def}}{=} \begin{cases} \theta'_R(q_A, q_B, \beta) & \text{if } \tau \notin \beta \text{ and } q_B \stackrel{b}{\not\to} \text{ for all } b \in \beta \setminus \{\mathsf{final}\} \\ (1-p) \cdot \theta'_R(q_A, q_B, \beta) + p \cdot \nu_R(q_A, q_B, \beta) & \text{otherwise} \end{cases}$$

$$\nu_L(q_A, q_B, \beta) \stackrel{\text{def}}{=} \operatorname*{avg}_{q_A \stackrel{a}{\to} q'_A} \max_{b \in \beta \setminus \{\mathsf{final}\}} \lambda_{A,B}(a, b) \cdot \mu(q'_A, q'_B)$$

$$\nu_R(q_A, q_B, \beta) \stackrel{\text{def}}{=} \operatorname*{avg}_{b \in \beta \setminus \{\mathsf{final}\}} \max_{q_A \stackrel{a}{\to} q'_A} \lambda_{A,B}(a, b) \cdot \mu(q'_A, q'_B)$$

where in the last two clauses q'_B is the state defined as:

$$q'_B \stackrel{\text{def}}{=} \begin{cases} q_B & \text{if } b = \tau \\ \delta_B(q_B, b) & \text{otherwise.} \end{cases}$$

As in [22], well-definedness of μ is established by showing that weighted matching is the unique greatest fixed point of the monotone transformer induced by the defining equations[1]. We lift μ to service automata and operating guidelines by defining $\mu(A, B, \Phi) \stackrel{\text{def}}{=} \mu(q_{0A}, q_{0B})$.

Theorem 1: A matches (B, Φ) iff $\mu(A, B, \Phi) = 1$.

We want to use the computation of the weighted matching between an automaton A and an operating guideline (B, Φ) to generate a correction automaton; this should be the most bisimilar service automaton to A among all service automata matching (B, Φ). To achieve this we need the notion of *synchronization automaton* (or graph), adapted from [14].

Definition 8 (Synchronization Automaton): Let $A = (Q_A, I, \delta_A, q_{0A}, \Omega_A)$ be a service automaton and (B, Φ) with $B = (Q_B, I, \delta_B, q_{0B}, \emptyset)$ an annotated service automaton over the same interface. The *synchronization automaton* of A and (B, Φ) is the automaton $A \odot (B, \Phi) = (Q_A \times Q_B, I^2, \delta, (q_{0A}, q_{0B}))$, where δ is defined so that $(q_A, q_B) \stackrel{(a,b)}{\to} (q'_A, q'_B)$ if and only if $q_A \stackrel{a}{\to} q'_A$, and either $b = \tau$ and $\tau \in \beta$ for some $\beta \in Sat_A(\Phi(q_B))$ and $q'_B = q_B$, or else $q_B \stackrel{b}{\to} q'_B$.

This automaton is essentially the playfield for the computation of $\mu(A, B, \Phi)$, and serves as the starting point for the extraction of a most bisimilar to A service automaton.

Definition 9 (Extracted Automaton): The automaton C extracted from the computation of $\mu(A, B, \Phi)$ is constructed as follows. For each pair of states (q_A, q_B) for which $\mu(q_A, q_B)$ has been computed from the equations of Definition 7, record a maximizing truth assignment $\beta_{q_A,q_B} \in Sat_A(\Phi(q_B))$. Then, starting from the synchronization automaton $A \odot (B, \Phi)$:

- (i) complete the automaton by adding transitions $(q_A, q_B) \stackrel{(\tau,b)}{\to} (q_A, q'_B)$ for every q_A, q_B and b such that $q_A \not\to$ and $q_B \stackrel{b}{\to} q'_B$;
- (ii) replace all edge labels (a, b) by their respective second component $b \in L_B$;
- (iii) for every node (q_A, q_B), remove all outgoing edges labelled with a label not in $\beta_{q_A,q_B} \setminus \{\mathsf{final}\}$; and
- (iv) equip the resulting automaton with a set of final states defined as $\Omega_C \stackrel{\text{def}}{=} \{(q_A, q_B) \mid \mathsf{final} \in \beta_{q_A,q_B}\}$.

We thus obtain a service automaton $C \stackrel{\text{def}}{=} (Q_A \times Q_B, I, \delta_C, (q_{0A}, q_{0B}), \Omega_C)$.

Theorem 2: Let A be a service automaton, (B, Φ) be an operating guideline and C be the service automaton extracted from the computation of $\mu(A, B, \Phi)$ as defined in Definition 9. Let q_A, q_B and q_C be states of A, B and C, respectively. Then:

2.1 C matches (B, Φ)

2.2 $q_C = (q_A, q_B)$ implies $\xi(q_A, q_C) \geq \mu(q_A, q_B)$

2.3 $q_C = (q'_A, q_B)$ implies $\xi(q_A, q_C) \leq \mu(q_A, q_B)$.

Moreover, if D is a deterministic service automaton that matches (B, Φ) and q_D is a state of D, then:

2.4 $\xi(q_A, q_D) \leq \mu(q_A, q_B)$

2.5 $\xi(A, D) \leq \xi(A, C)$

Notice that together, 2.2 and 2.3 imply $\xi(q_A, (q_A, q_B)) = \mu(q_A, q_B)$. The next result is a direct consequence of Theorem 2.

Corollary 1: Let $A = (Q_A, I, \delta_A, q_{0A}, \Omega_A)$ be a service automaton and (B, Φ) with $B = (Q_B, I, \delta_B, q_{0B}, \emptyset)$ be an annotated service automaton over the same interface. Then:

$$\mu(A, B, \Phi) \geq max_{D:\ D \text{ deterministic and matches } (B,\Phi)}\ \xi(A, D)$$

[1]The proofs of the theorems are omitted here for space reasons, but are available in http://www.csc.kth.se/~robertog/pst2015.pdf.

V. Privacy Preserving Service Matching

In this section we describe an implementation of weighted service matching and correction (extraction of most similar service automaton) on top of the SHAREMIND SMC platform [1]. The platform implements the *Arithmetic Black Box (ABB)* functionality [5] in the *outsourced secure computation* style — any number of *input parties* can provide data to a computation by secret-sharing it among three *computing parties*. The computing parties execute protocols to perform (certain) operations on secret-shared data, obtaining secret sharings of results. The shares of the results are sent to the parties that are supposed to learn the result of the computation. The protocols of SHAREMIND [2] provide security against one honest-but-curious party.

In our setting, one input party owns the automaton A, while another owns the operating guideline (B, Φ). To compute μ and the extracted automaton C, they secret-share the descriptions of A and (B, Φ) among the computing parties in a manner that facilitates the computations. After performing the computations, the computing parties send the shares of the description of C back to the first input party. In the following, we let $[\![v]\!]$ denote that a value v is secret-shared among the computing parties. The write-up $[\![u]\!] \otimes [\![v]\!]$, where $\otimes$ is some (arithmetic) operation, denotes the invocation of a protocol that computes secret-shared $u \otimes v$ from the shares of u and v.

Generally, SMC protocols do not attempt to hide the size of the inputs. We also make no attempt to hide them, hence we assume that $|Q_A|$, $|Q_B|$, $|\delta_A|$, $|\delta_B|$ and $|\Phi|$ are public. In principle, their size could be somewhat hidden by padding them to some maximum size. We represent Φ as a mapping from Q_B to subsets of 2^{Atoms_B} and leak $\sum_{q_B \in Q_B} |\Phi(q_B)|$, as well as $\sum_{q_B \in Q_B} \sum_{\beta \in \Phi(q_B)} |\beta|$.

To start the computation, first party secret-shares δ_A among the parties, obtaining $[\![\delta_A]\!]$. Here we consider δ_A to be a sequence of triples (q_A, a, q'_A) with $q_A, q'_A \in Q_A$ and $a \in I$. We identify Q_A with the set $\{0, \ldots, |Q_A| - 1\}$ and I with the set $\{0, \ldots, |I| - 1\}$. First party also secret-shares the characteristic vector of Ω_A.

The second party secret-shares the sequences δ_B, γ and γ'. Here γ is a sequence of triples (q_B, β, n), where $q_B \in Q_B$, $\beta \in Sat(\Phi(q_B))$ (represented as a bit-vector of length $|I|+1$) and $n = |\{(q_B \xrightarrow{b} q'_B \,|\, b \in \beta\}|$; and γ' is a sequence of quadruples (q_B, β, b, q'_B), where $q_B \xrightarrow{b} q'_B$, $\beta \in Sat(\Phi(q_B))$, and $b \in \beta$. The sequences γ and γ' have to be in the same order wrt. their first two components.

Actually, both input parties secret-share more data about A and (B, Φ). This additional data can in principle be computed from the secret-shared sequences described above. But as it can be computed from a single party's data, it is more efficient if that party computes it itself. We will elaborate below.

The label similarity function $\lambda_{A,B}$ (represented as a sequence of length $|I|^2$), and the discount parameters p and p_{fin} have also been secret-shared — these should be known by both input parties, but the computing parties should not learn them. The computing parties now initialize a $|Q_A| \cdot |Q_B|$-sized sequence $[\![\mu]\!]$ and iterate the transformer induced by the equations in Def. 7. The number of iterations affects the running time and is thus public. In our implementation, it is yet another parameter given to the computing parties; in this sense our solution is similar to [22, Sec. 5.2]. Fig. 4 gives a high-level overview of one iteration.

1) $[\![\Xi(q_A, q_B, a, b)]\!] \leftarrow [\![\lambda(a, b)]\!] \cdot [\![\mu(q_A, q_B)]\!]$
2) $[\![\nu^{\times}_R(q_A, b, q'_B)]\!] \leftarrow \max_{(q_A, a, q'_A) \in [\![\delta_a]\!]} [\![\Xi]\!](q'_A, q'_B, a, b)$
3) $[\![\nu^{\#}_R(q_A, q_B \xrightarrow{b} q'_B, \beta)]\!] \leftarrow [\![\nu^{\times}_R(q_A, b, q'_B)]\!]$
4) $[\![\nu_R(q_A, q_B, \beta)]\!] \leftarrow \text{avg}_{(b, q'_B)} [\![\nu^{\#}_R(q_A, q_B \xrightarrow{b} q'_B, \beta)]\!]$
5) $[\![\nu_L(q_A, q_B, \beta)]\!] \leftarrow \text{avg}_{(a, q'_A)} \max_{(q_B, b, q'_B, \beta) \in [\![\gamma']\!]} [\![\Xi]\!](q'_A, q'_B, a, b)$
6) Compute $[\![\mu_X(q_A, q_B, \beta)]\!]$ from $[\![\nu_X(\ldots)]\!]$ and $[\![\theta'_X(\ldots)]\!]$
7) $[\![\mu'(q_A, q_B, \beta)]\!] \leftarrow \min([\![\mu_L(q_A, q_B, \beta)]\!], [\![\mu_R(q_A, q_B, \beta)]\!])$
8) $[\![\mu(q_A, q_B)]\!] \leftarrow \max_\beta [\![\mu(q_A, q_B, \beta)]\!]$

Fig. 4. One iteration of computing $[\![\mu]\!]$

In Step 1, we multiply $\lambda(a, b)$ and $\mu(q_A, q_B)$ for all q_A, q_B, a, b. SHAREMIND's protocol set allows us to compute $[\![\Xi]\!]$ with $O(|\lambda| + |\mu|)$ workload, not with $O(|\lambda| \cdot |\mu|)$. All multiplications can be done in parallel.

In Step 2, we compute $[\![\nu^{\times}_R]\!]$, where each element is a maximum of several elements of $[\![\Xi]\!]$. The computation is done in parallel for all q'_B and b; let us fix them while discussing step 2. To compute $[\![\nu^{\times}_R(q_A)]\!]$ in parallel for all $q_A \in Q_A$, the first party prepares an arithmetic circuit C_A with (binary) $\max$-gates and (unary) identity gates. The circuit has $|Q_A| \cdot |I|$ inputs and $|Q_A|$ outputs; if the inputs are initialized with $\Xi(q'_A, a)$, then the outputs will be $\nu^{\times}_R(q_A)$. The circuit must be *oblivious* in the following sense: all wires connect gates on adjacent layers only, and the depth of the circuit, and the number of $\max$- and identity-gates in each layer must depend only on public values $|Q_A|$, $|I|$ and $|\delta_A|$. Only the connections between each pair of adjacent layers may depend on δ_A. The first party secret-shares their description (represented as vectors) among the computing parties. In Step 2, this circuit is obliviously evaluated by the computing parties. It is straightforward to compute the maxima, if the inputs to each $\max$-gate are known. To transfer the outputs of one layer to inputs of the next layer, the computing parties use the oblivious parallel reading protocol described in [11]. For reading x values according to private indices from an array of size y, this protocol has a setup cost of $O((x+y)\log(x+y))$ for the given indices, after which each read requires $O(x+y)$ communication in constant number of rounds.

In Step 3, consider $\nu^{\times}_R$ as a matrix with $|Q_A|$ rows and $|Q_B| \cdot |I|$ columns. We turn it to a matrix $\nu^{\#}_R$ with $|Q_A|$ rows and $|\gamma'|$ columns, where each column of $\nu^{\#}_R$ is a copy of some column of $\nu^{\times}_R$ (but it is private, which one). We can again use the oblivious parallel reading protocol [11] to obliviously read the columns of $[\![\nu^{\times}_R]\!]$; the indices are given in $[\![\gamma']\!]$.

In Step 4, all $|Q_A|$ rows of $\nu^{\#}_R$ are processed in parallel. We have to compute the averages of the segments of a row of length $|\gamma'|$. The lengths of these segments are given as the third components of $[\![\gamma]\!]$. First we compute the sums of these segments; this computation is described in Fig. 5, where $[\![v]\!]$ is initialized with the third components of $[\![\gamma]\!]$, and $[\![z]\!]$ is initialized with a row of $[\![\nu^{\#}_R]\!]$. The prefix sum and its inverse involve only local operations at computing parties, hence they

Given: Vectors $[\![v]\!]$ and $[\![z]\!]$ of lengths m and $\sum_{i=1}^{m} v_i$
Result: Vector $[\![x]\!]$, where $x_i = \sum_{j=w_{i-1}}^{w_i - 1} z_i$ and $w_i = \sum_{j=1}^{i} v_i$

(i) $[\![w]\!] \leftarrow \mathsf{prefixsum}([\![v]\!])$
(ii) $[\![z']\!] \leftarrow \mathsf{prefixsum}([\![z]\!])$
(iii) $[\![x']\!] \leftarrow \mathsf{obliviousRead}([\![z']\!]; [\![w_1]\!], \ldots, [\![w_m]\!])$
(iv) $[\![x]\!] \leftarrow \mathsf{prefixsum}^{-1}([\![x']\!])$

Fig. 5. Oblivious computation of partial sums

do not figure into our cost analysis. The oblivious parallel read selects the elements in positions $w_1, \ldots, w_m$ from $[\![z']\!]$. To obtain the averages, we pointwise divide [2, Alg. 9] the elements of the vector $[\![x]\!]$ with the third components of $[\![\gamma]\!]$.

In Step 5, which is an analogue to Steps 2–4, we compute $[\![\nu_L]\!]$, also using an oblivious maximum-finding circuit shared by second party and a vector of length $|Q_A|$ containing the out-degrees of all nodes of A shared by the first party.

Step 6 is computed according to Def. 7. The values of $[\![\theta'_L(q_A, q_B, \beta)]\!]$ and $[\![\theta'_R(q_A, q_B, \beta)]\!]$ have been computed before the iterations from $[\![\Omega]\!]$ and $[\![\gamma]\!]$. Step 7 is straightforward. Step 8 consists of the application of another oblivious maximum-finding circuit secret-shared by the second party.

The running time of a single iteration is asymptotically dominated by the execution of oblivious maximum-finding circuits, the size of which is linearithmic in the sizes of A and (B, Φ), while their execution time is linear in their size (with linearithmic preprocessing before starting the iterations). A reasonable number of iterations can be computed from the discount parameter $p$determining the speed of convergence.

Most of the values occurring in the computation are fractions in $[0, 1]$. We represent them as fix-point numbers [3], as SHAREMIND natively computes with integers only.

After computing $[\![\mu]\!]$, we determine the *extracted automaton* C (Def. 9). The set $Q_A \times Q_B$ of its vertices is public, and the set of its edges can be seen as a subset of $\delta_C^{\#} = \delta_A \times \delta_B$ (with some components removed). During Step 8 of the last iteration of computing $[\![\mu]\!]$, we will also record $[\![\beta_{q_A, q_B}]\!]$ for all q_A, q_B (note that $[\![\Omega_C]\!]$ is a part of it). We form $[\![\delta_C^{\#}]\!]$ and augment its with an extra component $[\![\beta_{q_A, q_B}]\!]$ (using oblivious parallel read). For each element of $\delta_C^{\#}$ we now decide whether it is an element of δ_C. We replace the other elements of $[\![\delta_C^{\#}]\!]$ with nonsense data, randomly shuffle $[\![\delta_C^{\#}]\!]$ [12], and send the shares of the resulting sequence to the first party.

We performed some preliminary studies of the running time of our implementation, to see whether our approach is feasible at all. We have executed all three computing parties on the same computer — a Lenovo Thinkpad X240 with an Inter Core i5-4300U CPU running at 1.9 GHz (with four cores) and 8GB memory, running Ubuntu 14.04. The computing parties communicated with each other over the loopback interface. The traffic volume between two parties always remained below 1Gbit/s. The network latency should not make a significant difference in execution times, because our protocols are highly parallelized. Hence the running times reported in Table I would be similar to those of three computing parties running in different machines, connected with high-bandwidth (1 Gbit/s) channels; this is a highly feasible configuration. These times are for the larger examples from [13]. For the examples we've considered, we give its name (from [13]), its size, characterized by the number of labels, the number of vertices and edges of the SA and the OG, and also the total size of satisfying assignments in all vertices of OG: $\sum_{q_B \in Q_B} |Sat(\Phi(q_B))|$. We report the time it took to run a single iteration of the algorithm in Fig. 4; the necessary number of iterations for each task has to be determined from its parameters, as we discussed above. We also report the time it took to set up the data used in the iterations. This time includes the computations of $[\![\theta'_L]\!]$ and $[\![\theta'_R]\!]$, as well as performing the set-up for oblivious reads during the iterations. The time for extracting the automaton C from the resulting $[\![\mu]\!]$ would be much smaller and is not measured We can definitely say that the obtained running times show the feasibility of our approach, we should be able to privacy-preserving match realistically sized service automata in a few hours.

service	$\lvert I\rvert$	$\lvert Q_{\mathrm{SA}}\rvert$	$\lvert \delta_{\mathrm{SA}}\rvert$	$\lvert Q_{\mathrm{OG}}\rvert$	$\lvert \delta_{\mathrm{OG}}\rvert$	$\lvert \Phi\rvert$	t_{prep}	t_{iter}
Online Shop	16	222	531	153	331	735	1m09s	2m29s
Internal Order	9	14	18	512	2304	19172	4m42s	2m33s
Purchase Order	10	137	437	168	548	2074	1m29s	4m14s

TABLE I. RUNNING TIMES OF PRIVACY-PRESERVING SERVICE MATCHING

VI. RELATED WORK

The need of enforcing soundness of interorganizational business processes lead to the introduction of choreography languages (e.g. WS-CDL [10], BPEL4Chor [6]). These contract-oriented approaches (and the corresponding formal methods like [15], [9]) are all top-down and require the manual definition and public disclosure of a global process; they cannot be used when a manual agreement phase is not an option. There are no existing bottom-up mechanisms that can check soundness of process composition while preserving the participant's privacy and confidentiality.

On the other hand, certain bottom-up mechanisms exist to engineer interorganizational processes while preserving the participant's privacy. For example, our earlier work [7] enables competitive partners to discover the possible executions (traces) of their collaboration without leaking the private workflows. However, this result cannot be used to check soundness of the process composition, since the mechanism is built on top of the notion of language equivalence, which does not preserve the widely adopted definitions of soundness.

We argue that if the goal is to enable correct collaboration by checking soundness of interorganizational processes, and these processes are to be kept secret, then a bottom-up approach can not limit its answers to Boolean results: If the soundness check fails, the participants must be guided how to fix their local procedures to form a sound collaboration. This requires some notion of *process similarity* to be taken into account to suggest corrections. In this paper we adapted to service automata the *behavioral similarity* introduced by Sokolsky et al [22]. Without taking privacy into account, Lohmann [14] proposed a mechanism to correct automata based on a different notion of *weighted matching* between the automata and operating guidelines, namely a structural similarity measure inspired by "edit distances", with the aim of finding the least number of structural changes required to obtain isomorphism between the input automaton and the corrected one.

VII. Concluding Remarks

We presented a bottom-up mechanism to match business processes and to suggest suitable corrections. Existing approaches that take into account secrecy of the input processes are top-down and require to disclose the complete set of strategies of the participants (by either publishing a public interface [24] or by disclosing the complete operating guideline). We go beyond these results, by only leaking one of the possible strategies for the partner's process.

The enabling result is the notion of weighted service matching introduced in Section IV. On top of this notion we devise an algorithm that constructs, from an operating guideline, the matching automaton that is behaviorally most similar to the input automaton. Differently from [14], our approach measures behavioral similarity rather edit distance. We present and prove the correctness of the extraction algorithm. The extraction enables one of the participants to receive a suitable correction without forcing them to disclose their internal processes.

We defined our notion of bisimulation degree by adapting the work of Sokolsky et al [22] to service automata. This allows us to build our mechanism for business processes matching on top of an existing and established behavioral similarity. The latter has been used in several applications, including malware detection [22], merging of statecharts [19] and model integration [4]. However, the adoption of this similarity measure limits the result of Theorem 1 to deterministic automata, since for certain infinite sets of non-deterministic automata, bisimulation degree does not have a maximum. This limitation can be avoided by adopting a different behavioral similarity, for example by substituting in the definition of χ_L (and χ_R) the average $avg_{q_B \xrightarrow{b} q'_B}$ with $avg_{b \in en(q_B)}$.

To achieve privacy, we implemented the algorithm in the programming language SecreC, executing on the Sharemind platform for secure multiparty computation. As a result, only the correction information is leaked to the first enterprise and no more. Moreover, the SMC algorithm presented in Section V can be executed on top of different Arithmetic Black Boxes.

Acknowledgment

The authors are indebted to Karsten Wolf for useful discussions in the early stages of this work. This work has been supported by the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 284731 "Usable and Efficient Secure Multiparty Computation (UaESMC)".

References

[1] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Proceedings of ESORICS'08*, pages 192–206, 2008.

[2] Dan Bogdanov, Margus Niitsoo, Tomas Toft, and Jan Willemson. High-performance secure multi-party computation for data mining applications. *Int. J. Inf. Sec.*, 11(6):403–418, 2012.

[3] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In *Proceedings of the Conference on Financial Cryptography and Data Security, FC'10*, volume 6052 of *LNCS*, pages 35–50. Springer, 2010.

[4] Marsha Chechik, Shiva Nejati, and Mehrdad Sabetzadeh. A relationship-based approach to model integration. *Innovations in Systems and Software Engineering*, 8(1):3–18, 2012.

[5] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *Proceedings of CRYPTO'03*, pages 247–264, 2003.

[6] Gero Decker, Oliver Kopp, Frank Leymann, and Mathias Weske. Bpel4chor: Extending bpel for modeling choreographies. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 296–303. IEEE, 2007.

[7] Roberto Guanciale, Dilian Gurov, and Peeter Laud. Private intersection of regular languages. In *Privacy, Security and Trust (PST), 2014 Twelfth Annual International Conference on*, pages 112–120. IEEE, 2014.

[8] Michel Hack. Petri net language. Technical report, 1976.

[9] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *ACM SIGPLAN Notices*, 43(1):273–284, 2008.

[10] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon, and Charlton Barreto. Web services choreography description language version 1.0. *W3C candidate recommendation*, 9, 2005.

[11] Peeter Laud. Privacy-Preserving Minimum Spanning Trees through Oblivious Parallel RAM for Secure Multiparty Computation. Cryptology ePrint Archive, Report 2014/630, 2014. http://eprint.iacr.org/.

[12] Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-Efficient Oblivious Database Manipulation. In *Proceedings of the 14th International Conference on Information Security. ISC'11*, volume 7001 of *LNCS*, pages 262–277. Springer, 2011.

[13] Niels Lohmann. Correcting deadlocking service choreographies using a simulation-based graph edit distance. In *Proceedings of BPM 2008*, volume 5240 of *Lecture Notes in Computer Science*, pages 132–147, 2008. Case studies available from http://service-technology.org/publications/lohmann_2008_bpm/.

[14] Niels Lohmann. *Correctness of services and their composition*. PhD thesis, Zugl.: Rostock, Univ., Diss. und Eindhoven, Techn. Univ., Diss., 2010, 2010.

[15] Niels Lohmann, Peter Massuthe, and Karsten Wolf. Operating guidelines for finite-state services. In *Proceedings of ICATPN'07*, volume 4546 of *Lecture Notes in Computer Science*, pages 321–341. Springer, 2007.

[16] Peter Massuthe and Karsten Schmidt. Operating guidelines-an automata-theoretic foundation for the service-oriented architecture. In *Quality Software, 2005.(QSIC 2005). Fifth International Conference on*, pages 452–457. IEEE, 2005.

[17] Peter Massuthe and Karsten Wolf. An algorithm for matching nondeterministic services with operating guidelines. In *Proceedings of Dagstuhl Seminar on The Role of Business Processes in Service Oriented Architectures*, 2006.

[18] Robin Milner. *Communicating and mobile systems: the pi calculus*. Cambridge university press, 1999.

[19] Shiva Nejati, Mehrdad Sabetzadeh, Marsha Chechik, Steve Easterbrook, and Pamela Zave. Matching and merging of statecharts specifications. In *Proceedings of the 29th international conference on Software Engineering*, pages 54–64. IEEE Computer Society, 2007.

[20] Davide Sangiorgi. A theory of bisimulation for the π-calculus. *Acta informatica*, 33(1):69–97, 1996.

[21] August-Wilhelm Scheer and Markus Nüttgens. *ARIS architecture and reference models for business process management*. Springer, 2000.

[22] Oleg Sokolsky, Sampath Kannan, and Insup Lee. Simulation-based graph similarity. In *Proceedings of TACAS'06*, volume 3920 of *Lecture Notes in Computer Science*, pages 426–440, 2006.

[23] Christian Stahl, Peter Massuthe, and Jan Bretschneider. Deciding substitutability of services with operating guidelines. volume 5460 of *LNCS*, pages 172–191. Springer, 2009.

[24] Wil MP van der Aalst, Niels Lohmann, Peter Massuthe, Christian Stahl, and Karsten Wolf. Multiparty contracts: Agreeing and implementing interorganizational processes. *The Computer Journal*, 53(1):90–106, 2010.

[25] Stephen A White. Introduction to BPMN. *IBM Cooperation*, 2, 2004.

[26] Karsten Wolf. Does my service have partners? In *Transactions on Petri Nets and Other Models of Concurrency II*, pages 152–171. Springer, 2009.