Operating Guidelines – an Alternative to Public View

Peter Massuthe and Karsten Schmidt

Institut für Informatik
Humboldt-Universität zu Berlin
Email: {massuthe,kschmidt}@informatik.hu-berlin.de

Abstract. We propose *operating guidelines* as artifacts for publishing information about how to communicate with a business process that is intended to be provided as a service. We present an approach to compute operating guidelines fully automatically. We compare operating guidelines with the concept of *public view*.

Keywords: Service composition, e-services technology, public view, operating guidelines, controller synthesis

1 Introduction

Business processes are trade secrets. Nevertheless, more and more business processes are designed as a *service* (according to the *service oriented architecture* [6]) for being automatically connected to their customers, using service brokers or similar concepts. Such a process P (the *service provider*) must publish some information about how to communicate with P. A currently quite popular approach to provide this information consists of publishing a so-called *public view* of P, an abstracted version of P that contains sufficient information about the communication structure of P, yet being different enough from P to keep P secret. It is assumed that a process R (the *service requestor*) that wants to communicate with P can derive its own correct communication behavior from an investigation of the public view of P.

In this paper, we consider abstract models of business processes and their communication partners. We abstract from almost every aspect except control flow and communication. In this setting, we come to the conclusion that a public view is a less than optimal tool for providing information about how to communicate with a business process P. Instead, we find that publishing *operating guidelines* generated by the owner of P are a much more suitable and elegant instrument for specifying the communication protocol with P.

Consider, as an example of a business process, a vending machine M. The public view of M would be some kind of abstract description of how M actually works, from which the customer is expected to figure out at which time he or she is expected to insert a coin, or to push a button. Such a description is not what we usually find pinned to a vending machine. Instead, we usually find (if anything)

operating guidelines, i.e. an algorithm that is expected to be run by the user (possibly including choices that are resolved depending on the user's intention). That is, rather than a description of M's behavior, there is a description of the expected customer's behavior.

This example illustrates our proposal: Instead of a public view of a process P (an abstract description of P's behavior), we suggest that the owner of P generates and publishes operating guidelines for all intended communication partners (partner roles) of P.

We show that it is in fact possible to automatically generate such operating guidelines from a business process model. In many situations, it is even possible to achieve completeness of these guidelines. Completeness means that *every* behavior of a communication partner that *violates* the operating guidelines may cause deadlocks, or erroneous messages.

Given a business process P, we basically view a single communication partner (resp. several communication partners) as a centralized (resp. decentralized) controller for P aiming at termination in a "good" end state. The generation of operating guidelines thus consists of synthesizing a controller.

In Sec. 2, we embed our contribution into related work. Sec. 3 is devoted to a discussion of the shortcomings of the public view approach for our particular perspective. In Sec. 4, we introduce basic concepts concerning business process models and controllers. Then, in Sec. 5, we outline the construction of operating guidelines for the case of a single communication partner. Sec. 6 deals with the case of more than one communication partner. Finally, Sec. 7 closes the paper, gives a summary of our work, and sketches further work.

2 Scope and Related Work

Business processes that are designed to be published and connected via service brokers or similar technologies, are sometimes referred to as *web services*. We do not use the term *web service* in this paper, since many readers have a much more technical perspective on web services than is considered in this paper.

For publishing a process P, some information about how to communicate with P is necessary. [7] and [8] propose generating a public view of P. A public view is seen as an abstract version of P such that every process that properly interacts with the public view, interacts properly with P itself. [7] does not propose a formal notion of public view, nor a formal approach to generating one. [8] proposes a formal relation between private and public view, but no generally applicable algorithm to compute a public view. The notion relies on a strict alternation between input and output phases and is therefore not suitable for concurrent communication with multiple partners. In contrast, our approach is well suited for the case of multiple communication partners.

[2] and [5] use the term *public view* in a different scenario: Several parties agree on public views of their intended parts of a common distributed business process and, subsequently, separately implement private views that fit to their own public view. There is a formal notion of accordance between public and

private view. In contrast to the approach mentioned before, the private view is now derived from a public view (not vice versa), and the public views are negotiated between all business partners.

In this paper, we use the term *public view* in the sense of [7, 8] and consider the scenario outlined there. We contribute an alternative to public views as considered there: operating guidelines. The concept of public view as proposed in [2] or [5] is outside our scope.

In our considerations, we study formal models that abstract from several aspects of business processes (roles, data, etc.). We concentrate on the control flow of a business process and on the external communication. According to the considered scenario, communication partners are not known to the business process itself. Communication is not supposed to be subject of negotiations between process owner and communication partner(s).

Our proposed operating guidelines are basically controllers that impose the property "termination in a valid end state" (we call this property *controllability*) on the business process. Controller synthesis has been widely studied [4, 11]. [8, 9] proposed synthesis of controllers for the same kind of models of business processes that are studied here. His goal, however, is deciding a property he called *usability* which basically expresses the existence of a communication partner (in our terms: a controller) such that process and partner terminate in valid end states. However, his kind of controllers is less general than the notions proposed here. The reason is that he only wants to decide *existence* of a controller while he does not want to characterize *all* controllers as we do. He cannot show the existence of most permissive controllers which is crucial for our approach, nor can his concepts be generalized to a distributed setting (communication with several independent partners). All these weaknesses have been resolved in [12]. Our considerations in this paper set up upon the latter article.

For characterizing *all* possible operating guidelines, we use annotations similar to [13]. While the annotations proposed there are expected to be modelled as part of a business process, we compute them automatically for characterizing the set of all possible operating guidelines. However, the concept of *process matching* proposed in [13] fits very well to our annotated operating guidelines.

3 Operating Guidelines Versus Public View

In this section, we point out the main differences between the public view and operating guidelines approaches.

The public view of a business process P describes P's behavior abstractly. It is assumed that it is easily possible for a partner process R to derive its supposed communication behavior from the public view of P. We propose instead that the owner of P should publish the communication behavior of R directly. This may seem to be a minor difference. There is, however, a number of arguments in favour of operating guidelines.

Since business processes are trade secrets, it is necessary that the information published about a process P hides a sufficient amount of information about

the details of P. Some approaches ([5], for instance) try to solve this problem by abstracting from critical actions by forming so called *virtual activities*. It is, however, dangerous to change, for instance, the branching structure of P's control flow. For a sound public view approach, it is thus necessary to provide a rigorous concept for the relation between public and private view, as well as a procedure to transform private into public views. We believe that these objectives are more difficult to achieve than it appears at first glance. [8] proposes a relation between the views that is based on the process algebraic notion of *simulation*. In addition, he proposes a preliminary, for practical purposes insufficient set of transformation rules. However, with this notion we only have that a partner that interacts properly with the public view, also interacts properly with the private view while the converse does not hold. That is, the public view may exclude partner behaviour that would work well with the private view. A notion that overcomes this problem would probably be close to the process algebraic concept of *bisimulation*. For this concept, however, it is common sense that it has only limited capabilities of hiding structure of the private view. We have not found a convincing notion of conformance between private and public view nor proposals for sufficient transformations from private to public views.

Our notion of operating guidelines has control theory as a solid theoretic basis. The guidelines can be computed fully automatically and directly from a business process P itself. The operating guidelines actually hide information about P since they only publish what a partner is required to do (which is the minimum information necessary for interacting with P) but no however abstract details about P's structure.

Furthermore, we believe that the operating guidelines approach actually requires less computational efforts than the public view approach: Consider a partner R that wants to access P. Assume that P has somehow generated and then published its public view to R. Nevertheless, R has to perform quite a number of steps on P's public view, including:

1. Extracting the communication sequences of P;
2. Deriving all valid communication sequences for potential partners;
3. Verifying whether R's actual communication matches a derived one.

We admit that these steps may be automatically performed. On the other hand, note that the first two steps are based on the public view and may thus be unprecise. They constitute sources of errors in the construction of the partner behaviour.

In our approach, P publishes operating guidelines to R, instead. So, only one step:

verifying whether R's actual communication matches the guidelines

is left for R, with no additional work for P, even.

Technically, this is much more elegant: The owner of P has access to full and exact internal information of P. Thus, the generated operating guidelines are more precise than the derived valid communication sequences from step 2., yet giving no secrets to the outside of P. The abstraction from internal (non-communication) activities comes for free. The efforts necessary for computing

the operating guidelines are comparable to the calculation of a public view for P. As we will show, it is in most cases even possible to achieve completeness, i.e. any violation of the operating guidelines results in the existence of reachable undesired states, e.g. required messages are not received or the corporate process is in a deadlock.

In practice, publishing operating guidelines can be seen as additional service for R.

Especially in the case of multiple partners (which do not interact) it is not clear what a public view should be: There are examples (for instance, Fig. 6) where a single public view for all partners is not sufficient to determine the behaviour of the communication partners. Even with separate public views for the partners, there may be interdependencies that cannot be resolved by any of the partners autonomously. If some partner process R has a choice, the public views of all other partners must be robust to its decision. An automated construction of such a set of public views appears to be rather hard, or restrictive.

With operating guidelines, the coordination between different partners is fully controlled by the owner of P. Necessary decisions are thus based on complete information about all ingredients.

To summarize: Operating guidelines meet our proposed high privacy requirements and reduce complexity. They have a solid theoretic basis, are supported by algorithms, and allow coordination between multiple partners.

4 Underlying Concepts

We model business processes as a special class of Petri nets, i.e. *workflow nets* [1], enriched with input and output places. The so called *workflow modules* base on [8]. From the control-theoretic perspective, these processes serve as plants.

Definition 1 (Petri net). *A Petri net N consists of two finite, disjoint sets P* (places) *and T* (transitions), *a* flow relation F ($F \subseteq (T \times P) \cup (P \times T)$), *and an* initial marking. *A* marking *is a mapping $m : P \to \mathbb{N}$.*

Places are depicted as circles, transitions as boxes, the flow relation as arrows, and markings as distributions of tokens on the places.

Definition 2 (Behavior of Petri nets). *Transition t is* enabled *in marking m if, for all places p, $[p,t] \in F$ implies $m(p) > 0$. Transition t may* fire *in marking m yielding a marking m' ($m \xrightarrow{t} m'$) if t is enabled in m, and for all p, $m'(p) = m(p) + W([t,p]) - W([p,t])$ where $W([x,y]) = 1$ if $[x,y] \in F$, and $W([x,y]) = 0$, else. With $R_N(m)$, we denote the set of markings that can be reached from m by firing any finite number of transitions.*

In this paper, we study only acyclic systems.

Definition 3 (Workflow module). *A Petri net N is a* workflow module M, *if*

- *P contains a distinguished* start place *α and a distinguished* end place *ω;*
- *P is disjoint union of sets P_M* (internal places)*, P_I* (input channels)*, and P_O* (output channels)*;*
- *$F \cap (P_O \times T) = \emptyset$, $F \cap (T \times P_I) = \emptyset$;*
- *$m_0(\alpha) = 1$, $m_0(p) = 0$, for all $p \neq \alpha$;*
- *F does not contain cycles (the transitive closure of F is irreflexive).*

In a workflow module, we call the marking m_f with $m_f(\omega) = 1$ and $m_f(p) = 0$ for $p \neq \omega$ the *final marking*. Fig. 1 shows a workflow module.

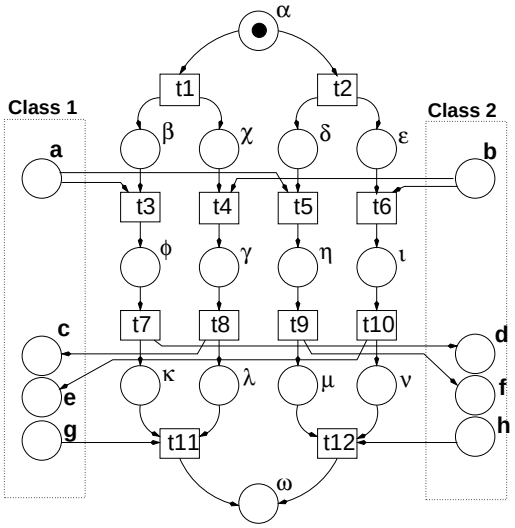

Fig. 1. A workflow module. Places with Greek names are internal, places a, b, g, h input channels, and c, d, e, f output channels. The dashed boxes represent an interface partition introduced for decentralized controllability.

Formally, the envisioned operating guidelines are controllers. We thus introduce a few control-theoretic concepts. In this paper, we model controllers as classical automata. Of course, it would be possible to model controllers as Petri nets, too. However, the constructions proposed here are essentially based on the concept of state. It is in fact possible to construct a Petri net out of a state space, either brute force resulting in a Petri net called *state machine*, or more sophisticated, through applying *region theory* [10, 3]. Since these constructions are well-understood, we decided to concentrate on the core construction thus introducing controllers as automata. In [12], we argue that we may, without loss of generality, restrict to automata which have tree shape (i.e., where a state is sufficient to recover the full history of reaching the state from the initial state). For this reason, we may identify a state with the sequence that reaches this state. The trees have a depth that is limited by a number l_M which can be computed

from the (acyclic) workflow module M. We omit an algorithm and assume l_M to be given. For an alphabet A, let $A*$ denote the set of finite words over A. Let $\underline{a}$ denote the multiset containing a once, and no other element.

Definition 4 (Tree controller). *Let M be a workflow module and $I \subseteq (P_I \cup P_O)$. A* tree controller *connected to I is an automaton with alphabet I, a set of states $Q \subseteq \{w | w \in I*, length(w) \leq l_M\}$ such that Q contains with q all prefixes of q, a transition function δ with $\delta(w,x) = \{wa\}$ if $wa \in Q$ and there is an a with $x = \underline{a}$, and $\delta(w,x) = \emptyset$, else, and λ (the empty word) as the initial state.*

Closure under prefixes is necessary and sufficient for reachability of all states from the initial state.

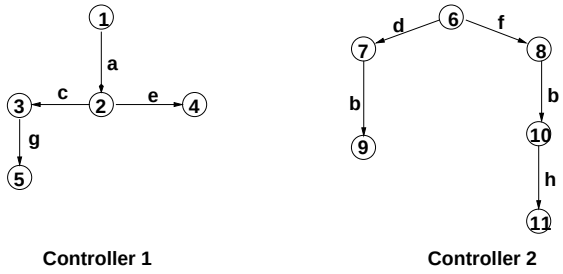

Fig. 2. A feasible set of controllers for the workflow module in Fig. 1.

Fig. 2 shows two tree controllers. A move in the controller describes an interaction with the workflow module, coded in the alphabet. The multiset involved in the transition stands for messages to be received or to be sent. Of course, only messages present on the output channels can be received. The following definition describes the interplay between a workflow module and a feasible set of controllers.

Definition 5 (Feasibility). *A set of controllers $\{C_1, \ldots, C_n\}$ is* feasible *for a workflow module M iff their alphabets are disjoint, and the union of their alphabets equals $P_I \cup P_O$.*

The set of the two partners in Fig. 2 is feasible for the workflow module in Fig. 1.

Definition 6 (Composed system). *Let M be a workflow module and let $\{C_1, \ldots, C_n\}$ be a feasible set of controllers for M. Then the* composed system *is a transition system with $R_N(m_0) \times Q_1 \times \cdots \times Q_n$ as set of states, and edges*

- *from $[m, q_1, \ldots, q_n]$ to $[m', q_1, \ldots, q_n]$ if there is a transition t in M with $m \xrightarrow{t} m'$;*

- *from* $[m, q_1, \ldots, q_i, \ldots, q_n]$ *to* $[m', q_1, \ldots, q_i', \ldots, q_n]$ *if there is a multiset* B *such that in* C_i $q_i' \in \delta_i(q_i, B)$, *for all* $p \in P_O$, $m(p) \geq B(p)$ *and* $m'(p) = m(p) - B(p)$, *for all* $p \in P_i$, $m'(p) = m(p) + B(p)$, *and for all* $p \in P_M$, $m'(p) = m(p)$.

The initial state is $[m_0, q_{0_1}, \ldots, q_{0_n}]$.

Fig. 3 depicts the system composed of the workflow module in Fig. 1 and the controllers in Fig. 2.

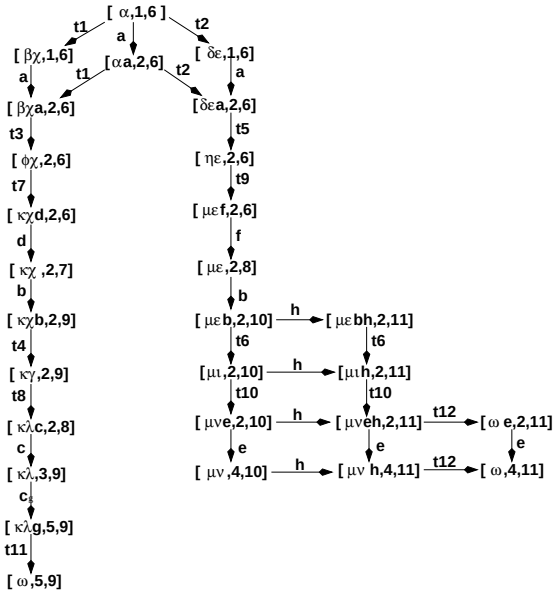

Fig. 3. System composed of the workflow module in Fig. 1 and the controllers in Fig. 2.

The goal of control is proper termination of the workflow module. This means that the workflow module is in marking m_f. In this marking, all channel places are empty.

Definition 7 (Strategy). *Let* M *be a workflow module. A feasible set of controllers is a* strategy *for* M *if, in the composed system, from every state reachable from the initial state, a state is reachable whose first component is* m_f.

In a tree controller C connected to I, states encode the full record of past interaction with the workflow module M. We assume that the behavior of M is fully unobservable for C, except the output channels in $I \cap P_O$. Nevertheless, knowing C and M, it is possible to deduce, in which markings M can possibly be while C is in a given state q. We formalize this knowledge that a state of C represents about the marking of M through a mapping K.

Definition 8 (Knowledge of the controller). *Let C be a controller connected to I and q a state of C. Let S be the set of states of the system composed of M, C, and noise$((P_I \cup P_O) \setminus I)$. Then $K(q) = \{m \mid \exists q' : [m, q, q'] \in S\}$ is called the* knowledge *of controller C in state q.*

K can be computed from C and M through building a composed system. The details are omitted. Fig. 4 shows a module and a controller for that module with its knowledge assigned to the nodes.

As a last pre-requisite, we classify deadlocks of the workflow module M.

Definition 9 (Deadlocks). *Let M be a workflow module. A* deadlock *of M is a marking that does not enable any transition. A deadlock m is* internal *if $m \neq m_f$, for every transition there is a $p \in P_M$ with $[p, t] \in F$ and $m(p) = 0$, and all output channels are empty. A deadlock is* external *if it is not internal nor equal to m_f.*

Internal deadlocks are permanent, i.e. no controller can navigate the module out of an internal deadlock. External deadlocks can be left by appropriate controller actions (producing tokens on unmarked inout channels).

5 Operating Guidelines for Single Partner

In this section, we study the generation of guidelines in the case that a business process has just one communication partner. In Sec. 5.1, we present results about most permissive strategies. The most permissive strategy turns out to form the most permissive operating guidelines. In Sec. 5.2, we show how annotations can be attached to the most permissive strategy such that *every* strategy can be derived from just the most permissive guideline and the annotations.

5.1 Most Permissive Guideline

In this section, we study strategies that consist of a single controller, connected to $P_I \cup P_O$. For simpler notation, we identify controller C with the set $\{C\}$.

Definition 10 (Most permissive strategy). *A tree controller C with set of states Q is a* most permissive strategy *for M if it is a strategy and every tree controller that is strategy for M has a set of states included in Q.*

Inclusion of the sets of states means that every strategy can be seen as a subtree of the behavior of the most permissive strategy.

Our approach to centralized controllability is based on the following characterization of centralized strategies.

Theorem 1 ([12]). *A tree controller C connected to $P_I \cup P_O$ is a strategy for workflow module M if and only if it has a nonempty set of states and the following two conditions hold for all $q \in Q$:*

- *$K(q)$ does not contain internal deadlocks;*

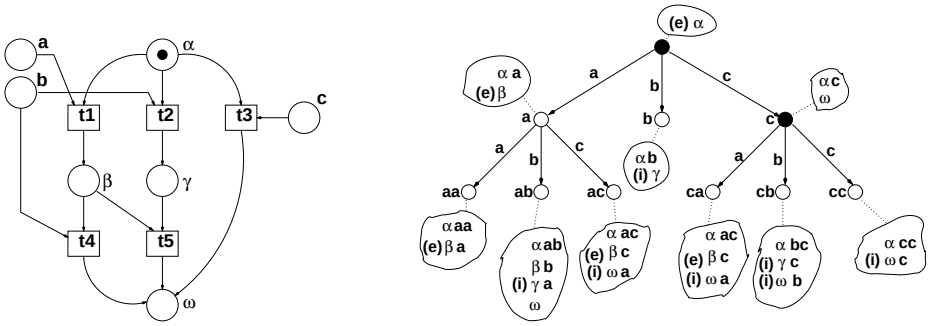

Fig. 4. Construction of a most permissive strategy (right) for the workflow module depicted left. Internal and external deadlocks are marked (i) and (e), resp.

- *for every external deadlock* $m \in K(q)$ *the system composed of* M *and* C *has a transition starting in* $[m, q]$.

With this characterization, we can construct a most permissive centralized strategy for M. Starting with $noise(P_I \cup P_O)$, we remove all states that may lead to internal deadlocks. Then, we repeatedly remove states where external deadlocks have no successors. Removing q requires removing all qw, too, since those states would become unreachable from the initial state. More details about the algorithm can be found in [12].

Fig. 4 shows an example for the application of this algorithm. Starting with a complete automaton (K-values are attached), we first remove all states q where $K(q)$ contains internal deadlocks: states ab, ac, b, ca, cb, and cc. Since b has an internal deadlock, we did not depict its successors. State aa is removed since it contains an external deadlock with no continuation. State a contains an external deadlock, too. Before removing aa, transition a is possible in β. However, after removal of aa, this external deadlock has no possible transition, hence state a needs to be removed, too. State c has no external deadlocks, and the initial state has an external deadlock with a remaining possible transition (c). Hence, the computed controller consists of the initial state, state c, and the transition the initial state to state c.

5.2 Complete Operating Guidelines

The most permissive strategy is one possible scenario for communicating with a process but not the only one. For example, some customer may always want to buy coffee from a vending machine and is not interested in the control flow for buying tea. Since the strategy computed in the previous subsection is most permissive, we know that *every* strategy, that is every possible behaviour of the communication partner, is a subtree of the most permissive strategy. Unfortunately, not every removal of subtrees from the most permissive strategy results in a working strategy. Remove, for instance, all states q where $K(q)$ contains ω, and you have a malfunctioning controller.

In this subsection, we add annotations to the states of the most permissive strategy. These annotations can be used to build exactly those subtrees of the most permissive strategy (i.e. specialized communication behaviour) that are guaranteed to work. So, the most permissive strategy and its annotations can be published as most permissive operating guidelines, containing all valid communication sequences with the process.

An *annotation* $A(q)$ to state q of a controller C is a boolean formula that has elements of the alphabet of C as propositions. Informally, the formula states which outgoing transitions must be present from q in order to arrive at a valid strategy.

Formally, a controller C *agrees* on annotation $A(q)$ for one of its states q, iff $A(q)$ evaluates to *true* for an assignment where *true* is assigned to x if a transition with x is present from q, and *false*, otherwise.

We can attach annotations to the most permissive strategy S such that the set of *all* strategies is exactly the set of subtrees of S that agree on the annotations to the states still present. In other words, we *may* remove a transition (and the subtree below it) from a state q as long as the remaining transitions in q still satisfy $A(q)$.

The key to the announced annotations is Thm. 1. A subtree of the most permissive strategy is a strategy if and only if the conditions of Thm. 1 are satisfied. The condition concerning internal deadlocks is satisfied for all subtrees since the most permissive strategy does not contain states q with internal deadlocks in $K(q)$. Thus, if a subtree is not a strategy then it violates the condition that for every external deadlock $d \in K(q)$ there must be a transition that is possible in d.

Consequently, we build $A(q)$ as a conjunction of subformulas with one subformula for each external deadlock $d \in K(q)$. The subformula corresponding to external deadlock d is the disjunction of all transitions possible in d. From this construction it is easy to see using Thm. 1:

Corollary 1. *When annotations to states of the most permissive strategy are as described above then the set of* all *strategies is exactly the set of subtrees of S that agree on the annotations to the states still present.*

Fig. 5 shows an example for an annotated controller and the resulting set of strategies. We believe that the concept of annotated automata is easy to use. Observe that the most permissive strategy itself agrees on all annotations since it satisfies Thm. 1.

If the controller (a) depicted in Fig. 5 with its annotations were published as operating guidelines for a process P, then exactly these 7 controllers in Fig. 5 represent valid partners for P.

6 Operating Guidelines for Multiple Partners

In this section, we consider business process models that communicate with several partners. That is, we need to calculate several operating guidelines, one for

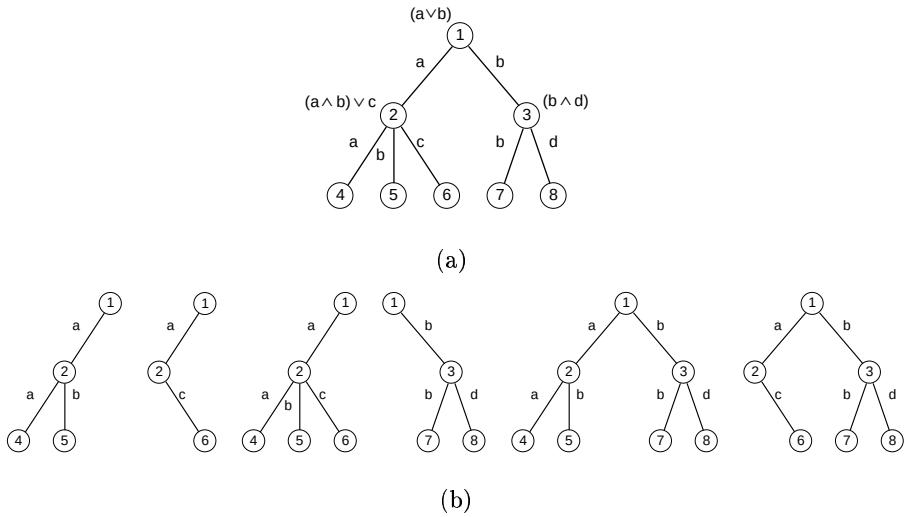

Fig. 5. An annotated controller (a) and all sub-controllers (b) (with annotations left out). All 7 controllers agree on all annotations, and no other.

each communication partner. In the control perspective, this problem is known as decentralized control.

6.1 Decentralized Controllability

Operating guidelines for multiple partners correspond to decentralized controllers. We study decentralized controllability always with respect to a given partition.

Definition 11 (Decentralized strategy). *A set of controllers* $\{C_1, \ldots, C_n\}$ *is a decentralized strategy w.r.t.* $U = \{I_1, \ldots, I_n\}$ *if, for all* i, C_i *is connected to* I_i *and* $\{C_1, \ldots, C_n\}$ *is a strategy.*

Our definitions imply that controllers involved in a decentralized strategy communicate with the module but not with each other. If we allowed controllers to communicate with each other, they would be able to implement every centralized strategy, so decentralized controllability would not be worth being studied. On the other hand, situations where different parties communicate with a process without communicating with each other appears to be typical in the world of business processes.

If a set of controllers forms a decentralized strategy then their parallel composition forms a central strategy. That is, we compute decentralized strategies by computing centralized strategies where actions that belong to different classes of U commute. Such strategies can then be partitioned to decentralized strategies. We skip the details. It is, however, necessary to point out that the calculation

may lead to more than one decentralized strategy, and there is not necessarily a unique most permissive strategy.

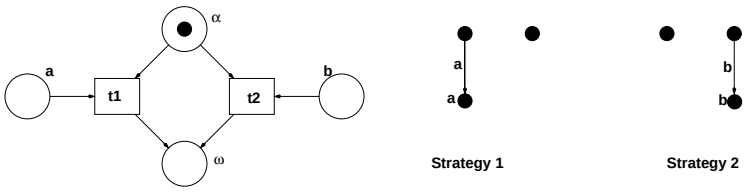

Fig. 6. A workflow module and two decentralized strategies for interface partition $\{\{a\},\{b\}\}$. The depicted strategies are the only decentralized strategies and none of them is more permissive than the other. The two communication partners cannot determine their own behaviour without negotiating with the module or the other partner about which of the strategies to use.

6.2 Decentralized Operating Guidelines

Each decentralized strategy forms a valid way to communicate with a process. However, it is impossible for independent partners to agree on the same strategy from all possible ones. So, the process itself must choose and publish only one such strategy to its partners (resp. their corresponding parts of that strategy). In the example in Fig. 6, the process owner must choose to publish either strategy 1 *or* strategy 2.

In this subsection, we develop annotations to an already chosen decentralized strategy (i.e. annotations for each controller in the strategy). It will turn out that, in analogy to decentralized strategies, there is no most permissive annotation to a decentralized strategy, but a set of incomparable annotations. Here also, the process owner must choose.

Let $\{C_1,\ldots,C_n\}$ be a decentralized strategy for process P. Then, the parallel composition $C = C_1 \parallel \ldots \parallel C_n$ is obviously a centralized controller for P. So, every state q of C relates to exactly one state q_i of C_i, for all $i \in \{1,\ldots,n\}$ [1]. This relation determines the annotation of a controller C_i: Informally, the annotations $A(q_1),\ldots,A(q_n)$ must ensure that the corresponding annotation $A(q)$, for $q = (q_1,\ldots,q_n)$, holds.

Formally, we say that a set of controllers $\{C_1,\ldots,C_n\}$ with states q_i of C_i *agrees on annotation* $A(q_1,\ldots,q_n)$, iff $A(q_1) \wedge \ldots \wedge A(q_n)$ implies $A(q_1,\ldots,q_n)$. Canonically, $\{C_1,\ldots,C_n\}$ agrees on the composed system C, iff $\{C_1,\ldots,C_n\}$ agrees on *all* annotations of C.

So, construction of annotations means solving a system of implications. Unfortunately, there may be multiple solutions of such a system and there is no most permissive one. Moreover, solving such a system is hard and turns out to

[1] Actually, we write q as a tuple $(q_1,\ldots,q_n)$ of its corresponding states q_i of C_i.

be rather unpractical. Nevertheless, we have found an efficient algorithm that computes annotations that are sufficient for our condition, but not complete. We skip the details here and only give an example:

Let controllers C_1 and C_2, with states q_1 and q_2, respectively, be a decentralized strategy. Let C be their parallel composition. Thus, C is a centralized strategy and has a state $q = (q_1, q_2)$. Assume $(a \wedge c) \vee (b \wedge d)$ is the annotation of state q in C and a partition, such that actions a, b belong to C_1 and actions c, d belong to C_2.

There are two minimal solutions, A_1 and A_2, to make the implication $A_i \Rightarrow (a \wedge c) \vee (b \wedge d)$ true: $A_1(q_1) = a,\ A_1(q_2) = c$ and $A_2(q_1) = b,\ A_2(q_2) = d$. In this example, our approximation algorithm would not return a minimal solution, but computes A_3, with $A_3(q_1) = (a \wedge b),\ A_3(q_2) = (c \wedge d)$. Note, that

(i) A_3 is a valid annotation, i.e. $A_3(q_1) \wedge A_3(q_2) \Rightarrow (a \wedge c) \vee (b \wedge d)$, and

(ii) A_3 is the intersection of A_1 and A_2, i.e. $A_3(q_1) \wedge A_3(q_2) \equiv A_1(q_1) \wedge A_1(q_2) \wedge A_2(q_1) \wedge A_2(q_2)$.

6.3 Local Controllability

Consider the following scenario: A company publishes a business process (or a public view thereof), we are one of the parties that wants to connect to that process via a given subset of the interface but do not know who else is connecting, nor how. Our task is to build a controller that, ideally, works properly with every set of controllers connected to the remaining parts of the interface. In this generality, our task cannot be accomplished, since we cannot prevent that a defecting controller denies to send a required message or sends too many messages of some kind. So it is at most possible to build a controller that works properly with arbitrary "non-defecting" controllers. In this section, we define *cooperative* controllers as a formalization for non-defection, and show that every feasible set of cooperative controllers forms a decentralized strategy. Unfortunately, cooperative controllers do not necessarily exist. For instance, in Fig. 6, assuming the interface partition $\{\{a\}, \{b\}\}$, cooperative controllers do not exist since it is impossible to choose among the two options depicted there without requiring that the other party picks the corresponding counterpart. We therefore define *local controllability* as the existence of cooperative strategies for all classes in a given interface partition. Locally controllable modules are therefore those whose publication is sufficient for enabling all communication partners to design their communication with the module independently of each other.

Consider a workflow module M and a class I_i of a given partition $\{I_1, \ldots, I_n\}$ of $P_I \cup P_O$. We establish two quite plausible requirements for calling a controller C_i connected to I_i cooperative: First, C_i should behave like a central controller assuming that the channels $(P_I \cup P_O) \setminus I_i$ were not present, and second, C_i should help in resolving external deadlocks, if it can. We formalize the first requirement by introducing the I_i-view of M.

Definition 12 (X-view of M). *Let M be a workflow module and $X \subseteq P_I \cup P_O$. Then the X-view of M, M_X is the workflow module obtained from$M$ by removing all places in $(P_I \cup P_O) \setminus X$ and all arcs connected to them.*

Definition 13 (Cooperative controller). *Let M be a module and $\{I_1, \ldots, I_n\}$ a partition of $P_I \cup P_O$. Then controller C_i connected to I_i is* cooperative *if it is a central strategy for M_{I_i}, and, considering M itself, for every state q of C where $K(q)$ contains an external deadlock m and an action $a \in I_i$ is possible in m, qa is a state of C, too.*

Theorem 2. *Every feasible set of cooperative controllers forms a decentralized strategy for M.*

Note, that this statement is only relevant if cooperative controllers exist for all classes of an interface partition.

This result justifies the definition of local controllability.

Definition 14. *A workflow module M is* locally controllable *w.r.t. a partition $\{I_1, \ldots, I_n\}$ of $P_I \cup P_O$ if, for all i, there exists a cooperative strategy for I_i.*

Given M and I_i, existence of a cooperative strategy can be decided similar to the centralized case.

Corollary 2. *If there is a cooperative controller for I_i then there is also a most permissive one.*

For the workflow module in Fig. 1, the controller depicted left in Fig.2 is cooperative while the one depicted right is not. In the latter one, action b is possible in the external deadlock $\beta\chi \in K(6)$ but not executed in 6. If both controllers behaved like the one depicted right (observe that the module is symmetric w.r.t. the interface partition) then the composed system would deadlock in $\beta\chi$. Since cooperative controllers exist (like the one depicted left), the module in Fig. 1 is locally controllable. The module in Fig. 6 is not locally controllable w.r.t. $\{\{a\}, \{b\}\}$.

Fortunately, cooperative strategies behave very similar to centralized strategies. In particular, whether or not a strategy is cooperative does not depend on the remaining communication partners. It is thus possible to characterize *all* cooperative strategies for a partner by an annotated most permissive cooperative strategy. The approach is so close to the one for centralized control that we decided to skip the details.

7 Conclusion

We proposed a technology for generating publishable operating guidelines for a business process that wants to provide a service according to the service oriented architecture. The operating guidelines are basically tree automata. It is, however, possible to compute condensed representations using standard automata theory (construction of minimal automata) and Petri net region theory (construction of operating guidelines that exhibit internal concurrency). We believe that it is easier to match an operating guideline with a prospective business partner than

to match public views. All constructions and matchings can be performed on the private processes without given away trade secrets.

In our approach, we disregarded aspects of business processes such as data flow and data integrity, and roles. We are, however, convinced that our approach can be extended to meet other requirements for business processes, too. We provided an approach on a solid theoretic basis. We are able to cope with single or multiple communication partners and pointed out the potential problems in the case of multiple partners.

In this paper, we studied acyclic workflow modules. Current research activities concerns the extension of our approach to systems containing cycles. At the moment, we are able to compute strategies that are practically satisfying but the strategies are not necessarily most permissive in the theoretical sense.

References

1. W. M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. W.M.P. van der Aalst and M. Weske. The P2P approach to interorganizational workflows. In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01)*, volume 2068 of *Lecture Notes in Computer Science*, pages 140–156. Springer-Verlag, Berlin, 2001.
3. E. Badouel and P. Darondeau. Theory of regions. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 529–586, 1998.
4. C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.
5. I. Chebbi, S. Tata, and S. Dustdar. Cooperation policies for inter-organizational workflows. In *Teamware: supporting scalable virtual teams in multi-organizational settings, The 2005 International Symposium on Applications and the Internet*, 2005.
6. K. Gottschalk. *Web Services architecture overview*. IBM developerWorks, Whitepaper, September 2000. http://ibm.com/developerWorks/web/library/w-ovr/.
7. F. Leymann, D. Roller, and M. Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2), 2002.
8. A. Martens. *Verteilte Geschäftsprozesse - Modellierung und Verifikation mit Hilfe von Web Services*. PhD thesis, Institut fr Informatik, Humboldt-Universität zu Berlin, 2004. WiKu-Verlag, Stuttgart.
9. A. Martens. Analyzing web service based business processes. In *FASE*, volume 3442 of *Lecture Notes in Computer Science*, 2005.
10. M. Nielsen, G. Rozenberg, and P.S. Thiagarajan. Elementary transition systems. *Theoretical Computer Science*, 96:3–33, 1992.
11. P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization*, 25(1):206–230, 1987.
12. K. Schmidt. Controlability of business processes. Techn. Report 180, Humboldt-Universitt zu Berlin, 2004.
13. A. Wombacher, P. Fankhauser, B. Mahleko, and E. Neuhold. Matchmaking for business processes based on choreographies. *International Journal of Web Services*, 1(4):14–32, 2004.