Processes, Roles and Their Interactions

Johannes Reich, johannes.reich@sophoscape.de

Taking an interaction network oriented perspective in informatics raises the challenge to describe deterministic finite systems which take part in networks of nondeterministic interactions. The traditional approach to describe processes as stepwise executable activities which are not based on the ordinarily nondeterministic interaction shows strong centralization tendencies. As suggested in this article, viewing processes and their interactions as complementary can circumvent these centralization tendencies.

The description of both, processes and their interactions is based on the same building blocks, namely finite input output automata (or transducers). Processes are viewed as finite systems that take part in multiple, ordinarily nondeterministic interactions. The interactions between processes are described as protocols.

The effects of communication between processes as well as the necessary coordination of different interactions within a processes are both based on the restriction of the transition relation of product automata. The channel based outer coupling represents the causal relation between the output and the input of different systems. The coordination condition based inner coupling represents the causal relation between the input and output of a single system.

All steps are illustrated with the example of a network of resource administration processes which is supposed to provide requesting user processes exclusive access to a single resource.

1 Introduction

We are living in an open world of interaction relations. As an example, Fig. 1 sketches a cutout of a network of business relations between a buyer, a seller, its stock, a post and a bank. Intuitively, these networks have two key features. First, they represent peer relations in the sense that none of the participants is in total control of all the other participants. The participant's actions are, in general, not determined by their interactions. Second, they are open in the sense that we will never be able to describe them completely. Each participant will also interact in many other roles in other networks.

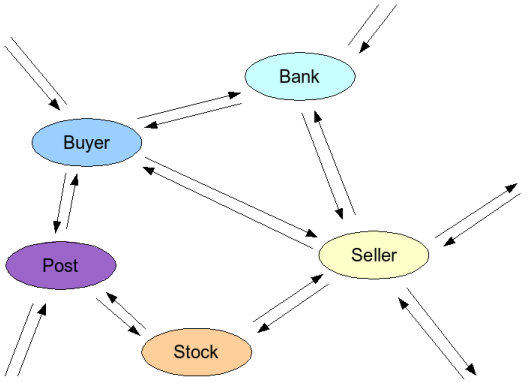

Figure 1: A cutout of an open business network.

J. Reich and B. Finkbeiner (Eds.): 2nd Int'l Workshop
on Interactions, Games and Protocols 2012 (IWIGP2012)
EPTCS 78, 2012, pp. 24–38, doi:10.4204/EPTCS.78.3

In informatics, the interacting systems are often called processes. So, taking a process or interaction network oriented perspective in informatics raises the challenge to describe deterministic finite systems which take part in networks of nondeterministic interactions. The traditional approach to model processes as stepwise executable activities leads to strong centralization tendencies (see [14, 2]). The contribution of this article is a formalism based on finite input output automata to describe processes and their interactions in a complementary way avoiding these centralization tendencies. We present two coupling mechanisms for system parts: the outer or interaction coupling, described by protocols and the inner or action coupling, leading to processes. The duality of system specifications from a process perspective and from an interaction perspective actually has been recognized as early as 1976 by Chris A. Vissers [16].

The rest of this article is structured as follows. Next, a concrete example of such an open interaction network is introduced. In section 2 the technical building blocks, the finite input output automata, are presented. Section 3 and 4 define the outer and inner coupling mechanisms. In section 5 related work is presented and in section 6 the results are discussed.

1.1 Preliminaries

Throughout this article, elements and functions are denoted by small letters, sets and relations by large letters and mathematical structures by large calligraphic letters. The components of a structure may be denoted by the structure's symbol or - in case of enumerated structures - index as subscript. The subscript is dropped if it is clear to which structure a component belongs.

For any character set or alphabet A, $A^\varepsilon := A \cup \{\varepsilon\}$ with ε is the empty character. For state value sets Q, $Q^\varepsilon := Q \cup \{\varepsilon\}$ with ε is the undefined value. If either a character or state value set $A = A_1 \times \ldots \times A_n$ is a Cartesian product then $A^\varepsilon = A_1^\varepsilon \times \ldots \times A_n^\varepsilon$. Additionally, a state vector $(p_1, \ldots, p_n)$ where p_k belongs to structure $\mathscr{A}_k$ is written as $\vec{p}$ and the change of this vector in a position k from p to q is written as $\vec{p}\left[\begin{smallmatrix} q \\ p \end{smallmatrix}, k\right]$. An n-dimensional vector of characters with the k-th component v and the rest ε is written as $\vec{\varepsilon}[v,k] = (\varepsilon_1, \ldots, \varepsilon_{k-1}, v, \varepsilon_{k+1}, \ldots, \varepsilon_n)$.

In the main article, I will talk about processes and their parts in a relational sense, which means that I actually talk about them at the level of their specification. The relation to the functional level of (finite) systems is shown in the appendix.

The part of a process that is directly related to an interaction is called a *role*.

1.2 Example

In Fig. 2 the open network of resource administration processes is shown. In its core, it consists of two different interactions. The first interaction is about the classical problem of mutual exclusion (e.g. [6]). Some process requests exclusive access to a resource and a resource administration process admits it. The second interaction concerns the coordination of the different resource administration processes for administering the exclusive access to the resource, where I chose the token ring protocol.

1.2.1 First Interaction: Protocol of Mutual Exclusion

The first interaction with its two roles $\mathscr{U}$ and $\mathscr{C}$ is displayed in Fig. 3. $\mathscr{U}$ has the state values $remn_{\mathscr{U}}$ (for remainder region), $try_{\mathscr{U}}$ (for trying region), $crit_{\mathscr{U}}$ (for critical region) and $exit_{\mathscr{U}}$ (for exit region), while $\mathscr{C}$ has the state values $remn_{\mathscr{C}}$, $try_{\mathscr{C}}$, $crit_{\mathscr{C}}$ and $exit_{\mathscr{C}}$.

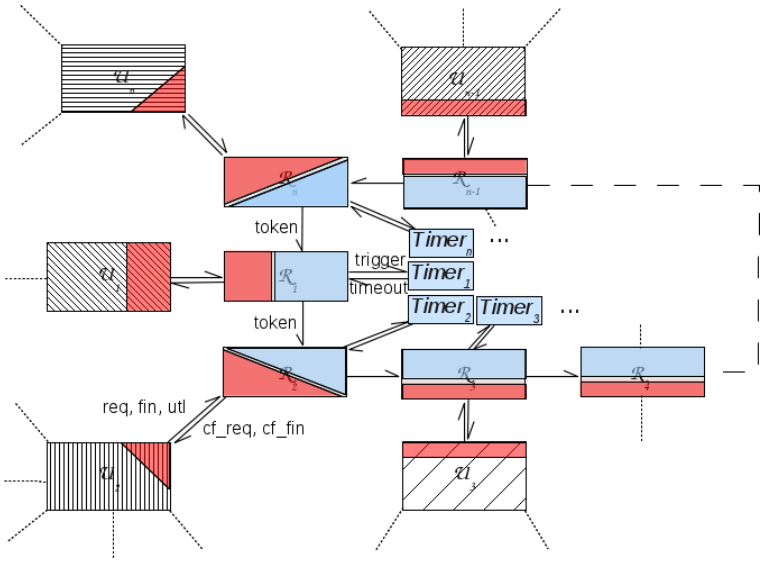

Figure 2: A network of interacting processes where n resource administrator processes $\mathscr{R}_i$ interact in a ring topology with timers and resource demanding processes $\mathscr{U}_i$, which themselves interact with an unknown number of other processes. The different colors mark the "roles" of the processes, that is those process parts that are directly related to the interactions. Red marks the protocol of mutual exclusion, blue marks the token ring protocol.

Initially, $\mathscr{U}$ and $\mathscr{C}$ are in the state $(remn_{\mathscr{U}}, remn_{\mathscr{C}})$. If $\mathscr{U}$ requests the resource, it notifies $\mathscr{C}$ with a *req*-character (for request) and transits into its $try_{\mathscr{U}}$ state. If the resource is free for use, $\mathscr{C}$ confirms the request with a *cf-req*-character, transits into its $crit_{\mathscr{C}}$ state and now provides the resource for usage. $\mathscr{U}$ receives the *cf-req*-character, transits to its $crit_{\mathscr{U}}$ state and now could use the resource (omitted). $\mathscr{U}$ releases the resource with sending a fin-character (for finalize), which is confirmed by $\mathscr{C}$ with sending a *cf-fin*-character back to $\mathscr{C}$, so that at the end of the cycle, both, $\mathscr{U}$ as well as $\mathscr{C}$ are again in their *remn* state.

1.2.2 Second Interaction: Token Ring Protocol

The token ring protocol is illustrated in Fig. 4 and consists of a quadruple interaction between a process, its two neighbor processes, and a timer for each process. Each $\mathscr{R}_i$ can take one out of three state values *abst* (for absent), *avlb* (for available) and *interm* (for intermediate). Each timer can take either *wait* or *triggered*. The protocol has to ensure that one $\mathscr{R}_i$ is not in *abst* where it is allowed to provide access to the resource. Initially, $\mathscr{R}_1$ is initialized with value *avlb* and $timer_1$ with *triggered* while all other processes are initialized with *abst* and all other timers with *wait*.

$\mathscr{R}_i$ waits in the state *abst* for reception of the token from $\mathscr{R}_{i-1}$ (or in case of $i = 1$, $\mathscr{R}_1$ waits for $\mathscr{R}_n$) to transit into state *avlb* in case of its reception and to trigger its timer. It rests there until it receives the timeout which results in a transition to *interm*. Finally, $\mathscr{R}_i$ hands over the token to $\mathscr{R}_{i+1}$ (or in case of $i = n$, $\mathscr{R}_n$ hands over to $\mathscr{R}_1$) and transits back to *abst*.

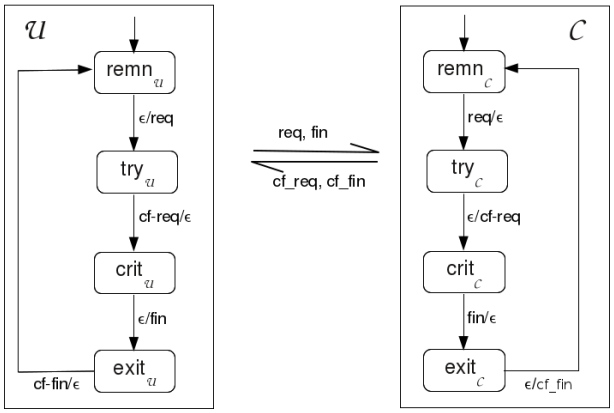

Figure 3: State diagram of the protocol of mutual exclusion where a resource administrator process in its role b$\mathscr{C}$ manages the exclusive usage of a single resource for the user process in its role $\mathscr{U}$.

2 The Building Blocks: Finite Input/Output Automata

Finite input output automata will later be used to describe both, processes and their interactions.

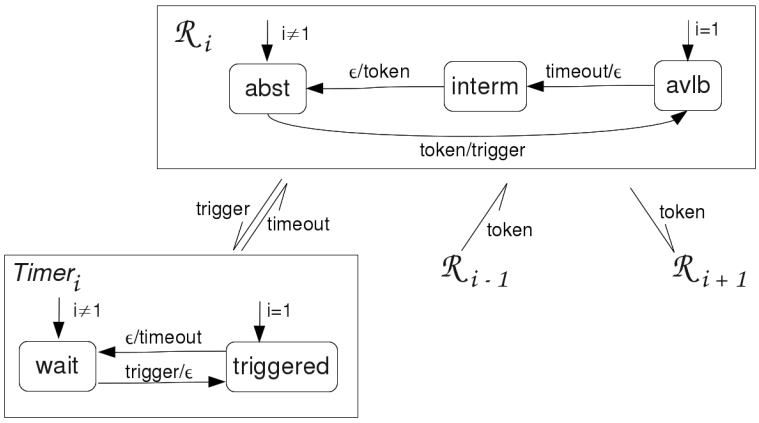

Figure 4: State diagram of the token ring protocol for managing exclusive access to a single resource by a resource administrator process $\mathscr{R}_i$.

Definition 1: A *nondeterministic finite I/O automaton (NFIOA)* is a tuple $\mathscr{A} = (Q, I, O, q_0, Acc, \Delta)$. Q is the non-empty finite set of state values, I and O are the finite, possibly empty input and output alphabets. The elements of Q, I or O are usually assumed to be vectors. For each element of I and O at most one component is unequal to ε. q_0 is the initial state value, Acc is the acceptance component and $\Delta \subseteq Q \times Q \times I^{\varepsilon} \times O^{\varepsilon}$ is the transition relation.

The acceptance component Acc represents the information needed to express the acceptance condition. It depends on the computational model of "acceptance". For a finite computation Acc is a set of final state values. For an infinite computation a representative acceptance condition is to accept all runs in which the set of infinitely often occurring state values is an element of the acceptance component (Muller-acceptance), that is $Acc \subseteq 2^Q$ [3].

Transitions with an ε-character as input are called *spontaneous*. In case that for each $(p,i) \in Q \times I^{\varepsilon}$ there is at most one transition $(p,q,i,o) \in \Delta$ then Δ specifies a function $\delta : Q \times I^{\varepsilon} \to Q \times O^{\varepsilon}$ with $(q,o) = \delta(p,i)$. If also no spontaneous transition exists, we have a deterministic automaton or *DFIOA* (a Mealy automaton [8]).

2.1 Product Automata

The precondition of any coupling of NFIOAs is to view them as a single automaton where all to-be-coupled NFIOAs are viewed together as a product automaton, stepping synchronously and where all acceptance conditions are to be fulfilled simultaneously.

Definition 2: The *weakly synchronized product* of a set of n NFIOAs $\mathscr{A}_k$ is defined by NFIOA $\mathscr{B} = (Q,I,O,\vec{q}_0,Acc,\Delta)$, with $Q_{\mathscr{B}} = \times Q_k$, $I_{\mathscr{B}} = \times I_k$, $O_{\mathscr{B}} = \times O_k$, $\vec{q}_{0\mathscr{B}} = (q_{01},\ldots,q_{0n})$, the common acceptance component represents the logical conjunction of the individual components, symbolized as $Acc_{\mathscr{B}} = \bigwedge Acc_k$, $\Delta_{\mathscr{B}} := \{(\vec{p},\vec{q},\vec{i},\vec{o})|$ the components of $\vec{p}$ are reachable states of the $\mathscr{A}_i$ and $\mathscr{A}_k$ provides a transition (p_k,q_k,i_k,o_k) with $\vec{q} = \vec{p}\left[\frac{q_k}{p_k},k\right]$ and $\vec{i} = \varepsilon[i_k,k]$ and $\vec{o} = \varepsilon[o_k,k]$ $\}$. I also write $\mathscr{B} = \bigotimes_{i=1}^{n} \mathscr{A}_i$.

As each step of the product automaton is achieved by only one component automaton, the input and output components of the product automaton again differ from epsilon at most in one component.

3 Channel Based Restriction and Protocols

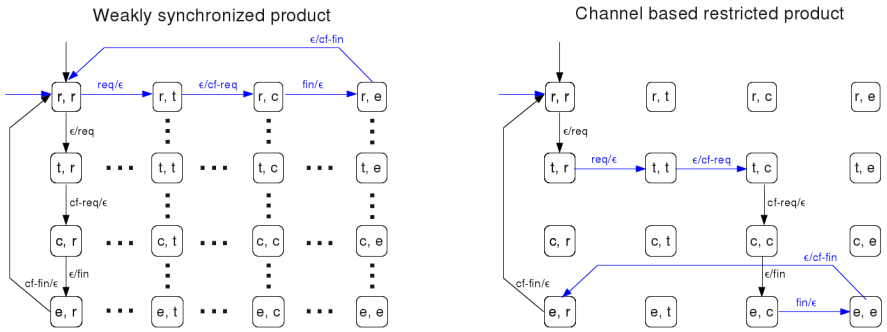

Figure 5: Left, you see the state diagram of the weakly synchronized product of both roles of the protocol of mutual exclusion. Right, you see the channel based restricted product. The names of the state values have been abbreviated.

Connecting two systems with a channel imposes certain restrictions on the transition relation of their weakly synchronized product automaton as it requires the receiving system to transit only after the sending system and with the input of the sending system's output. It thereby transforms the weakly synchronized product automaton into a stronger synchronized product automaton with a restricted transition relation.

As an example, Fig. 5 shows the state diagram of both, the weakly synchronized product and the channel based restricted product of the protocol of mutual exclusion.

Definition 3: Be $\mathscr{T}$ an NFIOA where O_k is the k-th component of the output alphabet and I_l is the l-th component of the input alphabet and $O_k \subseteq I_l$. The pair (k,l) is called a *channel* if it parametrizes

the *channel based restriction (cbr) operator* in the following way leading to the channel based restricted automaton $\mathscr{P} = cbr_{(k,l)}(\mathscr{T})$.

1. $Q_{\mathscr{P}} = Q_{\mathscr{T}}, I_{\mathscr{P}} = I_{\mathscr{T}}, O_{\mathscr{P}} = O_{\mathscr{T}}, Acc_{\mathscr{P}} = Acc_{\mathscr{T}}, q_{0\mathscr{P}} = q_{0\mathscr{T}}$.
2. Be $p \in Q_{\mathscr{P}}$ a reachable state of $\mathscr{P}$ (and thereby also of $\mathscr{T}$) which was reached by a transition with an output component $\varepsilon[o,k] \in O_{\mathscr{P}}$ (i.e. $o \in O_k$). Then all transitions of $\Delta_{\mathscr{T}}$ which start from p and have $\varepsilon[o,l] \in I_{\mathscr{P}}$ (i.e. $o \in I_l$) as input character are also elements of $\Delta_{\mathscr{P}}$. I call the transition providing the character *sending* transition, the ensuing transitions *receiving* transitions and such a state p *excited*.
3. Be $p \in Q_{\mathscr{P}}$ either the initial state or a reachable state of $\mathscr{P}$ which was reached by a transition with a character $o \in O_{\mathscr{P}}$ with $o_k = \varepsilon$ that is no output on the channel. Then all spontaneous transitions of $\mathscr{T}$ which start from p and all transitions of $\mathscr{T}$ which have an input not associated with the channel are elements of $\Delta_{\mathscr{P}}$. I call such a state p *relaxed*[1].

Input and output components which are not associated with a channel are called *open*. For an illustration of the resulting transition classes, see Tab. 1

Type of start state	Input	Output	Type of target state
relaxed	ε	ε	relaxed
relaxed	ε	$\varepsilon[\beta,k]$	excited
relaxed	ε	$\varepsilon[\beta,i\neq k]$	relaxed
relaxed	$\varepsilon[\alpha,i\neq l]$	ε	relaxed
relaxed	$\varepsilon[\alpha,i\neq l]$	$\varepsilon[\beta,k]$	excited
relaxed	$\varepsilon[\alpha,i\neq l]$	$\varepsilon[\beta,i\neq k]$	relaxed
excited	$\varepsilon[\alpha,l]$	ε	relaxed
excited	$\varepsilon[\alpha,l]$	$\varepsilon[\beta,k]$	excited
excited	$\varepsilon[\alpha,l]$	$\varepsilon[\beta,i\neq k]$	relaxed

Table 1: This table shows the possible transition classes for channel based restrictions. $\varepsilon[\alpha,i\neq l]$ means a class of n-dimensional character vectors with only some fix $v_{i\neq k} = \alpha \neq \varepsilon$ where α is a character variable and not an individual character.
A relaxed state allows for 6 possible transition classes. As input there is either ε for a spontaneous transition or there is a character not associated with the channel (k,l), namely $\varepsilon[\alpha,i\neq l]$. As output there is either a character with a non vanishing component k on the channel, leading to a new excited state, or a character with a non vanishing component $i \neq k$ somewhere else, or no output at all.
An excited state allows for three possible transition classes as there must be a channel associated input character α. The output is the same as for the relaxed state.

Proposition 1: Be $\mathscr{T}$ an NFIOA whose input and output alphabet are at least 2 dimensional and two channels (k,l) and (m,n) with $k \neq m$ and $l \neq n$. Then the *cbr*-operator for either channel commute: $cbr_{k,l}(cbr_{m,n}(\mathscr{T})) = cbr_{m,n}(cbr_{k,l}(\mathscr{T}))$. Thus, the *cbr*-operator can also be indexed with a set of channels.

[1]Please note that being excited or relaxed is per se not a static but a dynamic attribute of a state as it depends on how it was entered.

Proof: To prove the proposition, it suffice to show that the rules, prescribing the proper transitions for the combined operator do not depend on the sequence of application of the operators for both channels. This is illustrated in Tab. 2

Type of start state	Input	Output	Type of target state
relaxed	ε	ε	relaxed
relaxed	ε	$\varepsilon[\beta, i = k \vee i = m]$	excited
relaxed	ε	$\varepsilon[\beta, i \neq k \,\&\, i \neq m]$	relaxed
relaxed	$\varepsilon[\alpha, i \neq l \,\&\, i \neq n]$	ε	relaxed
relaxed	$\varepsilon[\alpha, i \neq l \,\&\, i \neq n]$	$\varepsilon[\beta, i = k \vee i = m]$	excited
relaxed	$\varepsilon[\alpha, i \neq l \,\&\, i \neq n]$	$\varepsilon[\beta, i \neq k \,\&\, i \neq m]$	relaxed
excited	$\varepsilon[\alpha, i = l \vee i = n]$	ε	relaxed
excited	$\varepsilon[\alpha, i = l \vee i = n]$	$\varepsilon[\beta, i = k \vee i = m]$	excited
excited	$\varepsilon[\alpha, i = l \vee i = n]$	$\varepsilon[\beta, i \neq k \,\&\, i \neq m]$	relaxed

Table 2: This table illustrates the restrictions of 2 consecutive channel operators onto a transition relation. The terminology is the same as in Tab. 1
A relaxed state allows for 6 different transition classes. Either there is a spontaneous transition or there is an input α neither associated with channel (k,l) nor with (m,n). As output there is either a character with a non vanishing component k or m on a channel, each leading to a new excited state, or a character with a non vanishing component i neither equaling k nor m, or no output at all, both leading to a new relaxed state.
An exited state allows for 3 possible transition classes. There must be a channel associated input character α, either from (k,l) or (m,n). The output is the same as for the relaxed state.
These rules obviously do not depend on the sequence of the application of the operators for both channels.

Proposition 2: Be $\mathscr{A}$ and $\mathscr{B}$ two NFIOAs and *restr* an operator that restricts the transition relation of $\mathscr{B}$. Then the following relation holds: $restr(\mathscr{A} \otimes \mathscr{B}) = \mathscr{A} \otimes restr(\mathscr{B})$.

The proof is trivial.

Definition 4: A *run* as a sequence of states has to be calculated according to the following rules:

1. Be p a relaxed state, then every spontaneous transition or every transition which have an open input (not associated with a channel) can be taken.
2. Be p an excited state, then any ensuing transition must process this character.

It is not a priori clear that this procedure creates something meaningful. A necessary condition obviously is that there has to be a receiving transition for each sending transition for each channel and that the acceptance condition can still be met. I therefore define in accordance with the protocol literature (e.g. [1]):

Definition 5: A channel based restricted automaton is called *well formed* if for every channel mediated transition which sends a character (different from ε) there exists an induced transition to process it.

Definition 6: A well formed channel based restricted automaton is called *consistent* if for each reachable state value either the acceptance condition is met or there is at least one continuation such that the acceptance condition can be met.

As a simple consequence, in a consistent channel based restricted automaton all continuations allow to meet the acceptance condition. Assuming the contrary that one continuation leads to a reachable

state where neither the acceptance condition is already met nor any such continuation exists is a direct contradiction to the definition.

3.1 Protocols

I would like to focus on the special case where several NFIOAs interact completely via a known set of channels, that is with no open input or output components.
Definition 7: A *protocol* is a channel based restricted product automaton with no open input or output components. I call the individual factor automata *roles*.

As was already mentioned in [13] the definition of "consistent" directly entails that a consistent protocol neither has deadlocks in the sense that there is a single reachable state without any continuation such that the acceptance condition holds nor livelocks, characterized by a set of periodical reached states without such a continuation.

Proposition 3: A weakly synchronized product of two protocols is again a protocol.

Proof: We have to show that for two protocols $\mathscr{P}_1 = cbr_{C_1} \bigotimes \mathscr{A}_i$ and $\mathscr{P}_2 = cbr_{C_2} \bigotimes \mathscr{B}_i$ with two sets of channels C_1 and C_2 the following equation holds:
$\mathscr{P}_1 \otimes \mathscr{P}_2 = (cbr_{C_1} \bigotimes \mathscr{A}_i) \otimes (cbr_{C_2} \bigotimes \mathscr{B}_i) = cbr_{C_1 \cup C_2}((\bigotimes \mathscr{A}_i) \otimes (\bigotimes \mathscr{B}_i)$.
This is obviously the case, as the restriction relates only either to the weakly synchronized $\mathscr{A}_i$ or $\mathscr{B}_i$ but never simultaneously to both. Therefore it gives the same result, if we weakly synchronize two channel based restricted weakly synchronized products or create the channel based restriction over the set union of channels of the weakly synchronized products of all automata.

4 Condition Based Restriction and Processes

The second mechanism to couple NFIOAs represents the causal relation between the input and output of a single system as an inner coupling determined by coordination rules. The goal is to restrict the transition set of the weakly synchronized product automaton such that at least a quasi-determinism of the formerly nondeterministic transition set is achieved in a sense that from each reachable state there is at most one transition for each input character, including the empty character, while the factor NFIOAs (the roles) still can be regained by projection.

In Fig 6 and 7 an example of such an inner coupling for the mutual exclusion protocol is illustrated where a process realizes a "man in the middle". It just pass-through the incoming requests and thereby combines both protocol roles complementary to example 1.2.1.
Definition 8: A *quasi deterministic* NFIOA has at most one transition for each input sign, including the empty character, from any reachable state.
Definition 9: Be $\mathscr{B}$ an NFIOA and $\pi = (\pi_Q, \pi_I, \pi_O)$ with $\pi_Q : Q_{\mathscr{B}} \to Q_{\mathscr{B}}$, $\pi_I : I_{\mathscr{B}} \to I_{\mathscr{B}}$, and $\pi_O : O_{\mathscr{B}} \to O_{\mathscr{B}}$ a projection function[2]. Then the projected automaton $\mathscr{A} = \pi(\mathscr{B})$ is given by $Q_{\mathscr{A}} = \pi_Q(Q_{\mathscr{B}})$, $I_{\mathscr{A}} = \pi_I(I_{\mathscr{B}})$, $O_{\mathscr{A}} = \pi_O(O_{\mathscr{B}})$, $q_{0\mathscr{A}} = \pi_Q(q_{0\mathscr{B}})$, $Acc_{\mathscr{A}} = \pi_Q(Acc_{\mathscr{B}})$, $\Delta_{\mathscr{A}} = \{(p',q',i',o') | (p,q,i,o) \in \Delta_{\mathscr{B}}$ and $p' = \pi_Q(p), q' = \pi_Q(q), i' = \pi_I(i), o' = \pi_O(o)\}$.

The condition based restriction is supposed to eliminate all transitions from non-reachable states as well as all transitions for which a given condition is true.
Definition 10: Be $\mathscr{T}$ an NFIOA. A set of conditions E parametrizes the *condition based restriction*

[2] A projection function π has the special property that $\pi = \pi \circ \pi$.

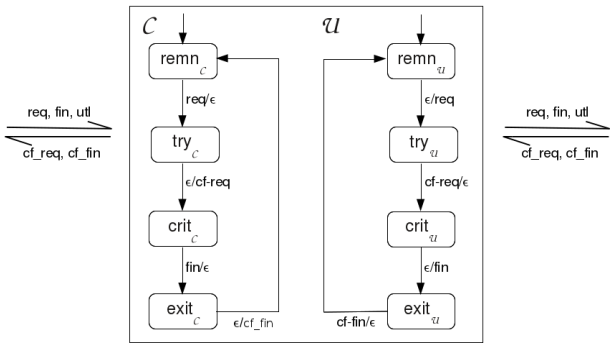

Figure 6: State diagram of the two (still independent) roles of a ”man in the middle” process within the mutual exclusion protocol.

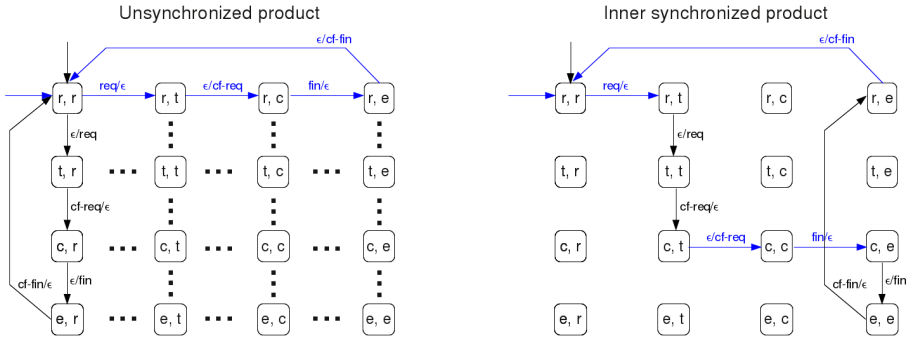

Figure 7: State diagram of the two roles of the mutual exclusion protocol, acting as a ”man in the middle”. Left, you see the weakly synchronized product of both roles. Right, you see the condition based restricted product.

($cond_E$) operator such that $\mathscr{P} = cond_E(\mathscr{T})$ with $Q_{\mathscr{P}} = Q_{\mathscr{T}}$, $I_{\mathscr{P}} = I_{\mathscr{T}}$, $O_{\mathscr{P}} = O_{\mathscr{T}}$, $Acc_{\mathscr{P}} = Acc_{\mathscr{T}}$, and $\Delta_{\mathscr{P}} = \{(p,q,i,o) \in \Delta_{\mathscr{T}} | p \text{ is a reachable state of } \mathscr{T}$ and $e(p,q,i,o) \in E$ is false$\}$.

The interesting property of the condition based restriction is, that it may not affect the projection:

Definition 11: A projection π of an NFIOA $\mathscr{T}$ is called *unaffected* by a condition based restriction $cond_E$ if $\pi(\mathscr{T}) = \pi(cond_E(\mathscr{T}))$.

Definition 12: A condition restricted automaton is called *consistent* if if for each reachable state value either the acceptance condition is met or there is at least one continuation such that the acceptance condition can be met.

4.1 Separating inner and outer coupling

Now, I would like to justify that looking at outer and inner coupling of roles can be separated. I first state that

Proposition 4: Channel and condition based restriction commute.

Proof: To prove the proposition we have to show again that the rules to restrict the transition relation do not depend on the sequence of application of both operators. Starting from the initial state, the set of reachable states is build up inductively. For the initial state, the set of transition to be eliminated is just the union of the to-be-eliminated transitions from channel and condition based restriction. The thereby determined set of reachable next level nodes is therefore independent of the sequence of application of both operators. The same argument applies inductively for the transitions starting from this set as well as for any further set of reachable nodes. So in fact, as both operator definitions relate to reachable states, the rules for both, excited states and relaxed states of definition 3 become only modified in the sense that it has to be checked whether the outgoing transitions are to be additionally excluded by the condition based restriction.

With this in mind, let us assume to have two independent protocols $\mathscr{P}_1 = cbr_{C_1}(\bigotimes \mathscr{A}_i)$ and $\mathscr{P}_2 = cbr_{C_2}(\bigotimes \mathscr{B}_i)$ where the roles $\mathscr{A}_n$ and $\mathscr{B}_1$ actually relate to the same process. The effective inner coupling between these two roles is expressed by a condition based restriction operator $cond_E$ relating only to $\mathscr{A}_n$ and $\mathscr{B}_1$. We then have:

$$\begin{aligned} & cond_E(\mathscr{P}_1 \otimes \mathscr{P}_2) \\ \overset{P.3}{=} \quad & cond_E(cbr_{C_1 \cup C_2}((\bigotimes \mathscr{A}_i) \otimes (\bigotimes \mathscr{B}_i))) \\ \overset{P.2\&4}{=} \quad & cbr_{C_1 \cup C_2}((\bigotimes_{i<n} \mathscr{A}_i) \otimes cond_E(\mathscr{A}_n \otimes \mathscr{B}_1) \otimes (\bigotimes_{i>1} \mathscr{B}_i))) \end{aligned}$$

So, applying the condition based restriction to some large protocol, involving many different roles boils down to applying it only to the roles that are actually coordinated within a given process. In the general case, where the condition based restriction relates to arbitrary NFIOA parts of the protocols, these NFIOA would have to be permuted to allow for this separation.

4.2 Processes

Now, what is a process? As a first try, we could define a process within the presented framework as a product automaton from at least two roles of different protocols that are restricted by some coordination conditions. Thereby, a process coordinates at least two different interactions.

Fig. 8 illustrates the example of the introduction. It shows the weakly synchronized product automaton of the two different roles of a resource administration process together with the to-be-eliminated transitions according to the following coordination conditions:

1. Being in state value $(*, abst)$, no spontaneous transition to state value $(crit, *)$ is allowed. That is, the token is needed for any entry into the critical region.
2. Being in state value $(try, *)$, no spontaneous transition to state value $(*, abst)$ is allowed. That is, after an incoming request, it is not allowed to dispose the token.
3. Being in state value $(crit, *)$, no spontaneous transition to state value $(*, abst)$ is allowed. That is, after an incoming request has been acknowledged, it is not allowed to dispose the token.
4. Being in state value $(*, interm)$, no spontaneous transition to state value $(remn, *)$ is allowed. That is, before sending a confirmation for finalization, the token has to be disposed.

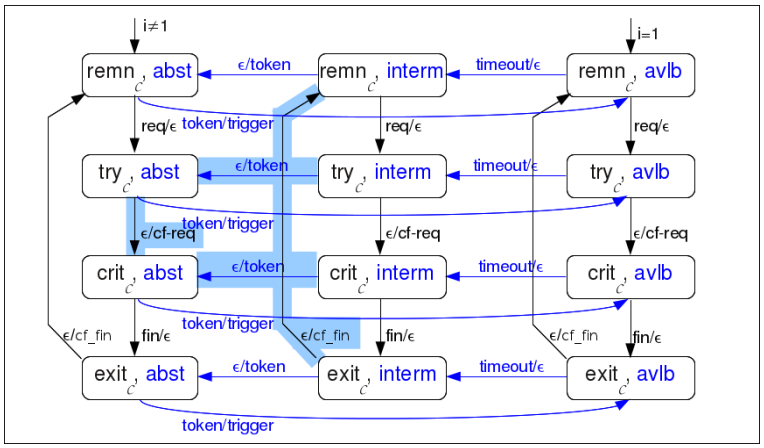

Figure 8: The weakly synchronized product automaton between the two different roles of a resource administration process is shown. The transitions which are eliminated for coordination are marked.

4.2.1 Determination

However, we not only want quasi deterministic NFIOAs but deterministic ones because deterministic ones specify finite systems (see appendix and e.g. [13]). Fig. 9 shows a simple example, where it seems straight forward to eliminate a spontaneous transition. With the usual interpretation of I/O automata, the two automata of Fig 9 would be viewed as being behaviorally equivalent.

Figure 9: Do the NFIOA on the left and on the right specify the same automaton behavior? In case of a channel based restricted product automaton it depends on what the rest of the automaton is doing.

But within a channel based restricted product automaton, both behaviors are equivalent only if the rest of the automaton waits until it receives character 'b' - or, in other words, if any other factor automaton process the result of the first automaton synchronously. With so called asynchronous processing, another factor automaton could transit and show some behavior. Eliminating the intermediate state implies to deprive all other factor automata from their chance to transit, possibly changing the behavior of the entire product automaton. Eliminating the spontaneous transitions changes the "atomicity of time steps".

Another problem is that the set of state values might change. As the acceptance component directly relates to the set of state values, it must be restated. Also, the projection relation between the restricted product automaton and its constituting roles gets lost. A third problem is that we might have to aggregate several output characters in one step.

It appears that moving from the quasi deterministic automaton to a deterministic one, the behavioral equivalence of the complete interaction has to be checked. For the quasi deterministic automaton of Fig. 8, representing the resource administration process, a behaviorally equivalent fully deterministic automaton can be found which is illustrated in Fig. 10. So, deterministic processes do coordinate at least two different interactions, but they may have lost their resemblance to the original roles used for the definition of their interactions they are involve in.

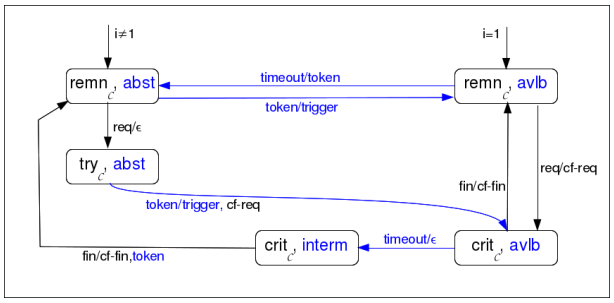

Figure 10: The completely deterministic resource administration process.

5 Related Work

There have been many different approaches to the issue of describing processes and their interactions, many of them can be subsumed under Richard Mayr's [7] process rewrite systems. He introduced a unifying view encompassing many formalisms based on named actions like finite state systems, Petri nets, pushdown processes, etc. The most general class he named process rewrite systems.

It presupposes a countably infinite set of atomic actions $Act = \{a,b,c,\ldots\}$ and a countably infinite set of process constants $Const = \{\varepsilon, X, Y, Z, \ldots\}$. All process terms that describe the states of the system have the form $t = \varepsilon | X | t_1.t_2 | t_1 || t_2$. The dynamics of a system is described by a finite set of rules Δ of the form $t_1 \xrightarrow{a} t_2$. For every $a \in Act$, the transition relation $\xrightarrow{a}$ is the smallest relation that satisfies the inference rules

$$\frac{(t_1 \xrightarrow{a} t_2) \in \Delta}{t_1 \xrightarrow{a} t_2}, \qquad \frac{t_1 \xrightarrow{a} t_1'}{t_1 || t_2 \xrightarrow{a} t_1' || t_2}, \qquad \frac{t_1 \xrightarrow{a} t_1'}{t_1.t_2 \xrightarrow{a} t_1'.t_2}$$

How would the first interaction, the protocol of mutual exclusion (see. section 1.2.1) be described? Instead of denoting the input and output, each transition would have to be arbitrarily named. E.g. for the user process $\mathscr{U}$: $rem \xrightarrow{try()} try$, $try \xrightarrow{cf-req()} crit$, $crit \xrightarrow{finalize()} exit$, $exit \xrightarrow{cf-fin()} rem$ and for the resource administration process: $rem \xrightarrow{request()} try$, $try \xrightarrow{send-cf-req()} crit$, $crit \xrightarrow{exit()} exit$, $exit \xrightarrow{send-cf-exit()} rem$.

Because these names or labels are arbitrary, they don't help in constructing the transition relation of the protocol as a restricted product automaton. Here is how the protocol would have to be described:

$$\begin{array}{ccc} rem_{\mathscr{U}} || rem_{\mathscr{C}} & \xrightarrow{try()} & try_{\mathscr{U}} || rem_{\mathscr{C}} \\ try_{\mathscr{U}} || rem_{\mathscr{C}} & \xrightarrow{request()} & try_{\mathscr{U}} || try_{\mathscr{C}} \\ & \ldots & \end{array}$$

One would not be allowed to mix this with the rules for the single protocol role. As a result, we can not use our knowledge of the parts (the roles) to construct the whole (the interactions).

In essence all approaches that are subsumed with process rewrite systems assume the "atomicity of actions" and provide names for identical actions only because they appear differently under different perspectives.

Other approaches that are also based on naming of actions suffer from the same shortcomings, for example Robin Milner's calculus of communicating systems [9, 10], Charles A. R. Hoare's communicating sequential processes [5] or Nancy Lynch's distributed algorithms [6].

The same holds for the currently quite popular BPMN proposal [11] to describe processes and their interactions driven by large industry players. There is some similarity between what the BPMN standard calls a private (internal) process and a process in my sense, an abstract (public) process and a role and a choreography or collaboration (global) process and a protocol. However, if a private process or orchestration of services is a combination of executable functionality, it remains unclear, what an 'end-to-end' process should be. The logical difference between a private and a public process is not seen as a projection but some sort of abstraction where only those activities that are used to communicate to other processes plus the order of these activities, are included. A choreography describes the way processes coordinate their interactions, which is claimed to allow the process designer to plan their processes for inter-operation without introducing conflicts, but no formal connection is made in the specification.

Also, a comparison to game theory is interesting, The interaction model of game theory is similar to the protocol model. As I have shown in [12], a protocol, enriched by a decision alphabet, can be mapped onto a corresponding extensive game without its payoff evaluation. With games, the nondeterminism problem of interaction, that the interactions don't determine the actions, is solved by introducing "decision" together with a payoff function. Then the decisions determine the actions and the question how to act becomes an optimization problem. In contrast, my approach to determine single interactions by other interactions leads to a coordination problem. However, there are already game theoretic methods to describe the simultaneous play of multiple games by single players which naturally are very similar to their process analogons (e.g. [4]).

6 Discussion

In this article I presented an approach to describe processes and their interactions with the same technical means, namely nondeterministic input output automata (NFIOAs). I presented two complementary ways to represent the causal relation between the output and the input of different systems, which I called "outer coupling", and between the input and output of the same system, which I called "inner coupling". Extending [14], an operator based approach was introduced.

The openness of the interaction networks is represented by the incompleteness of the interaction descriptions in the sense of protocols with respect to the interaction coordinating processes. Constructing a process, one needs to know the rest of the interaction network only up to the next interaction partner's role in the own interactions.

Why do protocols provide so many spontaneous transitions, while it is so difficult to get rid of this spontaneity when constructing deterministic processes? It is this spontaneity which provides the freedom to combine different interactions to coordinating processes. So, protocol design strives for providing role descriptions which can be most easily coordinated with other roles, a property which naturally gets lost in constructing deterministic processes.

Currently, instead of "protocol" and "processes", often the more fashionable terms "choreography" and "orchestration" are used, especially in the context of a so called "service oriented architecture (SOA)" [15]. If we identify "choreography" with "protocol" and "orchestration" with "process", we see that a single choreography is not supposed to specify the behavior of a system completely as well as an orchestration is not supposed to specify any "loose coupling" between two different applications. On the contrary it becomes understandable why a modeling of processes as stepwise executable activities which is not based on the interaction descriptions leads to strong centralization tendencies and tight coupling of the interacting systems.

The straight forward composition mechanism seems quite attractive for combining different roles

flexibly into quasi deterministic processes, just by adding a couple of rules. Unfortunately, things become more complex if we strive for determinism, which we must if we want to execute our processes with the traditional machines.

Acknowledgment Writing this article would not have been possible without an intense discussion with Hans-Jörg Kreowski beforehand and a daylong return journey by train. I also thank the reviewers for their helpful comments.

References

[1] Daniel Brand & Pitro Zafiropulo (1983): *On Communicating Finite-State Machines*. *J. ACM* 30(2), pp. 323–342, doi:10.1145/322374.322380.

[2] Nirmit Desai, Ashok U. Mallya, Amit K. Chopra & Munindar P. Singh (2005): *Interaction Protocols as Design Abstractions for Business Processes*. *IEEE Trans. Software Eng.* 31(12), pp. 1015–1027, doi:10.1109/TSE.2005.140.

[3] Berndt Farwer (2001): *omega-Automata*. In Erich Grädel, Wolfgang Thomas & Thomas Wilke, editors: *Automata, Logics, and Infinite Games*, *Lecture Notes in Computer Science* 2500, Springer, pp. 3–20, doi:10.1007/3-540-36387-4_1.

[4] Sujata Ghosh, Ramaswamy Ramanujam & Sunil Easaw Simon (2010): *Playing Extensive Form Games in Parallel*. In Jürgen Dix, João Leite, Guido Governatori & Wojtek Jamroga, editors: *CLIMA*, *Lecture Notes in Computer Science* 6245, Springer, pp. 153–170, doi:10.1007/978-3-642-14977-1_13.

[5] C.A.R. Hoare (1985/2004): *Communicating Sequential Processes*. Prentice Hall. Available at `http://www.usingcsp.com/cspbook.pdf`.

[6] Nancy A. Lynch (1996): *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc. San Francisco, California, USA.

[7] Richard Mayr (1999): *Process Rewrite Systems*. *Information and Computation* 156, pp. 264–286, doi:10.1006/inco.1999.2826.

[8] George H. Mealy (1955): *A method for synthesizing sequential circuits*. *Bell System Technical Journal* 34(5), pp. 1045–1079.

[9] Robin Milner (1980): *A Calculus of Communicating Systems*. Springer, Berlin, Heidelberg, New York.

[10] Robin Milner (1989): *Communication and Concurrency*. Prentice Hall.

[11] OMG (2011): *Business Process Model and Notation Specification, Version 2.0*. Technical Report, OMG.

[12] Johannes Reich (2009): *The relation between protocols and games*. In Stefan Fischer, Erik Maehle & Rüdiger Reischuk, editors: *GI Jahrestagung*, *LNI* 154, GI, pp. 3453–3464.

[13] Johannes Reich (2010): *Finite System Composition and Interaction*. In Klaus-Peter Fähnrich & Bogdan Franczyk, editors: *40. GI Jahrestagung (2)*, *LNI* 176, GI, p. 603.

[14] Johannes Reich (2011): *Process synthesis from multiple interaction specifications*. In Hans-Ulrich Heiß, Peter Pepper, Holger Schlingloff & Jörg Schneider, editors: *41. GI Jahrestagung*, *LNI* 192, GI, p. 309.

[15] R. W. Schulte & Y. V. Natis (1996): *"Service-Oriented" Architectures, Part 1*. SPA-401-068, Gartner Group.

[16] Chris A. Vissers (1976): *Interface, a dispersed architecture*. *SIGARCH Comput. Archit. News* 4(4), pp. 98–104, doi:10.1145/633617.803557.

7 Appendix

The importance of finite I/O automata for the theory of processes stems from the fact that DFIOAs can specify finite systems (in a physical sense). Such a system is represented as structure which comprises internal as well as externally accessible states in the sense of time dependent properties together with function which describes the time evolution of the states[3].

Definition 13: A *finite system* is defined by a tuple $\mathscr{S} = (T, succ, Q, I, O, x, in, out, f)$. T is the enumerable set of time values starting with 0 such that $succ : T \to T$ is the invertible time successor function. Q, I and O are the finite sets of state values for the internal, input and output states $(x, in, out) : T \to (Q, I^\varepsilon, O^\varepsilon)$. $f = (f^{ext}, f^{int}) : I^\varepsilon \times Q \to O^\varepsilon \times Q$ is a function describing the time evolution or system operation updating the internal and output state in one time step for each $t \in T$:

$$\begin{pmatrix} out(t+1) \\ x(t+1) \end{pmatrix} = \begin{pmatrix} f^{ext}(in(t), x(t)) \\ f^{int}(in(t), x(t)) \end{pmatrix}.$$

The state values of the I/O-states are also called *characters*. The n-fold application of the time successor function $succ$ is written as $t +_{\mathscr{S}} n := succ^n_{\mathscr{S}}(t)$.

Definition 14: A DFIOA $\mathscr{A} = (Q_{\mathscr{A}}, I_{\mathscr{A}}, O_{\mathscr{A}}, q_0, Acc, \Delta)$ *specifies* a finite system $\mathscr{S} = (T, succ, Q_{\mathscr{S}}, I_{\mathscr{S}}, O_{\mathscr{S}}, x, in, out, f)$ if $Q_{\mathscr{S}} \subseteq Q_{\mathscr{A}}$, $I_{\mathscr{S}} \subseteq I_{\mathscr{A}}$, $O_{\mathscr{S}} \subseteq O_{\mathscr{A}}$, $x(0) = q_0$ and for any point in time $t \geq 0$ in every possible sequence, $(x(t), x(t+1), in(t), out(t+1)) \in \Delta_{\mathscr{A}}$.

[3] This use of then term 'state' is the reason I used the term 'state values' where normally in automata theory the term 'state' is used.