Process Synthesis from Multiple Interaction Specifications

Johannes Reich, johannes.reich@sophoscape.de

Abstract: The traditional approach to model (business) processes as stepwise executable activities which interact with multiple partner processes by typed interfaces seems to lead to centralization or at least to a very tight coupling of the interacting systems.

Here, a unifying approach is presented that models the specifications of both, deterministic processes as well as their possibly nondeterministic interactions starting from the same building blocks, namely finite input output automata (or transducers).

Processes are viewed as finite systems that take part in multiple, possibly nondeterministic interactions. The interactions between processes are specified as protocols. The projection of a process specification onto one of its interactions is called a role.

The synthesis of a process specification from multiple roles is illustrated by the example of a resource administration process which is supposed to provide a requesting process exclusive access to a single resource (e.g. e printer or a car). This process has to coordinate its interaction with the requesting process, specified by the mutual exclusion protocol, and at the same time it has to coordinate its interactions with other resource administration processes, specified by a token exchange protocol, to guarantee the exclusivity of the access.

So, starting from an interaction centric perspective together with a synthesis procedure for process specifications, this approach circumvents the centralization tendencies and allows the construction of truly loosely coupled (finite) systems.

1 Introduction

The traditional approach to model (business) processes which interact with multiple partner processes as stepwise executable activities seems to lead to centralization or at least to a very tight coupling of the interacting systems which hinders their development and maintenance in the face of evolving requirements (see [DMCS05] for business processes).

In this article a different approach to synthesize processes is described. Processes are viewed as finite systems that take part in multiple, possibly nondeterministic interactions, specified by protocols. Unification of the specification of (deterministic) processes and their (nondeterministic) interactions is achieved by using the same building blocks for both specifications, namely finite input output automata (or transducers). The projection of a process specification onto one of its interactions is called a role.

The synthesis of a process specification from multiple roles is illustrated by the example of a resource administration process which is supposed to provide a requesting process exclusive access to a single resource (e.g. a printer or a car). This process has to coordinate its interaction with the requesting process, specified by the mutual exclusion protocol, and at the same time it has to coordinate its interactions with other resource administration processes, specified by a token exchange protocol, to guarantee the exclusivity of access.

So, starting from an interaction centric perspective together with a synthesis procedure for process specification, this approach circumvents the centralization tendencies and allows the construction of truly loosely coupled (finite) systems in the sense of Glassman [Gla73],

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4

Desai et al. [DMCS05] follow a very similar philosophy, but describe protocols and processes more on a business semantics level in terms of commitments. Choosing automata theory as a base has the advantage of providing more insight from a system theoretic point of view and thereby more insights into the foundations of software engineering [Bro95].

2 Preliminaries

All essential steps are illustrated by the example of interacting resource administration processes (see Fig. 1).

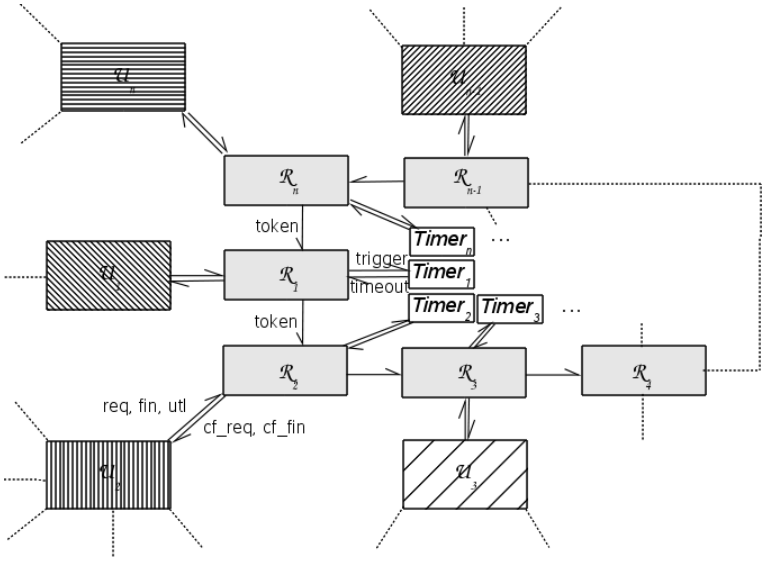

Figure 1: A network of interacting systems where n resource managers $\mathcal{R}_i$ interact in a ring topology with timers and resource demanding systems $\mathcal{U}_i$, which themselves interact with an unknown number of other systems

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, $A^\epsilon := A \cup \{\epsilon\}$ with ϵ is the empty character. For state value sets Q, $Q^\epsilon := Q \cup \{\epsilon\}$ 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 $A^\epsilon = A_1^\epsilon \times \ldots \times A_n^\epsilon$. Additionally, a state vector $(p_1, \ldots, p_n)$ where p_k belongs to automaton $\mathcal{A}_k$ is written as $\vec{p}$ and the change of this vector in a position k from p_k to q_k is written as $\vec{p}\left[\frac{q_k}{p_k}\right]$. An

n-dimensional vector of characters v with the k-th component v_k and the rest ϵ is written as $\epsilon^n[v_k] = (\epsilon_1, \ldots, \epsilon_{k-1}, v_k, \epsilon_{k+1}, \ldots, \epsilon_n)$. In case the dimension is clear, it may be omitted.

3 Systems and Their Roles in Interactions

The proposed system notion is physically motivated based on the function notion: a function describes how input and internal state values are transformed into output and new internal state values. It was introduced in [Rei10, Rei09] and is in line with system theory [Unb93][1].

Definition 1: A *finite system* is defined by a tuple $\mathcal{S} = (T, succ, Q, I, O, x, in, out, f)$. 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 $(x, in, out) : T \to (Q, I^\epsilon, O^\epsilon)$. $f = (f^{ext}, f^{int}) : I^\epsilon \times Q \to O^\epsilon \times Q$ 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 $t +_{\mathcal{S}} n := succ^n_{\mathcal{S}}(t)$.

The time dependency represents the desired succession property of the state values in a run in the sense of Leslie Lamports time notion [Lam78].

As a system's I/O-states are accessible from the outside, they can actually be shared between systems and thereby enable system interactions.

Definition 2: Be $\mathcal{S}, \mathcal{R}$ two systems where the k-th component of the output state of $\mathcal{S}$ is identical with the l-th component of the input state of $\mathcal{R}$. This shared state component, characterized by $\mathcal{C} = (\mathcal{S}, k, \mathcal{R}, l)$ is called a *channel*. If the output or input state is a scalar, the index is 0. $\mathcal{S}$ is also called a *sender* and $\mathcal{R}$ a receiver (system).

As a consequence, $O_{\mathcal{S},k} \subseteq I_{\mathcal{R},l}$ and the formal description of system interactions rests on identical names of the input and output characters. To reduce the interaction complexity, I require that in channel mediated system interactions with multiple channels, in any transition there is at most one channel-related input component different from ϵ.

The notion of "behavior" relates only to the accessible or "observable" I/O-states. It later on allows to define equivalence of systems in purely "behavioral" terms. **Definition 3:** A *trace* with respect to a set of channels $C = \{c_1, \ldots c_n\}$ is the sequence of characters of the channels $((a_0, c_{i0}), (a_1, c_{j1}), \ldots)$.

Traces therefore describe the sequence of characters which a given set of channels represent. In case it is clear which character is recorded from which channel, the channel

[1] As in physics, a state is a time dependent property of a system. Thereby what is usually named a "state" in automata theory becomes a "state value" in my proposed terminology.

information can be dropped. Relating to the set of channels of a system, we can define its *observable behavior* as the set of possible traces.

Taking explicitly into account the initial state of a system as well as expressing some sort of "goal", the behavior of finite systems is specified with nondeterministic finite I/O automata, also known as transducer [Sak09] .

Definition 4: A *nondeterministic finite I/O automaton (NFIOA)* is a tuple $\mathcal{A} = (Q, I, O, q_0, Acc, \Delta)$. Q is the non-empty finite set of state values, I and O are the finite input and output alphabets, q_0 is the initial state value, Acc is the acceptance component and $\Delta \subseteq Q \times Q \times I^\epsilon \times O^\epsilon$ 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 $Acc \subseteq 2^Q$ [Far02].

In case that for each $(p, i) \in Q \times I$ there is at most one transition $(p, q, i, o) \in \Delta$ then Δ defines a function $\delta : Q \times I \to Q \times O^\epsilon$ with $(q, o) = \delta(p, i)$. If no transition with ϵ-input exists, we then have a deterministic automaton or *DFIOA* (a Mealy automaton [Mea55]).

Especially when we describe interactions we will not be interested in the specifications of a complete system, but only on those parts which are directly involved in the interactions, that is only on the projection of a system onto the involved channels and affected inner states. I call such a projection a (concrete) role:

Definition 5: Be $\mathcal{S} = (T, succ, Q_\mathcal{S}, I_\mathcal{S}, O_\mathcal{S}, x, in, out, f)$ a finite system. A *concrete role* of $\mathcal{S}$ is defined by an NFIOA $\mathcal{A} = (Q_\mathcal{A}, I_\mathcal{A}, O_\mathcal{A}, q_0, Acc, \Delta)$ with $Q_\mathcal{S} \supseteq Q_\mathcal{A}$, $I_\mathcal{S} \supseteq I_\mathcal{A}$, and $O_\mathcal{S} \supseteq O_\mathcal{A}$, if a projection function[2] $\pi = (\pi_Q, \pi_I, \pi_O) : Q_\mathcal{S} \times I_\mathcal{S}^\epsilon \times O_\mathcal{S}^\epsilon \to Q_\mathcal{A} \times I_\mathcal{A}^\epsilon \times O_\mathcal{A}^\epsilon$ exists such that for any point in time $t \geq 0$ in every possible sequence, $(\pi_Q(x(t)), \pi_Q(x(t+1)), \pi_I(in(t)), \pi_O(out(t+1))) \in \Delta_\mathcal{A}$. The equivalence class of all concrete roles with identical behavior is called the *abstract role*.

Figure 2: From a behavioral point of view, both, the NFIOA on the left and on the right specify the same behavior.

As is illustrated in Fig. 2 an NFIOA may over-specify the behavior of a system. To get a behavioral equivalent DFIOA from an NFIOA it may be necessary to eliminate ϵ-transitions without input. This structural redundancy will become quite important for process synthesis from interactions (see below).

The question whether a given role "makes sense" can only be answered with the knowledge of the complete interaction (see below). A *robust* role can reach its acceptance condition

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

always independently on its interaction partner(s).

For technical reasons we also need the "embedding" relation of roles:

Definition 6: A role $\mathcal{A}$ embeds another role $\mathcal{B}$ ($\mathcal{A} \supseteq \mathcal{B}$) if for the related projection functions $\pi_{\mathcal{B}} = \pi_{\mathcal{B}} \circ \pi_{\mathcal{A}}$ holds and $Acc_B = \pi_{\mathcal{B}}(Acc_{\mathcal{A}})$.

4 Analysis of System Interactions: Protocols as the Outer Synchronized Product of Roles

The simplest way to combine 2 NFIOAs is to look at both simultaneously and to require all of their acceptance conditions to be fulfilled:

Definition 7: The *unsynchronized product* of a set of n NFIOAs $\mathcal{A}_k$ is defined by NFIOA $\mathcal{B} = (Q, I, O, \vec{q}_0, Acc, \Delta)$, with $Q_{\mathcal{B}} = \times Q_k, I_{\mathcal{B}} = \times I_k, O_{\mathcal{B}} = \times O_k, \vec{q}_{0\mathcal{B}} = (q_{0_1}, \ldots, q_{0_n})$, $Acc_{\mathcal{B}} = \bigwedge Acc_k$ is the common acceptance component, $\Delta_{\mathcal{B}} := \{(\vec{p}, \vec{q}, \vec{i}, \vec{o}) |\ \vec{p}$ is a reachable state and $\mathcal{A}_k$ provides a transition (p_k, q_k, i_k, o_k) with $\vec{q} = \vec{p}\left[\frac{q_k}{p_k}\right]$ and $\vec{i} = \epsilon[i_k]$ and $\vec{o} = \epsilon[o_k]\ \}$. I also write $\mathcal{B} = \bigotimes_{i=1}^{n} \mathcal{A}_i$.

"Synchronization" then means to eliminate some transitions from the transition relation $\Delta_{\mathcal{P}}$ and thereby to eliminate the symmetry of the unsynchronized product:

Definition 8: Be $\mathcal{B} = \bigotimes_{k=1}^{n} \mathcal{A}_k$ an unsynchronized product automaton. An automaton $\mathcal{B}^*$ with all components from $\mathcal{B}$ except a reduced transition set $\Delta_{\mathcal{B}^*} = \Delta_{\mathcal{B}} \setminus \Delta^e$ with nonempty $\Delta^e \subset \Delta_{\mathcal{B}}$ is a synchronized version of $\mathcal{B}$. I also write $\mathcal{B}^* = \mathcal{B} \setminus \Delta^e$

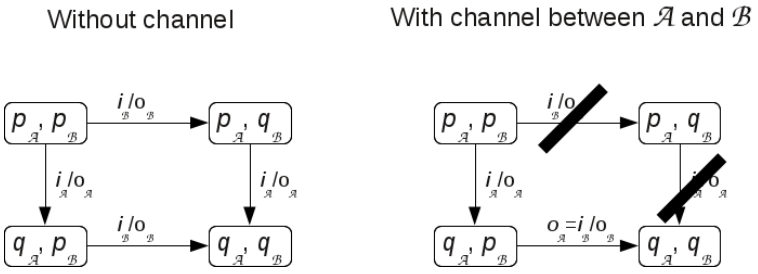

Figure 3: The time step-related constraint effect of a channel on the NFIOA specification of two systems.

A channel between two systems has the effect of a time step-related constraint on their corresponding NFIOA: it reduces the transition relation compared to their unsynchronized product as it requires that the receiving system is supposed to transit only after the sending system and with the input of the others output (see Fig. 3) - the induced transition. A common channel thereby eliminates all transitions from the product automaton attributable to the receiver where from a reachable state there is no input providing corresponding sender transition.

Definition 9: Be $A = \{\mathcal{A}_1, \ldots, \mathcal{A}_m\}$ a set of NFIOAs and $C = \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ a set of channels between the NFIOAs. The channel mediated synchronized product $\mathcal{P}$ is called an *outer (synchronized) product*. I also write symbolically $\mathcal{P} = \bigotimes_{k=1}^{m} \mathcal{A}_k \setminus \Delta^C$.

It is not a priori clear that a channel makes sense in the sense that all sent characters can be processed by the receiver (wellformedness) and that the acceptance condition of the receiver is still retained (consistency). I therefore define

Definition 10: An outer product of k NFIOAs $\mathcal{P} = \bigotimes_{k=1}^{n} \mathcal{A}_k \setminus \Delta^C$ is called *well formed* if for every channel mediated transition attributable to a sender which sends a character different from ϵ there exists an induced transition of the receiver.

Definition 11: A well formed outer product is called *consistent* if for each reachable product state value the product acceptance conditions still holds.

Proposition 1: Let $\mathcal{A}$ and $\mathcal{B}$ be two NFIOAs, representing projections of two systems which have one common channel $c = (\mathcal{A}, k, \mathcal{B}, l)$ whose outer product is consistent and where $\mathcal{B}$ does not receive any additional input and $\mathcal{A}$ does not provide any further output. Then a behavioral equivalent system specification can be derived which aggregates the $\mathcal{A}$- and $\mathcal{B}$-attributable transition pairs into one and eliminates the common input/output state.

Proof: We have to distinguish two cases: First, the $\mathcal{A}$-attributable transition provides an output character in the channel component. Then there is a corresponding $\mathcal{B}$-attributable transition taking it as input, leading to a characteristic step structure of the transition relation of the product automaton which can be aggregated into one transition. Second, the $\mathcal{A}$-attributable transition does not provide an output character in the channel component. Because $\mathcal{A}$ does not produce any further output, it doesn't behaviorally matter, if we first follow the output-less $\mathcal{A}$-attributable transition followed by the spontaneous $\mathcal{B}$-attributable transition or the other way around. Thus, from a behavioral point of view, both transitions can be aggregated into one. The aggregated transitions are described without the common input/output state.

One direct consequence of this proposition is that the consistent composition of two consecutive working systems is again a system, actually a super-system, as was already shown in [Rei10].

In the general case of network-like interactions these interactions are described by protocols:

Definition 12: A *protocol* is the outer product of different, non-embedding roles where all interactions happens through a set of channels between the roles.

The transition relation $\Delta_{\mathcal{P}} \subseteq Q_{\mathcal{P}} \times Q_{\mathcal{P}} \times I_{\mathcal{P}}^{\epsilon} \times O_{\mathcal{P}}^{\epsilon}$ of a protocol $\mathcal{P}$ has to be constructed inductively. Let $A = \{\mathcal{A}_1, \ldots, \mathcal{A}_m\}$ be a set of concrete roles and $C = \{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ be a set of channels between the roles. $Q_{\mathcal{P}} = \times Q_k$ is the set of protocol state values, $\vec{q_0}_{\mathcal{P}}$ is the initial state value, $I_{\mathcal{P}} = \times I_k$ and $O_{\mathcal{P}} = \times O_k$ are the set of input and output characters. $Acc_{\mathcal{P}} = \bigwedge Acc_k$ is the common acceptance component representing the logical conjunction of the individual acceptance conditions.

The structure of the set of input and output characters represents the fact that input and

ISBN 978-3-88579-286-4

output is achieved by shared states. Thus, the complete output character vector has as many components as all roles have output states. The same holds for the input character vector. If a state is shared by just two systems, both dimensions have to be the same. Instead of writing $\vec{i} = (\epsilon_1, \ldots, \epsilon_{k-1}, i_k, \epsilon_{k+1}, \ldots, \epsilon_n)$ indicating either an input (or output) character on the k-th channel, I write $i_{\mathcal{C}_k}$. This also clarifies to which role and which component of the role this character belongs, as this information is explicitly provided by the channel structure. In case, $i_k = \epsilon$, the channel index becomes dispensable.

To determine the elements of $\Delta_{\mathcal{P}}$, first, all spontaneous transitions which start from some reachable state of the roles (e.g. the initial state) are part of the relation. Then, assuming that a given transition t belongs to the transition relation, other elements t' of the transition relation can be constructed, depending on the role to which the transition relates.

1. Assuming that $\vec{p} \in Q_{\mathcal{P}}$ is a reachable state of the protocol and one of the roles $\mathcal{A}_k$ provides a spontaneous transition $(p_k, q_k, \epsilon, o_k)$ sending some component of $o_k \in O_k^{\epsilon}$ through channel $\mathcal{C}$, then $(\vec{p}, \vec{p}\left[\frac{q_k}{p_k}\right], \epsilon, \epsilon^m[o_k]) \in \Delta_{\mathcal{P}}$ is called a *spontaneous transition* of $\mathcal{P}$.
2. Assuming that $(\vec{p}, \vec{p}\left[\frac{q_k}{p_k}\right], \epsilon^m[i_k], \epsilon^m[o_k]) \in \Delta_{\mathcal{P}}$ with $i_k \in I_k^{\epsilon}$, $o_k \in O_k$ (and therefore $o_{\mathcal{C}_s} \neq \epsilon$) and $\mathcal{C} = (\mathcal{A}_k, r, \mathcal{A}_l, s)$, sends the r-th component of $o_k \neq \epsilon$ to the s-th component of input i_l of $\mathcal{A}_l$. If $\mathcal{A}_l$ provides an induced transition with $(p_l, q_l, i_l, o') \in \Delta_l$, then $(\vec{p}\left[\frac{q_k}{p_k}\right], \vec{p}\left[\frac{q_k}{p_k}, \frac{q_l}{p_l}\right], \epsilon[i_l], \epsilon[o']) \in \Delta_{\mathcal{P}}$ is called an *induced transition* of $\mathcal{P}$.

Directly from the definitions follows:

Proposition 2: A consistent protocol is free of deadlocks and livelocks.

4.1 Example

The protocol concept is illustrated by two example interactions. The first interaction is about the classical problem of mutual exclusion (e.g. [Lyn96]). Some process requests exclusive access to a resource in a role $\mathcal{U}$ and a resource administration process in its role $\mathcal{C}$ admits it such that $\mathcal{U}$ can now gain access to the resource and abandon it afterward.

The second interaction concerns the coordination of the different resource administration processes for administering the exclusive access to the resource. One possibility is the token protocol where all administrator processes share a single token and only the administrator process owning the token is allowed to provide its user process access to the resource.

4.1.1 First Interaction

The first interaction is displayed in Fig. 4. $\mathcal{U}$ has the state values $remn_{\mathcal{U}}$ (for remainder region), $try_{\mathcal{U}}$ (for trying region), $crit_{\mathcal{U}}$ (for critical region) and $exit_{\mathcal{U}}$ (for exit region),

while $\mathcal{C}$ has the state values $remn_{\mathcal{C}}$, $try_{\mathcal{C}}$, $crit_{\mathcal{C}}$ and $exit_{\mathcal{C}}$.

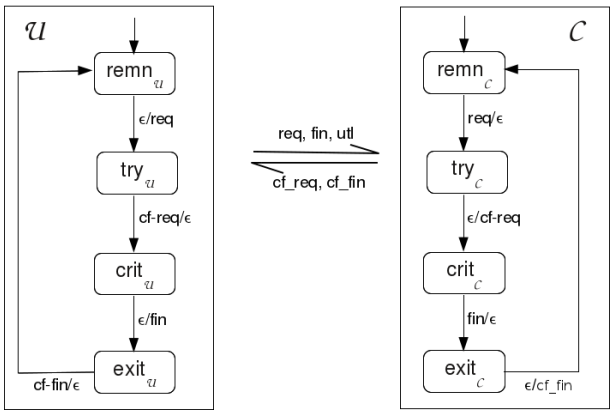

Figure 4: State diagram of the interactions between a process in a resource user role $\mathcal{U}$ and a resource administrator process in its access control role $\mathcal{C}$, managing the exclusive usage of a single resource for $\mathcal{U}$.

Initially, $\mathcal{U}$ and $\mathcal{C}$ are in the state $(remn_{\mathcal{U}}, remn_{\mathcal{C}})$. If $\mathcal{U}$ requests the resource, it notifies $\mathcal{C}$ with a req-character (for request) and transits into its $try_{\mathcal{U}}$ state. If the resource is free for use, $\mathcal{C}$ confirms the request with a *cf-req*-character, transits into its $crit_{\mathcal{C}}$ state and now provides the resource for usage. $\mathcal{U}$ receives the *cf-req*-character, transits to its $crit_{\mathcal{U}}$ state and now could use the resource (omitted). $\mathcal{U}$ releases the resource with sending a fin-character (for finalize), which is confirmed by $\mathcal{C}$ with sending a *cf-fin*-character back to $\mathcal{C}$, so that at the end of the cycle, both, $\mathcal{U}$ as well as $\mathcal{C}$ are again in their $remn$ state. It's important to note that we do not describe why $\mathcal{U}$ has to use the resource or how $\mathcal{C}$ coordinates eventually with other resource administration processes to guarantee exclusive access.

Fig. 5 illustrates the restricting effect of the channel mediated synchronization on the transition relation of this protocol.

4.1.2 Second Interaction

The token protocol is illustrated in Fig. 6 and consists of a quadruple interaction between a process in its token exchange role $\mathcal{T}$, its two neighbor processes in the same role and a timer.

Each $\mathcal{T}_i$ can take one out of three state values $abst$ (for absent), $avlb$ (for available) and $interm$ (for intermediate). The protocol has to ensure that one $\mathcal{T}_i$ is not in $abst$ where it is allowed to provide access to the resource. Initially, $\mathcal{T}_1$ is initialized with value $avlb$ while all other processes are initialized with $abst$.

$\mathcal{T}_i$ waits in the state $abst$ for reception of the token from $\mathcal{T}_{i-1}$ (or in case of $i = 1$, $\mathcal{T}_1$ waits for $\mathcal{T}_n$) to transit into state $avlb$ in case of its reception and to trigger the timer. It rests there

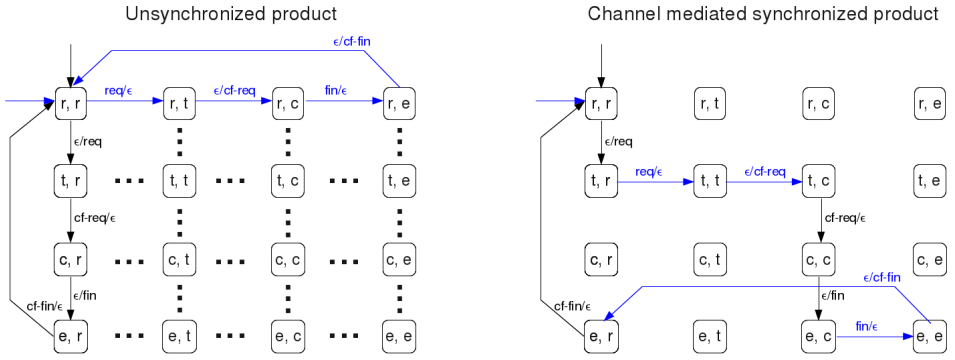

Figure 5: State diagram for the outer product of the interaction between a process in a resource user role $\mathcal{U}$ and a resource administrator process in its access control role $\mathcal{C}$. The state value names are abbreviated.

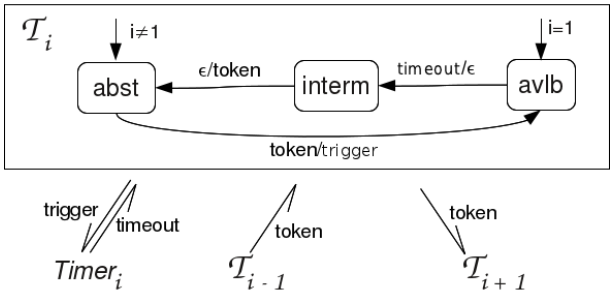

Figure 6: State diagram of the token protocol for managing exclusive access to a single resource by a resource administrator process $\mathcal{R}_i$ in its token exchange role $\mathcal{T}_i$.

until it receives the timeout which results in a transition to $interm$. Finally, $\mathcal{T}_i$ hands over the token to $\mathcal{T}_{i+1}$ (or in case of $i = n$, $\mathcal{T}_n$ hands over to $\mathcal{T}_1$) and transits back to $abst$.

5 Synthesis of Interacting Systems: Processes as the Inner Synchronized Product of Roles

I now investigate how different roles of the sames system relate to each other. Looking at the protocol definition 12, it could well be that several different roles relate to the same system.

Viewed as resulting from different projections of the same system specification, we have to figure out how the different roles become linked together. The whole approach is very similar to [Rei09] where I pursued the similarities between protocols and games. There, I

added an additional NFIOA with a "decision" alphabet as input and no output to the original NFIOA in a way that the original behavior was retained and a deterministic automaton resulted.

5.1 Synchronization

All roles together initially form an unsynchronized product automaton as defined in definition 7. To create the unsynchronized product automaton means to reconstruct a superset of the original system specification. Thereby transition elimination will bring us closer to the system specification. But, which transitions can be eliminated? First, the projection of the synchronized product automaton must result in the NFIOAs of the original roles:

Definition 13: Be $\mathcal{B}^* = \bigotimes_{k=1}^{n} \mathcal{A}_k \setminus \Delta^e$ a synchronized product automaton. A *projection* $\mathcal{A} = \pi_k(\mathcal{B}^*)$ of $\mathcal{B}^*$ onto its k-th role is defined as: $Q_{\mathcal{A}} = Q_k$, $I_{\mathcal{A}} = I_k$, $O_{\mathcal{A}} = O_k$, $q_{0\mathcal{A}} = q_{0k}$, $Acc_{\mathcal{A}} = Acc_k$, $\Delta_{\mathcal{A}} = \{(p', q', i', o') | (\vec{p}, \vec{q}, \vec{i}, \vec{o}) \in \Delta_{\mathcal{B}^*}$ and $p' = p_k, q' = q_k, i' = i_k, o' = o_k\}$

Second, the common acceptance condition must still be fulfilled:

Definition 14: Be $\mathcal{B} = \bigotimes_{k=1}^{n} \mathcal{A}_k$ an unsynchronized product automaton and $\mathcal{B}^* = \mathcal{B} \setminus \Delta^e$ a synchronized version of $\mathcal{B}$ where $\pi_k(\mathcal{B}^*) = \mathcal{A}_k$ for all $k = 1, \ldots, n$ as well as $Acc_{\mathcal{B}} = Acc_{\mathcal{B}^*}$ still hold. Then $\mathcal{B}^*$ is called *remainless synchronized* if any further transition elimination, that is any $\Delta^{e'} \supset \Delta^e$, implies either $\pi_k(\mathcal{B} \setminus \Delta^{e'}) \neq \mathcal{A}_k$ or impairs the fulfillment of the acceptance condition expressed by the $Acc_{\mathcal{B}} = Acc_{\mathcal{B}^*}$-component.

And third, the elimination must provide the coordination semantics of all related interactions.

5.1.1 Synchronization Example

Linking together both roles $\mathcal{C}$ and $\mathcal{T}$ of a resource administrator process, the coordination semantics is:

1. The transition to $crit$ requires both, the token and a request.
2. While being in $crit$, the token must not be given away.

In Fig. 7 the unsynchronized product automaton between the $\mathcal{C}$ and $\mathcal{T}$ role and the transitions, to be eliminated for remainless synchronization, is shown.

5.2 Determination

Only a DFIOA is a complete specification of a system. Remainless synchronization by itself does not necessarily lead to a DFIOA. Eventually, the resulting NFIOA has to be

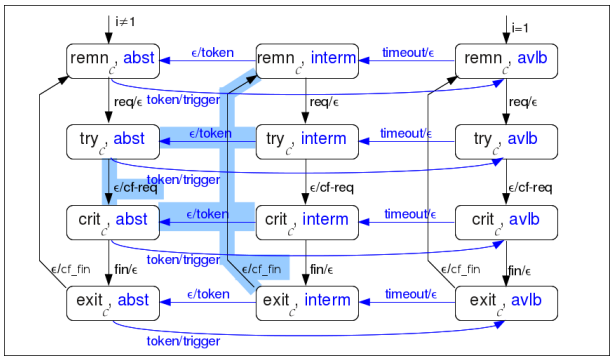

Figure 7: The unsynchronized product automaton between the $\mathcal{C}$ and $\mathcal{T}$ role is shown. The transitions which are eliminated for synchronization are marked. The (horizontal) transitions from the token protocol are colored blue, the (vertical) transitions from the mutual exclusion are colored black.

.

determined by elimination of behavioral unnecessary state values. I call this extension to [Rei09] ϵ-merge, because spontaneous transitions without any input can be used for this state value elimination (see Fig. 8).

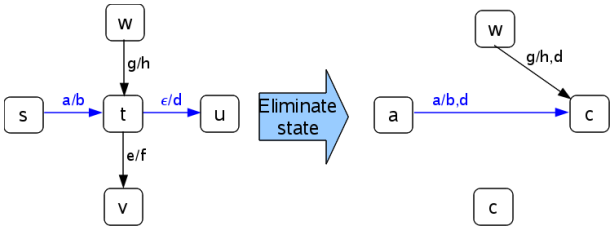

Figure 8: Merging a spontaneous transitions onto another transition in one interaction dimension might interfere with the interaction in another dimension resulting in additional loss of transitions.

If a DFIOA can be found which is behaviorally equivalent to a remainless synchronized NFIOA, I call it *completely synchronized.*

If we define a process as a system which takes part in protocol based interactions in at least two different roles, then we can say:

Proposition 3: A set of at least 2 roles of a consistent protocol for which a completely synchronized DFIOA can be constructed specifies a process.

5.2.1 Determination Example

If we merge all $\epsilon-$transitions from the remainless synchronized automaton of Fig. 7 then the completely synchronized DFIOA of Fig. 9 results.

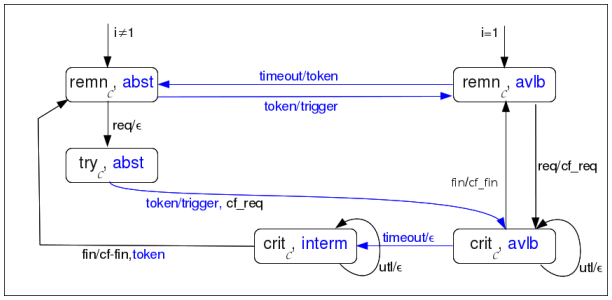

Figure 9: The completely synchronized and thereby executable deterministic process which coordinates both, the interaction to the resource processes as well as the interactions with the other resource administration processes.

.

6 Discussion

6.1 Related Work

There is no uniform well agreed system model in computer science. I just present two of them. There are many more.

For Manfred Broy [Bro95] system interact with their environment by exchanging messages through input and output channels. The relationship between the input and output messages determines what he calls the *black box behavior* or *interface*. For him, a process consists of all the actions carried out by a system. Sending and receiving messages are elementary actions. A key concept in his theory is a timed stream of messages from the set M which is represented by a mapping $s : \aleph \setminus \{0\} \rightarrow M^*$, where M^* is the set of finite sequences over M. The messages are transported by typed channels $c \in C$. The history x of such a typed channel attributes a timed stream to each channel, that is $x : C \rightarrow (\aleph \rightarrow M^*)$. The black box behavior of a component is then described by the relation between the histories of the input channels and the possible histories of the output channels, that is by a function f mapping the set of the histories of the input channels $\vec{I}$ onto the power set of histories of the output channels $2^{\vec{O}}$ of the component: $f : \vec{I} \rightarrow 2^{\vec{O}}$

The major difference to my approach seems to be the definition of the component's behavior as a relation between streams and not between single characters and state values. As the length of such a stream is finite but can be arbitrary large, it cannot be represented by a given finite state. Manfred Broy uses his system model to develop a theory of system

composition and refinement.

A group of authors around Barbara Jobstmann and Roderick Bloem use Moore machines as system representations to synthesize systems from formal specifications together with some qualitative requirements (e.g. [vEJ11, BGHJ09]) Their finite-state system $\mathcal{S} = (S, L, s_0, A, \delta, \tau)$ consists of a state set S, an input alphabet L, an output alphabet A, an initial state s_0, a transition function $\delta : L \times S \to S$ and an output function $\tau : S \to A$. With the assumption that every state is a 'safe' state, this system embeds a traditional finite automaton $\mathcal{A} = (L, S, s_0, \delta, S)$ with the property that given an input word w, the run of the system $\mathcal{S}$ on the word w is simply the run of $\mathcal{A}$ on the word w. So as a Moore automaton such a system outputs a character, then reads a letter, and then moves to the next state. It does not provide the nice interpretation of the DFIOA transition relation as being a specification of a finite system.

With respect to the relation of protocols and processes, Nirmit Desai et al. [DMCS05] follow a very similar philosophy compared to this article, by describing an agent-based approach for business processes. The terms protocol, role and process are used with almost identical meaning. To the authors, a protocol is a specification of the allowed interactions among two or more participants. The special semantics of messages within business processes is described in terms of their effects on the creation, discharge, cancellation, etc. of commitments. Thus, Nirmit Desai et al. see messages as operators over commitments. A (local) process of an agent is an executable composition of all roles the process is involved in, together, as the circumstances require, with some more business logic, necessary for determination. Different roles of a single agent become composed by stipulating composition axioms.

A protocol is called "closed" if (in my terminology) all its state values are determined from exchanged messages, and all the commitments created in the protocol can be discharged. Nirmit Desai et al. state that "a designer's goal is to obtain a closed protocol by repeated application of composition". Actually, business protocols, as they relate to networks of business interactions, are not to be expected to become closed in this sense by composition. I think what they mean is exactly what I demonstrated as process synthesis, aiming at a "closed" local process specification.

6.2 Concluding Remark

With this article, I tried to motivate the thesis that an interaction centric perspective together with a synthesis procedure for process specifications circumvents the centralization tendencies of traditional approaches and allows the construction of truly loosely coupled (finite) systems in the sense of Glassman [Gla73].

This insight has some far reaching consequences. First, following Manfred Broy [Bro95], with a system theoretic model as a base, computer science is essentially a - very interesting - sub discipline of system theory.

Second, choosing the means to describe these systems and their interactions, one must say that programming methods forcing everything into typed operations are inadequate to

ISBN 978-3-88579-286-4

describe processes based on their interactions. Although from an implementer perspective, a deterministic process is a finite system and can be described by a system operation, from an interaction perspective such a description is highly inconvenient. An interaction oriented - in contrast to an "object oriented" (which would be better called a "system oriented") - way to describe systems must adequately deal with the relation between the different roles of a process and the relation between the different roles of an interaction at the same time.

Nondeterminism is not introduced to abstract from details of implementation as Charles Hoare says [Hoa04](p.84) or only for convenience as Nancy Lynch says [Lyn96](p.257), but nondeterminism seems to be the precondition for an efficient description of interactions between many different systems, where all interactions happen on the same level of abstraction or peer-to-peer.

Third, it let us better understand the difficulties of developing software which is supposed to support business processes in the area of small and medium companies. Small and large companies more or less participate in the same external interactions of selling, buying, etc. However, it's the division of labor and therefore the additional internal interactions in a company which are decisive for the structure of the actual business processes. With an increasing number of employees, the extent of the division of labor first changes on a large scale, while reducing its effect within large companies. It is therefore to be expected that any software with preconfigured, traditionally constructed business processes will meet the needs of only a small segment of the total market of small to medium size companies. In contrast, large companies can afford the - expensive - adaptation of this software to their business needs. So, it follows that the dominance of a few ERP-vendors in the large enterprise segment in contrast to a highly segmented ERP-market for the small and medium enterprise segment might still have important technological reasons within the area of contemporary software engineering methodology.

Looking into the future, it will be interesting to study the presented formalism with its distinction between outer and inner synchronization mechanisms in more detail. Understanding better their differences and similarities may help in the design of new programming paradigms and perhaps bring up new aspects of the relation between the inside and the outside of computational systems.

Thanks: I warmly thank Hans-Jörg Kreowski for his encouragement and his critical but sympathetic comments on some earlier drafts of this article some time ago.

References

[BGHJ09] Roderick Bloem, Karin Greimel, Thomas A. Henzinger, and Barbara Jobstmann. Synthesizing robust systems. In *FMCAD*, pages 85–92. IEEE, 2009.

[Bro95] Manfred Broy. Mathematical system models as a basis of software engineering. In Jan van Leeuwen, editor, *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 292–306. Springer, 1995.

[DMCS05] Nirmit Desai, Ashok U. Mallya, Amit K. Chopra, and Munindar P. Singh. In-

teraction protocols as design abstractions for business processes. *IEEE Trans. Software Eng.*, 31(12):1015–1027, 2005.

[Far02] Berndt Farwer. ω-automata. In Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors, *Automata logics, and infinite games: a guide to current research*, pages 4–21. Springer, Berlin, Heidelberg, New York, 2002.

[Gla73] Robert. B. Glassman. Persistence and loose coupling in living systems. *Behavioral Science*, 18:83–98, 1973.

[Hoa04] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985/2004.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc. San Francisco, California, USA, 1996.

[Mea55] George H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.

[Rei09] Johannes Reich. The relation between protocols and games. In S. Fischer, E. Maehle, and R. Reischuk, editors, *Proceedings der 39. Jahrestagung der Gesellschaft für Informatik 2009 in Lübeck*, GI Lecture Notes in Informatics, pages 3453–3464. Dt. Gesellschaft für Informatik e.V., 2009.

[Rei10] Johannes Reich. Finite system composition and interaction. In Klaus-Peter Fähnrich and Bogdan Franczyk, editors, *GI Jahrestagung (2)*, volume 176 of *LNI*, pages 624–637. GI, 2010.

[Sak09] Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

[Unb93] Rolf Unbehauen. *Systemtheorie*. R. Oldenbourg Verlag München Wien, 6 edition, 1993.

[vEJ11] Christian von Essen and Barbara Jobstmann. Synthesizing systems with optimal average-case behavior for ratio objectives. In Johannes Reich and Bernd Finkbeiner, editors, *iWIGP*, volume 50 of *EPTCS*, pages 17–32, 2011.

ISBN 978-3-88579-286-4