Finite system composition and interaction*

Johannes Reich

Gerbersruhstraße 147, 69168 Wiesloch
Johannes.Reich@sophoscape.de

Abstract: In this article, it is proven for finite systems that if by reciprocal interaction, one finite system determines the action of another finite system, then both systems become subsystems of a larger supersystem.

To achieve this result, the notion of a finite system is formalized and the rules for sequential and parallel system composition are provided. The reciprocal interaction is captured by the protocol concept.

Being part of a larger supersystem is shown not to be a property which can be attributed to the system itself but depends on the context of its interaction, namely whether its interactions determine its behavior or not.

The result seems to be especially relevant in Information Systems and eCommerce, as it raises concerns about what end-to-end for example in a security context in a system theoretic sense really means. It also demonstrates the tight connection between our system and our function notion and thereby contributes to a better understanding, why approaches that rest mainly on the function notion struggle so much with network-like interacting systems and the necessary "loose coupling" in the sense of a sensible interaction with only little information about the internal state of the other actors.

1 Introduction

What is the difference between a system-to-system relation in the sense of peer-to-peer in contrast to system-to-subsystem? Shouldn't it be formally decidable, whether two interacting systems relate in one or the other way?

There are a lot of aspects, where this distinction matters. From a security point of view, it makes a big difference if a system relates to another system just as its subsystem or as another independent system. Especially in Information Systems and eCommerce, this raises concerns about what end-to-end really means. Actually, the debate about the end-to-end argument (SRC84) completely rests on a well defined system notion.

The well known OSI Reference Model (Zim80) was based on the assumption that protocols are well suited to describe peer-to-peer relations and functional interfaces are well suited to express system-to-subsystem relations. It is in full agreement with this understanding that security mechanisms like asymmetric signature and encryption harmonizes very well with protocol described interactions (Sch96). They presuppose the acquaintance of the participants in a logical sense as the endpoints of untyped channels. In contrast, there is a wide held belief in the process community that supersystem formation does not depend on the type of interaction but occurs by the interaction itself (Mil80; Mil89; Hoa04)

*published in: Klaus-Peter Fähnrich, Bogdan Franczyk (Eds), GI Lecture Notes in Informatics, Proceedings of the 40. Annual Conference of the German Gesellschaft für Informatik e.V. 2009 in Leipzig, Vol. 2, pp. 624-637.

This article ties in with the basic OSI-paradigm but shows that things are a bit more complex. The starting point in section 2 is that our notion of a system depends directly on the function notion: a system is characterized by its system operation or time evolution function mapping some input and internal state onto some output and internal state within one time step. To identify composed systems, it is therefore necessary to provide such system operations based on the interaction of other (sub-)systems. In section 3 I look at reciprocal system interaction, namely protocols. Protocols do not necessarily lead to super system formation in the sense of this article. I show that in the special case, where a system behaves deterministically within a reciprocal interaction, super system formation does occur. In section 4 , I refer to a similar approach of Jawad Drissi et al. (DYPvB98) to solve another problem in the area of system composition. In the last section, the results are discussed in a broader context.

2 Systems

Informally, a system is a set of states together with its system function. The system function governs the time-wise evolution of the system states. A state characterizes a time dependent property of a system.

I use the formal definition of a system, introduced in (Rei09), which is in line with system theory (Unb93). ϵ symbolizes the empty character in case of the sets of state values (or alphabets) I and O and $I^\epsilon = I \cup \epsilon$ and $O^\epsilon = O \cup \epsilon$. The n-fold application of the (invertible) time successor function $succ$ is written as $t +_{\mathcal{S}} n := succ^n_{\mathcal{S}}(t)$ (I may drop the subscript if the relation to the system is clear).

Definition 2.1: 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[1] describing the time evolution or system operation triggered by an update of its input parameters and updating the internal and output state in one time step for each $t \in T$:

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

Systems without internal states are called *stateless*. The fact that a system according to definition 2.1 provides a new output and internal state value one time step after it was provided some input is called the one-step I/O relation.

[1] In (Rei09) I defined this function as being partial. However, because this makes reasoning a bit more complex I use a total function in this definition here.

2.1 Composed systems

Being based on the function notion, the system notion as defined in def. 2.1 is recursive: a system can be composed from other systems. The composition rules have to guarantee that the composed systems can be viewed in exactly the same way as the parts. Especially, a time successor function has to be provided with an appropriate one-step I/O relation.

2.1.1 Sequential system composition

A simple way to compose systems sequentially is to feed one system's output into another system's input.

Definition 2.2: Two discrete systems $\mathcal{S}_i = (T_i, succ_i, Q_i, I_i, O_i, x_i, in_i, out_i, f_i)$, $i = 1, 2$ with $O_1 \subseteq I_2$ are said to be *(laggingly) concatenated* (from (τ_1, τ_2) onward) if $\tau_1 \in T_1$, $\tau_2 \in T_2$ exist such that $out_1(\tau_1 +_1 n +_1 1) = in_2(\tau_2 +_2 n)$ for all $n \geq 0$.

The following proposition holds

Proposition 2.1: Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two laggingly concatenated systems. Then, the structure $\mathcal{S} = (T, succ, Q, I, O, x, in, out, f)$ with $T = \{n \in \aleph_0 | \exists (t_1, t_2) \in T_1 \times T_2$ with $t_1 = \tau_1 +_1 n, t_2 = \tau_2 +_2 n\}$ with $succ : T \to T$ defined as $succ(t) = t + 1$, $Q = Q_1 \times Q_2$, $I = I_1$, $O = O_2$, $x = (x_1, x_2)$, $in = in_1$, $out = out_2$ is again a system according to definition 2.1

The proof follows directly from explicitly writing down the system function where $t = 0, 1, \ldots$ such that $t_1 = succ_1^t(\tau_1), t_2 = succ_2^t(\tau_2)$

$$\begin{pmatrix} out_1(t_1 +_1 1) \\ out_2(t_2 +_2 1) \\ x_1(t_1 +_1 1) \\ x_2(t_2 +_2 1) \end{pmatrix} = \begin{pmatrix} f_1^{ext}\,(in_1(t_1), x_1(t_1)) \\ f_2^{ext}\,(f_1^{ext}(in_1(t_1), x_1(t_1)), x_2(t_2)) \\ f_1^{int}\,(in_1(t_1), x_1(t_1)) \\ f_2^{int}\,(f_1^{ext}(in_1(t_1), x_1(t_1)), x_2(t_2)) \end{pmatrix}$$

I also say that the two laggingly concatenated systems $\mathcal{S}_1$ and $\mathcal{S}_2$ form a *sequential system* $\mathcal{S}$ and write $\mathcal{S} = \mathcal{S}_2 \circ \mathcal{S}_1$.

The second system therefore is always one step behind: $out_1(\tau_1 +_1 1) = in_2(\tau_2)$, $out_1(\tau_1 +_1 2) = in_2(\tau_2 +_2 1)$, etc. The difference between the inner and outer time structure of sequentially composed systems is expressed in the next proposition.

Proposition 2.2: A time step in a sequential system $\mathcal{S} = \mathcal{S}_2 \circ \mathcal{S}_1$ takes as long as the time steps of system $\mathcal{S}_1$ and $\mathcal{S}_2$ together.

Proof: Let $\tau_1 \in T_1$, $\tau_2 \in T_2$ be such that $out_1(\tau_1 +_1 n +_1 1) = in_2(\tau_2 +_2 n)$ for all $n \geq 0$. Then with $t_{1,2} = succ_{1,2}^t(\tau_{1,2})$, the evaluation of $f_{\mathcal{S}}$ at time t requires first to evaluate $(out_1(t_1 +_1 1), x_1(t_1 +_1 1)) = f_1(in_1(t_1), x_1(t_1))$ providing the input$in_2(t_2) = out_1(t_1 +_1 1)$ for the corresponding time step of system 2 as $(out_2(t_2 +_2 1), x_2(t_2 +_2 1)) = f_2(in_2(t_2), x_2(t_2))$.

The extension to more then two concatenated systems is straight forward.

2.1.2 Parallel system composition

Parallel processing systems can also be viewed as one system if there is a common input to more than one successor system, the parallel processing operations are independent and therefore well defined and both system operations synchronously finish their state evaluation in a single common step.

Definition 2.3: Two discrete systems $\mathcal{S}_i = (T, succ_i, Q_i, I_i, O_i, x_i, in_i, out_i, f_i), i = 1, 2$ with $I_1 = I_2$ are said to *work in parallel* (from (τ_1, τ_2) onward) if $x_1 \neq x_2$ $\tau_1 \in T_1$, $\tau_2 \in T_2$ exist such that $in_1(\tau_1 +_1 n) = in_2(\tau_2 +_2 n)$ for all $n \geq 0$.

Proposition 2.3: Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two parallel working systems. Then, the structure $\mathcal{S} = (T, succ, Q, I, O, x, in, out, f)$ defined by $T = \{n \in \aleph_0 | \exists (t_1, t_2) \in T_1 \times T_2$ with $t_1 = succ_1^n(\tau_1), t_2 = succ_2^n(\tau_2)\}$ with $succ : T \to T$ defined as $succ(t) = t+1$, $Q = Q_1 \times Q_2$, $I = I_1 = I_2$, $O = O_1 \times O_2$, $x = (x_1, x_2)$, $in = in_1 = in_2$ for $t_1 \geq \tau_1$ and $t_2 \geq \tau_2$ as well as $out = (out_1, out_2)$ is again a system according to definition 2.1

Again, the proof results directly from the definitions.

To work in parallel therefore means that the input state of two otherwise independent systems is identical (from a given point in time), whereby two parallel working systems can be viewed as a supersystem only if they perform their time step in a synchronized manner.

I also say that the two parallel working systems $\mathcal{S}_1$ and $\mathcal{S}_2$ form a *parallel system* $\mathcal{S}$ and write $\mathcal{S} = \mathcal{S}_2 || \mathcal{S}_1$.

The extension to more than two parallel working systems is again straight forward.

2.1.3 General system composition and subsystem relation

With sequential and parallel system composition defined separately, it is interesting to see which rules their combination obey.

As the application of an operation f preceding two parallel operations g and h is equivalent to the parallel execution of g after f and h after f, together with some bookkeeping on time steps and i/o-states, the following proposition is easy to prove:

Proposition 2.4: For three systems $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{S}$ where $\mathcal{P}_1$ and $\mathcal{P}_2$ are parallel systems and $\mathcal{P}_1 || \mathcal{P}_2$ and $\mathcal{S}$ are sequential systems, the right distribution law holds: $(\mathcal{P}_1 || \mathcal{P}_2) \circ \mathcal{S} = (\mathcal{P}_1 \circ \mathcal{S}) || (\mathcal{P}_2 \circ \mathcal{S})$.

As in general sequential system composition is non-commutative ($\mathcal{S}_1 \circ \mathcal{S}_2 \neq \mathcal{S}_2 \circ \mathcal{S}_1$), it follows that the left distribution law does not hold: $\mathcal{S} \circ (\mathcal{P}_1 || \mathcal{P}_2) \neq (\mathcal{S} \circ \mathcal{P}_1) || (\mathcal{S} \circ \mathcal{P}_2)$.

Equipped with a notion of system composition, the concept of super- or subsystem can be defined.

Definition 2.4: Let $\mathcal{S}$ be composed from systems $\mathcal{U}_k$ ($k = 1, \ldots, n$) e.g. according to definition 2.2 or 2.3. Then $\mathcal{S}$ is called the *supersystem* of the $\mathcal{U}_k$ and the $\mathcal{U}_k$ are the *subsystems* of $\mathcal{S}$.

2.2 Prescribing system behavior by I/O automata

The formal definitions of a system behavior and its description is again taken from (Rei09).

Definition 2.5: A *behavior* or *(observable) trace* of a finite system (or parts of it) is described by the sequence of incoming and outgoing characters $(in_0, out_0, in_1, out_1, \ldots)$. As ϵ-values are viewed as empty characters, they do not contribute to the behavior.

The behavior of a finite system can be prescribed with nondeterministic finite I/O automata. In the sense that the unobservable initial state value is part of the automata structure, "prescribing" is more than just "describing" observable behavior.

Definition 2.6: A *nondeterministic finite I/O automaton (NFIOA)* is defined by 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 where at least one of both is non empty, 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.

In case that for each $(p, i) \in Q \times I$ $(i \neq \epsilon)$ there is at most one transition $(p, q, i, o) \in \Delta$ then Δ defines a function $\delta : Q \times I \rightarrow Q \times O^\epsilon$ with $(q, o) = \delta(p, i)$. We then have a deterministic automaton or *DFIOA*. Actually a DFIOA is a Mealy automaton (Mea55).

Definition 2.7: A NFIOA $\mathcal{A} = (Q_\mathcal{A}, I_\mathcal{A}, O_\mathcal{A}, q_0, Acc, \Delta)$ *prescribes* the behavior of a projection of a system $\mathcal{S} = (T, succ, Q_\mathcal{S}, I_\mathcal{S}, O_\mathcal{S}, x, in, out, f)$, if a projection function $\pi = (\pi_Q, \pi_I, \pi_O) : Q_\mathcal{S} \times I_\mathcal{S}^\epsilon \times O_\mathcal{S}^\epsilon \rightarrow Q_\mathcal{A} \times I_\mathcal{A}^\epsilon \times O_\mathcal{A}^\epsilon$ 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 acceptance component Acc is traditionally given as a set of state values, as a set of state value sets or as a function from the set of state values to a finite set of natural numbers (Far02), depending on the "computational" model behind the automaton. For finite computation, a sequence of inputs is declared as accepted if the automaton ends up in a configuration considered to be final. For an infinite computation, more complex acceptance conditions have been defined like Büchi, Muller, Rabin or Street acceptance.

Usually, Q is named as the set of 'states' in the automata literature. However, with relation to the system concept as defined in section 2 it is more consistent to name it the set of 'state values' as it will denote the set of values a state can take.

3 Reciprocal system interactions and supersystem formation

As we will see, there are important system interactions that do not necessarily lead to super system formation. The description of these interactions are well known in informatics and are called "protocols" (Hol91; vB75; BZ83).

3.1 Protocols and channels

There is no uniform way to describe protocols in the literature. One class of descriptions achieve the coupling by synchronized execution of complementary operations (vB78). In this case, a send action of the sender system and a receive action of the receiver system are "directly coupled", as Gregor v. Bochmann says, and executed as one action. Another class of descriptions achieve the coupling by separating sending and receiving actions as different transitions explicitly by a channel (BZ83).

The way the system parts are described influence the messages' and channels' role in the structure of the protocol. In the former case, assuming stateless data transport, the notion of a message and channel disappears altogether and the system's send and receive actions become directly coupled in the sense of Gregor v. Bochmann. He calls this the "empty medium abstraction". In the latter case, channels are at least from a modeling perspective always stateful, requiring an extra analysis, whether states with empty channels are reachable (so called "stable states" according to Brand and Zafiropulo (BZ83)).

My idea to describe system interactions is neither to use synchronized actions, nor to separate sending and receiving actions by channels, but just to assume as I did with system composition that the output state of one system is the input state of another system. In the end, the presented approach is similar to Gregor v. Bochmann's "empty medium abstraction" as it relates the output of one system directly to the input of another system.

To describe the interaction of several systems, each described as an NFIOA according to definition 2.6, we additionally have to describe how the systems are connected, that is which output and input state of which system are identical for which transition. The following protocol definition is extended to more than two participants compared to (Rei09) where I used it to describe the tight relation between games and protocols. As the participating systems are enumerated, the system subscript is replaced by the enumeration subscript to simplify notation.

Definition 3.8: A *protocol* is defined by $\mathcal{P} = (S, Q, \vec{q_0}, I, O, Acc, \Omega, \Delta)$, with $S_\mathcal{P} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ is the set of participants described by NFIOAs, $Q_\mathcal{P} = \times Q_k$ is the set of protocol state values, $\vec{q_0}_\mathcal{P}$ is the initial state value, $I_\mathcal{P} = \bigcup I_k$ and $O_\mathcal{P} = \bigcup O_k$ are the set of characters, and $Acc_\mathcal{P} = \bigwedge Acc_k$ is the common acceptance component where all acceptance conditions are combined by logical conjunction. The receiver determination $\Omega_\mathcal{P} \subseteq \Delta_\mathcal{P} \times S_\mathcal{P}$ lists the participant to which the output character is sent [2].

[2]The separation between the receiver determination and the protocol transition relation has two advantages: first, the formalism is changed very little if the interaction between only two systems is considered as is done in (Rei09). Second, it can be more easily extended to allow for more than one output character in a single system

The transition relation is $\Delta_{\mathcal{P}} \subseteq Q_{\mathcal{P}} \times Q_{\mathcal{P}} \times I^{\epsilon}_{\mathcal{P}} \times O^{\epsilon}_{\mathcal{P}} \times S_{\mathcal{P}}$. Its elements are determined inductively from the transition relations of the participants. First, all spontaneous transitions which start from the participants initial state values (or any other reachable 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 participant 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 participants $\mathcal{A}_k$ provides a spontaneous transition (p_k, q_k, ϵ, o) with $o \in O^{\epsilon}_k$, then $(\vec{p}, \vec{p}\left[\frac{q_k}{p_k}\right], \epsilon, o, \mathcal{A}_k) \in \Delta_{\mathcal{P}}$ is called a *spontaneous transition* of $\mathcal{P}$.

2. Assuming that $(\vec{p}, \vec{p}\left[\frac{q_k}{p_k}\right], i, o, \mathcal{A}_k) \in \Delta_{\mathcal{P}}$ with $i \in I^{\epsilon}_k$, $o \in O_k$ (and therefore $o \neq \epsilon$), we can by means of Ω determine the receiver $\mathcal{R}_l$ for each component o_l of the output character o of the transition. If $\mathcal{R}_l$ provides an induced transition with $(p_{\mathcal{R}_l}, q_{\mathcal{R}_l}, o_l, o') \in \Delta_{\mathcal{R}_l}$, then $(\vec{p}\left[\frac{q_k}{p_k}\right], \vec{p}\left[\frac{q_k}{p_k}, \frac{q_{\mathcal{R}_l}}{p_{\mathcal{R}_l}}\right], o_l, o', \mathcal{R}_l) \in \Delta_{\mathcal{P}}$ is called an *induced transition* of $\mathcal{P}$.

Looking at the receiver determination Ω, we see that in each element $\omega = (t, \mathcal{R}) \in \Omega$ it contains a "receiver" system $\mathcal{R}$ and (within its transition $t = (p, q, i, o, \mathcal{S})$) a "sender" system $\mathcal{S}$. I therefore define:

Definition 3.9: An *(abstract) channel* is the class of all elements of the receiver determination Ω with the same sender and the same receiver system.

I thereby partition the receiver determination into equivalence classes or channels characterized by sender and receiver. Channels in this sense reflect the fact that the relation between the output states of the senders and the input states of the receivers is not necessarily static, but can be dynamic within an interaction, as a system may "send" by one and the same output state characters to different receivers in different transitions. This channel concept is obviously ignorant against any transported content.

As protocols obviously are pretty complex entities, I would like to mention some of their most important and already well known properties. Like Daniel Brand and Pitro Zafiropulo (BZ83) I define:

Definition 3.10: A protocol $\mathcal{P}$ is called *well formed* if for every transition $(\vec{p}, \vec{p}\left[\frac{q_k}{p_k}\right],$ $i, o, \mathcal{A}_k) \in \Delta_{\mathcal{P}}$ with $i \in I^{\epsilon}_k$, $o \neq \epsilon$ and each determined receiver $\mathcal{R}$ there exists an induced transition of $\mathcal{R}$

Being well formed, a protocol guarantees that every information exchanged between two participants is processed.

Definition 3.11: A well formed protocol $\mathcal{P}$ is called *consistent* if for each reachable protocol state value $\vec{p} \in Q_{\mathcal{P}}$ there exists a (finite) path such that the acceptance conditions of all participants hold.

A consistent protocol is free of deadlocks and livelocks. In the case of finite automata with

transition, which is not needed in this article..

the acceptance component of a set of final states $F_{\mathcal{P}} = \times F_i$, a consistent protocol always provides a finite path leading to a final protocol state.

A consistent protocol is self contained in a sense that it first needs no additional input beyond the output produced by the participants themselves to fulfill the acceptance condition and second does not provide any further output to unmentioned participants as all output is being processed. In this sense it is by itself no I/O-automaton anymore.

Someone has to start the interaction. Thus, the following simple proposition holds:

Proposition 3.5: If being in the initial state does not fulfill the acceptance condition of the protocol, then a consistent protocol has at least a single participant offering a spontaneous transition from an initial state value.

3.2 Protocol coupling to a system described as a DFIOA

The life of Edsger W. Dijkstra's humble programmer (Dij72) would be much easier if she could separate systems by their interactions as simple as I may have suggested with my distinction between "system interaction" and "system composition".

Unfortunately, this is not the case. Although protocol based interactions do not necessarily lead to super system formation, sometimes they do. Namely, if two systems interact such that within their interaction one of them can be described by a DFIOA. Then both systems are subsystems of a larger system. That is, the following proposition holds:

Proposition 3.6: Let $\mathcal{S}$ be a system determined by more than one consistent interaction and described by a DFIOA $\mathcal{D}$. It especially interacts with another system $\mathcal{U}$ described as a DFIOA $\mathcal{B}$ by the consistent protocol $\mathcal{P}(\mathcal{A}, \mathcal{B})$ with a set of final states as acceptance component, where $\mathcal{A}$ is an NFIOA describing only a projection of $\mathcal{S}$. Then $\mathcal{S}$ and $\mathcal{U}$ are subsystems of a larger system $\mathcal{T}$.

Proof: As $\mathcal{U}$ is deterministically described by DFIOA $\mathcal{B}$, the transition relation $\Delta_{\mathcal{B}}$ of $\mathcal{B}$ defines the system operation $f_{\mathcal{U}} : Q_{\mathcal{B}} \times I_{\mathcal{B}} \to Q_{\mathcal{B}} \times O_{\mathcal{B}}^{\epsilon}$ with $(o, x') = f_{\mathcal{U}}(i, x)$. Because the interaction between $\mathcal{U}$ and $\mathcal{S}$ is described by a consistent protocol, $\mathcal{U}$ reacts only to input of $\mathcal{S}$ and has no spontaneous (ϵ-) transitions.

$\mathcal{S}$ is interacting with $\mathcal{U}$ by protocol $\mathcal{P}(\mathcal{A}, \mathcal{B})$ and with other systems, denoted by $\sim \mathcal{U}$, by other protocols. The set of input and output characters of $\mathcal{S}$ therefore is the set union of the input and output characters from $\mathcal{U}$ and $\sim \mathcal{U}$: $I_{\mathcal{S}} = I_{\mathcal{S}}|_{\mathcal{U}} \cup I_{\mathcal{S}}|_{\sim \mathcal{U}}$ and $O_{\mathcal{S}} = O_{\mathcal{S}}|_{\mathcal{U}} \cup O_{\mathcal{S}}|_{\sim \mathcal{U}}$.

To prove the proposition, I construct the supersystem $\mathcal{T}$ of $\mathcal{S}$ and $\mathcal{U}$. The internal state of $\mathcal{T}$ is composed of the internal state of $\mathcal{S}$ and $\mathcal{U}$: $Q_T = Q_{\mathcal{S}} \times Q_{\mathcal{U}}$ and $x_{\mathcal{T}} = (x_{\mathcal{S}}, x_{\mathcal{U}})$.

For the time evolution of the supersystem $\mathcal{T}$, two cases have to be distinguished: first, there is no connection between $\mathcal{S}$ and $\mathcal{U}$ and therefore no transmitted characters between both

and second, according to the receiver determination, there is such a connection.

In the first case, $succ_{\mathcal{T}}(\tau_{\mathcal{T}}) = \tau_{\mathcal{T}} + 1$ which corresponds to $(succ_{\mathcal{S}}(\tau_{\mathcal{S}}), \tau_{\mathcal{U}})$ and $f_{\mathcal{U}} = (f_{\mathcal{S}}, id)$ where id is the identity function.

In the second case, the connection shall be established where the supersystem $\mathcal{T}$ is at its time step $\tau_{\mathcal{T}}$, $\mathcal{S}$ at its time step $\tau_{\mathcal{S}}$ and $\mathcal{U}$ is at its time step $\tau_{\mathcal{U}}$. The relevant transition of $\mathcal{S}$ (which was not induced by a character from $\mathcal{U}$) sends a character to $\mathcal{U}$, initiating an interaction chain between $\mathcal{S}$ and $\mathcal{U}$. Each such terminated interaction chain increments the time successor function of the supersystem $\mathcal{T}$ by one.

This interaction chain terminates under two circumstances

1. $\mathcal{U}$ transits such that no character is sent back to $\mathcal{S}$.
2. $\mathcal{S}$ transits because of an incoming character by $\mathcal{U}$ but does not sent any character back.

It actually always terminates, because on the one hand, the interaction between $\mathcal{S}$ and $\mathcal{U}$ is consistent and on the other hand, $\mathcal{S}$ is itself completely determined by $\mathcal{D}$. Thereby, any class of ambiguous transitions within $\mathcal{P}(\mathcal{A}, \mathcal{B})$, where a character from $\mathcal{U}$ seems to elicit more than one transitions within $\mathcal{S}$ is disambiguated within $\mathcal{D}$ by inputs of other systems. The interaction chain therefore terminates either at a disambiguated branching within $\mathcal{S}$ or at the latest in a state compatible with a terminal states of $\mathcal{P}$.

The system function of the super system results from the successive application of the system functions of $\mathcal{S}$ and $\mathcal{U}$ during the interaction chain, similar to subsection 2.1.1.

In the same way, it should be possible to prove that a system composition can result from an interaction of three participants involved in pairwise interactions, although none of the interacting partners is described as deterministic with respect to their pairwise interactions. This should be the case if the transitions of one of the participants are entirely determined by the interactions with the other two participants.

4 Related work

Jawad Drissi et al. (DYPvB98) use a related approach to consider another problem of system composition : given two deterministic I/O finite state machines (FSM) $\mathcal{A}$ and $\mathcal{C}$ that represent respectively the behavior of a desired system and the behavior of an existing (sub-)system, does there exist an I/O FSM X which, combined with C, exhibits the behavior of A? It is interesting to see the similarities and differences of their approach. Drissi et al. define an I/O nondeterministic FSM as a 5-tuple (Q, I, O, h, q_0) where Q is a non-empty finite set of state values, I, O are non-empty finite set of input and output symbols, q_0 is an initial state and h is a "total behavior" function $h : Q \times I^{\epsilon} \rightarrow \mathcal{P}(Q \times O^{\epsilon}) \setminus \{\emptyset\}$. Such an FSM becomes deterministic when $|\{h(q, i)\}| = 1$ for all $(q, i) \in Q \times I$. That is, they do not consider the acceptance component to be part of the behavior description. More importantly, they assume that transitions without input cannot change the internal state, or formally $h(q, \epsilon) = (q, \epsilon)$ for all $q \in Q$. They describe systems as labeled transitions

systems (LTS) by quadruples (Q, A, Δ, q_0) where Q is a non-empty finite set of states, q_0 is an initial state, A is a non-empty finite set of actions and $\delta \subseteq Q \times (A \cup \{\tau\}) \times Q$ with τ is a non-observable action. They say that a deterministic LTS *corresponds* to an FSM, if the trace of its observable actions is also generated by the FSM. The observable actions of the LTS therefore corresponds to I/O-characters of the I/O-FSM. However, the silent action τ does not seem to correspond to the empty character ϵ, as according to Drissi et al. the empty character cannot change the internal state, while the empty action probably can.

The one-step I/O-relation is explicitly modeled as an environment FSM, guaranteeing that a next external input is only submitted to the system after it has produced an external output.

They make no explicit reference to any state in the sense of this article. Instead, the coupling of multiple LTS is exclusively obtained by coincident naming. An output character of one LTS that is an input character of a coupled LTS is received by this coupled LTS. As a result, a transition like (q, q, a, a) with no other accepting transition for character a originating from q already leads to a live lock.

5 Discussion

In this article, I combined the descriptions of systems with the description of reciprocal interactions in a network context. With section 3.2, I have shown that super system formation occurs within a reciprocal interaction if a participant interacts deterministically.

First, this approach illustrates the complexity of our system and system composition notion. It shows that there is more to supersystem formation than mere interaction, as for example Robin Milner (Mil80; Mil89) or Charles Hoare (Hoa04) still thought. Supersystem formation in the sense of this article requires the presence of a respective time successor function. Whether such a time successor function exist or not isn't always obvious. It underlines that nondeterministic interactions and therefore nondeterministic automata have an important role in software engineering (e.g. in contrast to Helmut Balzert (Bal01), p.323).

One basic idea of this approach was to base the definition (2.1) of a system on an inside-outside distinction, namely to distinguish between externally accessible I/O-states and externally inaccessible or internal states. But, the system notion has like the function notion a recursive structure: systems can be combined to become subsystems in some larger supersystems. Then, what was formerly an external state with respect to some subsystem may become internal with respect to the supersystem. Thus, it is not a priori clear, whether a state is inside or outside a system.

Being a subsystem of some larger supersystem is not a property which can be attributed to the system itself but - as shown - depends on the context of its interaction, namely whether its interactions determine its behavior or not. In the case of a reciprocal interaction, a system becomes a subsystem if its behavior is entirely determined by its interaction "partner". Such an interaction results in a recursive system where some output becomes the input of a next time step, until the chain of recursive interaction terminates. This suggests that sequential and parallel system composition does not exhaust the space of system composition operations. Taking the relation between our system and function notion even further, it

could be speculated that as for recursive functions in the sense of Stephen Kleene (Kle36), the composition rules for systems comprises similar recursive rules.

As already mentioned in the introduction, well defined system borders are also a prerequisite for the ”end-to-end” argument (SRC84). In their article, Jerome H. Saltzer, David P. Reed and David D. Clark stated that ”choosing the proper boundaries between functions is perhaps the primary activity of the computer system designer” and suggested as a guiding design principle for placement of function among the modules of a distributed system that ”functions placed at low levels of a system may be redundant or of little value compared with the cost of providing them at that low level”. With operations representing only send or receive semantics and thereby not determining subsystem relations on the one hand and with protocol based interactions which in the deterministic case do determine subsystem relations, the questions where systems logically end may be difficult to decide.

Second, it shows that the borders of finite (computer) systems are theoretically drawn by logic and sometimes practically restricted by physics. The borders between the internal ”deterministic” parts of an engineered system and the parts that are coupled to it by non-deterministic interactions are drawn somewhat discretionary but are limited - from an engineering perspective - upon the required level of determinism. Effectively, determinism is fiction. In a recent study Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber (SPW09) measured the number of correctable errors of a DRAM-DIMM as nearly 4000 per year or 10 per day. Accordingly, the hardware manufacturer have to use error correction mechanism to bring this rate down below an acceptable limit. On the other hand, unreliable communication can make reaching consensus impossible, as the coordinated attack problem shows (Gra78), rendering enforcement of a one step semantic of the ”distributed” system impossible . So the faster and more reliable communication is, the more it allows to extend the deterministic fiction to control even ”remote” systems.

Third, the demonstrated tight connection between our system and our function notion contributes to a better understanding, why approaches that rest mainly on the function notion struggle so much with network-like interacting systems and the necessary ”loose coupling”. One example would be functional programming (Hud89), other examples are the many distributed object models like CORBA (Vin97), Java RMI (WRW96), COM/DCOM (Rog98), SOA (SN96), etc. With the term ”loose coupling” the degree of dependency between systems is denoted. Although this meaning is more or less common sense, the term is not used uniformly in the computer science literature. Within the object and service oriented communities, it is often stated that ”loose coupling” of systems is created by abstraction (Kay03). Though abstraction decouples from any specific implementation, it does not decouple from the particular implementation which is used during runtime and certainly not from the underlying logical mapping. Actually, this would imply to name a tire ”loosely coupled” to a car, just because it can be changed easily. It seems to me that this kind of ”loose coupling” is more concerned with the effort to replace a subsystem.

In contrast, Robert B. Glassman (Gla73) defines interacting systems [like cells, organisms or even societies] as being ”loosely coupled” if they have few states in common: several (autonomous) systems may sensibly interact without supersystem formation, but only provide very little information to each other compared to their private state. Determining the behavior of another system completely initiates supersystem formation and removes any

loose coupling in this second sense. This renders all object models in the object oriented sense which can be represented as deterministic I/O-automata to subsystems of some larger supersystem.

Last but not least, The approach to combine the descriptions of systems with the description of reciprocal interactions in a network context also suggests interesting directions of future research for example in program semantics. With some simplification, semantics of imperative programs are traditionally understood as "axiomatic"(Flo67; Hoa69), "denotational" (SS71) or "operational" (Plo81). "axiomatic" semantics deals more with the correctness-relation between specifications - or the intended meaning of a program - and the actually specified system itself. "denotational" semantics relates the meaning of a computer program to its mapping from input to output in the mathematical sense of a function. In "operational" semantics, a program together with the data is viewed as a description of a transition system and focuses on the operations a system can perform. With this article, the system notion comes more to the front. It would be interesting to further explore the possibility to view imperative programming languages as a combination of a system description calculus together with aspects to deal with the receiver determination problem of network-like interactions. Thereby the described entities would be systems and not something to simulate systems as was thought in the beginning of object orientation (DN66). Such a system description calculus would be complete, if every system (of a given sort) could be described and it would be sound, if every described system would indeed be a system under the appropriate interpretation.

Acknowledgments: I thank Prof. Hans-Jörg Kreowski and both referees for their very helpful comments

References

[Bal01] Helmut Balzert. *Lehrbuch der Softwaretechnik*. Spektrum Akademischer Verlag Heidelberg, Berlin, 2 edition, 2001.

[BZ83] Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *J. ACM*, 30(2):323–342, 1983.

[Dij72] Edsger W. Dijkstra. The humble programmer. *Commun. ACM*, 15(10):859–866, 1972.

[DN66] Ole-Johan Dahl and Kristen Nygaard. SIMULA - an ALGOL-based simulation language. *Commun. ACM*, 9(9):671–678, 1966.

[DYPvB98] J. Drissi, N. Yevtushenko, A. Petrenko, and G. v. Bochmann. On the design of a submodule based on the input/output FSM model. Technical Report 1120, University of Montreal, 1998.

[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.

[Flo67] Robert W. Floyd. Assigning Meaning to Programs. In J.T. Schwartz, editor, *Proceedings of American Mathematical Society Symposia in Applied Mathematics*, volume 19, pages 19–32. A.M.S., 1967.

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

[Gra78] Jim Gray. Notes on Data Base Operating Systems. In *Operating Systems, An Advanced Course*, pages 393–481, London, UK, 1978. Springer-Verlag.

[Hoa69] Charles A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM,*, 12(10):576–585, 1969.

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

[Hol91] Gerard J. Holzmann. *Design and validation of computer protocols*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[Hud89] Paul Hudak. Conception, evolution, and application of functional programming languages. *ACM Computing Surveys*, 21(3):359–411, 1989.

[Kay03] Doug Kaye. *Loosely Coupled: The Missing Pieces of Web Services*. RDS Press, 1 edition, 2003.

[Kle36] Stephen Kleene. General recursive functions of natural numbers. *Mathematische Annalen*, 112(5):727–742, 1936.

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

[Mil80] Robin Milner. *A Calculus of Communicating Systems*. Springer, Berlin, Heidelberg, New York, 1980.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Technical report, Aarhus University Computer Science Department, 1981. DAIMI FN-19.

[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.

[Rog98] Dale Rogerson. *Inside COM*. Microsoft Press, 1998.

[Sch96] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, 1996.

[SN96] R. W. Schulte and Y. V. Natis. *"Service-Oriented" Architectures, Part 1*. SPA-401-068, Gartner Group, 1996.

[SPW09] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. DRAM errors in the wild: a large-scale field study. In *SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, pages 193–204, New York, NY, USA, 2009. ACM.

[SRC84] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-To-End Arguments in System Design. *ACM Transactions on Computer Systems, Vol. 2, No. 4, November1984, Pages 277-288.*, 2(4):277–288, 1984.

[SS71] Dana Scott and Christopher Strachey. Toward a mathematical semantics for computer languages. In J. Fox, editor, *Proc. Symp. Computers and Automata*. Poytechnic Inst. of Brooklyn Press, 1971. Also Technical Monograph PRG-6, Programming Research Group, Oxford University.

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

[vB75] Gregor von Bochmann. Communication protocols and error recovery procedures. In *Applications, Technologies, Architectures, and Protocols for Computer Communication. Proceedings of the 1975 ACM SIGCOMM/SIGOPS workshop on Interprocess communications*, pages 45–50. ACM, 1975.

[vB78] Gregor von Bochmann. Finite State Description of Communication Protocols. *Computer Networks*, 2:361–372, 1978.

[Vin97] Steve Vinoski. CORBA: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communication*, 35(2):46 – 55, 1997.

[WRW96] Ann Wollrath, Roger Riggs, and Jim Waldo. A distributed object model for the javaTM system. In *COOTS'96: Proceedings of the 2nd conference on USENIX Conference on Object-Oriented Technologies (COOTS)*, pages 17–17, Berkeley, CA, USA, 1996. USENIX Association.

[Zim80] Hubert Zimmermann. OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, COM-28(4):425–432, 1980.