Compositionality of Safe Communication in Systems of Team Automata

Maurice H. ter Beek[1], Rolf Hennicker[2], and Jetty Kleijn[3]

[1] ISTI–CNR, Pisa, Italy
[2] Ludwig-Maximilians-Universität München, München, Germany
[3] LIACS, Leiden University, Leiden, The Netherlands

Abstract. We study guarantees for safe communication in systems of systems composed of reactive components that communicate through synchronised execution of common actions. Systems are modelled as (extended) team automata, in which, in principle, any number of component automata can participate in the execution of a communicating action, either as a sender or as a receiver. We extend team automata with synchronisation type specifications, which determine specific synchronisation policies fine-tuned for particular application domains. On the other hand, synchronisation type specifications generate communication requirements for receptiveness and responsiveness. We propose a new, liberal version of requirement satisfaction which allows teams to execute arbitrary intermediate actions before being ready for the required communication, which is important in practice. Then we turn to the composition of systems and show that composition behaves well with respect to synchronisation type specifications. As a central result, we investigate criteria that ensure the preservation of local communication properties when (extended) team automata are composed. This is particularly challenging in the context of weak requirement satisfaction.

1 Introduction

We study guarantees for safe communication in systems of systems of interconnected, reactive components that communicate through synchronised execution of shared actions. We focus on the prevention of output actions from not being accepted (i.e. no message loss) and input actions from not being provided (i.e. no indefinite waiting). The lack of safe communication in modular system models may reveal design problems before implementation. To guarantee safe communication in such models, a characterisation for compatibility of two component interactions free from message loss and indefinite waiting was given in [15] and lifted to n-ary interactions in multi-component systems in [16]. Both approaches support compatibility for synchronous communication. A first exploration on how to generalise compatibility notions to arbitrary synchronisation policies was performed in [7] in the framework of team automata.

Team automata [8, 23] are a transition system model for systems of reactive components differentiating input (passive), output (active), and internal (privately active) actions, in the line of I/O automata [19, 29], interface automata [20, 21], component-interaction automata [14], modal I/O automata [27], and contract automata [3,4]. The distinguishing feature of team automata is their very loose nature of synchronisation according to which, in principle, any number of component automata can participate in the synchronised execution of a shared communicating action, either as a sender or as a receiver. Team automata can determine specific synchronisation policies defining when and which actions are executed and by how many components.

Conditions for safe communication in terms of receptiveness and responsiveness were considered in [2,12,18,22] for (web) services and in [6,7,10,16] for team automata. Output actions not accepted as input by some component are considered as message loss or as unspecified receptions [13, 22]. If any (autonomously chosen) output action is accepted, we call this receptiveness [7]. Orthogonally, we recognise indefinite waiting for input to be received in the form of an appropriate output action provided by another component [15]. Since input relies on external choice, it is sufficient if only one of the enabled input actions is responded to (by other components), which we call responsiveness [6].

In [6], a representative set of synchronisation types was defined to classify synchronisation policies (e.g., binary communication, multi-cast communication, full synchronisation) realisable in team automata in terms of ranges for the number of sender and receiver components that can participate in a system communication. Moreover, a generic procedure was provided to derive requirements for receptiveness and responsiveness for each synchronisation type. Communication safety of team automata was expressed in terms of their compliance with receptiveness and responsiveness requirements. A team automaton is said to be compliant with a given set of communication requirements if in each reachable state of the team the desired communications can immediately occur; if the communication can eventually occur after some internal actions have been performed, it is said to be weakly compliant (à la weak compatibility [5, 25]).

In the short paper contribution [10], we briefly reviewed our previous approach from [6] and we identified some limitations leading to issues for future research. A first issue was that the assignment of a single synchronisation type to a team, as in [6], is too restrictive and that we need to fine tune the number of synchronising sending and receiving components *per action*. For this purpose we introduce, in the current paper, synchronisation type specifications which assign a synchronisation type individually to each communicating action. Such specifications uniquely determine a team formalised by an extended notion of team automaton (ETA). On the other hand, any synchronisation type specification generates communication requirements to be satisfied by the team.

A second issue was that we realised that even the weak compliance notion proposed in [6] is too restrictive for practical applications. In the current paper, we overcome this problem by introducing a much more liberal compliance notion: if a group $\mathcal{J}$ of components has issued a communication request, then we allow the team to execute arbitrary other actions, not limited to internal ones, before being ready for the required communication (with the components in $\mathcal{J}$). This leads to a powerful compliance notion not studied before (as far as we know).

This apparently simple generalisation has a significant consequence: among the 'arbitrary other actions' there may be output or input actions open to the environment. This is a potentially dangerous situation, since in this case local communication properties can be violated after composition with other teams.

This leads us to the third, perhaps most important, contribution of the current paper. We consider composition of systems and of teams. First, we show that composition behaves well with synchronisation types (Theorem 1). Then we investigate conditions under which communication properties are preserved by ETA composition. The principle idea is that for this it should be sufficient to consider interface actions and to check (global) compliance conditions for them. We formulate appropriate conditions, first for the case of (strong) receptiveness and responsiveness (Theorem 2) and then for the weak variant of the two, solving the problem sketched above (Theorem 3). An intuitive running example guides the reader through the paper.

Outline After introducing extended team automata (ETA) in Section 2, we consider synchronisation type specifications and ETA determined by them in Section 3. (Weak) compliance of ETA with communication requirements and safe communication are treated in Section 4. In Section 5, we define the composition of systems and of teams, and we show that this works well with synchronisation type specifications. In Section 6, we provide our main compositionality results. Full proofs and some insightful counterexamples of the results presented in the latter two sections can be found in [9]. After discussing related work in Section 7, we conclude the paper in Section 8.

2 Background and Extended Team Automata

In this section, we summarise the basic notions concerning team automata and introduce extended team automata. In contrast to the 'classical' team automata from [8, 23] and subsequent papers, extended team automata use system labels which, in addition to the executed action, specify the team members that participate in a synchronisation on an action. We start with some technical preliminaries concerning labelled transition systems which will be reused for the definitions of (local) component automata and (global) team automata.

A *labelled transition system* (LTS for short) is a quadruple $\mathcal{L} = (Q, \Sigma, \delta, I)$ consisting of a set Q of *states*, a set Σ of *actions* such that $Q \cap \Sigma = \varnothing$, a transition relation $\delta \subseteq Q \times \Sigma \times Q$ and a nonempty set $I \subseteq Q$ of *initial states*.

For an action $a \in \Sigma$, $\delta_a = \delta \cap (Q \times \{a\} \times Q)$ denotes the set of *a-transitions* of $\mathcal{L}$. Instead of $(p, a, p') \in \delta$ we may write $p \xrightarrow{a}_{\mathcal{L}} p'$. Action a is *enabled* in $\mathcal{L}$ at state $p \in Q$, denoted by $a\,\mathrm{en}_{\mathcal{L}}\,p$, if there exists $p' \in Q$ such that $p \xrightarrow{a}_{\mathcal{L}} p'$. For $\Gamma \subseteq \Sigma$, we write $p \xrightarrow{\Gamma}{}^{*}_{\mathcal{L}} p'$ if there exist $p_0 \xrightarrow{a_1}_{\mathcal{L}} p_1, \ldots, p_{j-1} \xrightarrow{a_j}_{\mathcal{L}} p_j$ for some $j \geq 0$, with $p_0, \ldots, p_j \in Q$, $a_1, \ldots, a_j \in \Gamma$, $p = p_0$, and $p' = p_j$. A state $p \in Q$ is *reachable* if $p_0 \xrightarrow{\Sigma}{}^{*}_{\mathcal{L}} p$ for some $p_0 \in I$. The set of reachable states of $\mathcal{L}$ is denoted by $\mathcal{R}(\mathcal{L})$.

Component automata are LTSs with an additional distinction between input and output actions.[4] They form the basic building block of systems.

Definition 1 (Component automaton). *A* component automaton *(CA for short) is an LTS $\mathcal{A} = (Q, \Sigma, \delta, I)$ such that Σ is the union of two disjoint sets Σ_{inp} and Σ_{out} of* input *and* output *actions, respectively.* □

In figures, we emphasise the role of actions by adding suffix ? to input actions and ! to output actions.

Systems A *system* is a pair $\mathcal{S} = (\mathcal{N}, (\mathcal{A}_i)_{i\in\mathcal{N}})$, where $\mathcal{N}$ is a finite, non-empty set of component names and $(\mathcal{A}_i)_{i\in\mathcal{N}}$ is an $\mathcal{N}$-indexed family of CA $\mathcal{A}_i = (Q_i, \Sigma_i, \delta_i, I_i)$ with actions $\Sigma_i = \Sigma_{i,inp} \cup \Sigma_{i,out}$. The *state space* of $\mathcal{S}$ is given by the Cartesian product $Q = \prod_{i\in\mathcal{N}} Q_i$. Hence a global *system state* is an $\mathcal{N}$-indexed family $q = (q_i)_{i\in\mathcal{N}}$ of local component states $q_i \in Q_i$. The *initial states* of $\mathcal{S}$ are given by the product $I = \prod_{i\in\mathcal{N}} I_i$. If $\varnothing \neq \mathcal{N}' \subseteq \mathcal{N}$ and $q = (q_i)_{i\in\mathcal{N}}$ is a system state, the *projection* of q to $\mathcal{N}'$ is defined by $proj_{\mathcal{N}'}(q) = (q_i)_{i\in\mathcal{N}'}$.

We refer to $\Sigma = \bigcup_{i\in\mathcal{N}} \Sigma_i$ as the set of *actions* of $\mathcal{S}$.[5] Within Σ, we identify $\Sigma_{com} = \bigcup_{i\in\mathcal{N}} \Sigma_{i,inp} \cap \bigcup_{i\in\mathcal{N}} \Sigma_{i,out}$ as the set of *communicating actions in* $\mathcal{S}$. Hence, an action of $\mathcal{S}$ is communicating in $\mathcal{S}$ if it occurs in (at least) one of its CA as an input action and in (at least) one of its CA as an output action.

For an action $a \in \Sigma$, we let $\mathrm{dom}_{a,inp}(\mathcal{S}) = \{\, i \mid a \in \Sigma_{i,inp} \,\}$ be its *input domain* (in $\mathcal{S}$) and $\mathrm{dom}_{a,out}(\mathcal{S}) = \{\, i \mid a \in \Sigma_{i,out} \,\}$ its *output domain* (in $\mathcal{S}$). Hence a communicating action of $\mathcal{S}$ is such that both its output and input domain in $\mathcal{S}$ are not empty.

Notation. *Up to and including Section 4, we fix $\mathcal{N}$ and $\mathcal{S}$ as above.* □

Example 1. Consider a distributed chat system, where buddies can interact once registered. For now, we consider two types of components: clients and servers, depicted in Fig. 1 (left and middle, respectively). The arbiter will join only later when we discuss system compositions. A server controls entries into the chat and exits from the chat, and coordinates the main activity: forwarding client messages to the chat. The communicating actions are partitioned into chat access actions (*join*, *leave*, *confirmJ*, *confirmL*) and chat messaging (*msg*, *fwdmsg*). The non-communicating actions are currently *ask*, *grant*, and *reject*. Let us assume a chat system $\mathcal{S}_{chat}$ consisting of two clients $\mathcal{A}_1$ and $\mathcal{A}_2$ and one server $\mathcal{A}_3$. Its state space consists of tuples (p, q, r) with client states p and q and server state r. □

We use extended labels as envisioned in [14] for multi component-interaction automata, to indicate explicitly which components are actively participating in

[4] In general, and in the classical team automata approach, a component automaton can also have a distinguished set of internal actions. Since internal actions are not really relevant for the scope of this paper, we omit them for the sake of simplicity.

[5] If component automata were equipped with internal actions, then a syntactic composability constraint would have to be applied to $\mathcal{S}$ requiring that each internal action of a component automaton is unique to that component automaton.

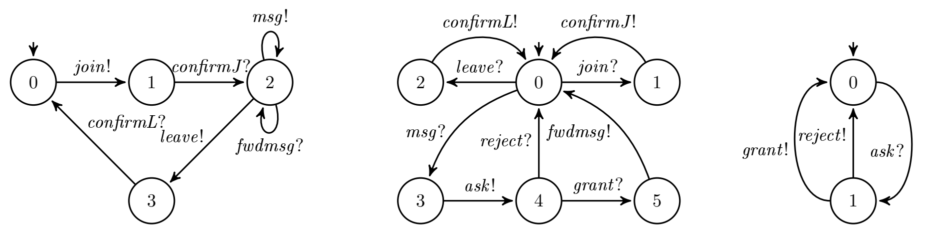

Fig. 1: [from left to right] CA for clients, servers, and arbiters [adapted from [10]]

system transitions. In the vector team automata of [11], vectors of component actions are used for this purpose, giving rise to a concurrent semantics.

Definition 2 (System labels). *Let $a \in \Sigma$. A* system label for a (in $\mathcal{S}$) *is a triple (out, a, inp) where $out, inp \subseteq \mathcal{N}$ are subsets of $\mathcal{N}$ such that $out \cup inp \neq \varnothing$ and $a \in \Sigma_{i,out}$ for all $i \in out$ and $a \in \Sigma_{i,inp}$ for all $i \in inp$. The set of system labels for a in $\mathcal{S}$ is denoted by $\Lambda_a(\mathcal{S})$, while $\Lambda(\mathcal{S}) = \bigcup_{a \in \Sigma} \Lambda_a(\mathcal{S})$ denotes the set of all* system labels *in $\mathcal{S}$.* □

System labels provide an appropriate means to describe which components in a system execute together a computation step, i.e. a system transition.

Definition 3 (System transitions). *A triple $(q, \sigma, q') \in Q \times \Lambda(\mathcal{S}) \times Q$ with system label $\sigma = (out, a, inp)$ is a* system transition on a *(in $\mathcal{S}$) if $(q(i), a, q'(i)) \in \delta_i$ for all $i \in out \cup inp$ and $q(i) = q'(i)$ for all $i \in \mathcal{N} \setminus (out \cup inp)$.*

For $a \in \Sigma$, the set of all system transitions on a in $\mathcal{S}$ is denoted by $E_a(\mathcal{S})$, while $E(\mathcal{S}) = \bigcup_{a \in \Sigma} E_a(\mathcal{S})$ denotes the set of all system transitions in $\mathcal{S}$*.* □

If $t = (q, (out, a, inp), q') \in E(\mathcal{S})$ then any CA $\mathcal{A}_i$ for which $i \in out \cup inp$ is said to *participate in t*. If $i \in out$, then $\mathcal{A}_i$ is a *sender in t*, otherwise it is a *receiver*. Since, by definition of system labels, $out \cup inp \neq \varnothing$, at least one CA is participating in any system transition in $\mathcal{S}$. Moreover, all system transitions in $E_a(\mathcal{S})$ are combinations of existing a-transitions from the CA in $\mathcal{S}$ and all possible combinations of a-transitions occur in $E_a(\mathcal{S})$. The elements of $E_a(\mathcal{S})$ are also referred to as *synchronisations on a*, even when only one CA participates. A synchronisation on a communicating action a in which a CA where a is an output action and a CA where a is an input action participate, is called a *communication*. Obviously, for a non-communicating action $a \in \Sigma$, either out or inp is empty in any system transition on a.

Example 2. The system transitions $E_{msg}(\mathcal{S}_{chat})$ of $\mathcal{S}_{chat}$ from Example 1 in which CA $\mathcal{A}_3$ participates are the following: $((2,2,0), (\varnothing, msg, \{\mathcal{A}_3\}), (2,2,3))$, $((2,2,0), (\{\mathcal{A}_1\}, msg, \{\mathcal{A}_3\}), (2,2,3))$, $((2,2,0), (\{\mathcal{A}_2\}, msg, \{\mathcal{A}_3\}), (2,2,3))$, and $((2,2,0), (\{\mathcal{A}_1, \mathcal{A}_2\}, msg, \{\mathcal{A}_3\}), (2,2,3))$. Using this notation, we thus express whether $\mathcal{A}_1$ or $\mathcal{A}_2$ participates or not in a synchronisation and hence whether or not a communication takes place. Note that not all system transitions are meaningful in applications. For instance, $((2,2,0), (\{\mathcal{A}_1, \mathcal{A}_2\}, msg, \{\mathcal{A}_3\}), (2,2,3))$ expresses that both clients join to send msg to the server. If we want to rule out undesired synchronisations we must declare a subset of admissible system transitions, which is the underlying idea of team automata. □

Extended Team Automata The CA combined in a system are meant to collaborate (form a team) through the simultaneous execution of shared actions. Such teams are formalised by our notion of extended team automaton. They are labelled transition systems with set of states Q and set of initial states I. Their transitions are always a subset ε of $E(\mathcal{S})$ containing the admissible system transitions. Such subset is called a *synchronisation policy*. From the software engineering perspective, it is the task of the team designer to determine an appropriate synchronisation policy for a given system of components. We use the system labels (out, a, inp) in $\Lambda(\mathcal{S})$ as the actions in team transitions. This is the main difference with the classical team automata from [8,23] and subsequent papers, where actions $a \in \Sigma$ would have been used in team transitions. However, to study communication properties and their compositionality, explicit rendering of the CA that actually participate in a transition of the team seems useful.

Definition 4 (Extended team automaton). *An extended team automaton (ETA for short) over $\mathcal{S}$ is an LTS $\mathcal{E} = (Q, \Lambda(\mathcal{S}), \varepsilon, I)$, where $\varepsilon \subseteq E(\mathcal{S})$ is a synchronisation policy over $\mathcal{S}$.* □

3 Synchronisation Type Specifications

In [6], we proposed *synchronisation types* to specify in a convenient, syntactic way synchronisation policies. A synchronisation type (snd, rcv) determines ranges for the number of senders and the number of receivers that may take part in a communication. Both, the *sending multiplicity* snd and the *receiving multiplicity* rcv are given by intervals. If $snd = [o_1, o_2]$ (with $0 \leq o_1 \leq o_2$) and $rcv = [i_1, i_2]$ (with $0 \leq i_1 \leq i_2$) then at least o_1 and at most o_2 senders and at least i_1 and at most i_2 receivers are allowed. While o_1 and i_1 are always natural numbers, the upper delimiters o_2 and i_2 can also be given as $*$, which indicates that no upper limit is imposed. On the other hand, at most one of the lower delimiters o_1 or i_1 can be zero. In this case an output (respectively, input) of a communicating action can be performed by components without a participating receiver (respectively, sender).

Notable synchronisation types that can be defined include binary communication $([1,1],[1,1])$ and multicast communication $([1,1],[0,*])$, in which exactly one CA outputs a communicating action while arbitrarily many CA input that action. We can also express full synchronisation on an action a by requiring as a synchronisation type for a (snd, rcv) with $snd = [\mathrm{dom}_{a,out}, \mathrm{dom}_{a,out}]$ and $rcv = [\mathrm{dom}_{a,inp}, \mathrm{dom}_{a,inp}]$.

For the following, recall that $\mathcal{S} = (\mathcal{N}, (\mathcal{A}_i)_{i \in \mathcal{N}})$ is a composable system with state space Q, actions Σ and communicating actions Σ_{com}. In [6], we considered the situation where all synchronisations in $\mathcal{S}$ follow a single synchronisation type used uniformly for all communicating actions of the system. In practice it is, however, necessary to relax this interpretation and define synchronisation types individually for each communicating action of the system. This leads to our new notion of synchronisation type specification.

Definition 5 (Synchronisation type specification). *A* synchronisation type specification *over $\mathcal{S}$ is a mapping st which assigns to all communicating actions $a \in \Sigma_{com}$ a synchronisation type $st(a) = (snd_{st(a)}, rcv_{st(a)})$.* □

For the non-communicating actions in $\mathcal{S}$ no synchronisation type is provided since this is only relevant when systems are composed; see Sections 5 and 6. We will now discuss how a synchronisation policy, and hence an ETA, can be deduced from a synchronisation type specification.

Let (snd, rcv) be a synchronisation type with $snd = [o_1, o_2]$ and $rcv = [i_1, i_2]$. A system transition $(q, \sigma, q') \in Q \times \Lambda(\mathcal{S}) \times Q$ with $\sigma = (out, a, inp)$ is of type (snd, rcv) if $o_1 \leq \#out \leq o_2$ and $i_1 \leq \#inp \leq i_2$, assuming $n \leq *$ for any $n \in \mathbb{N}$.

Remark 1. Note that for typing system transitions we use in a crucial way the information provided by system labels. If we had mere actions as transition labels of communications, as in team automata, it would not be clear whether a CA with a 'self-loop' participates in a communication or not. Consider, for instance, the system labels and transition from Example 2. In a team automaton over $\mathcal{S}_{chat}$ there could be a transition $((2,2,0), msg, (2,2,3))$ in which it is not clear whether one, two, or none of the clients participate, i.e. whether or not $msg!$ is actually executed and by whom. In team automata, this is typically resolved by implicitly assuming that a loop of a CA in a transition implies its execution, a 'maximal' interpretation of ambiguous participation. □

Each synchronisation type specification determines a unique synchronisation policy and hence a unique ETA in the following way:

Definition 6 (Typed synchronisation policy). *Let st be a synchronisation type specification over $\mathcal{S}$.*

1. *The* synchronisation policy determined by st*, denoted by $\varepsilon(st)$, is defined by $\varepsilon(st)_a = \{ t \in E_a(\mathcal{S}) \mid t$ is of type $st(a) \}$ if $a \in \Sigma_{com}$; and $\varepsilon(st)_a = E_a(\mathcal{S})$ if $a \in \Sigma \setminus \Sigma_{com}$.*
2. *The* ETA determined by st *is $\mathcal{E}(st) = (Q, \Lambda(\mathcal{S}), \varepsilon(st), I)$.* □

Note that for non-communicating actions $a \in \Sigma \setminus \Sigma_{com}$ for which no synchronisation type is specified, $\varepsilon(st)$ is 'maximal' in the sense that we set $\varepsilon(st)_a = E_a(\mathcal{S})$. This means that we allow all possible synchronisations in $\mathcal{S}$. This is in contrast with [6], where we allowed arbitrary subsets of $E_a(\mathcal{S})$ rather than equality. It is, however, needed to get the compositionality results later on.

Example 3. Consider global state $(2,0,5)$ of the chat system $\mathcal{S}_{chat}$ from Examples 1–2, where client $\mathcal{A}_1$ can (autonomously) decide to execute either its output action *leave* or its output action *msg*. To enforce receptiveness, there must be at least one other CA ready to execute either action as an input action. Server $\mathcal{A}_3$ only has output action *fwdmsg* locally enabled. If we set $st_{chat}(fwdmsg) = ([1,1],[0,*])$ as synchronisation type for *fwdmsg*, then the server is allowed to move to state 0 by executing its output action *fwdmsg* on its own (rather than in a communication) after which the server is ready to accept inputs as required. This synchronisation type is not suitable for the other actions, e.g. a client should be prohibited to *join* without acceptance by the server, thus $st_{chat}(join) = ([1,1],[1,1])$ would be appropriate. Therefore, we define a synchronisation type specification st_{chat} over $\mathcal{S}_{chat}$ such that $st_{chat}(fwdmsg) = ([1,1],[0,*])$ and $st_{chat}(a) = ([1,1],[1,1])$ for all other communicating actions of $\mathcal{S}_{chat}$. The ETA determined by st_{chat} is $\mathcal{E}_{chat}(st_{chat})$. $\square$

4 Communication Requirements and Compliance

The idea of communication-safety in team automata is as follows. At each reachable global state of a team, whenever a communicating action is enabled at the local states of some components $\mathcal{J}$ in accordance with the synchronisation type of that action, then all components in $\mathcal{J}$ can execute this action from their local states as a communication within the team.

Communication-safety of ETA in a nutshell Before giving formal definitions, let us explain in a nutshell how our approach works. Consider an ETA $\mathcal{E}(st)$ and a communication action a with, for instance, synchronisation type $st(a) = ([1,1],[1,*])$. Let $\mathcal{A}_i$ be a component of the system for which a is an output action and let q be a global state of $\mathcal{E}(st)$ such that a is enabled at the local state $q(i)$ of $\mathcal{A}_i$. Then we wish that a can be received by at least one other component in the team. We express this by a *receptiveness requirement* issued by component $\mathcal{A}_i$ and written as $\texttt{rcp}(\{i\},a)@q$. If the ETA $\mathcal{E}(st)$ is *compliant with* this requirement, it is guaranteed that in state q, component $\mathcal{A}_i$ can synchronise with other components in the team taking a as input.

Note that in case $\mathcal{A}_i$ could also execute another output action b with the same synchronisation type at state $q(i)$, subject to the corresponding receptiveness requirement, then the two requirements would be combined through a conjunction to $\texttt{rcp}(\{i\},a)@q \wedge \texttt{rcp}(\{i\},b)@q$. The reason for this is that components control their output actions and thus can internally decide which action to be sent. Hence, the choice of either of them should lead to a reception. The expression $\texttt{rcp}(\{i\},a)@q \wedge \texttt{rcp}(\{i\},b)@q$ is called a *receptiveness requirement generated by st*. Indeed, the information in the synchronisation type $([1,1],[1,*])$ determines, due to the lower bound 1 of the output multiplicity $[1,1]$, that already one component can induce a receptiveness requirement. On the other hand, the receive multiplicity $[1,*]$ tells us that a communication is really needed for the output of a. Indeed, if the receive multiplicity were $[0,*]$, then there would be no receptiveness requirement. If, however, the output multiplicity of a were $[2,*]$, then at least two components for which a is enabled in the current local states would be needed to issue a valid receptiveness requirement of the form $\texttt{rcp}(\mathcal{J},a)@q$ with $\mathcal{J}$ determining the set of output components.

For input actions one could require responsiveness with the intuition that enabled inputs should be served by appropriate outputs. Unlike output actions, however, input actions are controlled by the environment, i.e. input choice is external. Guaranteeing that for a choice of enabled inputs, *one of them* is supplied with an output of other components suffices for the progress of a component waiting for a signal. Hence, if component $\mathcal{A}_j$ enables input actions a and b in its local state $q(j)$, then the *responsiveness requirements*, denoted by $\texttt{rsp}(\{j\}, a)@q$ and $\texttt{rsp}(\{j\}, b)@q$ would be combined with a disjunction to $\texttt{rsp}(\{j\}, a)@q \vee \texttt{rsp}(\{j\}, b)@q$, which is called a *responsiveness requirement generated by st*. Of course, also responsiveness requirements can be issued by several components $\mathcal{J}$ instead of $\{j\}$.

In general, a team automaton $\mathcal{E}(st)$ over a system $\mathcal{S}$ is called *receptive* (respectively, *responsive*) if it is compliant with all receptiveness requirements (respectively, responsiveness requirements) generated by st at all reachable states of $\mathcal{E}(st)$. It is *communication-safe* if it is receptive and responsive.

Weak compliance In [6], we relaxed compliance to allow the team to execute some intermediate internal actions before being ready for the required communication. As anticipated in the introduction and in [10], in this paper we further relax the notion of *weak compliance* from [6]: if a group $\mathcal{J}$ of components has issued a communication request we allow the team to execute, without participation of $\mathcal{J}$, some arbitrary other actions before being ready for the required communication. This is a very flexible interpretation of interaction compatibility that, to the best of our knowledge, has not yet been studied in a similar way in the literature, likely because the permission of intermediate actions is dangerous for obtaining compositionality results. Its formal definition is given in Def. 9.

Formal definitions and examples In the remainder of this section, we provide the formal definitions of the concepts explained above, partly illustrated by examples. The definitions of communication requirements and compliance are taken from [6], the definition of weak compliance in this general form is new and the proposal to derive communication requirements from synchronisation type specifications is inspired by [6], but simplified and at the same time generalised to fit with synchronisation types per action.

We still assume given a composable system $\mathcal{S} = (\mathcal{N}, (\mathcal{A}_i)_{i\in\mathcal{N}})$ with state space Q, actions Σ and communicating actions Σ_{com}.

Definition 7 (Communication requirements). *Let $a \in \Sigma_{com}$ and $q \in Q$.*

- *Let $\varnothing \neq \mathcal{J} \subseteq dom_{\mathrm{a},out}(\mathcal{S})$ be such that $a\, \mathrm{en}_{\mathcal{A}_j}\, q(j)$ for all $j \in \mathcal{J}$. Then $\boldsymbol{rcp}(\mathcal{J}, a)@q$ is a* receptiveness requirement for *a* at *q.*
- *Let $\varnothing \neq \mathcal{J} \subseteq dom_{\mathrm{a},inp}(\mathcal{S})$ be such that $a\, \mathrm{en}_{\mathcal{A}_j}\, q(j)$ for all $j \in \mathcal{J}$. Then $\boldsymbol{rsp}(\mathcal{J}, a)@q$ is a* responsiveness requirement for *a* at *q.*
- *A* communication requirement at *q is either the trivial requirement true or a receptiveness or responsiveness requirement at q or a conjunction or disjunction of communication requirements at q.* □

When all non-trivial atomic requirements occurring in a communication requirement φ are receptiveness (respectively, responsiveness) requirements, we also refer to φ as a receptiveness (respectively, responsiveness) requirement.

Definition 8 (Compliance). *Let $\mathcal{E}$ be an ETA over $\mathcal{S}$ with synchronisation policy ε. Then $\mathcal{E}$ is* compliant with *a communication requirement φ at $q \in Q$ if either $q \notin \mathcal{R}(\mathcal{E})$ or $\varphi = true$, or one of the following holds:*

1. *$\varphi = \mathtt{rcp}(\mathcal{J}, a)@q$ for some $\mathcal{J} \subseteq \mathcal{N}$ and $a \in \Sigma_{com}$, and there exists $out \supseteq \mathcal{J}$ and $inp \neq \varnothing$ such that $q \xrightarrow{(out,a,inp)}_{\mathcal{E}} q'$*
2. *$\varphi = \mathtt{rsp}(\mathcal{J}, a)@q$ for some $\mathcal{J} \subseteq \mathcal{N}$ and $a \in \Sigma_{com}$, and there exists $out \neq \varnothing$ and $inp \supseteq \mathcal{J}$ such that $q \xrightarrow{(out,a,inp)}_{\mathcal{E}} q'$*
3. *$\varphi = \psi_1 \wedge \psi_2$ and $\mathcal{E}$ is compliant with ψ_1 at q and with ψ_2 at q*
4. *$\varphi = \psi_1 \vee \psi_2$ and $\mathcal{E}$ is compliant with ψ_1 at q or with ψ_2 at q* □

Note that when $\mathcal{E}$ is compliant with a requirement as in *1.* and *2.* above, then the components $\mathcal{J}$ can communicate through a synchronisation on a at q involving more CA from the output and input domains of a.

Recall that $\Lambda(\mathcal{S})$ denotes the labels of $\mathcal{E}$ and let $\Lambda_{\mathcal{J}}(\mathcal{S})$, with $\mathcal{J} \subseteq \mathcal{N}$, denote the set of labels in which CA from $\mathcal{J}$ participate, i.e. system labels (out, a, inp) such that $j \in out \cup inp$ for some $j \in \mathcal{J}$.

Definition 9 (Weak compliance). *Let $\mathcal{E}$ be an ETA over $\mathcal{S}$ with synchronisation policy ε. Weak compliance is defined analogously to Definition 8 but replacing* 1. *and* 2. *by the following items:*

1. *$\varphi = \mathtt{rcp}(\mathcal{J}, a)@q$ for some $\mathcal{J} \subseteq \mathcal{N}$ and $a \in \Sigma_{com}$, and there exists $p \in Q$, $out \supseteq \mathcal{J}$ and $inp \neq \varnothing$ such that $q \xrightarrow{\Lambda(\mathcal{S})\backslash\Lambda_{\mathcal{J}}(\mathcal{S})}{}^{*}_{\mathcal{E}}\ p \xrightarrow{(out,a,inp)}_{\mathcal{E}} q'$*
2. *$\varphi = \mathtt{rsp}(\mathcal{J}, a)@q$ for some $\mathcal{J} \subseteq \mathcal{N}$ and $a \in \Sigma_{com}$, and there exists $p \in Q$, $inp \supseteq \mathcal{J}$ and $out \neq \varnothing$ such that $q \xrightarrow{\Lambda(\mathcal{S})\backslash\Lambda_{\mathcal{J}}(\mathcal{S})}{}^{*}_{\mathcal{E}}\ p \xrightarrow{(out,a,inp)}_{\mathcal{E}} q'$* □

Compliance trivially implies weak compliance. Note that we require that the CA determined by $\mathcal{J}$ do not participate in the intermediate transitions. Moreover, it is possible that also CA not participating in the foreseen communication, do participate in the intermediate actions that are needed to reach the global team state where it can occur. This is a phenomenon known as 'state-sharing' (cf. [8, 24]). It allows CA to influence potential synchronisations through their local states without participating in the actual transition.

Example 4. We continue Example 3. In global state $(2, 0, 5)$, $\mathcal{A}_2$ is locally enabled to execute its output action *join*. Recall that $st_{chat}(join) = ([1, 1], [1, 1])$. Then the receptiveness requirement for *join* at state $(2, 0, 5)$ is $\mathtt{rcp}(\{\mathcal{A}_2\}, join)@(2{,}0{,}5)$, i.e. output action *join* of $\mathcal{A}_2$ must be received as input by at least one other CA. The only CA with *join* as an input action is the server $\mathcal{A}_3$, but *join* is not enabled at its local state 5. However, in an ETA $\mathcal{E}$ over $\mathcal{S}_{chat}$ where $\mathcal{A}_3$ can transit from state 5 to state 0 by a communication with $\mathcal{A}_1$ (or even alone) the CA $\mathcal{A}_3$ would subsequently be ready to execute *join* in a communication with $\mathcal{A}_2$. Since the intermediate move of the server to state 0 is allowed by our new notion of weak compliance, this $\mathcal{E}$ is weakly compliant with the given requirement. □

As discussed above, the guidelines for when which choice of communication requirements is suitable, must consider the synchronisation types of actions. Let st be a synchronisation type specification over $\mathcal{S}$, $q \in Q$ and $a \in \Sigma_{com}$ such that $st(a) = ([o_1, o_2], [i_1, i_2])$.

A receptiveness requirement $\texttt{rcp}(\mathcal{J}, a)@q$ is *valid for* $st(a)$ if $o_1 \leq |\mathcal{J}| \leq o_2$ and $i_1 \neq 0$. The *receptiveness requirement for* q *generated by* st is the conjunction[6]

$$\bigwedge\{\, \texttt{rcp}(\mathcal{J}, a)@q \mid a \in \Sigma_{com}, \texttt{rcp}(\mathcal{J}, a)@q \text{ is valid for } st(a) \,\}$$

A responsiveness requirement $\texttt{rsp}(\mathcal{J}, a)@q$ is *valid for* $st(a)$ if $i_1 \leq |\mathcal{J}| \leq i_2$, $o_1 \neq 0$ and, for each $j \in \mathcal{J}$, $q(j)$ is a state at which only input actions are enabled.[7] The *responsiveness requirement for* q *generated by* st is the disjunction[8]

$$\bigvee\{\, \texttt{rsp}(\mathcal{J}, a)@q \mid a \in \Sigma_{com}, \texttt{rsp}(\mathcal{J}, a)@q \text{ is valid for } st(a) \,\}$$

In summary, each synchronisation type specification st for a system $\mathcal{S}$ of components determines a synchronisation policy, i.e. an ETA $\mathcal{E}(st)$, and generates at the same time communication requirements. Then $\mathcal{E}(st)$ is (weakly) communication-safe if it is (weakly) compliant with all requirements. This means that the components in $\mathcal{S}$ coordinated by the synchronisation policy determined by st work properly together.

Definition 10. *Let $\mathcal{E}(st)$ be an ETA determined by a synchronisation type specification st. $\mathcal{E}(st)$ is* (weakly) receptive *(respectively,* (weakly) responsive*) if it is (weakly) compliant at all $q \in \mathcal{R}(\mathcal{E}(st))$ with the receptiveness (respectively, responsiveness) requirement for q generated by st. $\mathcal{E}(st)$ is* (weakly) communication-safe *if it is receptive and responsive.* □

Example 5. We continue Examples 1–4. Let st_{chat} be the chat system's synchronisation type specification as defined in Example 3. Let $\mathcal{E}_{chat}(st_{chat})$ be the ETA determined by st_{chat}. An example of a generated receptiveness requirement is $\texttt{rcp}(\{\mathcal{A}_1\}, \mathit{msg})@(2,0,5) \wedge \texttt{rcp}(\{\mathcal{A}_1\}, \mathit{leave})@(2,0,5) \wedge \texttt{rcp}(\{\mathcal{A}_2\}, \mathit{join})@(2,0,5)$. As explained in Example 4, $\mathcal{E}_{chat}(st_{chat})$ is weakly compliant with the second conjunct. It is also weakly compliant with the first conjunct as after moving on its own to state 0, the server can receive *msg*. Requirement $\texttt{rcp}(\{\mathcal{A}_1\}, \mathit{msg})@(2,0,3)$ is more tricky, since in state 3 the server already received a *msg* from client $\mathcal{A}_1$ who wants to send another *msg*. Due to the new weak compliance notion this is ok, since the server can execute its non-communicating external actions *ask* followed by, e.g., *reject* to return to state 0 where it can receive *msg*. An example of a generated responsiveness requirement is $\texttt{rsp}(\{\mathcal{A}_3\}, \mathit{join})@(0,0,0) \vee \texttt{rsp}(\{\mathcal{A}_3\}, \mathit{leave})@(0,0,0) \vee \texttt{rsp}(\{\mathcal{A}_3\}, \mathit{msg})@(0,0,0)$. Clearly, $\mathcal{E}_{chat}(st_{chat})$ is only compliant with the first disjunct since either client can provide the required output action *join*. It is important to note that requirements generated from inputs

[6] We use a conjunction here since outputs are autonomously decided by components.
[7] Otherwise the component has already a receptiveness requirement for an output.
[8] We use a disjunction here since inputs rely on external choice.

rely on external choice of the environment and therefore it is sufficient if one of the offered inputs is served, which is expressed by the disjunction. We could only discuss here a few requirements, but a thorough analysis shows that indeed $\mathcal{E}_{chat}(st_{chat})$ is weakly receptive and weakly responsive. $\square$

5 Systems of Systems and ETA Compositions

In this section, we consider systems of systems and the composition of ETA given for each individual system. This yields ETA over the combined global systems. We also show how a global synchronisation type specification can be constructed from the local ones respecting the underlying local ETA composition.

Let $n \geq 1$ and let, for $k = 1, \ldots, n$, $\mathcal{S}_k = (\mathcal{N}_k, (\mathcal{A}_{k,i})_{i \in \mathcal{N}_k})$ be a system with $(\mathcal{A}_{k,i})_{i \in \mathcal{N}_k}$ an $\mathcal{N}_k$-indexed family of CA $\mathcal{A}_{k,i} = (Q_{k,i}, \Sigma_{k,i}, \delta_{k,i}, I_{k,i})$ where $\Sigma_{k,i} = \Sigma_{k,i,inp} \cup \Sigma_{k,i,out}$. Hence, $\Sigma_k = \bigcup_{i \in \mathcal{N}_k} \Sigma_{k,i}$ is the set of actions in $\mathcal{S}_k$ and $\Sigma_{k,com} = \bigcup_{i \in \mathcal{N}_k} \Sigma_{k,i,inp} \cap \bigcup_{i \in \mathcal{N}_k} \Sigma_{k,i,out}$ is its set of communicating actions.

To compose the single systems we assume that communicating actions within one system cannot be used to interact with other systems. So, we say that the family of systems $(\mathcal{S}_k)_{k \in [n]}$ is *composable* if for all $k \in [n]$ and for all $k \neq l \in [n]$, $\mathcal{N}_k \cap \mathcal{N}_l = \varnothing$ and $\Sigma_{k,com} \cap \Sigma_l = \varnothing$. Hence, in a composable family of systems, component names and communicating actions are unique to a system.

Definition 11 (System composition). *The composition of a composable family $(\mathcal{S}_k)_{k \in [n]}$ is the system $\bigotimes_{k \in [n]} \mathcal{S}_k = (\bigcup_{k \in [n]} \mathcal{N}_k, (\mathcal{A}_{k,i})_{(k,i) \in [n] \times \mathcal{N}_k})$.* $\square$

Notation. *For the rest of the paper, we fix $n \in \mathbb{N}^{>0}$ and $\mathcal{S}_k$ for $k \in [n]$, as above. We moreover assume that $(\mathcal{S}_k)_{k \in [n]}$ is composable and that $\mathcal{S} = \bigotimes_{k \in [n]} \mathcal{S}_k$ with state space Q and initial states I.* $\square$

The set of actions of $\mathcal{S}$ is $\Sigma = \bigcup_{(k,i) \in [n] \times \mathcal{N}_k} \Sigma_{k,i} = \bigcup_{k \in [n]} \Sigma_k$ and the set of communicating actions in $\mathcal{S}$ is

$$\Sigma_{com} = \bigcup_{(k,i) \in [n] \times \mathcal{N}_k} \Sigma_{k,i,inp} \cap \bigcup_{(k,i) \in [n] \times \mathcal{N}_k} \Sigma_{k,i,out}$$

Obviously, Σ_{com} contains the communicating actions $\Sigma_{k,com}$ of each sub-system $\mathcal{S}_k$ but also actions which occur as input action in a component of one sub-system and as output action in a component of another sub-system. The latter are called *interface actions* and defined by $\Sigma_{inf} = \Sigma_{com} \setminus \bigcup_{k \in [n]} \Sigma_{k,com}$.

Example 6. We now add an arbiter to the chat system $\mathcal{S}_{chat}$ from Examples 1–4 to regulate message forwarding by composing $\mathcal{S}_{chat}$ with the singleton system $\{\mathcal{A}_4\}$, depicted in Fig. 1 (right). The idea is that the server must *ask* the arbiter to *grant* or *reject* permission to forward a message. The two systems $\mathcal{S}_{chat}$ and $\{\mathcal{A}_4\}$ form a composable family of systems; *ask*, *grant*, and *reject* are the interface actions. $\square$

Given the composable family of systems $(\mathcal{S}_k)_{k\in[n]}$, we describe how to compose an extended team automaton over $\mathcal{S} = \bigotimes_{k\in[n]} \mathcal{S}_k$ from given ETA $\mathcal{E}_k = (Q_k, \Lambda(\mathcal{S}_k), \varepsilon_k, I_k)$ over $\mathcal{S}_k$, $k \in [n]$.

An ETA obtained as a composition of $(\mathcal{E}_k)_{k\in[n]}$ is an ETA over $\mathcal{S}$. So it has state space Q, set of initial states I, and set of actions Σ as defined above. The essential part concerns the choice of the synchronisation policy ε for the composed ETA. We proceed as follows to define the system transitions of ε for each $a \in \Sigma$:

1. For each non-communicating action $a \in \Sigma \setminus \Sigma_{com}$, we define

$$\varepsilon_a = \{\, (q, (out, a, in), q') \in E_a(\mathcal{S}) \mid \text{for all } k \in [n], \text{s.t.}\, \mathcal{N}_k \cap (out \cup in) \neq \varnothing : \\ (proj_{\mathcal{N}_k}(q), (out \cap \mathcal{N}_k, a, in \cap \mathcal{N}_k), proj_{\mathcal{N}_k}(q')) \in \varepsilon_{k,a} \,\}$$

   Hence, ε_a is the set of all system transitions for a in $\mathcal{S}$ whose projections to sub-systems $\mathcal{S}_k$ having action a belong to $\varepsilon_{k,a}$.
2. If $a \in \Sigma_{com}$ (a communicating action in $\mathcal{S}$) we distinguish two cases:
   (a) $a \in \bigcup_{k\in[n]} \Sigma_{k,com}$: Then, by composability, there is exactly one $k \in \mathcal{K}$ such that $a \in \Sigma_{k,com}$ and a is unique to $\mathcal{S}_k$. Fix this k and define

$$\varepsilon_a = \{\, (q, (out, a, in), q') \in E_a(\mathcal{S}) \mid \\ (proj_{\mathcal{N}_k}(q), (out, a, in), proj_{\mathcal{N}_k}(q')) \in \varepsilon_{k,a} \,\}$$

   Due to the uniqueness of a to $\mathcal{S}_k$, ε_a is the set of all extensions of system transitions in $\varepsilon_{k,a}$ to the state space of $\mathcal{S}$.

   (b) $a \in \Sigma_{inf}$: Now a is an action shared as input and output action of some components of $\mathcal{S}$ but not shared as an input and output action of components in any sub-system $\mathcal{S}_k$. Therefore it is the design choice of the overall system architect to determine a set of system transitions $\varepsilon_a \subseteq E_a(\mathcal{S})$.

This procedure leads to a unique ETA once a set of system transitions is provided for each interface action $a \in \Sigma_{inf}$.

Definition 12 (Composition of ETA). *Let for all $k \in [n]$, $\mathcal{E}_k$ be an ETA over $\mathcal{S}_k$ with ε_k its synchronisation policy. Let $\varepsilon_{inf} \subseteq \bigcup_{a\in\Sigma_{inf}} E_a(\mathcal{S})$ be a set of system transitions in $\mathcal{S}$ for the interface actions from Σ_{inf}.*

The extended team automata composition $\bigotimes^{\varepsilon_{inf}}_{k\in[n]} \mathcal{E}_k$ *of $(\mathcal{E}_k)_{k\in[n]}$ w.r.t. ε_{inf} is the ETA over $\mathcal{S}$ with synchronisation policy ε such that:*

1. *For all $a \in \Sigma \setminus \Sigma_{com}$, ε_a is defined as in item 1. above.*
2. *For all $k \in [n]$ and $a \in \Sigma_{k,com}$, ε_a is defined as in item 2.(a) above.*
3. *For all $a \in \Sigma_{inf}$, $\varepsilon_a = (\varepsilon_{inf})_a$.*

If $n = 2$ we write $\mathcal{E}_1 \otimes^{\varepsilon_{inf}} \mathcal{E}_2$ for $\bigotimes^{\varepsilon_{inf}}_{k\in[n]} \mathcal{E}_k$. □

The next theorem shows the relationship between ETA composition and synchronisation types. If a family of ETAs is given, each one determined by a certain synchronisation type, then it is enough to specify synchronisation types for the interface actions in order to get the composition of the ETAs as an ETA generated by a single synchronisation type.

Theorem 1. *Let for all $k \in [n]$, $\mathcal{E}_k(st_k)$ be an ETA determined by synchronisation type specification st_k and let $st_{inf}(a)$ be a synchronisation type for each interface action $a \in \Sigma_{inf}$. Then $\bigotimes^{\varepsilon_{inf}}_{k\in[n]} \mathcal{E}_k(st_k) = \mathcal{E}(st)$, where for all $a \in \Sigma_{inf}$, $(\varepsilon_{inf})_a = \{\, t \in E_a(\mathcal{S}) \mid t \text{ is of type } st_{inf}(a) \,\}$ and st is the synchronisation type specification over $\mathcal{S}$, defined by:*

1. *$st(a) = st_k(a)$ for all $k \in [n]$ and $a \in \Sigma_{k,com}$ and*
2. *$st(a) = st_{inf}(a)$ for all $a \in \Sigma_{inf}$.*

Proof. (sketch) The proof is straightforward using the (syntactic) composability assumption for systems and the definition of ETA composition. □

6 Compositionality of Communication Properties

We now study compositionality of communication properties. The issue here is to investigate conditions under which communication properties are preserved by ETA composition. The principle idea is that for this it should be sufficient to consider interface actions and to check (global) compliance conditions for them. We start by considering receptiveness and responsiveness.

Theorem 2. *Let $\mathcal{E}_k(st_k)$, $st_{inf}(a)$, and $\mathcal{E}(st)$ be as in Theorem 1 for all $k \in [n]$ and $a \in \Sigma_{inf}$.*

1. *Assume that $\mathcal{E}_k(st_k)$ is receptive for all $k \in [n]$. If, for all $q \in \mathcal{R}(\mathcal{E}(st))$ and $a \in \Sigma_{inf}$, $\mathcal{E}(st)$ is compliant with all receptiveness requirements $\boldsymbol{rcp}(\mathcal{J}, a)@q$ that are valid for $st(a) = st_{inf}(a)$, then $\mathcal{E}(st)$ is receptive.*
2. *Assume that $\mathcal{E}_k(st_k)$ is responsive for all $k \in [n]$. If, for all $q \in \mathcal{R}(\mathcal{E}(st))$ for which the responsiveness requirement generated by st has the form*

$$\bigvee \{\, \boldsymbol{rsp}(\mathcal{J}, a)@q \mid a \in \Sigma_{inf}, \boldsymbol{rsp}(\mathcal{J}, a)@q \text{ is valid for } st(a) = st_{inf}(a) \,\}^{[9]} \quad (1)$$

$\mathcal{E}(st)$ is compliant with (1), then $\mathcal{E}(st)$ is responsive.

Proof. (sketch) The proof relies on the fact that projections $proj_{\mathcal{N}_k}(q)$ of globally reachable states $q \in \mathcal{R}(\mathcal{E}(st))$ to a sub-system $\mathcal{S}_k$ are reachable in $\mathcal{E}_k(st_k)$. Then one can propagate communication properties concerning communicating actions $a \in \Sigma_{k,com}$ from $\mathcal{E}_k(st_k)$ to $\mathcal{E}(st)$. For interface actions compliance of $\mathcal{E}(st)$ with communication requirements is anyway assumed as a proof obligation. □

[9] Thus, (1) involves only responsiveness requirements concerning interface actions.

The following corollary reformulates the second case in Theorem 2 to make it symmetric to the first case. This yields, however, a strengthening of the condition in the second case of Theorem 2 disregarding the fact that for responsiveness requirements it is always sufficient if just one of the input alternatives is served.

Corollary 1. *Let $\mathcal{E}_k(st_k)$, $st_{inf}(a)$, and $\mathcal{E}(st)$ be as in Theorem 1 for all $k \in [n]$ and $a \in \Sigma_{inf}$. Assume that $\mathcal{E}_k(st_k)$ is responsive for all $k \in [n]$. If, for all $q \in \mathcal{R}(\mathcal{E}(st))$ and $a \in \Sigma_{inf}$, $\mathcal{E}(st)$ is compliant with all responsiveness requirements $\boldsymbol{rsp}(\mathcal{J}, a)@q$ that are valid for $st(a) = st_{inf}(a)$, then $\mathcal{E}(st)$ is responsive.* □

Next we consider compositionality of the weak notions of receptiveness and responsiveness. The idea is to require weak compliance of the global team with all communication requirements concerning interface actions and then to rely on weak receptiveness (respectively, weak responsiveness) of the sub-teams. How this works is demonstrated by the following example.

Example 7. Consider from Examples 1–6 the chat system $\mathcal{S}_{chat}$ and the singleton system $\{\mathcal{A}_4\}$ consisting of the arbiter. From Example 3 we take the synchronisation type specification st_{chat} for the chat system and the ETA $\mathcal{E}_{chat}(st_{chat})$ determined by st_{chat}. Since $\{\mathcal{A}_4\}$ has no communicating actions, there are no synchronisation types and the ETA determined by the empty synchronisation type specification $st_\varnothing$ is $\mathcal{E}_{arbiter}(st_\varnothing)$ which coincides with $\mathcal{A}_4$ but has system labels $(\varnothing, ask, \{\mathcal{A}_4\})$, $(\{\mathcal{A}_4\}, grant, \varnothing)$, and $(\{\mathcal{A}_4\}, reject, \varnothing)$ instead of *ask*, *grant*, and *reject*. The latter are the interface actions. For them we choose the synchronisation type $st_{inf}(ask) = st_{inf}(grant) = st_{inf}(reject) = ([1, 1], [1, 1])$.

Now consider the ETA composition $\mathcal{E}_{chat}(st_{chat}) \otimes \mathcal{E}_{arbiter}(st_\varnothing) = \mathcal{E}(st)$, where st is defined by st_{chat} and st_{inf} as described in Theorem 1. To show how weak receptiveness of $\mathcal{E}_{chat}(st_{chat})$ (cf. Example 5) can be propagated to $\mathcal{E}(st)$, we consider as an example the receptiveness requirement $\texttt{rcp}(\{\mathcal{A}_1\}, msg)@(2, 0, 3, 0)$ concerning the communicating action *msg* of $\mathcal{E}_{chat}(st_{chat})$ at the global state $(2, 0, 3, 0)$ of $\mathcal{E}(st)$. In state 3 the server already received a *msg* from client $\mathcal{A}_1$ who wants to send another *msg*. The weak compliance of the sub-team $\mathcal{E}_{chat}(st_{chat})$ with $\texttt{rcp}(\{\mathcal{A}_1\}, msg)@(2, 0, 3)$ has shown us (cf. Example 5) that in the scope of $\mathcal{E}_{chat}(st_{chat})$ the server can execute the interface action *ask* followed by, e.g., *reject* to return to state 0 where it can receive *msg*. The crucial point is now that in the global scope of $\mathcal{E}(st)$ the server can communicate with the arbiter such that server and arbiter together perform *ask* followed by, e.g., *reject* and thus the server returns to state 0 where it can receive *msg*. Hence the compliance of $\mathcal{E}_{chat}(st_{chat})$ with $\texttt{rcp}(\{\mathcal{A}_1\}, msg)@(2, 0, 3)$ is propagated to $\mathcal{E}(st)$. □

The example shows that weak compliance of the overall ETA $\mathcal{E}(st)$ with communication requirements concerning interface actions (*ask* and *reject* in the example) is a crucial assumption needed for compositionality. But there is still a subtle point to be taken into account as illustrated in the next example.

Example 8. Let $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$ be the following three component automata:

$$\mathcal{A}_1 : p_0 \xrightarrow{a!} p_1, p_0 \xrightarrow{c?} p_2 \quad \mathcal{A}_2 : q_0 \xrightarrow{b!} q_1 \xrightarrow{a?} q_2 \quad \mathcal{A}_3 : r_0 \xrightarrow{c!} r_1 \xrightarrow{b?} r_2$$

Let system $\mathcal{S}_1$ consist of $\mathcal{A}_1$ and $\mathcal{A}_2$ and $\mathcal{S}_2$ be the system consisting only of $\mathcal{A}_3$. The only communicating action in $\mathcal{S}_1$ is a, since b and c are interface actions. Let $\mathcal{E}_1(st_1)$ be the ETA over $\mathcal{S}_1$ determined by $st_1(a) = ([1,1],[1,1])$. There are two receptiveness requirements generated by st_1 which are $\mathtt{rcp}(\{\mathcal{A}_1\},a)@(p_0,q_0)$ and $\mathtt{rcp}(\{\mathcal{A}_1\},a)@(p_0,q_1)$. Obviously, $\mathcal{E}_1(st_1)$ is weakly compliant with both. For instance the first one is satisfied by the system transitions

$$(p_0,q_0) \xrightarrow{(\{\mathcal{A}_2\},b,\varnothing)}_{\mathcal{E}_1(st_1)} (p_0,q_1) \xrightarrow{(\{\mathcal{A}_1\},a,\{\mathcal{A}_2\})}_{\mathcal{E}_1(st_1)} (p_1,q_2)$$

The intermediate transition before accepting a executes interface action b.

Since $\mathcal{S}_2$ has no communicating actions, there are no synchronisation types and the ETA determined by the empty synchronisation type specification $st_\varnothing$ is $\mathcal{E}_2(st_\varnothing)$ which coincides with $\mathcal{A}_3$ when actions are replaced by system labels. Now we set $st_{\mathit{inf}}(b) = st_{\mathit{inf}}(c) = ([1,1],[1,1])$ and consider the ETA composition $\mathcal{E}_1(st_1) \otimes \mathcal{E}_2(st_\varnothing) = \mathcal{E}(st)$. For interface action b we get receptiveness requirement $\mathtt{rcp}(\{\mathcal{A}_2\},b)@(p_0,q_0,r_0)$ and for c we get $\mathtt{rcp}(\{\mathcal{A}_3\},c)@(p_0,q_0,r_0)$. $\mathcal{E}(st)$ is weakly compliant with the first one and compliant with the second one. So all looks fine. In the first case the weak compliance holds because of the transitions

$$(p_0,q_0,r_0) \xrightarrow{(\{\mathcal{A}_3\},c,\{\mathcal{A}_1\})}_{\mathcal{E}(st)} (p_2,q_0,r_1) \xrightarrow{(\{\mathcal{A}_2\},b,\{\mathcal{A}_3\})}_{\mathcal{E}(st)} (p_2,q_1,r_2)$$

The subtle point is here that in the first transition $\mathcal{A}_3$ 'calls back' to a component in $\mathcal{S}_1$, namely $\mathcal{A}_1$, before satisfying the receptiveness requirement of $\mathcal{A}_2$. This creates a kind of cycle which makes the overall team $\mathcal{E}(st)$ not weakly compliant with $\mathtt{rcp}(\{\mathcal{A}_1\},a)@(p_0,q_0,r_0)$. Indeed, if $\mathcal{A}_1$ wants to send a then $\mathcal{A}_2$ must first send b to $\mathcal{A}_3$ which must first send c to $\mathcal{A}_1$. Hence the requirement of $\mathcal{A}_1$ to send b is not satisfiable in the overall team. Therefore we must exclude the possibility of such 'call backs' when checking weak compliance for interface actions. This is taken into account in the conditions (a) and (b) of Theorem 3, where we consider weak compliance without participation of components of the sub-system where a requirement stems from. $\square$

The necessary assumptions discussed so far for obtaining compositionality in the cases of weak receptiveness and weak responsiveness are summarised in the following theorem. It additionally requires determinism of the sub-teams in order to get a unique lifting of intermediate activities in sub-teams when weak compliance is considered.

Theorem 3. *Let $\mathcal{E}_k(st_k)$, $st_{\mathit{inf}}(a)$, and $\mathcal{E}(st)$ be as in Theorem 1 for all $k \in [n]$ such that $st_{\mathit{inf}}(a) = ([o_1, \#\mathit{dom}_{a,\mathit{out}}(\mathcal{S})],[i_1, \#\mathit{dom}_{a,\mathit{inp}}(\mathcal{S})])$ with $o_1, i_1 \in \{0,1\}$ for all $a \in \Sigma_{\mathit{inf}}$. Let each $\mathcal{E}_k(st_k)$ be deterministic and weakly receptive (respectively, weakly responsive).*

Assume that $\mathcal{E}(st)$ is weakly compliant, for all $a \in \Sigma_{\mathit{inf}}$ and $q \in \mathcal{R}(\mathcal{E}(st))$, with all atomic communication requirements $\mathtt{rcp}(\mathcal{J},a)@q$ and $\mathtt{rsp}(\mathcal{J},a)@q$ that are valid for $st_{\mathit{inf}}(a)$. Moreover, assume that if $\varnothing \neq \mathcal{J} \subseteq \mathcal{N}_k$ for some $k \in [n]$, then the weak compliance holds 'without participation of components in $\mathcal{N}_k \setminus \mathcal{J}$', i.e. we assume that:

(a) $\mathtt{rcp}(\mathcal{J}, a)@q$ *holds since there exist* $p, q' \in Q$*,* $out \supseteq \mathcal{J}$ *and* $\varnothing \subseteq inp$ *such that* $out \cap \mathcal{N}_k = \mathcal{J}$ *and* $q \xrightarrow{\Lambda(\mathcal{S}) \setminus \Lambda_{\mathcal{N}_k}(\mathcal{S})}{}^{*}_{\mathcal{E}(st)}\ p \xrightarrow{(out,a,inp)}_{\mathcal{E}(st)} q'$*, and*

(b) $\mathtt{rsp}(\mathcal{J}, a)@q$ *holds since there exist* $p, q' \in Q$*,* $\varnothing \subseteq out$ *and* $inp \supseteq \mathcal{J}$ *such that* $inp \cap \mathcal{N}_k = \mathcal{J}$ *and* $q \xrightarrow{\Lambda(\mathcal{S}) \setminus \Lambda_{\mathcal{N}_k}(\mathcal{S})}{}^{*}_{\mathcal{E}(st)}\ p \xrightarrow{(out,a,inp)}_{\mathcal{E}(st)} q'$*.*

Then $\mathcal{E}(st)$ *is weakly receptive (respectively, weakly responsive).*

Proof. (sketch) For interface actions weak compliance of $\mathcal{E}(st)$ with communication requirements is anyway assumed as a proof obligation. For non-interface actions we have to propagate communication properties concerning communicating actions $a \in \Sigma_{k,com}$ from a sub-team $\mathcal{E}_k(st_k)$ to $\mathcal{E}(st)$. The tricky point is here that the notion of weak compliance in a sub-team is so flexible that it allows some intermediate actions before a desired output (respectively, input) is accepted by the sub-team $\mathcal{E}_k(st_k)$. In particular, the intermediate actions can be interface actions. Then it must be guaranteed that those interface actions can be executed as synchronisations in the overall team $\mathcal{E}(st)$ which is ensured by conditions (a) and (b). □

Example 9. Consider the ETA composition $\mathcal{E}_{chat}(st_{chat}) \otimes \mathcal{E}_{arbiter}(st_\varnothing) = \mathcal{E}(st)$ of Example 7. The ETA $\mathcal{E}_{chat}(st_{chat})$ is deterministic, weakly receptive, and weakly responsive (cf. Example 5) and so is, trivially, the ETA $\mathcal{E}_{arbiter}(st_\varnothing)$. Then for the interface actions *ask*, *grant*, and *reject* we obtain the receptiveness requirements $\mathtt{rcp}(\{\mathcal{A}_3\}, ask)@(p, q, 3, 0)$, $\mathtt{rcp}(\{\mathcal{A}_4\}, grant)@(p, q, 4, 1)$, and $\mathtt{rcp}(\{\mathcal{A}_4\}, reject)@(p, q, 4, 1)$ for any states p and q of the two clients such that the given global state is reachable within $\mathcal{E}(st)$. Obviously, $\mathcal{E}(st)$ is (even) compliant with these requirements. Hence condition (a) of Theorem 3 holds.

Now on the other hand, for the interface actions we obtain the responsiveness requirements $\mathtt{rcp}(\{\mathcal{A}_3\}, grant)@(p, q, 4, 1)$, $\mathtt{rcp}(\{\mathcal{A}_3\}, reject)@(p, q, 4, 1)$, and $\mathtt{rcp}(\{\mathcal{A}_4\}, ask)@(p, q, r, 0)$ for any local states p, q, r such that the given global state is reachable within $\mathcal{E}(st)$. Obviously, $\mathcal{E}(st)$ is (even) compliant with the first two requirements and with the third requirement if $r = 3$. In all other possible cases for r, $\mathcal{E}(st)$ is weakly compliant with $\mathtt{rcp}(\{\mathcal{A}_4\}, ask)@(p, q, r, 0)$. For instance, in the initial state $(0, 0, 0, 0)$ there is a path in $\mathcal{E}(st)$ without participation of the arbiter reaching state $(2, 0, 3, 0)$ (where the server already received a *msg* from client $\mathcal{A}_1$). Then the input *ask* of the arbiter can be served by the server. Since this holds similarly in all other cases, condition (b) of Theorem 3 is satisfied. Hence, as a consequence of Theorem 3, $\mathcal{E}(st)$ is weakly receptive and weakly responsive. □

7 Related Work

In the literature, compatibility notions are typically restricted to receptiveness requirements in system models with binary, synchronous commmunication [4, 25–28]. Our approach is generic, generating notions of communication safety for various kinds of synchronisation types. Concerning receptiveness and synchronisation type $st(a) = ([1, 1], [1, 1])$ for all actions a, it subsumes, e.g., compatibility notions of [4, 15] and [20] (for closed systems), and, for a limited weak case, those of [22] and [5].

We are aware of just a few approaches that consider notions of compatibility with respect to responsiveness. Both [15] and [22] consider system models with synchronous composition. Notably, in [15] responsiveness is captured by deadlock-freeness, while in [22] responsiveness is expressed as part of the definition of bidirectional complementarity compatibility. The latter, however, does not support a choice of input actions like we do. Finally, [17] can express sending constraints on partners in an asynchronous environment. It supports two kinds of communication styles: client/server and peer-to-peer.

Synchronisation types constrain the number of components which can simultaneously execute a *shared* action. There are approaches which do not rely on shared actions but specify possible interactions by determining which actions may or must synchronise. This originates already in Winskel's synchronisation algebras [31] providing an abstract model to specify different synchronisation styles for parallel composition. For describing system architectures BIP [1] proposes interaction models using connectors for ports (actions) of components. Typical architecture styles can be graphically represented [30]. Also compositionality results are provided but the focus is not on the analysis of input and output compatibilities.

8 Conclusion

We considered ETA, their specification by synchronisation types, and their (weak) compliance with communication requirements generated from synchronisation type specifications. In this sense our approach is generic, generating notions of communication safety for various kinds of synchronisation types. An essential contribution concerns the composition of systems and of ETA, and the investigation of criteria ensuring preservation of communication properties by composition. Verification of communication requirements in concrete cases is still a tedious task, which should be supported by appropriate future tools. Moreover, the validation of our approach on the basis of larger case studies is a future goal. From a software engineering perspective we are also interested in hierarchical designs where sub-teams are first encapsulated into CA by hiding communicating actions to make analysis of larger systems feasible, e.g., by using techniques of minimisation with respect to observational equivalence. Moreover, to support reusability we could also add an explicit notion of system connector to match actions of different systems by renaming. A further desired extension concerns the introduction of designated states in CA where execution can stop but may also continue, in addition to states where progress is required. As sketched in [10], their addition has significant and useful consequences for the derivation of communication requirements and compliance.

Acknowledgments We thank the anonymous reviewers for their useful comments. The first author is supported by MIUR PRIN 2017FTXR7S project IT MaTTerS (Methods and Tools for Trustworthy Smart Systems).

References

1. Attie, P.C., Baranov, E., Bliudze, S., Jaber, M., Sifakis, J.: A general framework for architecture composability. Formal Asp. Comput. **28**(2), 207–231 (2016). https://doi.org/10.1007/s00165-015-0349-8
2. Bartoletti, M., Cimoli, T., Zunino, R.: Compliance in Behavioural Contracts: A Brief Survey. In: Bodei, C., Ferrari, G.L., Priami, C. (eds.) Programming Languages with Applications to Biology and Security. LNCS, vol. 9465, pp. 103–121. Springer (2015). https://doi.org/10.1007/978-3-319-25527-9_9
3. Basile, D., ter Beek, M.H., Legay, A.: Timed service contract automata. Innovations Syst. Softw. Eng. **2**(16), 199–214 (2020). https://doi.org/10.1007/s11334-019-00353-3
4. Basile, D., Degano, P., Ferrari, G.L.: Automata for Specifying and Orchestrating Service Contracts. Log. Meth. Comp. Sci. **12**(4:6), 1–51 (2016). https://doi.org/10.2168/LMCS-12(4:6)2016
5. Bauer, S.S., Mayer, P., Schroeder, A., Hennicker, R.: On Weak Modal Compatibility, Refinement, and the MIO Workbench. In: Esparza, J., Majumdar, R. (eds.) TACAS'10. LNCS, vol. 6015, pp. 175–189. Springer (2010). https://doi.org/10.1007/978-3-642-12002-2_15
6. ter Beek, M.H., Carmona, J., Hennicker, R., Kleijn, J.: Communication Requirements for Team Automata. In: Jacquet, J.M., Massink, M. (eds.) COORDINATION'17. LNCS, vol. 10319, pp. 256–277. Springer (2017). https://doi.org/10.1007/978-3-319-59746-1_14
7. ter Beek, M.H., Carmona, J., Kleijn, J.: Conditions for Compatibility of Components: The Case of Masters and Slaves. In: Margaria, T., Steffen, B. (eds.) ISoLA'16. LNCS, vol. 9952, pp. 784–805. Springer (2016). https://doi.org/10.1007/978-3-319-47166-2_55
8. ter Beek, M.H., Ellis, C.A., Kleijn, J., Rozenberg, G.: Synchronizations in Team Automata for Groupware Systems. Comput. Sup. Coop. Work **12**(1), 21–69 (2003). https://doi.org/10.1023/A:1022407907596
9. ter Beek, M.H., Hennicker, R., Kleijn, J.: Compositionality of Safe Communication in Systems of Team Automata. Tech. rep., Zenodo (September 2020). https://doi.org/10.5281/zenodo.4050293
10. ter Beek, M.H., Hennicker, R., Kleijn, J.: Team Automata@Work: On Safe Communication. In: Bliudze, S., Bocchi, L. (eds.) COORDINATION'20. LNCS, vol. 12134, pp. 77–85. Springer (2020). https://doi.org/10.1007/978-3-030-50029-0_5
11. ter Beek, M.H., Kleijn, J.: Associativity of Infinite Synchronized Shuffles and Team Automata. Fundam. Inform. **91**(3-4), 437–461 (2009). https://doi.org/10.3233/FI-2009-0051
12. Bordeaux, L., Salaün, G., Berardi, D., Mecella, M.: When are Two Web Services Compatible? In: Shan, M.C., Dayal, U., Hsu, M. (eds.) TES'04. LNCS, vol. 3324, pp. 15–28. Springer (2005). https://doi.org/10.1007/978-3-540-31811-8_2
13. Brand, D., Zafiropulo, P.: On Communicating Finite-State Machines. J. ACM **30**(2), 323–342 (1983). https://doi.org/10.1145/322374.322380
14. Brim, L., Cerná, I., Vareková, P., Zimmerova, B.: Component-Interaction Automata as a Verification-Oriented Component-Based System Specification. ACM Softw. Eng. Notes **31**(2) (2006). https://doi.org/10.1145/1118537.1123063
15. Carmona, J., Cortadella, J.: Input/Output Compatibility of Reactive Systems. In: Aagaard, M.D., O'Leary, J.W. (eds.) FMCAD'02. LNCS, vol. 2517, pp. 360–377. Springer (2002). https://doi.org/10.1007/3-540-36126-X_22

16. Carmona, J., Kleijn, J.: Compatibility in a multi-component environment. Theor. Comput. Sci. **484**, 1–15 (2013). https://doi.org/10.1016/j.tcs.2013.03.006
17. Carrez, C., Fantechi, A., Najm, E.: Behavioural Contracts for a Sound Assembly of Components. In: König, H., Heiner, M., Wolisz, A. (eds.) FORTE'03. LNCS, vol. 2767, pp. 111–126. Springer (2003). https://doi.org/10.1007/978-3-540-39979-7_8
18. Castagna, G., Gesbert, N., Padovani, L.: A Theory of Contracts for Web Services. ACM Trans. Program. Lang. Syst. **31**(5), 19:1–19:61 (2009). https://doi.org/10.1145/1538917.1538920
19. David, A., Larsen, K.G., Legay, A., Nyman, U., Wąsowski, A.: Timed I/O Automata: A Complete Specification Theory for Real-time Systems. In: Proceedings of the 13th International Conference on Hybrid Systems: Computation and Control (HSCC'10). pp. 91–100. ACM (2010). https://doi.org/10.1145/1755952.1755967
20. de Alfaro, L., Henzinger, T.A.: Interface Automata. In: Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE'01). pp. 109–120. ACM (2001). https://doi.org/10.1145/503209.503226
21. de Alfaro, L., Henzinger, T.A.: Interface-Based Design. In: Broy, M., Grünbauer, J., Harel, D., Hoare, T. (eds.) Engineering Theories of Software Intensive Systems, NATO Science Series, vol. 195, pp. 83–104. Springer (2005). https://doi.org/10.1007/1-4020-3532-2_3
22. Durán, F., Ouederni, M., Salaün, G.: A generic framework for n-protocol compatibility checking. Sci. Comput. Program. **77**(7-8), 870–886 (2012). https://doi.org/10.1016/j.scico.2011.03.009
23. Ellis, C.A.: Team Automata for Groupware Systems. In: Proceedings of the 1st International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge (GROUP'97). pp. 415–424. ACM (1997). https://doi.org/10.1145/266838.267363
24. Engels, G., Groenewegen, L.: Towards Team-Automata-Driven Object-Oriented Collaborative Work. In: Brauer, W., Ehrig, H., Karhumäki, J., Salomaa, A. (eds.) Formal and Natural Computing, LNCS, vol. 2300, pp. 257–276. Springer (2002). https://doi.org/10.1007/3-540-45711-9_15
25. Hennicker, R., Bidoit, M.: Compatibility Properties of Synchronously and Asynchronously Communicating Components. Log. Meth. Comp. Sci. **14**(1), 1–31 (2018). https://doi.org/10.23638/LMCS-14(1:1)2018
26. Hennicker, R., Knapp, A.: Moving from interface theories to assembly theories. Acta Inf. **52**(2-3), 235–268 (2015). https://doi.org/10.1007/s00236-015-0220-7
27. Larsen, K.G., Nyman, U., Wąsowski, A.: Modal I/O Automata for Interface and Product Line Theories. In: De Nicola, R. (ed.) ESOP'07. LNCS, vol. 4421, pp. 64–79. Springer (2007). https://doi.org/10.1007/978-3-540-71316-6_6
28. Lüttgen, G., Vogler, W., Fendrich, S.: Richer interface automata with optimistic and pessimistic compatibility. Acta Inf. **52**(4-5), 305–336 (2015). https://doi.org/10.1007/s00236-014-0211-0
29. Lynch, N.A., Tuttle, M.R.: An Introduction to Input/Output Automata. CWI Q. **2**(3), 219–246 (1989), https://ir.cwi.nl/pub/18164
30. Mavridou, A., Baranov, E., Bliudze, S., Sifakis, J.: Architecture Diagrams: A Graphical Language for Architecture Style Specification. EPTCS, vol. 223, pp. 83–97 (2016). https://doi.org/10.4204/eptcs.223.6
31. Winskel, G.: Synchronization trees. Theor. Comput. Sci. **34**(1), 33–82 (1984). https://doi.org/10.1016/0304-3975(84)90112-9