A Shared-Medium Communication Architecture for Distributed Discrete Event Systems

K. Schmidt*, E.G. Schmidt†, J. Zaddach*
* Lehrstuhl für Regelungstechnik, Universität Erlangen-Nürnberg, Erlangen, Germany
†Department of Electrical and Electronics Engineering, Middle East Technical University, Ankara, Turkey

***Abstract*— Recently, several efficient supervisor synthesis approaches for distributed discrete event systems (DES) have been established. In this paper, the implementation of such supervisors on interacting distributed programmable logic controllers (PLCs) on a network is considered for the hierarchical and decentralized control approach elaborated in our previous work. A communication model that captures the controller behavior relevant for communication is developed, and a network architecture together with a scheduling policy that ensures correct operation of the networked controllers is proposed. In addition to the formal statements, simulation results for an example system are presented.**

I. Introduction

In the recent years, a variety of supervisor design methods for discrete event systems that result in interacting *modular* and *decentralized* controllers have been developed [1], [2], [3], [4], [5], [6], [7], [8], [9].

In these works, controller interaction is modeled by *shared event* occurrences that have to be synchronized among controllers. However, the realization of this interaction is not addressed. As long as the controllers are implemented on a single device (PC, PLC, etc.), the interaction takes place internally, e.g. via shared memory. In contrast, if each controller is situated in a different physical location, communication is required. This work presents the theoretical framework of a *communication architecture* on a shared-medium such as Ethernet for the hierarchical and decentralized control approach in [6]. In our *communication model*, the system is represented as a set of *nodes* that communicate in order to execute system *tasks* (shared events). Communication messages for a task are identified with *jobs* that have to comply with real-time requirements. Timely message transmission is ensured by a scheduling policy that exploits the deterministic system structure. Although our communication architecture is not specifically designed for Ethernet, it can easily be implemented using off-the shelf Ethernet ICs.

We show that there exists a lower bound on the network speed such that our communication architecture works correctly. We also present a preliminary simulation study to investigate the average communication performance.

The paper outline is as follows. In Section II, we present the underlying hierarchical and decentralized control approach. The communication model and the communication architecture are developed in Section III and in Section IV, respectively. Section V provides simulation results and we give conclusions in Section VI.

II. Preliminaries

A. Basic notation

We recall the basic notations regarding DES [10].

For a finite alphabet Σ, the set of all finite strings over Σ is denoted Σ^*. We write $s_1 s_2 \in \Sigma^*$ for the concatenation of two strings $s_1, s_2 \in \Sigma^*$, and $s_1 \leq s$ when s_1 is a *prefix* of s, i.e. if $s = s_1 s_2$ with $s_2 \in \Sigma^*$. The empty string is denoted $\varepsilon \in \Sigma^*$, i.e. $s\varepsilon = \varepsilon s = s$ for all $s \in \Sigma^*$. A *language* over Σ is a subset $H \subseteq \Sigma^*$. The *prefix closure* of H is defined by $\overline{H} := \{s_1 \in \Sigma^* \mid \exists s \in H \text{ s.t. } s_1 \leq s\}$. The *natural projection* $p_i : \Sigma^* \to \Sigma_i^*$, $i = 1,2$, for the union $\Sigma = \Sigma_1 \cup \Sigma_2$ is defined iteratively: (1) let $p_i(\varepsilon) := \varepsilon$; (2) for $s \in \Sigma^*$, $\sigma \in \Sigma$, let $p_i(s\sigma) := p_i(s)\sigma$ if $\sigma \in \Sigma_i$, or $p_i(s\sigma) := p_i(s)$ otherwise. The set-valued inverse of p_i is denoted $p_i^{-1} : \Sigma_i^* \to 2^{\Sigma^*}$, $p_i^{-1}(t) := \{s \in \Sigma^* \mid p_i(s) = t\}$. The *synchronous product* of $H_i \subseteq \Sigma_i^*$, $i = 1,2$ is $H_1 || H_2 = p_1^{-1}(H_1) \cap p_2^{-1}(H_2) \subseteq \Sigma^*$.

A *finite automaton* is a tuple $G = (X, \Sigma, \delta, x_0, X_m)$ with the finite set of *states* X, the finite alphabet of *events* Σ, the partial *transition function* $\delta : X \times \Sigma \to X$, the *initial state* $x_0 \in X$, and the set of *marked states* $X_m \subseteq X$. We write $\delta(x,\sigma)!$ if $\delta(x,\sigma)$ is defined. In order to extend δ to a partial function on $X \times \Sigma^*$, recursively let $\delta(x,\varepsilon) := x$ and $\delta(x, s\sigma) := \delta(\delta(x,s),\sigma)$, whenever both $x' = \delta(x,s)$ and $\delta(x',\sigma)!$. Also we write $\delta(x,\Sigma') = x'$ if $\delta(x,\sigma) = x'$ for each $\sigma \in \Sigma' \subseteq \Sigma$. $L(G) := \{s \in \Sigma^* : \delta(x_0,s)!\}$ and $L_m(G) := \{s \in L(G) : \delta(x_0,s) \in X_m\}$ are the *closed* and *marked language* of G, respectively. G is denoted *nonblocking* if $L(G) = \overline{L_m(G)}$. The synchronous composition of two automata G_1 and G_2 is defined such that $L(G_1||G_2) = L(G_1)||L(G_2)$ (see e.g. [10]).

As supervisor synthesis (see e.g. [10]) is not the focus of this paper, we just consider the closed-loop behavior after supervisor design and model it as an automaton R.

B. Hierarchical and Decentralized Approach

In practice, DES can be modeled as a set of finite automata (e.g. different components of a manufacturing system) that can be composed to an overall system model. However, supervisor design for such a *compound DES* is often computationally infeasible due to the fact that the number of states grows exponentially with the number of system components (*state space explosion problem*).

Several control approaches circumvent this problem [1], [2], [3], [4], [5], [6], [7], [8], [9]. These methods employ *modular*, *decentralized* and *hierarchical* synthesis techniques in order to reduce the computational effort for supervisor computation and representation.

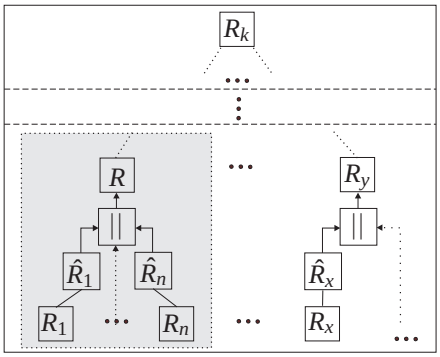

Fig. 1. Hierarchical and decentralized architecture

In this paper, we employ the hierarchical and decentralized approach developed in [6]. It results in a set of supervisors on small state spaces in a hierarchical relationship as depicted in the gray box of Fig. 1. The supervisors are represented by finite automata, where $R_1,\dots,R_n$ ($R_i=(X_i,\Sigma_i,\delta_i,x_{0,i},X_{\mathrm{m},i})$) are low-level supervisors, R is a high-level supervisor and the abstractions $\hat{R}_1,\dots,\hat{R}_n$ ($\hat{R}_i=(\hat{X}_i,\hat{\Sigma}_i,\hat{\delta}_i,\hat{x}_{0,i},\hat{X}_{\mathrm{m},i})$) of the respective R_i are used to compute R. The interaction of the supervisors is defined such that the compound behavior is represented by the language $L(R)||(||_{i=1}^{n}L(R_i))$. The following properties relate the high level and the low level of the hierarchy:

- the natural projections $\hat{p}_i:\Sigma_i^*\to\hat{\Sigma}_i^*$ are used for abstraction: $L(\hat{R}_i)=\hat{p}_i(L(R_i))$, $L_{\mathrm{m}}(\hat{R}_i)=\hat{p}_i(L_{\mathrm{m}}(R_i))$.
- it is required that the shared events are included in the abstraction alphabets, i.e. $\bigcup_{j=1,j\neq i}^{n}(\Sigma_i\cap\Sigma_j)\subseteq\hat{\Sigma}_i$ for all $i=1,\dots,n$.
- $L(R)\subseteq L(||_{i=1}^{n}\hat{R}_i)$,
- nonblocking control is guaranteed, i.e. $L(R)||(||_{i=1}^{n}L(R_i))=\overline{L_{\mathrm{m}}(R)||(||_{i=1}^{n}L_{\mathrm{m}}(R_i))}$.

As elaborated in [6], the design process can be repeated on multiple hierarchical levels as shown in Fig. 1.

From the communication point of view, the controller synthesis provides a set $\mathcal{R}=\{R_1,\dots,R_k\}$ of m distributed supervisors in a parent-children hierarchical relationship that interact via their shared events. We define the maps $p_{\mathcal{R}}:\mathcal{R}\to\mathcal{R}$ and $c_{\mathcal{R}}:\mathcal{R}\to 2^{\mathcal{R}}$, where $p_{\mathcal{R}}(R_i)$ is the parent and $c_{\mathcal{R}}(R_i)$ is the set of children of $R_i\in\mathcal{R}$.

Example 1 Fig. 2 shows a simple hierarchical architecture with two levels of abstraction and $k=6$ automata. The shared event φ can only be executed if the components R_1, R_2 and R_5 can participate in the respective states 3, 1 and 1. Observe that $L(R_5)\subseteq L(||_{i,R_i\in c_{\mathcal{R}}(R_5)}\hat{R}_i)$, where $c_{\mathcal{R}}(R_5)=\{R_1,R_2,R_3\}$ is the set of children of R_5. □

III. Communication Model

A. Communication via Jobs

Communication is required if the supervisors as computed in Section II-B are implemented in a number of (e.g. k) PLCs that are situated in distinct physical locations and connected by a network, e.g. on a factory floor. In this case, the occurrence of shared events $\Sigma_\cap:=\bigcup_{i,j=1,i\neq j}^{k}(\Sigma_i\cap\Sigma_j)$ has to be communicated, i.e. all supervisors that share some event must agree on its execution.

Our communication idea is illustrated in the following example. We denote shared events *tasks* and identify the required communication messages with so-called *jobs*. In addition to the conditions for the hierarchical controller synthesis in [6], we require that if an event $\sigma\in\Sigma_\cap$ is possible in a state of R_i, then there is no local string such that σ is no longer possible, i.e. $\forall s\in L(R_i),\sigma\in\Sigma_\cap$ with $s\sigma\in L(R_i):\nexists u\in(\Sigma_i-\Sigma_\cap)^*$ s.t. $su\in L(R_i)\wedge su\sigma\notin L(R_i)$.

Example 2 We consider the system in Fig. 2 with each automaton in its initial state. The first task is α (shared by R_1, R_3, R_4, R_5 and R_6). We propose to propagate the information about the execution of α from the highest-level component that contains α to the lower level components using the parent-children relationship. That is, R_6 first asks R_4 and R_5 "is α possible?" (we identify this *question* with a job $?\alpha_{R_6}$). R_4 can directly answer "α is possible!" ($!\alpha_{R_4}$), while R_5 has to execute φ before α. As R_5 is the highest-level component for φ, it asks R_1 and R_2 "is φ possible?" ($?\varphi_{R_5}$). R_2 can directly answer "φ is possible!" ($!\varphi_{R_2}$), while R_1 has to wait for the occurrence of the local string `kl` (independent of the other supervisors) and then answers "φ is possible!" ($!\varphi_{R_1}$). This triggers the *command* job "execute φ" (φ_c) from R_5 to R_1 and R_2 and causes state changes in R_5 ($1\to 2$, R_1 ($3\to 4$) and R_2 ($1\to 2$). Now, R_5 has to ask R_1 and R_3 "is α possible?" ($?\alpha_{R_5}$) and this question-answer procedure continues until all answers for α ($!\alpha_{R_4}$ and $!\alpha_{R_5}$) have arrived at R_6. This triggers the command "execute α" (α_c) and the respective state changes in R_6,R_5,R_1,and R_3. Now, the highest-level supervisor R_6 for the tasks β, γ and δ starts a new question-answer-command procedure. □

We formalize the ideas presented in Example 2 by defining a *communication structure* $CS_i^{x,\sigma}=(C_i^{x,\sigma},\mathcal{J}_i^{x,\sigma},\nu_i^{x,\sigma},c_{0,i}^{x,\sigma},C_{\mathrm{m},i}^{x,\sigma})$. It captures the job combinations for a transition $x':=\delta(x,\sigma)$ defined in $x\in X_i$ (see Fig. 3). If the parent $p_{\mathcal{R}}(R_i)$ contains σ, communication of R_i starts with a question $?\sigma_{p_{\mathcal{R}}(R_i)}$. Otherwise, we set

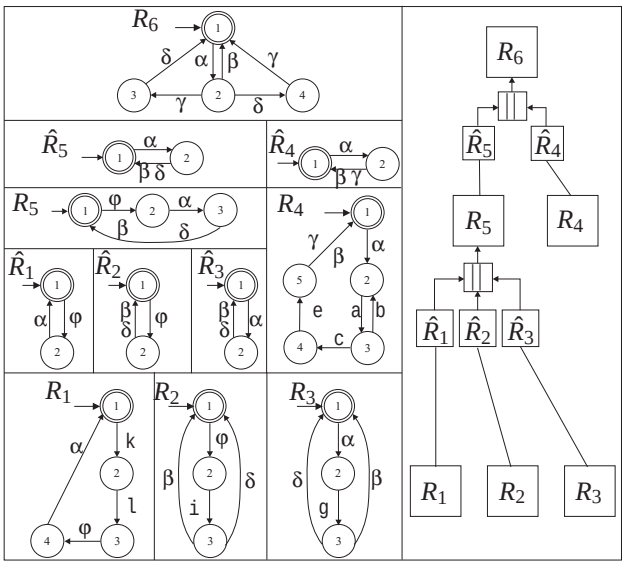

Fig. 2. Distributed Supervisors (left); Architecture (right)

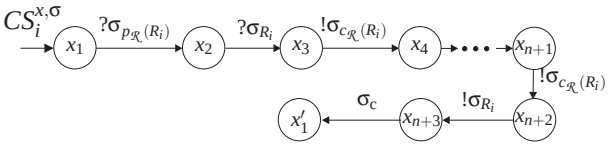

Fig. 3. Communication Structure

$?\sigma_{p_{\mathcal{R}}(R_i)} = \varepsilon$. If there are children of R_i that contain σ, the question $?\sigma_{R_i}$ is asked, and an answer is expected for each child that contains σ. We denote the set of answers $!\sigma_{c_{\mathcal{R}}(R_i)}$, and the number n of such children is $n = |!\sigma_{c_{\mathcal{R}}(R_i)}|$. If there are no children with σ, both $?\sigma_{R_i} = \varepsilon$ and $!\sigma_{c_{\mathcal{R}}(R_i)} = \varepsilon$. Next, the answer $!\sigma_{R_i}$ to the question $?\sigma_{p_{\mathcal{R}}(R_i)}$ is given and the "command" job σ_c terminates the communication for σ. The labeling of the states is defined as in Fig. 3, i.e. the states $x_1,\ldots,x_{n+3}$ correspond to x, and x'_1 corresponds to x'. Furthermore, $x_1,\ldots,x_{n+3}$ are marked if and only if x is marked. Note that the jobs $?\sigma_{p_{\mathcal{R}}(R_i)}$ and $!\sigma_{R_i}$ are shared with the parent $p_{\mathcal{R}}(R_i)$, the job $?\sigma_{R_i}$ is shared with all children of R_i that contain σ, each job in $!\sigma_{c_{\mathcal{R}}(R_i)}$ is shared with a child of R_i, and the command job σ_c is shared with all supervisors that contain σ. Thus, communication is represented by shared jobs between distributed supervisors. For later use in algorithmic computations, we introduce the function $\mathtt{cs}(x,x',?\sigma_{p_{\mathcal{R}}(R_i)},?\sigma_{R_i},!\sigma_{c_{\mathcal{R}}(R_i)},!\sigma_{R_i},\sigma_c,R_i^\sigma)$, that adds the communication structure in Fig. 3 to an automaton R_i^σ between the states x and x'.

The set of jobs of an event σ is defined as follows.

Definition 3.1 (Set of Jobs) Let $\mathcal{R} = \{R_1,\ldots,R_k\}$ be a set of distributed supervisors and $\sigma \in \Sigma_\cap$ be a shared event. The *set of jobs* $\mathcal{J}^\sigma$ for σ is defined as

$$\mathcal{J}^\sigma := \bigcup_{i,\sigma\in\Sigma_i} \mathcal{J}_i^\sigma,$$

where $\mathcal{J}_i^\sigma$ is the alphabet common to all communication structures $CS_i^{x,\sigma}$. The set of jobs transmitted by R_i is

$$\Sigma_{\text{out},i} := \textstyle\bigcup_{\sigma\in\Sigma_i}\{?\sigma_{R_i},!\sigma_{R_i}\} \cup \bigcup_{\sigma\in(\Sigma_i-\hat\Sigma_i)}\{\sigma_c\} \qquad \square$$

B. Logical Communication Model

The communication model for a distributed supervisor is formulated as a composition of automata that capture the job sequences for each of its events.

We compute the automaton $R_i^\sigma = (X_i^\sigma,\Sigma_i^\sigma,\delta_i^\sigma,x_{0,i}^\sigma,X_{\text{m},i}^\sigma)$ that defines the order of jobs for the event σ and a supervisor $R_i \in \mathcal{R}$ with the following algorithm.

Algorithm 3.1 (Computation of R_i^σ) Given: R_i, σ.
Initialize: $X_i^\sigma = X_i$; $\Sigma_i^\sigma = \mathcal{J}_i^\sigma \cup \bigcup_{\tau\in\Sigma_i-\{\sigma\}}\{\tau_c\}$; $x_{0,i}^\sigma = x_{0,i}$; $X_{\text{m},i}^\sigma = X_{\text{m},i}$
for each $x \in X_i^\sigma$
  replace x by x_1 in R_i^σ
  if $\delta_i(x,\sigma)!$
    $\mathtt{cs}(x,\delta_i(x,\sigma),?\sigma_{p_{\mathcal{R}}(R_i)},?\sigma_{R_i},!\sigma_{c_{\mathcal{R}}(R_i)},!\sigma_{R_i},\sigma_c,R_i^\sigma)$
for each $x_k \in X_i^\sigma$
  for each $\tau \in \Sigma_i - \{\sigma\}$
    if $x' := \delta_i(x,\tau)!$ and $x'_k \in X_i^\sigma$
      set $\delta_i^\sigma(x_k,\tau_c) := x'_k$
    else
      $\delta_i^\sigma(x_k,\tau_c) = x'_1$ $\square$

Algorithm 3.1 is initialized with the state set, initial state and marked states of R_i. The first "for"-loop imposes the communication structure for event σ in each state of R_i where σ is possible. The second "for" loop adds a "command" job for each event different from σ and remembers previously communicated jobs.

Algorithm 3.2 computes the automaton $R'_i = (X',\Sigma',\delta',x'_0,X'_\text{m})$ that defines when jobs in a supervisor $R_i \in \mathcal{R}$ can be transmitted.

Algorithm 3.2 (Computation of R'_i) Given: R_i, $\hat\Sigma_i$
Initialize: $X'_i = X_i$; $\Sigma'_i = \bigcup_{\sigma\in\hat\Sigma_i}(\mathcal{J}_i^\sigma - \{?\sigma_{p_{\mathcal{R}}(R_i)}\}) \cup \bigcup_{\sigma\in\Sigma_i}\sigma_c$; $x'_{0,i} = x_{0,i}$; $X'_{\text{m},i} = X_{\text{m},i}$
for each $x \in X'_i$
  for each $\sigma \in \Sigma_i$
    if $x' = \delta_i(x,\sigma)!$
      set $\delta'_i(x,\sigma_c) := x'$
      if $\sigma \in \hat\Sigma_i$
        set $\delta'_i(x,\mathcal{J}_i^\sigma - \{\sigma_c,?\sigma_{p_{\mathcal{R}}(R_i)}\}) := x$ $\square$

Algorithm 3.2 is initialized with the state set, initial state and marked states of the supervisor automaton R_i. Each transition in R_i is replaced by a transition with the command job for the respective event in R'_i. Additionally, the jobs in $\mathcal{J}_i^\sigma - \{\sigma_c,?\sigma_{p_{\mathcal{R}}(R_i)}\}$ for an event $\sigma \in \hat\Sigma_i$ shall only be feasible in states where the "command" job σ_c is feasible and are added as selfloops.

Using Algorithms 3.1 and 3.2, the communication model CM_{R_i} for $R_i \in \mathcal{R}$ is constructed by composing the automata R'_i and R_i^σ and $\hat R_i^\sigma$ for shared events in R_i.

Definition 3.2 (Logical Communication Model)
Let $\mathcal{R} = \{R_1,\ldots,R_k\}$ be distributed supervisors and $\hat R_1,\ldots,\hat R_{k-1}$ the corresponding abstractions. We denote the shared events in $\sigma \in \Sigma_\cap$ *tasks* with their respective sets of jobs $\mathcal{J}^\sigma$, according to Definition 3.1. The communication model is defined as an automaton $CM_{R_i} = (Q_i,\mathcal{J}_i,\nu_i,q_{0,i},Q_{\text{m},i})$ for each R_i, $i = 1,\ldots,k$ with

$$CM_{R_i} := R'_i||(||_{\sigma\in\hat\Sigma_i}\hat R_i^\sigma)||(||_{\sigma\in\Sigma_i-\hat\Sigma_i}R_i^\sigma) \qquad \square$$

Example 3 Fig. 4 shows the automata $\hat R_1^\varphi$ for $\hat R_1$, $\hat R_1^\alpha$ for $\hat R_1$, R'_1 for R_1 and the communication model $CM_{R_1} = R'_1||\hat R_1^\varphi||\hat R_1^\alpha||R_1^k||R_1^l$. $\square$

In order to compare the system behavior with communication to the behavior of the distributed discrete event supervisors, we define the overall communication alphabet $\mathcal{J} := \bigcup_{\sigma\in\Sigma_\cap}\mathcal{J}^\sigma$ and the *reporter map* $\theta : \mathcal{J}^* \to \Sigma^*$ such

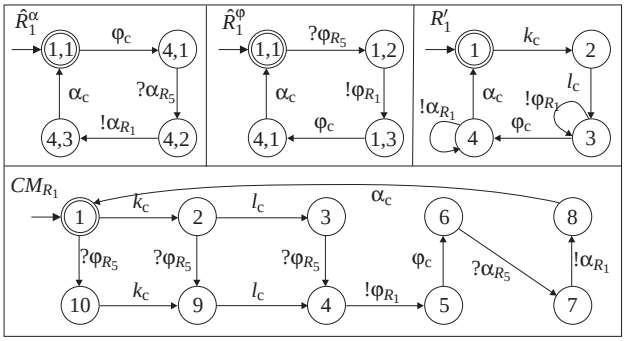

Fig. 4. Communication Model CM_{R_1} and related automata $\hat R_1^\varphi$, $\hat R_1^\alpha$, R'_1

that $\theta(\varepsilon)=\varepsilon$ and $\theta(wj)=\theta(w)\sigma$ if $j=\sigma_c$, $\theta(wj)=\theta(w)$ otherwise, where $w\in\mathcal{J}^*$ and $j\in\mathcal{J}$. That is, θ determines the events executed in a communication sequence in $\mathcal{J}^*$ by remembering the respective "command" jobs.

The following theorem states that the behavior with communication is nonblocking and complies with the behavior of the original distributed supervisors.[1]

Theorem 3.1 (Communication Equivalence) Let $\mathcal{R}=\{R_1,\ldots,R_k\}$ be a set of distributed supervisors according to Section II-B and let $CM_{R_1},\ldots,CM_{R_k}$ be the corresponding communication models in Definition 3.2. Then

$$\overline{||_{i=1}^{k}L_m(CM_{R_i})} = ||_{i=1}^{k}L(CM_{R_i})$$
$$\theta(||_{i=1}^{k}L(CM_{R_i})) = ||_{i=1}^{k}L(R_i) \qquad \square$$

To sum up; we consider that supervisors synthesized according to [6] are implemented in a distributed manner and communicate via a network. The *communication structure* in Section III-A determines the necessary jobs to synchronize the shared events among the distributed supervisors. We then algorithmically compute a *communication model* for each supervisor and thus define rules for job communication that establish equivalence between the behavior of the distributed supervisors and the supervisors according to [6] in Theorem 3.1.

C. Deadlines for Communication Messages

In the previous section, the *logical* system behavior is considered by defining a logical communication model CM_{R_i} for each node R_i. Our communication model is designed for distributed systems on a network where the communication causes delay which affects the system operation. Hence, the logical model has to be extended with real-time requirements such that the equivalence in Theorem 3.1 still holds in case of communication delays.

In this paper, we introduce timing restrictions for shared events as a map $r:\Sigma_\cap\to\mathbb{R}$, where $r(\sigma)$ represents the maximal allowable time between the physical occurrence of an event $\sigma\in\Sigma_\cap$, e.g. a sensor edge, and its execution, i.e. the command job σ_c has been send. [2]

The execution of an event $\sigma\in\Sigma_\cap$ requires the communication of all the jobs in $\mathcal{J}^\sigma$, and the actual event can happen any time between the first and the last job of $\mathcal{J}^\sigma$. Hence, we associate a *deadline* $d_J:=\frac{r(\sigma)}{|\mathcal{J}^\sigma|}$ with each job $J\in\mathcal{J}^\sigma$, assuming that each job in $\mathcal{J}^\sigma$ has the same deadline. In our framework, the deadline indicates that if a job J is ready to be transmitted by its corresponding supervisor at time t_0, then it has to be sent at t_0+d_J latest.

Definition 3.3 (Communication Model with Deadlines) A *communication model with deadlines* is a logical communication model according to Definition 3.2 with a map $r:\Sigma_\cap\to\mathbb{R}\cup\{\infty\}$ as defined above. Also we denote $d_J:=\frac{r(\sigma)}{|\mathcal{J}^\sigma|}$ the *deadline* of $J\in\mathcal{J}^\sigma$. $\square$

Example 4 Consider the timing function r s.t. $r(\alpha)=0.07$, $r(\beta)=0.07$, $r(\gamma)=0.03$, $r(\delta)=0.06$ and $r(\varphi)=0.06$. The corresponding deadlines are $d_{J_\alpha}=0.01$ for $J_\alpha\in\mathcal{J}_\alpha$, $d_{J_\beta}=0.01$ for $J_\beta\in\mathcal{J}_\beta$, $d_{J_\gamma}=0.01$ for $J_\gamma\in\mathcal{J}_\gamma$, $d_{J_\delta}=0.01$ for $J_\delta\in\mathcal{J}_\delta$ and $d_{J_\varphi}=0.015$ for $J_\varphi\in\mathcal{J}_\varphi$. $\square$

The communication model with deadlines in Definition 3.3 is constructed such that jobs $?\sigma_{R_i}$ or $!\sigma_{R_i}$ that are transmitted by R_i, are received by all supervisors that contain the respective job. In doing so, it has to be ensured that whenever a supervisor needs to transmit a job, it has access to the network before the respective deadline. This issue is addressed in the next section.

IV. Proposed Communication Architecture and Operation for Distributed Systems

The distributed supervisors in $\mathcal{R}=\{R_1,\ldots,R_k\}$ according to Section II-B can be represented by a set of corresponding network *nodes* $\mathcal{N}=\{N_1,\ldots,N_k\}$ that are situated in different physical locations (e.g. on PLCs, PCs) and can communicate via a network.

Our communication architecture is based on shared-medium networks, i.e. all network nodes are connected to the same medium as can be seen in Fig. 5. When a message is transmitted on the shared medium, all of the nodes check the message destination address and receive the message accordingly.

The shared medium networks are simple, inexpensive, and they provide inherent broadcast and multicast capabilities. However, if two nodes attempt to communicate at the same time a *collision* occurs and the messages of both of the nodes are destroyed. Different techniques for granting network access to the nodes are developed to cope with the collisions. In Ethernet networks, each node is allowed to communicate at arbitrary time instants. In case of collision, the nodes retransmit after a random interval of time which makes it impossible to provide delay guarantees for the communicated messages.

In this paper, we propose time-slotted operation and a scheduling policy to provide collision-free communication. The time slots are of fixed size t_s, and we assume that there is a synchronization mechanism such that all of the nodes are synchronized with these time slot. Note that such synchronization with an accuracy up to 100ns is for example provided by the IEEE 1588 standard for Ethernet [11] which is already implemented in the Intel IXP465 network processor and integrated in PLCs. Synchronization is vital in our approach as the collision avoidance described below relies on the fact that all nodes know which node will transmit a message in each time slot. This is achieved by exploiting the deterministic structure of the supervisor automata and the hierarchical relationship between supervisors.

a) **Network Node:** In our setting, a network node $N_i\in\mathcal{N}$ consists of the following entities.

N1 an automaton CM_{R_i} according to Definition 3.2,
N2 an *output buffer* that stores messages to be sent,

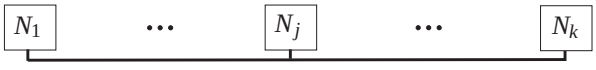

Fig. 5. Nodes on a shared medium

[1] Proofs are omitted in this paper due to space limitations.

[2] Examples for timing restrictions include actuator or sensor events in manufacturing systems that have to be enforced or detected on time.

N3 an *input buffer* that stores received messages,
N4 a set of *active tasks* that contains the tasks (event communications) currently initiated by the node,
N5 a *priority queue* that stores *communication requests* in the form of a tuple (N,e,d,T), where N is a node, $e \in \mathbb{R}$ is an *eligibility time*, $d \in \mathbb{R}$ is a deadline and T is the active task that issued the request. Note that N is required to have access to the shared medium network before d to transmit its message. The priority queue is ordered according to the deadlines of the communication requests. Hence, the communication request that has the smallest deadline is granted first.

We address the fact that it takes time for the nodes to compute e.g. state updates or output messages. We denote this computation time the *eligibility time*. The eligibility time is used to determine when a node is ready to transmit a message in the communication operation.

b) **Message:** Communication between nodes requires the exchange of jobs. This communication is provided by fixed-size *messages*. At most one message is transmitted in each time slot. We assume that the time slot duration is selected long enough to acommodate the longest message to be transmitted.

A message M of a sender node $N_i \in \mathcal{N}$ contains:

M1 a set of *jobs to be sent* by N_i,
M2 a set of *receiver nodes*,
M3 a *minischedule* with a set of communication requests (N_r,e,d,T) for the jobs to be sent from N_i to N_r,
M4 a set of tasks that have been terminated in N_i.

c) **Construction of an Output Message:** The message constructed for the output buffer of a node N_i depends on the current state $q \in Q_i$ of the communication model CM_{R_i} of N_i. The message is computed as follows.
M1 Set of jobs:

- if a "command job" $\sigma_c \in \Sigma_{\text{out},i}$ of N_i is feasible, i.e. $\nu_i(q,\sigma_c)!$, then the string s that starts with σ_c and contains a maximal number of answers and questions of CM_{R_i}, i.e. $s \in \sigma_c\Sigma^*_{\text{out},i}$ with $\nu_i(q,s)!$ is computed.[3] The set of jobs of the message contains all jobs in s.
- otherwise, $s \in \Sigma^*_{\text{out},i}$ is evaluated as above, and the set of jobs is constructed accordingly.

M2 Set of receiver nodes:

- if $s \neq \varepsilon$, then the nodes that share jobs in the set of jobs constructed above are receiver nodes.
- otherwise, N_i is the only receiver node.

M3 Minischedule:

- if $s \neq \varepsilon$, then for each job J in the set of jobs, a request (N_r,e,d,T) with the receiver node N_r, a deadline d, an eligibility time e and the task T (shared event) corresponding to J is generated. Note that both d and e are not related to the current job J and the node N_i, respectively, but to the receiver node N_r. Both values can be computed from the hierarchical node relationship. This concept is further explained in the next example.
- otherwise, the request (N_i,e,d,T) is generated, where e, d and T are eligibility time, deadline and task of the previous incoming request, respectively.

M4 Terminated tasks:

Let $\mathcal{T}$ be the set of tasks (shared events initiated by node N_i) in state q and let $\mathcal{T}'$ be the set of tasks in state $\nu_i(q,s)$ (s is derived as shown above). Then the set of terminated tasks in the message is set to $\mathcal{T}-\mathcal{T}'$ as these tasks are no longer active and valid.

Altogether, messages constructed by a node N_i contain information about the current jobs to be sent, the times when receiving nodes have to transmit their next messages and tasks that are valid at the moment.

d) **Communication Operation:** The nodes transmit the messages prepared as defined above. The transmission times of each node are determined by the priority queue that exists in each node. At system startup, the nodes are initialized as follows:

- the highest-level node N_k constructs the output message for its initial state $x_{0,k}$ and the output buffers for the remaining nodes are empty
- all nodes put $(N_k,0,1,-)$ in their priority queue

After initialization, in each time slot

- each node takes out the first eligible communication request from its priority queue. As the requests are sorted by deadline, Earliest Deadline First [12] scheduling is applied. If there are requests with the same deadlines, ties are resolved in the same unique way in all of the nodes.
- the node in this communication request transmits the message in its output buffer
- the receiver nodes put the incoming jobs in their input buffer and compute their according state update (evaluation of the transition function for the incoming jobs) and the message in the output buffer
- all nodes receive the minischedule. New communication requests are inserted in, and communication requests with terminated tasks are removed from the priority queue. Note that this update ensures that all nodes have the same priority queue by communicating only the minischedule.

Example 5 illustrates the communication operation.

Example 5 The node N_1 in Fig. 4 in state 2 of its communication model CM_{R_1} is investigated, assuming that: the current time is $t = 100msec$; the time slot is $t_s = 1msec$; the eligibility time is $1msec$; a message is sent from node N_5 with the *receiver nodes* N_1,N_2, the *job to be sent* $?\varphi_{R_5}$, the *minischedule* $(N_1,1,10,\varphi)(N_2,1,10,\varphi)$ and an empty set of *terminated tasks*. N_1 operates as follows:

- the communication requests $(N_1,101,110,\varphi)$ and $(N_2,101,110,\varphi)$ are added to the priority queue.
- computation for the input buffer with the job $?\varphi_{R_5}$: state update of CM_{R_1} to state 9.
- computation of the output buffer for $s = \varepsilon$: *receiver node*: N_1; set of *jobs to be sent*: $\{\}$, *minischedule*: $(N_1,1,10,\varphi)$; set of *terminated tasks*: empty.
- if the local event l occurs, then the new state of CM_{R_1} is 4. Output message for $s = !\varphi_{R_1}$: *receiver*

[3] It can be shown that such a string with maximal length exists in each state of CM_{R_i}.

TABLE I

NETWORK BEHAVIOR FOR VARYING t_s AND $t_e = 1$ MSEC

slot time (msec)	0.1	0.2	0.5	1.0
completed tasks/sec	20.12	20.1	18.86	16.23
message delay (msec)	1.08	1.14	1.81	3.35
network utilization (%)	31.17	59.20	93.81	98.62

TABLE II

NETWORK BEHAVIOR FOR VARYING t_e AND $t_s = 1$ MSEC

eligibility time (msec)	1	2	5
completed tasks/msec	20.12	18.06	14.74
message delay (msec)	1.08	2.13	5.29
network utilization (%)	31.17	15.12	5.75

node: N_5; set of *jobs to be sent*: $!\varphi_{R_1}$; *minischedule*: $(N_5, 1, 10, \varphi)$; set of *terminated tasks*: empty.

- suppose the first eligible communication request in the priority queue is $(N_1, 101, 110, \varphi)$ at time $t = 105$ (it is eligible as $101 < 105$). N_1 sends the answer to N_5 if all local tasks are completed. Otherwise it transmits the communication request to itself. □

e) **Correct Networked System Operation:** Correct system operation is achieved if all jobs that are ready to be sent by the nodes in $\mathcal{N}$ meet their deadlines. It can be shown that the communication operation as defined above guarantees that the jobs are sent and processed in the order specified by the communication model and that a communication request for each job to be sent is put into the priority queue of each node before its deadline.

Proposition 4.1 (Job order) Let $J_1 J_2 \cdots J_s$ be a job sequence according to the communication operation as defined above and assume that J_l has to be sent by node $N_{i_l} \in \mathcal{N}$ between time e_l and t_l, $l = 1, \ldots, s$. Then $J_1 J_2 \cdots J_s \in L(CM)$ and there exists a communication request for N_{i_l} between e_l and t_l in the priority queue. □

Correct operation of the distributed supervisors follows from Proposition 4.1 if the network is sufficiently fast.

Theorem 4.1 (Correct Operation) Let $\mathcal{N}$ be a set of nodes on a shared medium with the communication operation as defined above. There exists a lower bound on the network speed such that correct communication operation is guaranteed, i.e. all jobs are communicated in the correct order and all job deadlines are met. □

A procedure to determine the required network speed is addressed in future work.

V. SIMULATION STUDY

Theorem 4.1 states the conditions to guarantee the correct operation of the networked system. In this section we present a preliminary simulation study of the system in Fig. 2 to observe the performance of the distributed supervisors communicating on a network. We assume that the low-level events occur randomly within arbitrary but fixed time ranges in the order of 10-100 msec. Deadlines are chosen as in Example 4 and we assume that the eligibility time t_e is equal for all messages.

We simulate the distributed controller implementation according to Section IV. for different values of the time slot t_s and the eligibility time t_e. We also simulate the timed behavior of a monolithic (one PLC with cycle time 1 msec) implementation to provide a reference. The number of *completed tasks/sec* for the monolithic simulation is 20.4. We summarize the results in the following tables.

All messages meet their deadlines in the experiments presented in Table 1 and Table 2. There are messages that miss their deadlines for $t_s = 5msec$ and for $t_e = 8msec$ resulting in incorrect system behavior. In this preliminary simulation study, we observe that a slower network and a growing processing overhead slow down the networked implementation with respect to the monolithic implementation finally leading to incorrect operation.

VI. CONCLUSIONS AND FUTURE WORK

An efficient method for the computation of hierarchical and decentralized supervisors for discrete event systems has been elaborated in our previous work ([6]). In this paper, the *distributed* implementation of such supervisors on a *shared-medium network* has been investigated. Based on the deterministic hierarchical system structure, a *communication model* has been developed that ensures correct system behavior if the specified *deadlines* for communication messages are met. Additionally, a *communication architecture* and *scheduling policy* have been proposed such that the communication messages satisfy their deadlines if the network speed exceeds a certain lower bound. This result has been illustrated by simulations. Future work includes efficient criteria to determine bounds on the required network speed for correct operation and communication models for other distributed control architectures.

REFERENCES

[1] G. Barett and S. Lafortune, "Decentralized supervisory control with communicating controllers," *IEEE Transactions on Automatic Control*, vol. 45, pp. 1620–1638, 2000.

[2] M. de Queiroz and J. Cury, "Modular supervisory control of large scale discrete event systems," in *Workshop on Discrete Event Systems (WODES)*, 2000.

[3] J. van Schuppen, "Decentralized control with communication between controllers," *Unsolved Problems in Mathematical Systems and Control Theory, Princeton University Press*, 2004.

[4] R. Leduc, M. Lawford, and W. Wonham, "Hierarchical interface-based supervisory control-Part II: Parallel case," *IEEE Transactions on Automatic Control*, vol. 50, pp. 1336–1348, 2005.

[5] J. Komenda, J. van Schuppen, B. Gaudin, and H. Marchand, "Modular supervisory control with general indecomposable specification languages," in *Conference on Decision and Control*, 2005.

[6] K. Schmidt, *Hierarchical Control of Decentralized Discrete Event Systems Theory and Application*. Ph.D. Thesis, Technische Fakultät der Universität Erlangen-Nürnberg, 2005.

[7] A. Mannani, Y. Yang, and P. Gohari, "Decentralized embedded supervisory control of discrete-event systems," in *Workshop on Discrete Event Systems (WODES)*, 2006.

[8] R. Su and J. Thistle, "A distributed supervisor synthesis approach based on weak bisimulation," in *Workshop on Discrete Event Systems (WODES)*, 2006.

[9] L. Feng and W. Wonham, "Computationally efficient supervisor design: Abstraction and modularity," in *Workshop on Discrete Event Systems (WODES)*, 2006.

[10] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[11] (2002) IEee 1588tm-2002 standard for a precision clock synchronization protocol for networked measurement and control systems. [Online]. Available: http://ieee1588.nist.gov

[12] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE J. Select. Areas Commun.*, vol. 8, pp. 368–379, 1990.