Reliable and Safe Operation of Distributed Discrete-Event Controllers: A Networked Implementation with Real-time Guarantees

Klaus Schmidt * **Ece G. Schmidt** ** **Jorgos Zaddach** ***

* *Lehrstuhl für Regelungstechnik, Universität Erlangen-Nürnberg, Germany (e-mail: klaus.schmidt@rt.eei.uni-erlangen.de).*
** *Department of Electrical and Electronics Engineering, Middle East Technical University Ankara, Turkey (e-mail: eguran@metu.edu.tr)*
*** *Siemens AG, Industrial Solutions and Services, Germany (e-mail: jorgos-johannes.zaddach@siemens.com)*

Abstract: Efficient controller synthesis approaches for discrete-event systems mostly provide a set of interacting distributed controllers that are potentially implemented in networked controller devices. Although the fulfillment of specified requirements and the absence of deadlocks is guaranteed by such methods on a logical level, timing issues due to controller communication are not incorporated. Recently, a formal communication model including real-time requirements for the reliable and safe operation of distributed discrete-event controllers has been proposed by the authors. In this paper, the real-time communication operation of such distributed controllers is discussed, and a sufficient condition for the network bandwidth in order to meet the specified real-time requirements is derived. A simulation study of a manufacturing system model with 50 distributed controllers supplements the theoretical result.

1. INTRODUCTION

The efficient controller synthesis for discrete event systems (DES) has been an area of intensive study in recent years. Approaches such as Barett and Lafortune (2000); de Queiroz and Cury (2000); Leduc et al. (2005); Komenda et al. (2005); Schmidt et al. (2007a); Hill and Tilbury (2006); Su and Thistle (2006); Feng and Wonham (2006) result in interacting *modular* and *decentralized* controllers, where controllers interact via *shared events* that have to be synchronized. However, since the above approaches focus on controller synthesis, the realization of this interaction remains an open question.

As long as the controllers are implemented on a single device (PC, PLC, etc.), the interaction can take place internally, e.g., via shared memory. In contrast, if each controller is placed in a different physical location, communication is required. This issue is addressed in Schmidt et al. (2007b), where we propose a *communication model* and a *communication operation* on a shared-medium network for the control approach in Schmidt et al. (2007a). In this context, *communication messages* have to be sent before a certain specified *deadline*.

Reliability (continuity of correct service) and *safety* (avoidance of catastrophic consequences) are components of *dependable* system operation as in Avizienis et al. (2004). In this paper, the results in Schmidt et al. (2007b) are extended by deriving a lower bound for the *network bandwidth* that is required for the reliable and safe operation of the distributed controllers. Additionally, a large-scale manufacturing system model with 50 distributed controllers is simulated in order to validate the formal results and to investigate the average performance.

The paper outline is as follows. In Section 2, we briefly discuss our communication model. Reliable and safe communication operation are investigated in Section 3. Section 4 provides a simulation study, and we give conclusions in Section 5.

2. COMMUNICATION MODEL FOR DISTRIBUTED DISCRETE EVENT CONTROLLERS

2.1 Distributed Discrete Event Controllers

In this paper we employ the hierarchical and decentralized control approach in Schmidt et al. (2007a) for a distributed controller implementation. The approach is based on the assumption that a large-scale DES is composed of several interacting system components, and results in a set $\mathcal{R} = \{R_1, \ldots, R_k\}$ of k DES controllers for the different components in a hierarchical relationship as indicated in Fig. 1 (a). Each controller is represented by a finite automaton $R_i = (X_i, \Sigma_i, \delta_i, x_{0,i}, X_{\mathrm{m},i})$ with a finite set of *states* X_i, a finite alphabet of *events* Σ_i, a partial *transition function* $\delta_i : X_i \times \Sigma_i \to X_i$, an *initial state* $x_{0,i} \in X_i$, and a set of *marked states* $X_{\mathrm{m},i} \subseteq X_i$ following the notation in Cassandras and Lafortune (1999). We also introduce $\Gamma_i(x) := \{\sigma \in \Sigma_i | \delta_i(x,\sigma) \text{ exists}\}$ as the set of *feasible events* in each state $x \in X_i$. Interaction among the different controllers is modeled by *shared events* that have to occur synchronously in all controllers that share the event. Formally, this interaction is given by the *synchronous composition* of the controllers. Let $R_i, R_j \in \mathcal{R}$ be finite automata. Then, the synchronous composition $R_i || R_j$ of R_i and R_j is defined as the finite automaton $R_{i||j} := (X_{i||j}, \Sigma_{i||j}, \delta_{i||j}, x_{0,i||j}, X_{\mathrm{m},i||j})$ with $X_{i||j} = X_i \times X_j$, $\Sigma_{i||j} = \Sigma_i \cup \Sigma_j$, $x_{0,i||j} = x_{0,i} \times x_{0,j}$, $X_{\mathrm{m},i||j} = X_{\mathrm{m},i} \times X_{\mathrm{m},j}$. For a state $(x_i, x_j) \in X_{i||j}$ and an event $\sigma \in \Sigma_{i||j}$, the transition function is

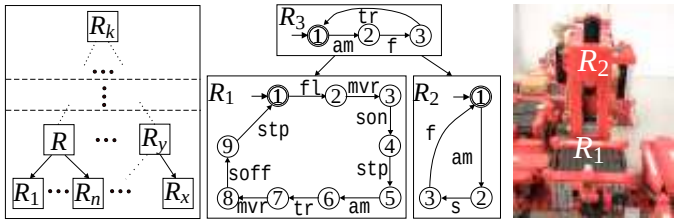
$$\delta_{i||j}((x_i,x_j),\sigma) := \begin{cases} (\delta_i(x_i,\sigma), \delta_j(x_j,\sigma)) & \text{if } \sigma \in \Gamma_i(x_i) \cap \Gamma_j(x_j) \\ (\delta_i(x_i,\sigma), x_j) & \text{if } \sigma \in \Gamma_i(x_i) - \Sigma_j \\ (x_i, \delta_j(x_j,\sigma)) & \text{if } \sigma \in \Gamma_j(x_j) - \Sigma_i \\ \text{undefined} & \text{otherwise} \end{cases}$$

Accordingly, the overall system representation of the hierarchical and decentralized controllers evaluates to a finite automaton $R := ||_{i=1}^{k} R_i$, and the controller synthesis procedure in Schmidt

et al. (2007b) guarantees that R is nonblocking, i.e., from each of its states there is a sequence of transitions to a marked state. However, note that the state space of R need not be enumerated explicitly, but is implicitly given by the decentralized representation of the controllers and the rule of interaction via the synchronous composition, which avoids the *state space explosion problem* encountered by monolithic implementations.

Example 1 illustrates the controller interaction.

Example 1. Fig. 1 (b) shows a simple hierarchical architecture with two levels and $k = 3$ automata. It describes the operation of a manufacturing unit with a conveyor belt (R_1) and a machine (R_2, see Fig. 1 (c)) that is controlled by a high-level controller R_3. The conveyor belt notices if a product has to be transported (fl/tr – product from left/to right) and moves accordingly (mvr – move to right). It stops (stp) when a sensor signals the product arrival at the machine (son), which is indicated by the shared event am (product at machine). After am, the machine R_2 starts processing (s) and finishes processing (f) after some time. The high-level controller R_3 ensures that the *shared events* am, f and tr occur such that the product is not transported to the right before the machine finished processing.



(a) (b) (c)

Fig. 1. (a) Hierarchical and decentralized architecture (b) simple example hierarchy (c) machine and conveyor belt.

2.2 Logical Communication Model

The decentralized controller representation introduced in Section 2.1 is profitable especially if the respective controller devices (e.g., PLCs) are placed in distinct physical locations and connected by a network, e.g., on a factory floor. Nevertheless, in this case, the occurrence of *shared events* $\Sigma_\cap$ with $\Sigma_i \cap \Sigma_j \subseteq \Sigma_\cap$ for all $i, j = 1,\ldots,k$, $i \neq j$, has to be communicated and synchronized. Consequently, each controller that shares an event $\sigma \in \Sigma_\cap$ must know when σ is possible in all of the other controllers that share σ. Using the hierarchical system structure, a communication model automaton (CMA) C_{R_i} for each controller $R_i \in \mathcal{R}$ has been constructed algorithmically in Schmidt et al. (2007b). The communication is modeled by identifying shared events with system *tasks* that have to be completed by communicating *jobs* among the distributed controllers. Due to the hierarchical system structure, high-level controllers know about shared event occurrences in their lower-level controllers. This is reflected in the sequential order of job transmissions of the proposed *communication model* by initiating communication in the highest level and propagating it to the lower-level controllers along the hierarchy. The main features of this communication model are briefly outlined in Example 2.

Example 2. Fig. 2 depicts the CMA C_{R_i}, $i = 1,2,3$ for the respective controllers in Fig. 1 (b). Every state in C_{R_i} corresponds to a state in the controller R_i, and the state labels and markings are chosen accordingly (e.g., 1_1, 1_2, 1_3, 1_4 in C_{R_3} correspond to 1 in R_3). Assuming that each controller in Fig. 1 is in its initial state, the communication is as follows.

- R_3 can execute the task (shared event) am and needs to know when this event is possible in the low-level controllers that share am. Thus, R_3 issues a *question job* $?\mathtt{am}_{R_3}$ for am to R_1 and R_2. This is realized by the transition $?\mathtt{am}_{R_3}$ in state 1_1 of C_{R_3}. C_{R1} and C_{R2} receive $?\mathtt{am}_{R_3}$ (transition from 1_1 to 1_2 with $?\mathtt{am}_{R_3}$).
- In the initial state of R_2, am is feasible. Thus, R_2 can directly send the *answer job* $!\mathtt{am}_{R_2}$ and C_{R_2} changes from 1_2 to 1_3.
- in R_1, am only becomes feasible after the string of non-shared events fl mvr son stp occurred, and the answer job $!\mathtt{am}_{R_1}$ is given in the corresponding state 5_2 in C_{R_1}.
- If all answers ($!\mathtt{am}_{R_1}$ and $!\mathtt{am}_{R_2}$) have been received by R_3, it can send the *command job* $\mathtt{am}_c$ in the corresponding state 1_4 of C_{R_3} to make all controllers execute the shared event am.[1] All low-level controllers can process the command in their communication model: C_{R_1} changes from 5_3 to 6_1, and C_{R_2} changes from 1_3 to 2_1.
- The question-answer-command procedure repeats with R_3 initiating communication for f.

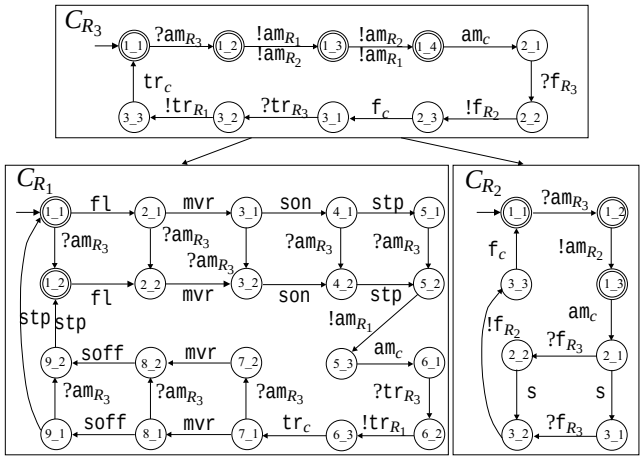


Fig. 2. Communication model for the manufacturing unit.

Formally, the outcome of the communication model construction is a tree structure $T_C = (\mathcal{C}, C_k, c_\mathcal{C}, p_\mathcal{C})$ (see e.g., Hopcroft and Ullman (1975)) that captures the hierarchical relationship of the distributed controllers. In this paper, the *set of vertices* $\mathcal{C}$ denotes the set of CMAs $C_{R_i} = (Q_i, \mathcal{J}_i, \nu_i, q_{0,i}, Q_{\mathrm{m},i})$ for the controllers R_i, $i = 1,\ldots,k$ with the *set of jobs* $\mathcal{J}_i = \mathcal{J}_{\mathrm{out},i} \dot\cup \mathcal{J}_{\mathrm{in},i}$ that are communicated from ($\mathcal{J}_{\mathrm{out},i}$) and to ($\mathcal{J}_{\mathrm{in},i}$) R_i as described in Example 2. Furthermore, C_k is the *root vertex* and $c_\mathcal{C} : \mathcal{C} \to 2^\mathcal{C}$ and $p_\mathcal{C} : \mathcal{C} \to \mathcal{C}$ are the *children map* and the *parent map* such that $c_\mathcal{C}(C_i)$ is the *set of children* and $p_\mathcal{C}(C_i)$ is the *parent* of $C_i \in \mathcal{C}$, respectively. Every vertex without children is called a *leaf*. We also distinguish the set of jobs $\mathcal{J}_\sigma$ that are sent for each $\sigma \in \Sigma_\cap$, and call σ_c the command job for σ.

Observing that again interaction between the CMAs via the exchange of jobs is modeled by jobs shared between CMAs, the overall communication model $C = (Q, \mathcal{J}, \nu, q_0, Q_\mathrm{m})$ is obtained as the synchronous composition of the CMAs: $C := ||_{i=1}^k C_k$. In particular, each state of the overall communication model is composed of the state values of its distributed components. The following properties can be deduced from the communication model construction in Schmidt et al. (2007b).

[1] The communication operation in Section 3.1 ensures the synchronous arrival of $\mathtt{am}_c$ at all controllers.

Properties: Let $q = (q_1, \ldots, q_k) \in Q$ and $J \in \mathcal{J}_\sigma - \{\sigma_c\}$ for $\sigma \in \Sigma_\cap$ s.t. $\nu_i(q_i, J)$ exists for some $1 \leq i \leq k$. Then

(1) for all j s.t. $J \in \mathcal{J}_j$ it follows that $\nu_j(q_j, J)$ exists
(2) for all j s.t. $J \in (\mathcal{J}_j - \mathcal{J}_{\text{out},j})$ it holds that there is a $J' \in \mathcal{J}_{\text{out},j} \cap \mathcal{J}_\sigma$ s.t. $\nu_j(q_j, JJ')$ exists.

Property (1) states that whenever a job is communicated, all CMA that contain the job either send or receive the job, while property (2) makes clear that every CMA that received a job for an event σ can send a follow-up job for this event. Additionally, it holds that C is nonblocking and exhibits the same behavior as the original controllers.

2.3 Requirements and Issues for Reliable and Safe Operation

The communication model introduced above describes the logical behavior of the communication, i.e., the sequential order of job transmissions. However, the fact that the communication model is designed for distributed systems on a network, where possible communication delays affect the system operation, also has to be addressed. Specifically, issues such as *system reliability and safety* (the occurrence of a shared event has to be detected fast in order to prevent an undesired situation) and *system performance* (the occurrence of a shared event has to be detected fast such that the communication does not slow down the system operation) have to be accounted for.

Considering the controller representation, a shared event $\sigma \in \Sigma_\cap$ *theoretically* occurs if each controller R_i that shares σ is in a state $x_i \in X_i$ where $\delta_i(x_i, \sigma)$ exists. According to the distributed implementation with the communication model, σ *physically* happens when the command job σ_c is transmitted. Depending on the physical interpretation of σ, it has to be ensured that the time between its theoretical and its physical occurrence remains below an appropriate bound in order to fulfill safety and performance requirements.

In our work, we incorporate such real-time requirements in the communication model be introducing a map $r : \Sigma_\cap \to \mathbb{R} \cup \{\infty\}$ for the shared events, where $r(\sigma)$ represents the maximal allowable time between the theoretical and physical occurrence of an event $\sigma \in \Sigma_\cap$ (e.g., the reaction time to a sensor event). The execution of an event $\sigma \in \Sigma_\cap$ in the worst case requires the communication of all jobs related to σ, while the actual event can happen any time between the transmission of the first and the last job for σ. Denoting N_σ the number of jobs for a task σ, a *deadline* $d_J := \frac{r(\sigma)}{N^\sigma}$ is associated with each job $J \in J^\sigma$. In this framework, d_J indicates that if J is ready to be transmitted by its corresponding controller at time t_0, then it has to be sent at $t_0 + d_J$ latest. A communication model with a map $r : \Sigma_\cap \to \mathbb{R} \cup \{\infty\}$ as defined above is denoted a *communication model with deadlines*.

To sum up; the case where controllers synthesized according to Schmidt et al. (2007a) are implemented in a distributed manner and communicate via a network has been considered. The *communication model* with deadlines for each controller defines rules for job communication such that the behavior of the communicating controllers and the original controllers is equivalent. It is constructed such that jobs that are transmitted by R_i, are received by all controllers that contain the respective job. In doing so, it has to be ensured that whenever a controller needs to transmit a job, it has access to the network before the job deadline. This issue is addressed in the next section.

3. NETWORKED IMPLEMENTATION

3.1 Shared-Medium Operation

According to Schmidt et al. (2007b), the CMAs in Section 2.3 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 shared-medium network as in Fig. 3 (a).

Shared-medium networks have a simple and low-cost architecture. However, *collisions* occur if more than one node send messages at the same time. We provide a *collision avoidance* policy for messages to be sent on the network. In the first step, we propose *time-slotted* operation with fixed size time slots t_s such that the time instants for message transmissions are synchronized among all nodes (see Fig. 3 (b)). Note that such synchronization with an accuracy up to 100ns is for example provided by the IEEE 1588 standard for Ethernet in IEEE (2002) which is already implemented in the Intel IXP465 network processor and integrated in PLCs. Secondly, we exploit the deterministic structure of the controller automata and the hierarchical relationship between controllers as follows. Each node that sends a job knows which nodes will have to send a job next, and attaches this information to the job in the form of a *communication request* (CR). All of the nodes process this CR and deterministically compute which node will transmit next. To this end, the *time-slotted* operation together with the described scheduling policy ensure that in each time instant, each node uniquely knows the next node to send a message. Furthermore, due to the inherent broadcast on the shared medium, all of the nodes can receive all messages synchronously.

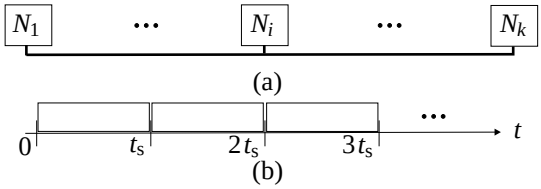


Fig. 3. (a) shared-medium network; (b) time-slotted operation.

3.2 Network Node

A network node $N_i \in \mathcal{N}$ implements the following entities.

N1 a CMA C_{R_i},
N2 an *output buffer* that stores *messages* to be sent,
N3 an *input buffer* that stores received messages,
N4 a set of *active tasks* (shared event communications) currently initiated by the node,
N5 a *priority queue* (PQ) that stores *communication requests* as a tuple (N, e, d, T), where N is a node to transmit, $e \in \mathbb{R}$ is an *eligibility time*, $d \in \mathbb{R} \cup \{\infty\}$ is a deadline and T is the active task that issued the request. The PQ is ordered such that the CR with the smallest deadline is granted first.

In this setting, a CR (N, e, d, T) states that the node N has to access the shared medium before the deadline d. The fact that each node needs a certain amount of time to react to incoming messages is captured by the *eligibility time* e. It determines the earliest time instant when a node is ready to transmit a message. Hence, the eligibility time and the deadline define the time interval, where the message has to be sent, and can be derived from the process parameters (e.g., the cycle time of a PLC) and the communication model with deadlines, respectively.

3.3 Message

According to Section 2.2, communication between nodes requires the exchange of jobs. In our approach, jobs are sent via *messages* that are constructed offline for each node N_i and each state $q \in Q_i$ of its associated CMA C_{R_i}.

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

- M1 A set of *jobs to be sent* by N_i. To this end, the longest sequence of outgoing jobs $s = J_1 J_2 \cdots J_m \in \mathcal{J}_{\text{out},i}$ is computed s.t. $q' := \nu_i(q,s)$ exists.[2] The set of jobs of the message contains all jobs in $J_1 J_2 \cdots J_m$.
- M2 A set of *receiver nodes*. If s is not empty, then all nodes that share jobs in the set of jobs constructed above are receiver nodes. Otherwise, there is no receiver node.
- M3 A *minischedule* with CRs. If s is not empty, then for each job, a request (N_r, e, d, σ) with the receiver node N_r, an eligibility time e, a deadline d and the task σ of the job is generated. Otherwise, a *self request* (N_i, e, d, σ) is generated, where e is the next time when N_i can send a message, and d is the deadline of the valid task σ in q.[3]
- M4 A set of tasks that have been terminated in N_i. If in a set of competing tasks, one task finishes first, the requests for the other tasks become invalid, and have to be erased from the PQ. Let $\mathcal{T}$ be the set of tasks 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 in M1). Then the set of *terminated tasks* is set to $\mathcal{T} - \mathcal{T}'$ as these tasks are no longer active and valid.

Altogether, messages constructed by a node N_i in its state $q \in Q_i$ 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. Note that the collision avoidance policy demands that at most one message is sent per time slot. Hence, t_s has to accommodate the longest message frame with a frame length $F_{\max}$ which can be computed during the offline message construction process of the individual nodes.

3.4 Communication Operation

The nodes transmit the messages prepared as defined above, where the transmission times are determined by the respective PQ. At system startup, the nodes are initialized as follows:

- O1 Only the highest-level node N_k constructs the output message for its initial state $q_{0,k}$.
- O2 All nodes put the CR $(N_k, 0, 1, -)$ in their PQ.

After initialization, in each time slot

- O3 Each node takes out the first eligible CR from its PQ.
- O4 The node in this CR sends the message in its output buffer.
- O5 All nodes insert the CRs in the minischedule in their PQ, while adding the current time to both eligibility time and deadline. CRs with terminated tasks are removed from the PQ s.t. all nodes have the same PQ by exchanging CRs.
- O6 The receiver nodes put the incoming jobs in their input buffer and compute their according state update (evaluation of the transition function for incoming jobs) and the message in the output buffer (according to Section 3.3).

Example 3 illustrates the communication operation.

Example 3. Assume that at time $t = 0\,\text{ms}$, all nodes are in the initial states of their respective communication model in Fig. 2; the time slot is $t_s = 1\,\text{ms}$; the eligibility times of N_1, N_2 and N_3 are 1 ms, 0.5 ms and 1 ms, respectively, and $d_{\texttt{am}} = 50\,\text{ms}$. Then the high-level node N_3 has a message in its output buffer with the *receiver nodes* N_1, N_2, the *job to be sent* $?\texttt{am}_{R_3}$, and the *minischedule* $(N_1, 1\,\text{ms}, 50\,\text{ms}, \texttt{am})(N_2, 0.5\,\text{ms}, 50\,\text{ms}, \texttt{am})$. Note that 1 ms and 0.5 ms are the eligibility times of N_1 and N_2, respectively. Initially, each PQ contains the CR $(N_3, 0\,\text{ms}, 1\,\text{ms}, -)$ (O1). At $t = 1\,\text{ms}$, N_3 sends the content of its output buffer (O3,O4). The operation of node N_1 is as follows:

(1) PQ: the CRs $(N_1, 2\,ms, 51\,ms, \texttt{am})$ and $(N_2, 1.5\,ms, 51\,ms, \texttt{am})$ are added (O5).
(2) input buffer computation: state update of C_{R_1} to state 1_2 with received job $?\texttt{am}_{R_3}$ (O6).
(3) output buffer computation for s empty (M1-M4): *receiver nodes*: {}; set of *jobs to be sent*: {}, *minischedule*: $(N_1, 1\,ms, 50\,ms, \texttt{am})$; set of *terminated tasks*: {}. By sending this message, N_1 gives itself the opportunity to transmit again until the answer job $!\texttt{am}_{R_1}$ can be sent.
(4) if the local string `flmvr son stp` occurs, then the new state of C_{R_1} is 4_2. Output message for $s = !\texttt{am}_{R_1}$: *receiver node*: N_3; set of *jobs to be sent*: $!\texttt{am}_{R_1}$; *minischedule*: $(N_3, 1\,ms, 50\,ms, \texttt{am})$; set of *terminated tasks*: {}.
(5) suppose the first eligible CR in the PQ is $(N_1, 2\,ms, 51\,ms, \texttt{am})$ at time $t = 4\,\text{ms}$ (it is eligible as $2\,\text{ms} < 4\,\text{ms}$). N_1 sends the answer in (4) to N_3 if it is in state 4_2. Otherwise it transmits the CR in (3) to itself.

3.5 Reliability and Safety Guarantees

Reliable and safe system operation is achieved if all jobs that are ready to be sent by the nodes in $\mathcal{N}$ meet their deadlines and are transmitted in the order specified by the communication model. In this section, we first recall a result from Schmidt et al. (2007b). It states that the communication operation in Section 3.4 guarantees that all jobs are sent in the order specified by the communication model and that a CR for each corresponding message is put into the PQ of each node before its deadline.

Proposition 3.1. (Job order). Let $J_1 J_2 \cdots J_s$ be a job sequence according to the defined communication operation with a set of nodes $\mathcal{N}$, and assume that J_l has to be sent by $N_{i_l} \in \mathcal{N}$ between time e_l and t_l, $l = 1, \ldots, s$. Then $J_1 J_2 \cdots J_s$ is a job sequence in C and there exists a CR for N_{i_l} between e_l and t_l in the PQ.

Additionally, in order to guarantee reliable and safe system operation, the *network bandwidth* B has to be high enough to send the message associated to each CR in the PQ before its deadline. By intuition, the required B increases with the *maximum priority queue length* $Q_{\max}$, the *frame length* of the maximum size message $F_{\max}$, the reaction time of the slowest controller (maximum eligibility time) $e_{\max}$, and the *minimum job deadline* $d_{\min}$. We first establish a result for $Q_{\max}$, and then provide a sufficient condition for B to guarantee reliable and safe system operation based on the above parameters.

Proposition 3.2. (Maximum Queue Length). Let $\mathcal{N}$ be a set of nodes with the communication model tree structure T_C and the communication operation as defined above. Then, the maximum number $Q_{\max}$ of communication requests in the PQ is finite and can be computed algorithmically.

Lemma 3.1 supports the proof of Proposition 3.2.

Lemma 3.1. (Requests per State and Event). Given the prerequisites in Proposition 3.2, assume that $q = (q_1, \ldots, q_k) \in Q$ and $\sigma \in \Sigma_\cap$. Let C_j be the highest-level node such that $\mathcal{J}_j \cap \mathcal{J}_\sigma \neq \emptyset$ and define the subtree T_q^σ of T_C as follows:

[2] It can be shown that such a sequence exists in each state of C_{R_i}.
[3] Such task exists because the communication model is nonblocking.

- T_q^σ is empty if $\nu_j(q_j, J)$ does not exist for any $J \in \mathcal{J}_j \cap \mathcal{J}_\sigma$.
- otherwise, $T_q^\sigma = (\mathcal{C}^\sigma, C_j, c_C^\sigma, p_C^\sigma)$, where each $C_i \in \mathcal{C}^\sigma$ has the property that $c_C(C_i) = \{C_l \in \mathcal{C}^\sigma | \nu_l(q_l, J') \text{ exists for some } J' \in \mathcal{J}_l \cap \mathcal{J}_\sigma\}$, i.e., $\mathcal{C}^\sigma$ contains all nodes in $\mathcal{C}$ such that each node that lies on a branch is in a state where some $J' \in \mathcal{J}_i \cap \mathcal{J}_\sigma$ is possible.

Then, the maximum number CR_q^σ of CRs in the PQ associated with σ in state q equals the number of leaves of T_q^σ and is finite.

Proof (Sketch) If T_q^σ is not empty, then C_j can send a message with a job $J \in \mathcal{J}_j \cap \mathcal{J}_\sigma$. By Property (1) in Section 2.2, $\nu_i(q_i, J)$ exists for all $C_i \in c_C^\sigma(C_j)$, i.e., C_j sends $|c_C^\sigma(C_j)|$ CRs. Each of these C_i has a follow-up job $J_i \in \mathcal{J}_i \cap \mathcal{J}_\sigma$ such that $\nu_i(q_i, J_i)$ exists because of Property (2). Hence, sending jobs with associated CRs to the children nodes in T_q^σ can be repeated until the leafs of T_q^σ are reached. The maximum number CR_q^σ of CRs for σ occurs if each leaf has a CR. □

Now, Proposition 3.2 can be proved.

Proof Applying Lemma 3.1, the maximum number CR_q of requests in the PQ for a certain state q evaluates to $CR_q = \sum_{\sigma \in \Sigma_\cap} CR_q^\sigma$. Then, taking the maximum over all states $q \in Q$ gives the desired result $Q_{\max} = \max_{q \in Q} CR_q$. □

The computation of $Q_{\max}$ suggests the enumeration of the overall state space of C that was deemed computationally infeasible for the controller synthesis. However, the above result only shows the existence of $Q_{\max}$. Practically, the hierarchical system structure can be exploited to efficiently compute $Q_{\max}$.

We now deduce an upper bound $t_{s,\max}$ for the time slot such that each CR leaves the PQ before its respective deadline, and then conclude reliable and safe system operation for network bandwidth higher than $B_{\min} = \frac{F_{\max}}{t_{s,\max}}$.

Lemma 3.2. (Meeting Deadlines). Let $rq = (N, e, d, T)$ be a communication request that enters the PQ at time t_0. Then, rq can be scheduled before its absolute deadline $t_0 + d$ if

$$t_s \leq t_{s,\max} := \frac{d_{\min} - e_{\max}}{Q_{\max} + 1}.$$

Proof First assume that rq has the minimum deadline $d = d_{\min}$. For notational purposes, the entries in the PQ are numbered from 1 to $Q_{\max}$, the set of all CRs $\mathcal{RQ}$ is defined, and the map $q : \mathbb{R} \times \mathcal{RQ} \to \{0, \ldots, Q_{\max}\}$ is introduced, where $q(t, rq)$ denotes the entry of the CR rq in the PQ at time t and $q(rq, t) = 0$ if rq is not in the queue at t. It has to be shown that rq leaves the PQ before $t_0 + d_{\min}$, i.e., $q(t_0 + d_{\min}) = 0$.

Because of Proposition 3.2, it holds that $q(t_0, rq) \leq Q_{\max}$. Observing that no CR can enter the PQ in front of rq for $t > t_0$, and that all CRs in the PQ become eligible after $e_{\max}$ latest, the position of rq in the PQ at times $t > t_0$ evaluates to

$$\begin{aligned} q(t, rq) &\leq q(t_0, rq) + \lceil \frac{e_{\max}}{t_s} \rceil - \lfloor \frac{t - t_0}{t_s} \rfloor \leq \\ &\leq Q_{\max} + 1 - \lceil \frac{-e_{\max} + (t - t_0)}{t_s} \rceil \leq \\ &\leq Q_{\max} + 1 - \lceil \frac{t - t_0 - e_{\max}}{(d_{\min} - e_{\max})} (Q_{\max} + 1) \rceil. \end{aligned}$$

That is, $q(t_0 + d_{\min}, rq) \leq Q_{\max} + 1 - \lfloor \frac{d_{\min} - e_{\max}}{d_{\min} - e_{\max}} (Q_{\max} + 1) \rceil = 0$.

Let $d > d_{\min}$. Then there is $t' > t_0$ s.t. $t_0 + d = t' + d_{\min}$ and $q(t', rq) \leq Q_{\max}$. With rq as a CR with deadline $d_{\min}$ that arrives at t', the same argument shows that $q(t' + d_{\min}, rq) \leq 0$. □

Theorem 3.1. (Bound on Network Bandwidth). Let $\mathcal{N}$ be a set of nodes with the communication model tree structure T_C and the communication operation as defined above. Then a network bandwidth $B \geq B_{\min} := \frac{F_{\max}}{t_{s,\max}}$ is sufficient for reliable and safe operation of the distributed controllers.

Proof Because of Proposition 3.1 and Lemma 3.2, each message to be sent has a CR in the PQ before its deadline, and this CR is served before its deadline. $C \geq F_{\max}/t_{s,\max}$ ensures that the message can be sent until the next transmission starts. Conversely, if $C < F_{\max}/t_{s,\max}$ this requirement is violated. □

Theorem 3.1 implies that a lower bound on the network bandwidth for reliable and safe operation can be computed offline using $d_{\min}$, $Q_{\max}$, $e_{\max}$ and $F_{\max}$ for a given distributed system.

4. SIMULATION

4.1 Laboratory Setup

In Section 2 and 3, the communication model and the operation are formally described, and statements for the worst case network usage are employed to derive real-time guarantees. In this section, an extensive simulation study of the large-scale manufacturing system model in Fig. 4 with 50 distributed controllers on 5 hierarchical levels is carried out. The distributed controller design for this system which comprises manufacturing components such as the machine and the conveyor belt in Example 1 has been elaborated in Schmidt et al. (2007a). The communication models with deadlines and the corresponding message sets for each node are constructed algorithmically. Considering the measured system characteristics, jobs with a *minimum deadline* of $d_{\min} = 20$ ms ensure reliable and safe system operation. Additionally, this study assumes that all nodes implement PLCs with a cycle time $e_{\max}/2$ such that the *eligibility time* $e_{\max}$ is chosen. Furthermore, the simulator implements all network components as described in Section 3.2 - 3.4. In order to achieve a realistic simulation, the timed behavior of all manufacturing components that interact with the distributed controllers has been modeled in the form of timed automata, where the timing characteristics of transitions are in the order of 1 s. The entire simulator that incorporates the component models as well as the communication operation and network model is developed in C++ based on the `libfaudes` software library for DES in `libfaudes` (2007). All of the results in the following sections are obtained after simulating the manufacturing system for 10 minutes of operation.

The goal of the study in this paper is the validation of the *theoretical results* in Section 3.5. In addition to that, we conduct an investigation of the *average performance* of our real-time communication operation.

Fig. 4. Manufacturing system example.

4.2 Experiments and Results

According to the result in Proposition 3.2, the maximum length of the PQ in each node could be determined as $Q_{\max} = 32$. Noting that the longest message frame is $F_{\max} = 708$ bits, this results in a required network bandwidth of up to $B_{\min} = 1800$ Mbit/s. For $e_{\max}$ between 0.2 ms and 7 ms, a maximum time slot of $t_{s,\max} = \frac{(20\,\text{ms} - e_{\max})}{32+1}$ between 0.6 ms and 0.4 ms is required according to Theorem 3.1.

In the following experiments, we first investigate how the variation of $t_s/t_{s,\max}$ and $e_{\max}$ affects the number of missed deadlines (NMD) which is a metric to indicate reliability and safety of the system operation. The deadline misses in Figure 5 (a) could only be observed for combinations of large $e_{\max}$ (≥ 3 ms) and/or very large $t_s/t_{s,\max}$ (≥ 6), which clearly violates Theorem 3.1. The maximum observed queue size is 15 and thus significantly smaller than the theoretical value $Q_{\max} = 32$.

Furthermore we study the *average used bandwidth* (AUB in Mbit/s) and the *number of completed tasks* (NCT) as metrics for the average system performance. In particular, we want to find out how to spare network resources (bandwidth) without slowing down the communicating controllers.

As can be seen in Fig. 5 (b), it is favorable to choose a large value of $e_{\max}$ while keeping a large value of $t_s/t_{s,\max}$ to achieve a small AUB. This is expected as on the one hand messages cannot be sent frequently (large $e_{\max}$) and on the other hand, messages are only sent as frequently as necessary. Furthermore, the variation of NCT for different values of $e_{\max}$ and $t_s/t_{s,\max}$ is below 2% (see Fig. 5 (c)). This is the case as the occurrence of tasks (shared events) rather depends on the timing characteristics of the system evolution which are in the order of seconds. Note that the slight decrease of NCT with larger $e_{\max}$ is due to the increased average CR delay (ARD) (see Fig. 5 (d)).

Together, it has been observed from the simulation that there is a trade-off between AUB and NTC. The operating point $t_s = 0.54$ ms (this corresponds to $B = 1.3$ Mbit/s) and $e_{\max} = 2.1$ ms is a good choice, as it yields a good system performance (1010 tasks) and at the same time results in a small value for NTS. The PLC cycle time of $e_{\max}/2 \approx 1$ ms is standard in current PLCs.

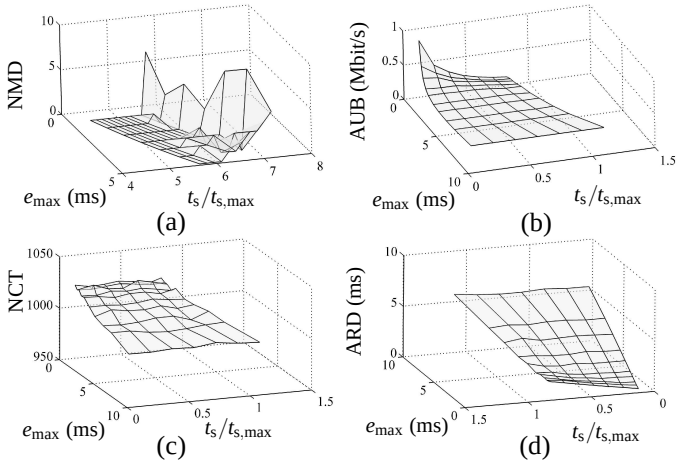


Fig. 5. Simulation results: (a) missed deadlines (b) bandwidth usage (c) completed tasks (d) request delay.

5. CONCLUSION

In this paper, the *distributed* implementation of hierarchical and decentralized DES controllers on a *shared-medium network* has been investigated. Based on the deterministic hierarchical system structure, a *communication model* has been developed, and a communication operation has been proposed such that communication messages are transmitted according to the communication model. Using this operation, it has been formally proved that a lower bound for the network bandwidth that guarantees reliable and safe system operation can be computed depending on the dynamic system properties and the real-time requirements in form of message deadlines. A simulation study of a large-scale distributed DES with 50 controllers has been performed to validate the formal results and to characterize the average behavior of our communication architecture. Future work aims at the incorporation of timing information of the discrete event system models in the communication model, and the hardware implementation of the proposed approach.

REFERENCES

A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

G. Barett and S. Lafortune. Decentralized supervisory control with communicating controllers. *IEEE Transactions on Automatic Control*, 45:1620–1638, 2000.

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

M.H. de Queiroz and J.E.R. Cury. Modular supervisory control of large scale discrete event systems. In *Workshop on Discrete Event Systems*, 2000.

L. Feng and W.M. Wonham. Computationally efficient supervisor design: Abstraction and modularity. In *Workshop on Discrete Event Systems*, 2006.

R. Hill and D. Tilbury. Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction. *Workshop on Discrete Event Systems*, 2006.

J.E. Hopcroft and J.D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, 1975.

IEEE. 1588tm-2002 standard for a precision clock synchronization protocol for networked measurement and control systems, 2002. URL `http://ieee1588.nist.gov`.

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.

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

K. Schmidt, Th. Moor, and S. Perk. Nonblocking hierarchical control of decentralized discrete event systems. *accepted in IEEE Transactions on Automatic Control*, 2007a.

K. Schmidt, E.G. Schmidt, and J. Zaddach. A shared-medium communication architecture for distributed discrete event systems. In *Mediterranean Conference on Control and Automation*, 2007b.

R. Su and J. Thistle. A distributed supervisor synthesis approach based on weak bisimulation. In *Workshop on Discrete Event Systems*, 2006.

`libfaudes`. software library, 2007. URL `http://www.rt.eei.uni-erlangen.de/FGdes/faudes/index.php`.