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 Baret 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 \( R = \{R_1, \ldots, R_k\} \) 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 \( R_i = (X_i, \Sigma_i, \delta_i, x_{0,i}, X_{m,i}) \) with a finite set of states \( X_i \), a finite alphabet of events \( \Sigma_i \), a partial transition function \( \delta_i : \Sigma_i \times X_i \rightarrow X_i \), an initial state \( x_{0,i} \in X_i \), and a set of marked states \( X_{m,i} \subseteq X_i \) 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 \( R_i, R_j \in R \) be finite automata. Then, the synchronous composition \( R_i \parallel R_j \) of \( R_i \) and \( R_j \) is defined as the finite automaton \( R_{ij} := (X_{ij}, \Sigma_{ij}, \delta_{ij}, x_{0,ij}, X_{m,ij}) \) with \( X_{ij} = X_i \times X_j \), \( \Sigma_{ij} = \Sigma_i \cup \Sigma_j \), \( x_{0,ij} = (x_{0,i}, x_{0,j}) \), and \( X_{m,ij} = X_{m,i} \times X_{m,j} \). For a state \( (x_i, x_j) \in X_{ij} \) and an event \( \sigma \in \Sigma_{ij} \), the transition function is \( \delta_{ij}((x_i, x_j), \sigma) := \begin{cases} \delta_i(x_i, \sigma), \delta_j(x_j, \sigma) & \text{if } \sigma \in \Gamma_i(x_i) \cap \Gamma_j(x_j) \\ \delta_i(x_i, \sigma), x_j & \text{if } \sigma \in \Gamma_i(x_i) \cap \Sigma_j \\ x_i, \delta_j(x_j, \sigma) & \text{if } \sigma \in \Gamma_j(x_j) \cap \Sigma_i \\ \text{undefined} & \text{otherwise} \end{cases} \)

Accordingly, the overall system representation of the hierarchical and decentralized controllers evaluates to a finite automaton \( R := \parallel_{i=1}^{k} R_i \), 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 (\( \text{mvr} \rightarrow \text{product from left/to right} \)) and moves accordingly (\( \text{mvr} \rightarrow \text{move to right} \)). It stops (\( \text{stp} \)) when a sensor signals the product arrival at the machine (\( \text{son} \)), which is indicated by the shared event \( \text{am} \) (product at machine). After \( \text{am} \), 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 \( \text{am}, f \) and \( \text{tr} \) occur such that the product is not transported to the right before the machine finished processing.

![Fig. 1](image)

(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 in state 1.1 can be realized by a high-level controller \( R_3 \). The conveyor belt notices if a product has to be transported (\( \text{mvr} \rightarrow \text{product from left/to right} \)) and moves accordingly (\( \text{mvr} \rightarrow \text{move to right} \)). It stops (\( \text{stp} \)) when a sensor signals the product arrival at the machine (\( \text{son} \)), which is indicated by the shared event \( \text{am} \) (product at machine). After \( \text{am} \), 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 \( \text{am}, f \) and \( \text{tr} \) occur such that the product is not transported to the right before the machine finished processing.

![Fig. 2](image)

Fig. 2. Communication model for the manufacturing unit.

Formally, the outcome of the communication model construction is a tree structure \( T_C = (c, C_k, c_c, p_c) \) (see e.g., Hopcroft and Ullman (1979)) which captures the hierarchical relationship of the distributed controllers. In this way, the set of vertices \( C \) denotes the set of CMAs \( C_k = (Q_k, \Sigma, \nu, \gamma, \delta, Q_m) \) for the controllers \( R_k, i = 1, \ldots, k \) with the set of jobs \( \gamma = \delta_{\text{out}} \cup \delta_{\text{in}} \) that are communicated from \( \delta_{\text{out}} \) and to \( \delta_{\text{in}} \) as described in Example 2. Furthermore, \( c_c \) is the root vertex and \( c_c : c \rightarrow 2^c \) and \( p_c : c \rightarrow c \) are the children map and the parent map such that \( c_c(G_i) \) is the set of children and \( p_c(G_i) \) is the parent of \( G_i \in C \). Every vertex without children is called a leaf. We also distinguish the set of jobs \( \gamma_k \) that are sent for each \( \sigma \in \Sigma_k \), and call \( \sigma \) the command job for \( \sigma \).

Observing that again interaction between the CMAs via the exchange of jobs is modeled by jobs shared between CMAs, the overall communication model \( C = (Q, \gamma, \nu, \gamma_0, Q_m) \) is obtained as the synchronous composition of the CMAs: \( C := \prod_{i=1}^{k} C_k \).

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 \( \text{am} \) at all controllers.
Properties: Let \( q = (q_1, \ldots, q_k) \in Q \) and \( J \in J_{\sigma} - \{\sigma_c\} \) for \( \sigma \in \Sigma \cap \sigma \) s.t. \( \nu_i(q_i, J) \) exists for some \( 1 \leq i \leq k \). Then

1. For all \( j \) s.t. \( J \in J_j \) it follows that \( \nu_j(q_j, J) \) exists.
2. For all \( j \) s.t. \( J \in J_{\text{out}, j} \) it holds that there is a \( J' \in J_{\text{out}, j} \cap J_0 \) s.t. \( \nu_j(q_j, J') \) 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 \( \sigma \) 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 \( \sigma \in \Sigma \cap \sigma \) theoretically occurs if every controller \( R_i \) that shares \( \sigma \) is in a state \( x_i \in X_c \) where \( \delta_i(x_i, \sigma) \) exists. According to the distributed implementation with the communication model, \( \sigma \) physically happens when the command job \( \sigma_c \) is transmitted. Depending on the physical interpretation of \( \sigma \), 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 by introducing a map \( r : \Sigma \cap \sigma \rightarrow \mathbb{R} \cup \{\infty\} \) for the shared events, where \( r(\sigma) \) represents the maximal allowable time between the theoretical and physical occurrence of an event \( \sigma \in \Sigma \cap \sigma \) (e.g., the reaction time to a sensor event). The execution of an event \( \sigma \in \Sigma \cap \sigma \) in the worst case requires the communication of all jobs related to \( \sigma \), while the actual event can happen any time between the transmission of the first and the last job for \( \sigma \). Denoting \( N_0 \) the number of jobs for a task \( \sigma \), a deadline \( d_j := \frac{r(\sigma)}{N_0} \) is associated with each job \( J \in J_j \). In this framework, \( d_j \) indicates that if \( J \) is ready to be transmitted by its corresponding controller at time \( t_0 \), it has to be sent at \( t_0 + d_j \) latest. A communication model with a map \( r : \Sigma \cap \sigma \rightarrow \mathbb{R} \cup \{\infty\} \) 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 \( N = \{N_1, \ldots, N_k\} \) 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 \( t_0 \) 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.](image-url)

3.2 Network Node

A network node \( N_i \in N \) implements the following entities.

- **N1**: a CMA \( C_{\sigma_i} \),
- **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, \( e \in R \) 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 \(q \in Q\) of its associated CMA \(C_{R_i}\).

A message \(M\) of a sender node \(N_i \in \mathcal{N}\) in state \(q \in Q\) contains:

- **M1** A set of jobs to be sent by \(N_i\). To this end, the longest sequence of outgoing jobs \(s = J_1J_2 \ldots J_m \in \mathcal{J}_{out}\) is computed s.t. \(q' := v_1(q,s)\) exists. \(^2\) The set of jobs of the message contains all jobs in \(J_1J_2 \ldots J_m\).

- **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 \(R\)ers. If \(s\) is not empty, then for each job, a request \((N_i,e,d,\sigma)\) with the receiver node \(N_i\), an eligibility time \(e\), a deadline \(d\) and the task \(\sigma\) of the job is generated. Otherwise, a self request \((N_i,e,d,\sigma)\) is generated, where \(e\) is the next time when \(N_i\) can send a message, and \(d\) is the deadline of the valid task \(\sigma\) 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 \(v_1(q,s)\) (\(s\) 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 \(q \in Q\) 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 \(F_{\text{max}}\) 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 described 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 \(N_0\) constructs the output message for its initial state \(q_{0,k}\).
- **O2** All nodes put the CR \((N_i,0,1,-)\) 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 \(t = 0\) ms, all nodes are in the initial states of their respective communication model in Fig. 2; the time slot is \(t_s = 1\) ms; the eligibility times of \(N_1\), \(N_2\) and \(N_3\) are 1 ms, 0.5 ms and 1 ms, respectively, and \(d_{\text{min}} = 50\) 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 \(\{a_{11}, r_{11}\}\), and the minischedule \((N_1, 1\, \text{ms}, 50\, \text{ms}, \emptyset)\).\((N_2, 0.5\, \text{ms}, 50\, \text{ms}, \emptyset)\). 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 \(t = 1\) ms, \(N_3\) sends the content of its output buffer (O3, O4). The operation of node \(N_1\) is as follows:

1. **Output message for its initial state**. Assume that at time \(t = 1\) ms, node \(N_1\) has a job \((A, 0.5\, \text{ms}, 1\, \text{ms})\) for the receiver node \(N_2\) in its input buffer. The node constructs the output message \((A, 0.5\, \text{ms}, 1\, \text{ms})\). Then the node updates its state to \((A, 0.5\, \text{ms}, 1\, \text{ms})\) and sends the message in the output buffer (O5).
2. **Input buffer computation**. State update of the CRs in its input buffer. The node updates its state to \((A, 0.5\, \text{ms}, 1\, \text{ms})\), and the CR is added to the PQ (O5).
3. **Proposition 3.1**. Let \(N_1\) be a node with the communication model tree structure \(T_C\) and the communication operation as defined above. Then, the maximum number \(Q_{\text{max}}\) 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 \(q = (q_1, \ldots, q_k) \in \mathcal{Q}\) and \(\sigma \in \Sigma_q\). Let \(C_j\) be the highest-level node such that \(f_j \cap f_0 \neq \emptyset\) and define the subtree \(T_{\theta_j}\) 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.
Theorem 3.1. (Bound on Network Bandwidth.) Let $\mathcal{X}$ be a set of nodes with the communication model tree structure $T_c$ and the communication operation as defined above. Then a network bandwidth $B \geq B_{\text{min}} \equiv \frac{F_{\text{max}}}{h_{\text{max}}}$ 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. $C \geq F_{\text{max}}/h_{\text{max}}$ ensures that the message can be sent until the next transmission starts. Conversely, if $C < F_{\text{max}}/h_{\text{max}}$ 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 $d_{\text{min}}, Q_{\text{max}}, \epsilon_{\text{max}}$ and $F_{\text{max}}$ 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 $d_{\text{min}} = 20 \text{~ms}$ ensure reliable and safe system operation. Additionally, this study assumes that all nodes implement PLCs with a cycle time $\epsilon_{\text{max}}/2$ such that the eligibility time $\epsilon_{\text{max}}$ 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.](image-url)
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 $t_s = \frac{(20\text{ms} - e_{max})}{32\text{ms}}$ 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 $e_{max}$ ($\geq 3$ ms) and/or very large $t_s/t_{s, max}$ ($\geq 6$), 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 NCT. 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.

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


