Asian Journal of Control, Vol. 10, No. 1, pp. 55–66, January 2008
Published online in Wiley InterScience (www.interscience.wiley.com) DOI: 10.1002/asjc.006

CONTROLLER DESIGN FOR FLOW NETWORKS OF SWITCHED SERVERS WITH SETUP TIMES: THE KUMAR-SEIDMAN CASE AS AN ILLUSTRATIVE EXAMPLE

Erjen Lefeber and J. E. Rooda

ABSTRACT

In this paper we consider the control of a reentrant manufacturing system with setup times, as introduced by Kumar and Seidman. In most literature on control of a network of servers with setup times, first a policy is introduced and then the resulting network behavior is analyzed. In manufacturing systems the network typically is fixed and given *a priori*. Furthermore, optimal steady state behavior is desired. Therefore, this paper follows a different approach. First optimal steady state network behavior is determined, then a feedback controller is presented which makes the network converge towards this desired steady state behavior. The resulting controller is a non-distributed controller: each server needs global state information. For a manufacturing system this is not a problem, since global information typically is available. Finally it is shown that a distributed controller (each server needs only local state information) can also be used to achieve the same result.

Key Words: Hybrid system, network control, distributed control, tracking control.

I. INTRODUCTION

Reentrant manufacturing systems might show some unexpected behavior. In [1] it was shown by simulation that even when each server has enough capacity to serve all jobs, these networks can be unstable, that is, the total number of jobs in the network explodes as time evolves. Whether this happens or not depends on the policy used to control the flows through the network. In [4] it was shown analytically that using a clearing policy (serve the queue you are currently serving until it is empty, then switch to another queue), certain networks become unstable, even deterministic systems with no setup times. In [8] several clearing policies have been introduced, the so-called clear a fraction (CAF) policies. It was shown that these policies are stable for a single server in isolation in a deterministic environment. Furthermore, it was shown that a CAF policy stabilizes a multi-server system, provided the network is acyclic. A network is called acyclic if the servers can be ordered in such a way that jobs can only move from one server to a server higher in the ordering. A network is called non-acyclic if such an ordering is not possible. The example in [4] shows that there are non-acyclic networks that can not be stabilized by a CAF policy.

The main reason why CAF policies can fail for a non-acyclic network is because they spend too long on serving one type of job. This results in starvation of other servers, which is a waste of their capacity. Due to this waste, the effective capacity of these other servers is not sufficient anymore, resulting in an unstable system. This observation has led to the development

Manuscript received October 13, 2006; revised September 14, 2007; accepted October 4, 2007.

Both authors are with the Systems Engineering Group, Department of Mechanical Engineering, Eindhoven University of Technology, PO Box 513, 5600 MB Eindhoven, The Netherlands (e-mail: {A.A.J.Lefeber,J.E.Rooda}@tue.nl).

The authors sincerely thank Joost van Eekelen for providing the code for the simulations.

of so-called buffer regulators [3, 9] or gated policies. The main idea is that each buffer contains a gate, so the buffer is split into two parts (before and after the gate). Instead of switching depending on the total buffer contents, switching is now determined based on the buffer contents after the gate. As a result, a server might now leave a buffer earlier, avoiding long periods of serving one type of job. It has been shown in [9] that under certain conditions on these regulators the (possibly non-acyclic) network is stabilized. Since non-acyclic networks are only unstable under certain conditions, applying buffer regulators is not always necessary. Needlessly applying buffer regulators results in a larger mean number of jobs in the network, which from a performance point of view is undesired.

In [10, 11] a different approach has been developed. First, the minimal period is determined during which the network is able to serve all jobs that arrive during that period. Given this minimal period, or any longer period, one can determine how long each server should serve each step. Next, a distributed controller is proposed where each server serves its buffers in a cyclic order until either the buffer becomes empty or the server has spend the time reserved for serving that step. If necessary the next setup is prolonged to make sure that the time reserved for serving a step is fully used. In [10, 11] it was shown that this policy guarantees that all trajectories of the controlled system are bounded and that for constant arrival rates the behavior of the network eventually becomes periodic, *i.e.*, regular behavior is achieved. The policy introduced in [10, 11] has two disadvantages. First, it is not really a state feedback: the current mode of each server has been fixed *a priori*, independent of the current state. Second, if a large number of jobs is in the system initially, the resulting periodic steady state behavior also contains a large number of jobs in the system, as no attempt is made to reduce the number of jobs, which is undesired from a performance point of view.

The above mentioned references are only a few of the large amount of papers that have been published in this area, or related subjects, such as (dynamic) lot sizing problems and polling models, see also [12] and references therein. However, most of these papers have one thing in common: first a policy is proposed, and then the resulting behavior of the network under this policy is considered. When performance is considered, mostly the system behavior is optimized over a family of considered policies. One strength of existing results is that they often can be applied to any network and that stability is guaranteed. However, stability is only a prerequisite for a good network control policy. It is usually unclear if the presented policies yield suitable network performance. In particular, it is not clear how to obtain desired network behavior.

Instead of having one policy which guarantees stability for an arbitrary network, we aim at a general methodology for designing a tailor made policy for a specific network under consideration, depending on its desired performance. In particular, if a network is known *a priori* (and not subject to change), which typically is the case for manufacturing systems, this is a reasonable approach from a control theoretical point of view. Therefore, we propose an entirely different way of looking at the problem of controlling a network of switching servers with setup times. Instead of starting from a policy and then analyzing the proposed policy, we start from *a priori* specified desired network behavior. This behavior might be optimal, but not necessarily. Using this desired behavior for the network under consideration as a starting point, we derive a policy for this network which guarantees convergence of the system towards this desired behavior.

This approach enables us to make a distinction between, on the one hand, the trajectory generation problem, and, on the other hand, the tracking control problem. Though the trajectory generation problem (determining feasible network behavior) is a challenging problem, the main focus of this paper is on the tracking control problem. In tracking control typically two problems are studied: first, full state information is assumed (state feedback); second, only partial information is assumed (output feedback). In [7] we presented an approach to derive a state feedback controller for an arbitrary given network and arbitrary feasible behavior for this network. This approach guarantees regular behavior for the closed-loop network according to the desired behavior. As in [10, 11], additional jobs might remain in the buffers, but the controller resulting from the approach introduced in [7] outperforms the one introduced in [10, 11].

The controllers resulting from the approach introduced in [7] are central controllers (or state-feedback controllers) which determine for each server when to switch to what job type, based on global state information, whereas in certain cases decentralized or distributed controllers (or output-feedback controllers) are needed. The controllers introduced in [10, 11] are an example of such a distributed controller.

In this paper we show, by means of an example, that this new way of looking at the problem of controlling a network of switching servers with setup times, as introduced in [7], can be used for deriving tailor-made network controllers which achieve *a priori* specified network behavior and can be implemented in a distributed way. The resulting distributed controllers are different from the distributed controllers so far proposed in literature.

II. THE KUMAR-SEIDMAN CASE

In [4], Kumar and Seidman presented the manufacturing system shown in Fig. 1. A single job-type is considered which first visits machine A, then machine B, then machine B again, and finally machine A again. The successive buffers visited will be denoted by $1, 2, 3$, and 4, respectively. The constant input rate λ into buffer 1 is 1 job/time-unit, while the maximal process rates required at the buffers are $\mu_1 = 1/0.3, \mu_2 = 1/0.6, \mu_3 = 1/0.3$, and $\mu_4 = 1/0.6$, respectively. Lastly, the times for setting up buffers 1 and 4 at machine A are $\sigma_{41} = 50$ and $\sigma_{14} = 50$, the times for setting up to buffers 2 and 3 at machine B are $\sigma_{32} = 50$ and $\sigma_{23} = 50$.

Even though for this system each machine has enough capacity, *i.e.*, $(\lambda/\mu_1) + (\lambda/\mu_4) < 1$ and $(\lambda/\mu_2) + (\lambda/\mu_3) < 1$, it has been shown in [4] that since $(\lambda/\mu_2) + (\lambda/\mu_4) > 1$ and setup times are all positive, using a clearing policy for both machines results in an unstable system.

The state of this system is not only given by the buffer contents x_1, x_2, x_3, and x_4, but also by the remaining setup time at machine A, x_0^A, the remaining setup time at machine B, x_0^B, and the current mode $m = (m^A, m^B) \in \{(1,2), (1,3), (4,2), (4,3)\}$. We say that the system is in mode $(1,2)$ when machine A is processing or setting up for step 1 and machine B is processing or setting up for step 2. Similar for the other modes.

The input of this system is given by rates $u_1 \leq \mu_1$, $u_2 \leq \mu_2$, $u_3 \leq \mu_3$, and $u_4 \leq \mu_4$, at which respectively buffers 1, 2, 3, and 4 are being served (a machine not necessarily has to serve at full rate), as well as the current activity for machine A, $u_0^A \in \{\text{❶}, \text{①}, \text{❹}, \text{④}\}$, and for machine B, $u_0^B \in \{\text{❷}, \text{②}, \text{❸}, \text{③}\}$. The activity ❶ denotes setting up for serving step 1, whereas ① denotes serving step 1. Similarly, the activities for steps $2, 3$, and 4 can be distinguished.

The dynamics of this system is hybrid. On the one hand, we have the discrete event dynamics:

$$x_0^A := \sigma_{14}; \quad m^A := 4 \quad \text{if } u_0^A = \text{❹} \text{ and } m^A = 1$$

$$x_0^A := \sigma_{41}; \quad m^A := 1 \quad \text{if } u_0^A = \text{❶} \text{ and } m^A = 4$$

$$x_0^B := \sigma_{23}; \quad m^B := 3 \quad \text{if } u_0^B = \text{❸} \text{ and } m^B = 2$$

$$x_0^B := \sigma_{32}; \quad m^B := 2 \quad \text{if } u_0^B = \text{❷} \text{ and } m^B = 3.$$

In words, if the system is currently in a mode, and according to the input the current activity becomes "set up to a different mode", both the remaining setup time and current mode change.

On the other hand, we have the continuous dynamics

$$\dot{x}_0^A(t) = \begin{cases} -1 & \text{if } u_0^A \in \{\text{❶}, \text{❹}\} \\ 0 & \text{if } u_0^A \in \{\text{①}, \text{④}\} \end{cases} \qquad \dot{x}_0^B(t) = \begin{cases} -1 & \text{if } u_0^B \in \{\text{❷}, \text{❸}\} \\ 0 & \text{if } u_0^B \in \{\text{②}, \text{③}\} \end{cases}$$

$$\dot{x}_1(t) = \lambda - u_1(t) \qquad \dot{x}_2(t) = u_1(t) - u_2(t)$$

$$\dot{x}_4(t) = u_3(t) - u_4(t) \qquad \dot{x}_3(t) = u_2(t) - u_3(t).$$

Furthermore, at each time instant the input is subject to the constraints $u_1 \geq 0$, $u_2 \geq 0$, $u_3 \geq 0$, $u_4 \geq 0$, and

$$\begin{array}{lll}
u_0^A \in \{\text{❶}, \text{❹}\} & u_1 = 0 & u_4 = 0 \\
 & & \text{for } x_0^A > 0 \\
u_0^A \in \{\text{①}, \text{❹}\} & u_1 \leq \mu_1 & u_4 = 0 \\
 & & \text{for } x_0^A = 0, x_1 > 0, m^A = 1 \\
u_0^A \in \{\text{①}, \text{❹}\} & u_1 \leq \lambda & u_4 = 0 \\
 & & \text{for } x_0^A = 0, x_1 = 0, m^A = 1 \\
u_0^A \in \{\text{❶}, \text{④}\} & u_1 = 0 & u_4 \leq \mu_4 \\
 & & \text{for } x_0^A = 0, x_4 > 0, m^A = 4 \\
u_0^A \in \{\text{❶}, \text{④}\} & u_1 = 0 & u_4 \leq \min(u_3, \mu_4) \\
 & & \text{for } x_0^A = 0, x_4 = 0, m^A = 4 \\
u_0^B \in \{\text{❷}, \text{❸}\} & u_2 = 0 & u_3 = 0 \\
 & & \text{for } x_0^B > 0 \\
u_0^B \in \{\text{②}, \text{❸}\} & u_2 \leq \mu_2 & u_3 = 0 \\
 & & \text{for } x_0^B = 0, x_2 > 0, m^B = 2 \\
u_0^B \in \{\text{②}, \text{❸}\} & u_2 \leq \min & u_3 = 0 \\
 & (u_1, \mu_2) & \\
 & & \text{for } x_0^B = 0, x_2 = 0, m^B = 2 \\
u_0^B \in \{\text{❷}, \text{③}\} & u_2 = 0 & u_3 \leq \mu_3 \\
 & & \text{for } x_0^B = 0, x_3 > 0, m^B = 3 \\
u_0^B \in \{\text{❷}, \text{③}\} & u_2 = 0 & u_3 \leq u_2 \\
 & & \text{for } x_0^B = 0, x_3 = 0, m^B = 3.
\end{array}$$

In words, these constraints say that in case the server is setting up, no jobs can be served. Furthermore, in case a setup has been completed, only the job type

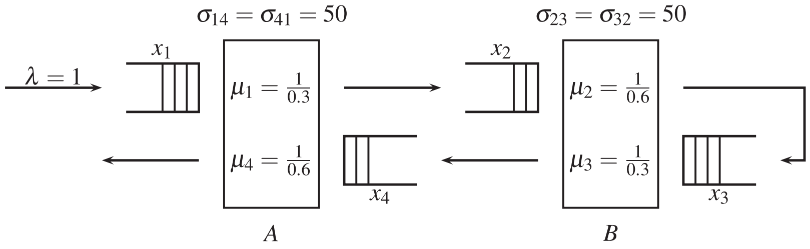

Fig. 1. The system introduced in [4].

can be processed for which the server has been set up. This processing takes place at a rate which is at most μ_i if jobs of step i are available in the buffer and at the arrival rate if no jobs of type i are available in the buffer ($i \in \{1,2,3,4\}$). Also, it is possible to either stay in the current mode, or to switch to the other mode. In particular it is possible during a setup to leave that setup and start a setup to the other type again. The latter setup is assumed to take the entire setup time.

Having defined the state, input, dynamics and constraints for the system, we can consider the problem of controlling this system, *i.e.*, designing an input u which satisfies the constraints and achieves desired behavior. But before we can do so we first need to specify desired behavior.

III. DESIRED PERIODIC BEHAVIOR

As mentioned in the introduction, we do not want to start from a policy which works for a general network and analyze the resulting closed-loop behavior, but given this specific manufacturing system we want to start from desired system behavior and determine a feedback controller which makes the system converge towards this desired behavior. This implies that we first need to define desired periodic behavior, *i.e.*, we have to first solve a trajectory generation problem. For manufacturing systems this would typically be behavior for which the mean amount of jobs in the system is minimal, since from Little's law we know that this results in the smallest mean flow time (the time a job spends in the system). More precisely, we would like to minimize

$$J = \frac{1}{T}\int_0^T x_1(t)+x_2(t)+x_3(t)+x_4(t)\,\mathrm{d}t$$

over the set of feasible periodic orbits, where T denotes the period of the periodic orbit under consideration.

A first observation is that the set of feasible periodic orbits is not empty. Note that, even though in [4] it has been shown that using a clearing policy for both machines renders the system of Section II unstable, it has been made clear in [10, 11] and other papers that this is not a system property, but due to the policy used. To make the latter more clear, consider machine A. Each job needs $0.3+0.6=0.9$ time-units of processing. During a cycle of serving both step 1 and step 4, in total $50+50=100$ time-units are lost due to setups. Therefore, during a cycle of 1000 time-units, the 1000 jobs that arrive can also be processed. The same holds for machine B (since the parameters of machine B are identical). These 1000 time-units are also the minimal cycle period as observed in [11], but a longer time period can be used as well.

A second observation is that a periodic orbit with the smallest mean amount of jobs does not necessarily also have the smallest period, as presented in [2, 5]. Having a longer period in some cases reduces the mean amount of jobs in the system. At first glance this is counterintuitive. Having a longer period implies not serving jobs at the highest possible rate. Can this be efficient? It turns out that a trade-off needs to be made. Since a small period implies that on average more time is wasted on setups. So one either has to waste capacity by frequent switching, or by occasionally not serving at the highest possible rate.

Since the main focus of this paper is not on the trajectory generation problem but on the problem of controlling the network towards given desired periodic behavior, we simply start from the desired network behavior as depicted in Fig. 2. It turns out that this behavior minimizes both the mean amount of jobs in the system and the mean amount of work in the system, but the interested reader is referred to [6] for a proof of these statements.

The desired network behavior can be described as follows.

- From $t=0$ till $t=350$ the system is in mode $(1,2)$. From $t=0$ till $t=50$ both machines are setting up, from $t=50$ till $t=350$ both machines are serving at

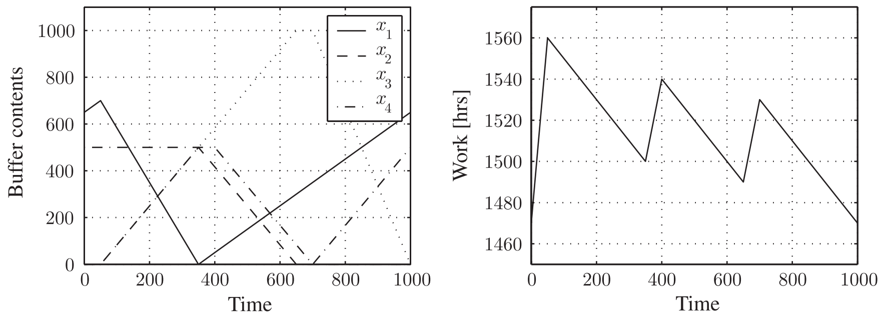

Fig. 2. Evolution over time of both buffer contents and amount of work for the desired periodic behavior.

full rate. At the end of this mode $x_1 = 0$, $x_2 = 500$, $x_3 = 500$, and $x_4 = 500$.
- From $t = 350$ till $t = 650$ the system is in mode $(4, 2)$. From $t = 350$ till $t = 400$ machine A is setting up, from $t = 400$ till $t = 650$ machine A is serving at full rate. Machine B is serving at full rate all the time. At the end of this mode $x_1 = 300$, $x_2 = 0$, $x_3 = 1000$, and $x_4 = 83\frac{1}{3} = 50/0.6$.
- From $t = 650$ till $t = 1000$ the system is in mode $(4, 3)$. Machine A is serving at full rate all the time. From $t = 650$ till $t = 700$ machine B is setting up. From $t = 700$ till $t = 1000$ machine B is serving at full rate. At the end of this mode $x_1 = 650$, $x_2 = 0$, $x_3 = 0$, and $x_4 = 500$.

For this periodic orbit the mean amount of jobs in the system equals 1350 and the mean amount of work equals 1515 time-units. Notice that for the desired periodic orbit the system never is in mode $(1, 3)$. Furthermore, the largest amount of work in the system is reached at $t = 50$, in mode (1,2). Finally, the above mentioned description of the desired behavior is not a policy yet.

IV. NON-DISTRIBUTED FEEDBACK

In the previous section we presented desired periodic behavior for the manufacturing system introduced in Section II. In [6] it has been shown that for the periodic orbit as depicted in Fig. 2, both the mean amount of jobs and the mean amount of work in the system is minimal. In particular this implies that the mean flow time, *i.e.* the time a job spends in the system, is minimal. Given this desired periodic behavior, we next want to have a feedback controller which makes the manufacturing system introduced in Section II converge towards this desired behavior from any initial condition. Assuming we are in a manufacturing setting, global information is available, which in particular implies we do not have to restrict ourselves to distributed controllers. A non-distributed controller can be implemented relatively easily.

In [7] we introduced an approach for deriving a feedback controller from a given desired periodic orbit for arbitrary networks of switching servers with setup times. This approach guarantees convergence similar to the desired periodic orbit. However, it might be that some buffers always contain a fixed additional number of jobs, compared to the desired periodic orbit. Something similar holds for the controllers presented in [10, 11], but the controller that follows from applying the ideas in [7] results in smaller additional amount of jobs in the system.

Before presenting the controller, we first consider the desired behavior as depicted in Fig. 2. The system cyclically visits the modes $(1, 2)$, $(4, 2)$, and $(4, 3)$. In particular the system does not visit mode $(1, 3)$. Similar to the controller presented in [7] we therefore let the feedback be such that the system cyclically visits the modes $(1, 2), (4, 2)$, and $(4, 3)$. If the system happens to be initially in mode $(1, 3)$, we switch to mode $(1, 2)$, since the largest amount of work in the system is reached in mode $(1, 2)$, *cf.* [7].

Next, we need to determine when to leave each mode. Consider mode $(1, 2)$ of the desired periodic orbit. In this mode, only contents of buffer 1 decreases and reaches the value 0. Buffers 2 and 3 both increase. Since machine B serves from buffer 2 to buffer 3, these buffers can be considered together. Both buffers increase until $x_2 + x_3 = 1000$. Therefore, the feedback leaves mode $(1, 2)$ when $x_1 = 0$ and $x_2 + x_3 \geq 1000$. Notice that it might happen that machine A needs to continue serving buffer 1 at the arrival rate whenever $x_1 = 0$ and $x_2 + x_3 < 1000$. Similarly, when $x_1 = x_2 = 0$ and $x_3 < 1000$, machine B might also need to continue serving buffer 2 at the arrival rate.

In mode $(4,2)$ the contents of both buffer 2 and buffer 4 decreases, until $x_2=0$ and $x_4=83\frac{1}{3}$, respectively. The contents of both buffer 1 and buffer 3 increases, until $x_1=300$ and $x_3=1000$, respectively. Notice that by staying in mode $(4,2)$ it is not possible for x_3 to become arbitrarily large. Therefore, the feedback leaves mode $(4,2)$ when $x_2=0$, $x_4\leq 83\frac{1}{3}$, and $x_1\geq 300$. All three conditions need to be met. Therefore, it might be required for machine A or machine B to idle.

Similarly, the feedback leaves mode $(4,3)$ when both $x_3=0$ and $x_1\geq 650$. Again, it might be required for machine A or machine B to idle.

Notice that the condition on x_1 for leaving mode $(4,2)$ and mode $(4,3)$ are automatically fulfilled once the system has been in mode $(1,2)$. Therefore, we drop these conditions in the feedback.

The above can be summarized as follows:

Proposition 1. Consider the system as depicted in Fig. 1 in closed-loop with the following feedback:

- If initially in mode $(1,3)$, switch to mode $(1,2)$.
- If in mode $(1,2)$, stay in this mode until both $x_1=0$ and $x_2+x_3\geq 1000$. Then switch to mode $(4,2)$. Both machines serve at the highest possible rate (which might be the arrival rate).
- If in mode $(4,2)$, stay in this mode until both $x_2=0$ and $x_4\leq 83\frac{1}{3}$. Then switch to mode $(4,3)$. Both machines serve at the highest possible rate (which might be 0).
- If in mode $(4,3)$, stay in this mode until $x_3=0$. Then switch to mode $(1,2)$. Both machines serve at the highest possible rate (which might be 0).

Then the resulting closed-loop system converges towards the behavior as depicted in Fig. 2.

Proof. Let $t_{12}^{(k)}$ denote the time at which mode $(1,2)$ is entered for the k^{th} time $(k\geq 1)$, and let $(x_1^{(k)},x_2^{(k)},x_3^{(k)},x_4^{(k)})$ denote the buffer contents at $t_{12}^{(k)}$ for buffers 1, 2, 3, and 4, respectively. Let $t_{42}^{(k)}$ and $t_{43}^{(k)}$ denote the moment at which mode $(4,2)$ and mode $(4,3)$ is entered, respectively, and define the durations of these modes as $\tau_{12}^{(k)}=t_{42}^{(k)}-t_{12}^{(k)}$, $\tau_{42}^{(k)}=t_{43}^{(k)}-t_{42}^{(k)}$, and $\tau_{43}^{(k)}=t_{12}^{(k+1)}-t_{43}^{(k)}$.

It needs to be shown that

$$\lim_{k\to\infty}(x_1^{(k)},x_2^{(k)},x_3^{(k)},x_4^{(k)})=(650,0,0,500). \qquad (1)$$

A first observation is that at $t=t_{43}^{(k)}$ we have $x_2=0$ and $x_3\geq 1000$, since in mode $(1,2)$ at least 1000 jobs are processed by machine A and in mode $(4,2)$ buffer 2 is emptied. As a result, at $t=t_{12}^{(k+1)}$ we have $x_2^{(k+1)}=0$, $x_3^{(k+1)}=0$, $x_4^{(k+1)}\geq 500$.

A second observation is that $\tau_{42}^{(k+1)}\geq 300$ (since $x_4^{(k+1)}\geq 500$), and $\tau_{43}^{(k)}\geq 350$ (since machine B needs to process at least 1000 jobs and requires a setup).

From these observations it follows that without loss of generality we can assume $x_1^{(k)}\geq 650$, $x_2^{(k)}=0$, $x_3^{(k)}=0$, $x_4^{(k)}\geq 500$ by considering $k\geq 2$. Under these assumptions we would like to determine $x_1^{(k+1)}$ and $x_4^{(k+1)}$.

During mode $(1,2)$, first both machines need to be set up, which takes 50 time-units. At $t_{12}^{(k)}+50$ buffer 1 contains $x_1^{(k)}+50$ jobs. If machine A serves at full rate, x_1 effectively reduces at a rate of $(1/0.3)-1=7/3$ jobs/solidus time-unit. Therefore, clearing buffer 1 takes $\frac{3}{7}(x_1^{(k)}+50)$, during which $\frac{10}{7}(x_1^{(k)}+50)$ jobs are being processed by machine A. Notice that since $x_1^{(k)}\geq 650$ also 1000 jobs have been processed, so $\tau_{12}^{(k)}=50+\frac{3}{7}(x_1^{(k)}+50)$. Also, at $t_{42}^{(k)}=t_{12}^{(k)}+\tau_{12}^{(k)}$ we have $x_2=x_3=\frac{5}{7}(x_1^{(k)}+50)$.

Next, from the condition on x_2 we obtain $\tau_{42}^{(k)}\geq\frac{3}{7}(x_1^{(k)}+50)$, and from the condition on x_4 we obtain $\tau_{42}^{(k)}\geq\frac{3}{5}x_4^{(k)}$, so $\tau_{42}^{(k)}=\max(\frac{3}{7}(x_1^{(k)}+50),\frac{3}{5}x_4^{(k)})$.

At $t_{43}=t_{42}^{(k)}+\tau_{42}^{(k)}$ we have $x_3=\frac{10}{7}(x_1^{(k)}+50)$, so $\tau_{43}=50+\frac{3}{7}(x_1^{(k)}+50)$.

Since $x_4=0$ at $t_{43}+50$ we get $x_4^{(k+1)}=\frac{5}{7}(x_1^{(k)}+50)$. Furthermore, $x_1^{(k+1)}=\tau_{42}^{(k)}+\tau_{43}^{(k)}=\max(\frac{3}{7}(x_1^{(k)}+50),\frac{3}{5}x_4^{(k)})+50+\frac{3}{7}(x_1^{(k)}+50)$.

To summarize, we have for $k\geq 2$:

$$x_1^{(k+1)}=\max\left(\frac{3}{7}(x_1^{(k)}+50),\frac{3}{5}x_4^{(k)}\right)+50+\frac{3}{7}(x_1^{(k)}+50) \qquad (2a)$$

$$x_2^{(k+1)}=0 \qquad (2b)$$

$$x_3^{(k+1)}=0 \qquad (2c)$$

$$x_4^{(k+1)}=\frac{5}{7}(x_1^{(k)}+50). \qquad (2d)$$

Using this map we need to show that (1) holds.

To simplify analysis of (2), define $y_1^{(k)}=\frac{1}{7}(x_1^{(k)}+50)-100$ and $y_4^{(k)}=\frac{1}{5}x_4^{(k)}-100$, or $x_1^{(k)}=7y_1^{(k)}+650$, and $x_4^{(k)}=5y_4^{(k)}+500$. Then using (2) we obtain for

$k \geq 2$:

$$y_1^{(k+1)} = \frac{3}{7}\max(y_1^{(k)}, y_4^{(k)}) + \frac{3}{7}y_1^{(k)} \quad y_1^B \geq 0 \qquad (3a)$$

$$y_4^{(k+1)} = y_1^{(k)} \quad y_4^B \geq 0. \qquad (3b)$$

From (3a) we have

$$0 \leq y_1^{(k+1)} \leq \frac{6}{7}\max(y_1^{(k)}, y_4^{(k)}).$$

Also

$$0 \leq y_1^{k+2} \leq \frac{6}{7}\max\left(\frac{6}{7}\max(y_1^{(k)}, y_4^{(k)}), y_1^{(k)}\right)$$

$$\leq \frac{6}{7}\max(y_1^{(k)}, y_4^{(k)}).$$

Therefore,

$$0 \leq \max(y_1^{k+2}, y_4^{k+2}) \leq \frac{6}{7}\max(y_1^{(k)}, y_4^{(k)})$$

from which we can conclude that

$$\lim_{k\to\infty} y_1^{(k)} = \lim_{k\to\infty} y_4^{(k)} = 0,$$

which shows that (1) holds. □

Remark 2. Proposition 1 only claims convergence towards the steady state cycle depicted in Fig. 2. No claims are made concerning optimal transient behavior.

V. DISTRIBUTED CONTROLLER IMPLEMENTATION

The controller derived in the previous section is a non-distributed controller. Machine A needs information about the state at machine B to determine what to do, and machine B requires information about the state at machine A. Since we consider a manufacturing problem, global information is available and a non-distributed controller can be implemented in a rather straightforward way.

Nevertheless, in this case the controller can be implemented in a distributed way. That is, such that machine A does not require information about the state at machine B and machine B does not require information about the state at machine A. This becomes clear from the proof of Proposition 1.

Notice that we know from the proof of Proposition 1 that for $k \geq 2$ when machine B switches from serving step 2 to serving step 3, $x_2 = 0$ and $x_2 + x_3 \geq 1000$. That is, buffer 2 needs to be empty, and machine B should have served at least 1000 jobs (for $k \geq 2$). Furthermore, notice that before switching from mode $(4, 2)$ to mode $(4, 3)$ it might happen that machine B needs to idle (in case x_4 is still too large). However, instead of first idling and then serving step 3, machine B can also first switch and serve step 3 and then idle for the same duration. As long as machine A stops serving step 4 at the same time as in the feedback of Proposition 1, the mapping (2) still holds. So as long as the behavior of machine A is still according to that specified by the feedback of Proposition 1 we can implement the following controller for machine B:

- Serve step 2 at the highest possible rate (which might be at the arrival rate or even idling) until both $x_2 = 0$ and at least 1000 jobs have been served. Then switch to step 3.
- Serve step 3 at maximal rate until $x_3 = 0$. Then switch to step 2.

It remains to determine a controller for machine A. As mentioned above, this controller needs to make sure that, after a finite transient, the overall system behavior still satisfies mapping (2). Notice that in this mapping only x_1 and x_4 play a role. This enables us to come up with a controller for machine A. From the proof of Proposition 1 we know that, for $k \geq 2$, during mode $(1, 2)$ machine A serves $\bar{x}_1^{(k)} = \frac{10}{7}(x_1^{(k)} + 50)$ jobs. From (2d) we know that after machine A has served step 4, machine B is empty and $\frac{1}{2}\bar{x}_1^{(k)}$ jobs remain in buffer 4. At the time machine A starts serving step 4, buffer 4 contains $x_4^{(k)}$ jobs. From this it follows that machine A has served $x_4^{(k)} + \bar{x}_1^{(k)} - \frac{1}{2}\bar{x}_1^{(k)} = x_4^{(k)} + \frac{1}{2}\bar{x}_1^{(k)}$ jobs. Therefore, we can implement the following controller for machine A:

- Serve step 1 at the highest possible rate (which might be at the arrival rate) until both $x_1 = 0$ and at least 1000 jobs have been served. Then switch to step 4. Let $\bar{x}_1$ denote the number of jobs served.
- Let $\bar{x}_4$ denote the number of jobs in buffer 4. Serve $\bar{x}_4 + \frac{1}{2}\bar{x}_1$ jobs from buffer 4. Then switch to step 1.

The above can be summarized in the following.

Proposition 3. Consider the system as depicted in Fig. 1 in closed-loop with the following feedback:

- Controller for machine A:
  - ◦ If serving step 1, continue until both $x_1 = 0$ and at least 1000 jobs have been served. Then switch to step 4. Let $\bar{x}_1$ be the number of jobs served during this mode.
  - ◦ Let $\bar{x}_4$ denote the number of jobs in buffer 4 when the setup to serving step 4 has completed. If serving step 4, continue until $\bar{x}_4 + \frac{1}{2}\bar{x}_1$ jobs have been served. Then switch to step 1.

- Controller for machine B:
  - Serve step 2 at the highest possible rate (which might be at the arrival rate or even idling) until both $x_2 = 0$ and at least 1000 jobs have been served. Then switch to step 3.
  - Serve step 3 at maximal rate until $x_3 = 0$. Then switch to step 2.

Then the resulting closed-loop system converges towards the behavior as depicted in Fig. 2.

Proof. Notice that the cycle for machine B always takes at least 1000 time-units, since at least 1000 jobs need to be served by each step during the cycle. First, assume that the cycle at machine B repeatedly takes exactly 1000 time-units. In that case machine A also takes a period of 1000 time-units. Furthermore, serving step 1 takes 300 time-units and serving step 4 takes 600 time-units, which implies that buffer 4 contains 500 jobs when machine A starts serving step 4. This implies that the system operates according to the desired periodic orbit.

As soon as the cycle for machine B takes strictly more than 1000 time-units, machine B will synchronize with machine A, *cf.* Section III, that is, after a finite transient, $x_2 = x_3 = 0$ and machine B is waiting for jobs to arrive from machine A. After this transient, machine A guarantees that (2a) holds again, which guarantees (1). □

VI. SIMULATION EXPERIMENTS

To support our claims we performed several simulations in which we compared the distributed controller presented in the previous section with the controller proposed in [11]. We chose the parameters of the latter controller according to the desired periodic orbit, *i.e.*, this controller is able to stay on the desired periodic orbit once the system is on it.

In the first simulation we started with an empty system, *i.e.* $x_1 = x_2 = x_3 = x_4 = 0$, where we initiated the controllers for machine A and B in “serving step 1” and “serving step 2”, respectively. The resulting responses of the controlled systems are given in Fig. 3 for both controllers.

In Fig. 3 we see that the controller proposed in [11] achieves regular behavior, as claimed in [11]. Furthermore, we see that buffer 1 is never emptied, resulting in a larger mean number of jobs in the system. For the controller of Proposition 3 we obtain convergence towards the desired periodic orbit. Notice that the total number of jobs oscillates between 1150 at the end of mode $(4, 3)$ and 1550 at the time the system starts serving step 4, as should be the case.

In the second simulation we do not start with an empty system, but with initially 1000 jobs in each buffer, *i.e.* $x_1 = x_2 = x_3 = x_4 = 1000$, where the controllers were initiated as in the first simulation. The resulting responses of the controlled systems are given in Fig. 4 for both controllers.

In Fig. 4 we clearly see the focus on regular behavior for the controller introduced in [11]. None of the buffers is cleared and the mean amount of jobs in the system is not reduced. However, the controller presented in Proposition 3 is able to reduce the number of jobs in the system and makes the system converge again to the desired optimal behavior.

Our third simulation experiment is a discrete event simulation. First of all, the hybrid fluid model as introduced in Section II has been replaced by its discrete event counterpart. Second, all process times and setup times are made stochastic by drawing them from independent exponential distributions. That is, process times for step 1 are drawn from an exponential distribution with mean 0.3, process times for step 2 are drawn from an exponential distribution with mean 0.6, setup times for switching from step 1 to step 4 are drawn from an exponential distribution with mean 50, etc. Again we initiated the system in $x_1 = x_2 = x_3 = x_4 = 1000$, but this time assuming that the setups to serving step 1 and step 2 have already been completed. Implementing the controller of Proposition 3 in this stochastic setting is rather straightforward. The controller introduced in [11], however, requires predetermined maximum durations. Therefore, we considered a preemptive resume policy. That is, in case the machine has not yet completed a job but it is time to switch, the machine stops serving the job and completes this service as soon as the machine has switched back. The resulting responses of the controlled systems in this stochastic discrete event environment are given in Fig. 5 for both controllers.

By controlling the seeds of the random generator we made sure to have a fair comparison between both controllers. The successive inter arrival times are the same for both experiments, as are the successive process times at each machine and the successive setup times. Looking at the results depicted in Fig. 5, we can make similar remarks to those made for the second simulation. Due to the stochasticity, we see some varying behavior over time. The controller introduced in [11] keeps the number of jobs in the system between 4000 and 6000, whereas the controller of Proposition 3 keeps

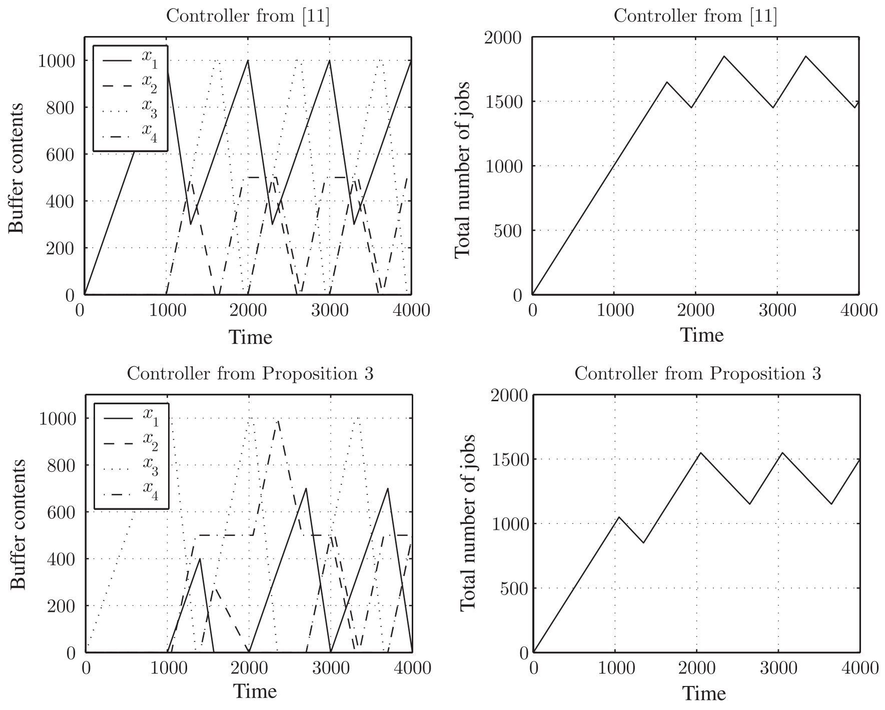

Fig. 3. The buffer contents and total number of jobs for a deterministic system initiated in $(x_1, x_2, x_3, x_4) = (0, 0, 0, 0)$ for both the distributed controller proposed in [11] and the distributed controller of Proposition 3.

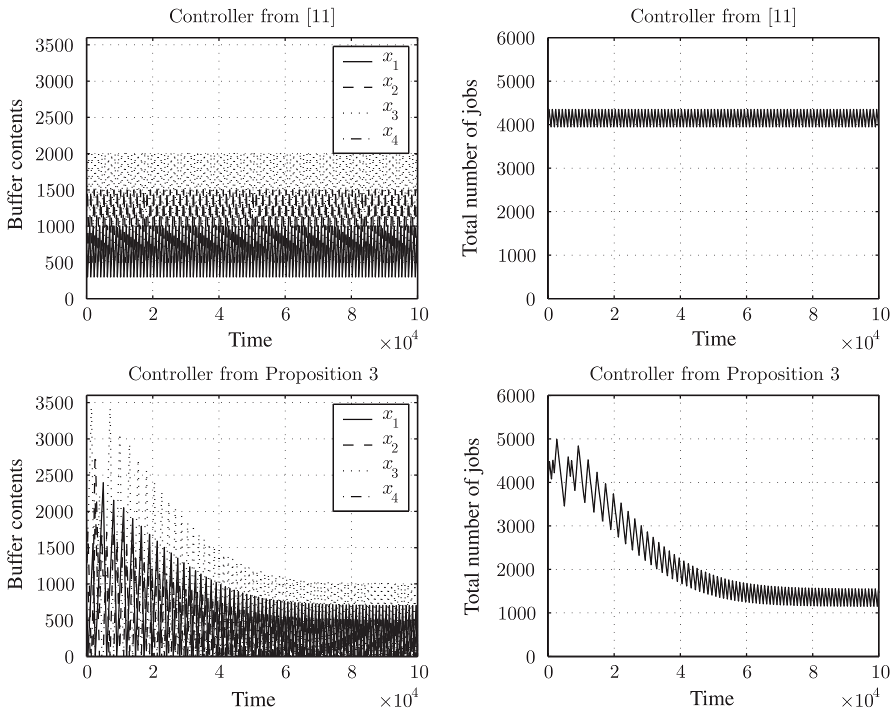

Fig. 4. The buffer contents and total number of jobs for a deterministic system initiated in $(x_1, x_2, x_3, x_4) = (1000, 1000, 1000, 1000)$ for both the distributed controller proposed in [11] and the distributed controller of Proposition 3.

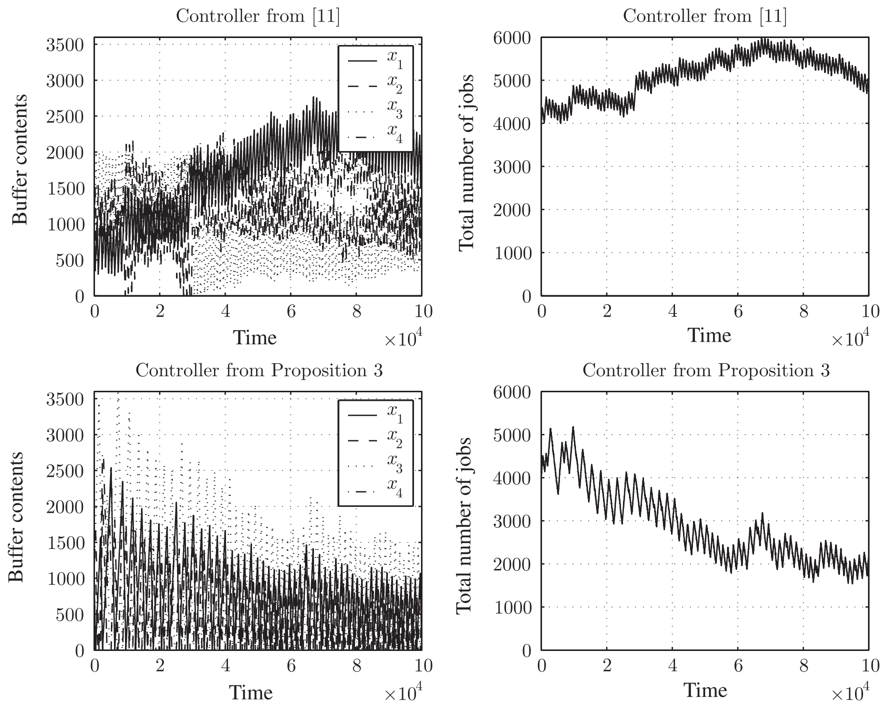

Fig. 5. The buffer contents and total number of jobs for a stochastic system initiated in $(x_1, x_2, x_3, x_4) = (1000, 1000, 1000, 1000)$ for both the distributed controller proposed in [11] and the distributed controller of Proposition 3.

the number of jobs in the system between 1500 and 2500.

VII. CONCLUSIONS

We considered the controller design for networks of switching servers with setup times, *e.g.*, manufacturing systems or urban road networks (traffic light control). Control of these networks is difficult, since using controllers that are stable for a server in isolation might render the network unstable. So far, in literature, most people first propose a policy, and then study the resulting behavior of the network under this policy.

In this paper we propose an entirely different way of looking at the problem of controlling a network of switching servers with setup times. Instead of starting from a policy and then analyzing the proposed policy, we start from *a priori* specified desired network behavior. Using this desired behavior for the network under consideration as a starting point, we *derive* a policy which guarantees convergence of the system towards this desired behavior. Though the policy is tailor-made for both the network under consideration and its desired behavior, we aim for a general methodology that is applicable to arbitrary networks and arbitrary feasible network behavior.

As an illustration we considered the reentrant system introduced by Kumar and Seidman. Since we are interested in optimal network behavior, instead of proposing a control policy for this reentrant network and analyzing the resulting network behavior, we first determined the optimal behavior for this network and then determined a policy which guarantees convergence towards this optimal behavior.

The resulting controller was a non-distributed controller, *i.e.*, each machine needs to have global information for determining when to switch. Since we are in a manufacturing setting, this can easily be implemented. However, it turned out that this non-distributed controller can be implemented in a distributed way, *i.e.*, each machine only requires local information for determining when to switch. We showed that these distributed controllers also guarantee convergence of the system towards the desired behavior. The proposed distributed controllers have been implemented successfully in a discrete event simulation study containing stochasticity.

Most of the literature on distributed control on networks starts from a policy which works for an arbitrary network and then analysis its performance.

This is a good approach for networks that are unknown or frequently subject to change. However, for networks that are fully known, and not subject to change, a different approach should be used. In this paper, we showed that it is possible to come up with distributed controllers that have been designed for a particular network in order to arrive at optimal network behavior. The resulting distributed controllers are different for each server and take into account knowledge of the network topology. Furthermore, the controllers do not fit in the class of standard controllers, such as clearing, gated, or k-limited policies.

The results in this paper show that tailor-made design of distributed controllers is possible, but still a lot of work remains to be done. An important question that we are currently addressing relates to the notion of observability: under what conditions is it possible for a workstation to reconstruct the global network state based on local observations only. As it currently seems, this observability is mainly possible for non-acyclic networks. For acyclic networks the upstream servers are not able to reconstruct the state at the downstream servers. Fortunately, as mentioned in the introduction, distributed policies usually have no difficulty in stabilizing the system for acyclic networks. However, non-acyclic networks are precisely the ones for which distributed controllers have difficulties in stabilizing the system. Using this notion of observability one should be able to come up with new distributed controllers that both stabilize the network and achieve desired network behavior, as shown in this paper by means of an example.

REFERENCES

1. Banks, J. and J. G. Dai, "Simulation studies of multiclass queueing networks," *IIE Trans.*, Vol. 29, pp. 213–219 (1997).
2. Eekelen, J. A. W. M. v., E. Lefeber and J. E. Rooda, "Feedback control of 2-product server with setups and bounded buffers," *Proc. Amer. Contr. Conf.*, Minneapolis, MN, USA (2006).
3. Humes, C., Jr, "A regulator stabilization technique: Kumar Seidman revisited," *IEEE Trans. Automat. Contr.*, Vol. 39, No. 1, pp. 191–196 (1994).
4. Kumar, P. R. and T. I. Seidman, "Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems," *IEEE Trans. Automat. Contr.*, Vol. 35, No. 3, pp. 289–298 (1990).
5. Lan, W. M. and T. L. Olsen, "Multiproduct systems with both setup times and costs: Fluid bounds and schedules," *Operations Research*, Vol. 54, No. 3, pp. 505–522 (2006).
6. Lefeber, E. and J. E. Rooda, "Control of a reentrant manufacturing system with setup times: The Kumar-Seidman case," SE Report 2006-04, Eindhoven University of Technology, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven, The Netherlands (2006), Available at http://se.wtb.tue.nl/sereports.
7. Lefeber, E. and J. E. Rooda, "Controller design of switched linear systems with setups," *Physica A*, Vol. 363, No. 1 (2006).
8. Perkins, J. and P. R. Kumar, "Stable, distributed, real-time scheduling of flexible manufacturing/assembly/disassembly systems," *IEEE Trans. Automat. Contr.*, Vol. 34, No. 2, pp. 139–148 (1989).
9. Perkins, J. R., C. Humes, Jr and P. R. Kumar, "Distributed scheduling of flexible manufacturing systems: Stability and performance," *IEEE Trans. Robot. Automat.*, Vol. 10, No. 2, pp. 133–141 (1994).
10. Savkin A. V., "Regularizability of complex switched server queueing networks modelled as hybrid dynamical systems," *Syst. Contr. Lett.*, Vol. 35, pp. 291–299 (1998).
11. Savkin, A. V., "Optimal distributed real-time scheduling of flexible manufacturing networks modeled as hybrid dynamical systems," *Proc. 42nd Conf. Decis. Contr.*, Honolulu, Hawaii, USA, pp. 5468–5471 (2003).
12. Takagi, H., "Analysis and application of polling models," in: G. Haring, C. Lindemann and M. Reiser, Eds, *Performance Evaluation*: *Origins and Directions*, Springer: Berlin, Vol. 1769 of *Lecture Notes in Computer Science*, pp. 423–442 (2000).

Erjen Lefeber received the M.Sc. degree in applied mathematics in 1996 and the Ph.D. degree in tracking control of nonlinear mechanical systems in 2000, both from the University of Twente, Enschede, The Netherlands. Since 2000, he has been an assistant professor in the Systems Engineering Group of the Department of Mechanical Engineering at Eindhoven University of Technology, Eindhoven, The Netherlands. His current research interests include modeling and control of manufacturing systems.

J.E. Rooda received the M.Sc. degree from the Wageningen University of Agriculture Engineering and the Ph.D. degree from the Twente University of Technology, The Netherlands. Since 1985 he is professor of (Manufacturing) Systems Engineering at the Department of Mechanical Engineering of Eindhoven University of Technology, The Netherlands. His research fields of interest are modeling and analysis of manufacturing systems. His interest is especially in control of manufacturing lines and in supervisory control of manufacturing machines.