5th Annual IEEE Conference on Automation Science and Engineering
Bangalore, India, August 22-25, 2009

Scalable scheduling algorithms for wireless networked control systems

Alessandro D'Innocenzo[1,2], Gera Weiss[1], Rajeev Alur[1], Alf J. Isaksson[4], Karl H. Johansson[3], George J. Pappas[1]

Abstract— **In this paper, we address the problem of designing scalable scheduling and routing policies over a time-triggered multi-hop control network, when closing a considerable number of control loops on the same network. The key idea is to formally define by means of regular languages the set of schedules for each control loop that satisfy a given control specification, and to exploit operators on regular languages to compute the set of schedules for the whole system. In order to test our methodology, we address a mineral floatation control problem derived from the Boliden (a swedish mining company) mine in Garpenberg, and propose a scheduling solution that can be implemented on systems compliant with communication protocols for wireless networks (e.g. the WirelessHART specification).**

I. Introduction

Wireless networked control systems are spatially distributed control systems where the communication between sensors, actuators, and computational units is supported by a shared wireless communication network. The use of wireless networked control systems in industrial automation results in flexible architectures and generally reduces installation and maintenance costs with respect to wired networks. Wide deployment of wireless industrial automation requires substantial progress in wireless transmission, networking and control, in order to provide formal tools to quantify performance and robustness of a wireless networked control system. The design of the control system has to take into account the presence of the network, as it represents the interconnection between the plant and the controller, and thus affects the dynamic behavior of the closed loop system. Using a wireless communication medium, new issues such as fading and time-varying throughput in communication channels have to be addressed, and communication delays and packet losses may occur. Moreover analysis of stability, performance, and reliability of real implementations of networked control systems requires addressing issues such as scheduling and routing for real communication protocols.

On this line of research, we proposed in [1] a mathematical framework inspired by the WirelessHART specification [2], for modeling and analyzing multi-hop wireless networked control systems. The framework is designed for systems consisting of multiple control loops closed over a multi-hop communication network. In this paper, we address the problem of designing scalable scheduling and routing policies when closing a considerable number of control loops on the same communication network. The key idea is to formally define by means of regular languages the set of schedules for each control loop that satisfy a given control specification, and to exploit operators on regular languages to compute the set of schedules for the whole system.

While our previous work (c.f. [1]) is focused on modeling the dynamics of the control loops as switched linear systems and analyzing stability of these systems, in this paper we propose an approach based on task-graph abstraction [3], [4], [5]. The main difference between our work and other studies of task-graph abstractions is that we focus on finding the set of all schedules that satisfy the task-graph constraints as a basis for further analysis, while most of the research is focused on finding individual optimal schedules (see e.g. [3]).

In order to test our methodology, we address a mineral floatation control problem derived from the Boliden (a swedish mining industry) mine in Garpenberg, and propose a scheduling solution that can be implemented on systems compliant with communication protocols for wireless networks (e.g. the WirelessHART specification).

The paper is organized as follows. In Section II we recall the definition of Networked Control System proposed in [1] as a mathematical framework for modeling and analysis of control, topology, routing, and scheduling on multi-hop communication networks, and propose modeling sets of periodic schedules using Deterministic Finite Automata (DFA) and regular languages. In Section III we propose an algorithm for constructing a DFA that accepts the language of admissible schedules for each control loop, and an algorithm for composing the DFAs that accept the language of admissible schedules of all control loop, in order to compute a DFA that accepts the admissible schedules of all loops. In Section IV we propose a case study on mineral processing to test the methodology and algorithms developed in Section III, and illustrate the simulation results in Section V.

II. Formal definition of Networked Control Systems and scheduling policies

In this section we first recall the definition of Networked Control System proposed in [1] as a mathematical framework for modeling and analysis of control, topology, routing, and scheduling on multi-hop communication networks. Then, we propose modeling sets of scheduling policies using Deterministic Finite Automata (DFA) and regular languages.

Definition 1: A Networked Control System (NCS) is a tuple $\mathscr{N} = \langle \mathscr{D}, \mathscr{G}, \mathbb{O}, \mathbb{I}, \omega, \mathscr{R} \rangle$, where:

- $\mathscr{D} = \{\mathscr{D}_i\}_{i=1}^{p} = \{\langle\langle A_i, B_i, C_i\rangle, \langle \tilde{A}_i, \tilde{B}_i, \tilde{C}_i\rangle\rangle\}_{i=1}^{p}$ models the control loops. Each control loop in $\mathscr{D}$ is modeled by a pair of triplets of matrices. The first triplet in each pair defines the dynamics of the plant and the second

[1]University of Pennsylvania, Philadelphia PA. [2]University of L'Aquila, Italy. [3]Royal Institute of Technology, Stockholm, Sweden. [4] ABB, Vasterås, Sweden.

triplet defines the dynamics of the control algorithm, both in terms of matrices of Linear Time Invariant (LTI) systems. The number of columns in B_i must be the same as the number of rows in $\tilde{C}_i$, which is the number of inputs to the plant. Similarly, the number of rows in C_i must be the same as the number of columns in $\tilde{B}_i$, which is the number of measurable outputs from the plant. Let $\mathbb{I} = \cup_{i=1}^{p} \{y_{i,1} \ldots, y_{i,n_i}\}$ be the set of input signals for the plants, where n_i is the number of columns in B_i (rows in $\tilde{C}_i$). Let $\mathbb{O} = \cup_{i=1}^{p} \{u_{i,1} \ldots, u_{i,m_i}\}$ be the set of output signals from the plants, where m_i is the number of rows in C_i (columns in $\tilde{B}_i$). The matrices of the controller induce a switched system with two operation modes by $\tilde{A}_i(Active) := \tilde{A}_i, \tilde{B}_i(Active) := \tilde{B}_i, \tilde{C}_i(Active) := \tilde{C}_i, \tilde{A}_i(Idle) := \mathbf{1}$ (identity matrix) , $\tilde{B}_i(Idle) := \mathbf{0}$ (zero matrix) and $\tilde{C}(Idle) := \tilde{C}$. The *Idle* mode corresponds to times when the controller is inactive and the *Active* mode models times where the controller applies a transformation of its state and computes a new control command.

- $\mathscr{G} = \langle V, E \rangle$ is a directed graph that models the radio connectivity of the network, where vertices are nodes of the network, and an edge from v_1 to v_2 means that v_2 can receive messages transmitted by v_1. We denote with v_c the special node of V that corresponds to the controller. Let $\mathscr{P}$ be the set of simple paths in $\mathscr{G}$ that start or end with the controller;
- $\omega \colon \mathbb{I} \cup \mathbb{O} \to V$ assigns to every input and output signal the node that implements, respectively, sensing or actuation;
- $\mathscr{R} \colon \mathbb{I} \cup \mathbb{O} \to 2^{\mathscr{P}}$ is a map, which associates to each input (resp. output) signal a set of allowed simple paths from (resp. to) the controller. We require that all elements in $\mathscr{R}(y)$ (resp. $\mathscr{R}(u)$) start (resp. end) with $\omega(y)$ (resp. $\omega(u)$) and end (resp. start) with the controller, for all $y \in \mathbb{I}$ (resp. $u \in \mathbb{O}$).

To define the scheduling of this system we construct the memory slots graph which is obtained by splitting every node in the connectivity graph, as follows. The nodes of the memory slots graph are pairs $\langle v, s \rangle$ where $v \in V$ is a node in the connectivity graph and $s \in \mathbb{I} \cup \mathbb{O}$ is a signal (input or output). The nodes $\langle v_1, s_1 \rangle$ and $\langle v_2, s_2 \rangle$ are connected iff $\langle v_1, v_2 \rangle \in E$ and $s_1 = s_2$. This graph models the memory slots reserved to each signal in every physical node. Edges model the ability to copy data from a slot to another (when the physical nodes are communicating).

Definition 2: Given an NCS $\mathscr{N} = \langle \mathscr{D}, \mathscr{G}, \mathbb{O}, \mathbb{I}, \omega, \mathscr{R} \rangle$ we define a communication and a computation schedule as a tuple $\langle \eta, \mu \rangle$, where:

- A communication schedule is a function $\eta \colon \mathbb{N} \to 2^{E \times (\mathbb{I} \cup \mathbb{O})}$. The intended meaning of this schedule is that $\langle \langle v_1, v_2 \rangle, s \rangle \in \eta(t)$ iff at time t the data related to the signal s in v_1 is copied to the space reserved for the data related to s in v_2. We require that if $\langle \langle v_1, v_2 \rangle, s \rangle \in \eta(t)$ then for every $v_3 \neq v_1$, $\langle \langle v_3, v_2 \rangle, s \rangle \notin \eta(t)$. Namely, we do not allow assignment of two values to the same memory slot.
- A computation schedule for the ith control loop (corresponding to $\mathscr{D}_i$) is a function $\mu_i \colon \mathbb{N} \to \{Idle, Compute\}$. The meaning of this function is that $\mu_i(t)$ defines the mode of the controller at time t.

In order to formally define sets of communication-computation schedules and apply compositional operators, we use the formalism of Deterministic Finite Automata and regular languages. A Deterministic Finite Automaton (DFA) is a tuple $\mathscr{F} = \langle Q, \Sigma, q^0, Q^F, \delta \rangle$, where Q is a finite set of states, Σ is a finite alphabet, $q_0 \in Q$ is the initial state, $Q_f \subseteq Q$ is the set of final (accepting) states, and δ is the transition function. A set (language) $\mathscr{L} \subseteq \Sigma^*$ is called a *regular language* if there exists a DFA that accepts all and only the strings in $\mathscr{L}$. For more details, the reader is referred to [6].

Given a NCS $\mathscr{N}$, let $\Sigma = 2^{E \times (\mathbb{I} \cup \mathbb{O})} \cup \{Idle\}$: any regular language over the alphabet Σ defines a set of schedules for $\mathscr{N}$. Regular languages allow modeling of a large class of scheduling policies including periodic schedules, which are mandatory in usual time-triggered protocols such as WirelessHART (see [7] for more details).

III. Algorithms for scheduling design

Given an NCS $\mathscr{N}$ consisting of p control loops, we propose one algorithm to construct a DFA that accepts the language of admissible schedules for each control loop, and one algorithm to compose the DFAs that accept the language of admissible schedules of each control loop, in order to define a DFA that accepts the admissible schedules of all loops.

The first algorithm takes as input an NCS $\mathscr{N} = \langle \mathscr{D}, \mathscr{G}, \mathbb{O}, \mathbb{I}, \omega, \mathscr{R} \rangle$ and generates as output, for each control loop $\mathscr{D}_i$, a deterministic finite automaton $\mathscr{F}_i$ that accepts a regular language of allowed schedules for the control loop $\mathscr{D}_i$, that transmit all sensor data to the controller, and all control commands to the actuators.

Given a NCS $\mathscr{N}$, the associated routing map $\mathscr{R}$, and any node $v \in V$, we define the graph $\mathscr{R}_v = \langle V_v, E_v \rangle$ which models the set of all paths that can be used to reach the destination node v from any other node.

Algorithm 1: Given an NCS $\mathscr{N} = \langle \{\mathscr{D}_i\}, \mathscr{G}, \mathbb{O}, \mathbb{I}, \omega, \mathscr{R} \rangle$, we define for each $j \in \mathbb{O} \cup \mathbb{I}$ a deterministic finite automaton $\mathscr{F}_j = \langle Q_j, \Sigma_j, q_j^0, Q_j^F, \delta_j \rangle$ as follows:

$$Q_j = \begin{cases} V_{\omega(j)} & \text{if } j \in \mathbb{I} \\ V_c & \text{if } j \in \mathbb{O} \end{cases}$$

$$\Sigma_j = \begin{cases} E_{\omega(j)} \cup \{Idle\} & \text{if } j \in \mathbb{I} \\ E_c \cup \{Idle\} & \text{if } j \in \mathbb{O} \end{cases}$$

$$q_j^0 = \begin{cases} \omega(j) & \text{if } j \in \mathbb{O} \\ c & \text{if } j \in \mathbb{I} \end{cases}$$

$$Q_j^F = \begin{cases} \{\omega(j)\} & \text{if } j \in \mathbb{I} \\ \{c\} & \text{if } j \in \mathbb{O} \end{cases}$$

We define δ_j as follows:

$$\forall e = \langle v, v' \rangle \in \Sigma_j \setminus \{Idle\}, \delta_j(v, e) = v',$$
$$\forall v \in Q_j, \delta_j(v, Idle) = v.$$

Let $|\mathbb{O}| = m$ and $|\mathbb{I}| = n$. Given $\{\mathscr{F}_j\}_{j=1}^{m+n}$, define a deterministic finite automaton $\mathscr{F} = \langle Q, \Sigma, q^0, Q^F, \delta \rangle$ as follows:

$$Q = Q_1 \times \cdots \times Q_{m+n} \times \{0,1\}$$
$$\Sigma = E_1 \cup \cdots \cup E_{m+n}$$
$$q^0 = \langle q_1^0, \ldots, q_{m+n}^0, 0 \rangle$$
$$Q^F = Q_1^F \times \cdots \times Q_{m+n}^F \times \{1\}$$

We remark that given a state $q = \langle q_1, \ldots, q_{m+n}, q_{m+n+1} \rangle \in Q$ we interpret each component as follows. The first m components correspond to a state $q \in Q_j$, with $j \in \mathbb{O}$, and model that the measurement performed by the node $\omega(j)$ is currently stored in node q. The following n components correspond to a state $q \in Q_j$, with $j \in \mathbb{I}$, and model that the control command destined to the node $\omega(j)$ is currently stored in node q. The last component q_{m+n+1} is 1 if the control command for the control loop has been computed, and is 0 otherwise.

For $\sigma \in \Sigma, q = \langle q_1, \ldots, q_{m+n+1} \rangle \in Q$, let

$$\delta(q, \sigma) = \langle \delta|_1(q_1, \sigma), \ldots, \delta|_{m+n+1}(q_{m+n+1}, \sigma) \rangle.$$

where δ is defined as follows.

$\forall e \in \Sigma \setminus \{Idle\}$,

$$\delta|_j(q_j, e) = \begin{cases} \delta_j(q_j, e) & \text{if } (0 \le j \le m \text{ and } q_{m+n+1} = 0) \\ & \text{or} \\ & (m+1 \le j \le m+n \text{ and } q_{m+n+1} = 1) \\ q_j & \text{otherwise} \end{cases}$$

$$\delta(\langle \underbrace{c, \ldots, c}_{m+n}, 0 \rangle, Idle) = \langle \underbrace{c, \ldots, c}_{m+n}, 1 \rangle$$

$\forall q \in Q, \delta(q, Idle) = q$

Iterating the above algorithm for each control loop $\mathscr{D}_i \in \mathscr{D}$, we generate a set of deterministic finite automata $\{\mathscr{S}_i\}_{i=1}^{p}$.

Proposition 1: The cardinality of the state space Q_i of $\mathscr{S}_i$ is upper bounded by $\prod_{j=1}^{m} |Q_j| + \prod_{j=m+1}^{m+n} |Q_j|$.

The second algorithm takes as input the set of deterministic finite automata $\{\mathscr{S}_i\}_{i=1}^{p}$, and generates as output a deterministic finite automaton $\mathscr{S}$ that accepts the set of allowed schedules for all control loops.

Algorithm 2: Given a NCS $\mathscr{N} = \langle \{\mathscr{D}_i\}_{i=1}^{p}, \mathscr{G}, \mathbb{O}, \mathbb{I}, \omega, \mathscr{R} \rangle$, and a set of deterministic finite automata $\{\mathscr{S}_i\}_{i=1}^{p}, \mathscr{S}_i = \langle Q_i, \Sigma_i, q_i^0, Q_i^F, \delta_i \rangle$ generated by Algorithm 1, define a deterministic finite automaton $\mathscr{S} = \langle Q, \Sigma, q^0, Q^F, \delta \rangle$ as follows:

$$Q = Q_1 \times \cdots Q_p$$
$$\Sigma = E \cup \{Idle\}$$
$$q^0 = \langle q_1^0, \ldots, q_p^0 \rangle$$
$$Q^F = Q_1^F \times \cdots \times Q_p^F$$

For any $\sigma \in \Sigma$ and $q = \langle q_1, \ldots, q_p \rangle \in Q$, we define δ as follows:

$$\delta(q, \sigma) = \langle \delta_1(q_1, \sigma), \ldots, \delta_p(q_p, \sigma) \rangle.$$

Proposition 2: The cardinality of the state space Q of $\mathscr{S}$ is upper bounded by $\prod_{i=1}^{p} |Q_i| = \prod_{i=1}^{p} \left(\prod_{j=1}^{m} |Q_j| + \prod_{j=m+1}^{m+n} |Q_j| \right)$.

IV. Mineral floatation control problem

In this section, we propose a case study on mineral processing to test the methodology and algorithms developed in the previous section. Mineral processing of ores is performed to recover minerals or metal from the extracted raw ore. Processes have to be optimized to yield an acceptable purity of the recovered mineral or metal. The main steps involved in mineral and metal recovery from the ore are size reduction, concentration of the pulp and de-watering. Fore more details on mineral processing we refer to [8], [9] and references therein. The general approach in the mineral processing is to use several consecutive flotation cells to form a flotation bank. As illustrated in Figure 1, the flotation cell is a tank

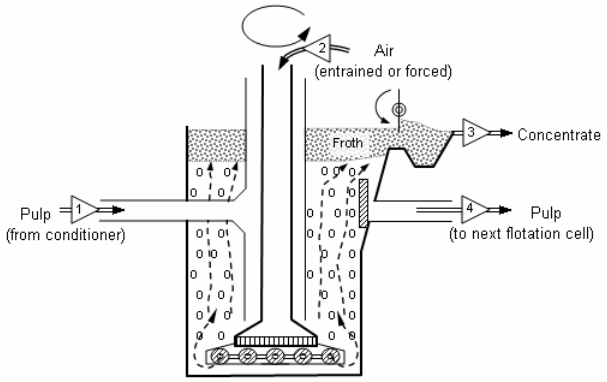

Fig. 1. Diagram of froth flotation cell [8].

with a pulp feed, an outlet, and froth launders to recover the concentrate. The separation of minerals in froth flotation depends primarily on the differences in the hydrophobicity of the particles, as they must selectively attached to air bubbles to be floated. Some minerals can be directly floated, but in most cases reagents have to be added to make the flotation process possible.

The process plant of the Boliden mine in Garpenberg is designed to produce four concentrates: zinc, copper, lead and precious metals. In this paper we consider the zinc flotation process, as the zinc is the most important metal extracted from the Garpenberg mine.

The controlled variables are listed in Table I, where T_s is the sampling interval. Each controlled variable represents a control loop, i.e. the number of control loops is equal to the number of controlled variables. In this work, we will only consider the main control loops, i.e. *air flow*, *pulp level* and *reagent*. We abstract each control loop by a time constraint, which specifies the maximum delay between sensing and actuation. For each control loop illustrated in Table I, we will set this constraint equal to the corresponding sampling interval T_s. In the following section, we will propose a scalable methodology to design scheduling policies that allows data transmission from the sensors to the controller

TABLE I

CONTROLLED VARIABLES FOR THE GARPENBERG PLANT (FROM [10]).

LOOP CATEGORY	Number of loops	Loop name	T_s
AIR FLOW	9	FA301_FC1	2
		FA302_FC1	2
		FA303_FC1	2
		FA304_FC1	2
		FA305_FC1	2
		FA101_FC1	2
		FA102_FC1	2
		FA103_FC1	2
		FA104_FC1	2
PULP LEVEL	6	FA302_LC1	2
		FA303_LC1	1
		FA305_LC1	8
		FA102_LC1	8
		FA103_LC1	8
		FA104_LC1	8
REAGENTS	2	BL031_FC1	2
		FA300_FC2	1

and from the controller to the actuators for each control loop, within the corresponding sampling interval T_s. The main novelty of our approach is that we focus on finding the set of all schedules that satisfy the constraints for one control loop (e.g. the sampling interval T_s) as a basis for on-line or off-line admission of further control loops, instead of finding individual (optimal) schedules.

V. A MATHEMATICA NOTEBOOK FOR SCHEDULING THE MINERAL FLOATATION NETWORKED CONTROL SYSTEM

To experiment with the algorithms developed in Section III, we extended the Mathematica based tool described in [1]. With the extension, the tool supports specifications of the form "the data from all sensors has to be sent to the controller, then a computation of the control signals is carried, and after that all the control signals are sent to the actuators". This requirement needs to be satisfied in each period of the schedule, and the designer can also specify an upper time bound for the round trip to complete.

The tool translates the requirement to an automaton based on the algorithms described in Section III. Once the set of schedules is specified by an automaton, one can extract schedules that satisfy the requirement (corresponding to paths from the initial state to a final state). The main advantage of using automata to represent sets of schedules is that this representation allows operation such as intersection, concatenations and union. For example, we can intersect the requirements of one control loop with the requirements of another loop. Automata based representations can be used to analyze combinations of constraints including timing (as studied in this paper), stability (as studied in [11]), and periodic requirements (as discussed in [1]).

We continue with a more technical description of the tool extension we developed for this study.

We propose to use symbolic representation of automata because direct explicit application of Algorithm 1 and Algorithm 2 have potential scalability limitations. Specifically, as stated in Proposition 2, the number of states in the product automaton grows exponentially with the number of control loops. For example, with the seventeen loops present in the Mineral Flotation Control case study, the computation of the product automaton is not practical with direct application of current versions of tools such as GOAL [12], JFLAP [13], and SPIN [14].

Symbolic algorithms avoid building the automaton explicitly; instead, they maintain a compact representation of its transition relation. For example, we use the NuSMV symbolic model checker [15] which combines Binary Decision Diagrams (BDDs) [16] and SAT based model checking [17]. The tool allows verification of Temporal Logic [18] properties of a transition system expressed in the SMV input language [15]. Specifically, for each control loop, we use Algorithm 1 to generate a module in the SMV modeling language representing the transition relation of the automaton $\mathscr{F}$ described in Section III. Then, we apply the compositional semantics of the SMV modeling language to combine the modules according to the semantics described in Algorithm 2.

A. Example

Before we discuss the case-study itself, let us begin with a simple example that shows how the tool is used and demonstrates the translation. For this example, consider the topology depicted in Figure 2. The SMV code generated for this example is listed in Figure 3.

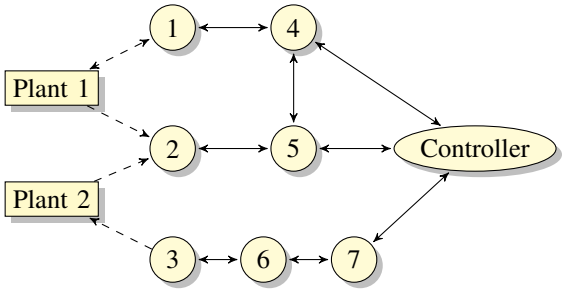

Fig. 2. An example of a multi-hop control network. Circles represent nodes with wireless communication capabilities, solid lines represent radio connectivity and dashed lines represent actuation/sensing. Specifically, node 1 is both a sensor and an actuator of Plant 1, node 2 is a sensor of Plant 1 and of Plant 2 and node 3 is an actuator of Plant 2.

The code consists of three sections, as follows. The first section corresponds to the constraints imposed by the second control loop (Plant 2), the second section models the constraints imposed by the first control loop (Plant 1), and the last section contains the main module defining the shared variables and the temporal logic property that SMV is asked to dispute. A more detailed look into this code follows.

Looking more closely into the first section of the code, one can see that it specifies a partial order of assignment to the shared variable `bus`. More specifically, the flag `done` can only be positive if there are times $t_5 \geq t_4 \geq t_3 > t_2 \geq t_1$ such that `bus=e2to5` at time t_1, `bus=e5toc` at time t_2, `bus=ecto7` at time t_4, `bus=e7to6` at time t_5, and `bus=e6to3` at time t_6. The idea is that an assignment of the form `bus=e`i`to`j at time t models that the bus is scheduled to send a message from node i to node j at the tth slot of the schedule. In particular, all sequences of assignments to the variable `bus` that are consistent with the first module (called

```
MODULE loop2(bus)
VAR
    cnt:0..6;
ASSIGN
    init(cnt):=0;
    next(cnt):=case
        bus=e2to5 & cnt=0 : 1;
        bus=e5toc & cnt=1 : 2;
        bus=bus   & cnt=2 : 3;
        bus=ecto7 & cnt=3 : 4;
        bus=e7to6 & cnt=4 : 5;
        bus=e6to3 & cnt=5 : 6;
        1:cnt;
    esac;
DEFINE
       done := cnt=6;

--------------------------------------
MODULE loop1(bus)
VAR
    in1:0..2;
    in2:0..2;
    out1:0..3;
ASSIGN
    init(in1):=0;
    init(in2):=0;
    init(out1):=0;

    next(in1):=case
        bus=e1to4 & in1=0 : 1;
        bus=e4toc & in1=1 : 2;
        1:in1;
    esac;

    next(in2):=case
        bus=e2to5 & in2=0 : 1;
        bus=e5toc & in2=1 : 2;
        1:in2;
    esac;

    next(out1):=case
        bus=bus   &  allin & out1= 0 : 1;
        bus=ecto4 & allin  & out1= 1 : 2;
        bus=e4to1 & allin  & out1= 2 : 3;
        1 : out1;
    esac;
DEFINE
       allin := in1=2 & in2=2;
       done  := out1=3;

--------------------------------------
MODULE main
VAR
    bus:{e1to4, e2to5, e4to1, e4toc, e5toc, e6to3,
         e7to6, ecto4, ecto7, Idle};
    l1:loop1(bus);
    l2:loop2(bus);
SPEC
    AG !(l1.done & l2.done);
```

Fig. 3. SMV code generated for the system depicted in Figure 2.

loop2) correspond to schedules in which data is sent from node 2 (the only sensor of Plant 2) to node 5 and later on from node 5 to the controller and so on. Note that t_3 must be strictly grater than t_2 because we want to allow time for the computation of the feedback signal. This corresponds to the line `bus=bus & cnt=2 : 3;` in the SMV code.

The second section of the code is similar in nature to the first one. However, we have to be more careful because Plant 1 has two sensors. In the case of multiple sensors, we have to make sure that all inputs are sent to the controller before the computation of feedback signal is carried (more than one step before starting to send messages towards the actuator). This is achieved by defining the flag `allin` which becomes positive only after both inputs are sent to the controller and conditioning the run of the counter `out1` on that. This assures that the flag `done` becomes positive only if the schedule contains a sequence of messages that allows an update of the actuator based on data from both sensors.

Note that each module in the SMV code corresponds to the automaton $\mathscr{F}$ described in Section III. Specifically, the state of the automaton $\mathscr{F}_j$, capturing the node that contains the recent value of the signal j, is modeled by the variables `in1`,...,`in`n and `out1`,...,`out`m that count how much of the routing path of the signal is executed (e.g. the state of $\mathscr{F}_{y_1}$ is the `in1`'th element of `routing[`y_1`]`) .

The last section of the code defines the shared variable `bus`. As mentioned above, sequences of assignments to this variable correspond to schedules. Instantiating copies of `loop1` and of `loop2`, with this variable passed as a parameter, guarantees that only schedules that are consistent with the constraints posed by the two control loops are possible. Then, we ask NuSMV to refute the temporal logic property `AG !(l1.done & l2.done)` which says that no schedule can be consistent with the constraints posed by both loops. In the case of the example that we are considering, since there are schedules that are consistent with both loops, NuSMV spits a counter example which is a sequence of assignments to `bus`. This sequence of assignments can be interpreted as a schedule that allows, in every cycle, to collect data from all sensors, carry the computations, and send commands to all actuators.

B. Case Study

We continue by summarizing our experience with application of the technique, demonstrated by the above example, to the mineral floatation networked control system.

The input to the tool is a textual description of the graph depicted in Figure 4. As illustrated in the figure, the network has three layers. Each node in each layer is connected to all the nodes in the layer below it. Only the nodes in the third layer can communicate with the controller. All communication links in this case-study are bidirectional. Each wireless node is both a sensor and an actuator of a single-input-single-output plant.

Given a description of the above topology, the tool can generate an SMV code similar to the one shown in Figure 3. Note that, in principle, we also need to specify a routing path for each signal. In practice however, if a routing path is not explicitly specified, the tool automatically selects a minimal path from the sensor to the controller (for input signals) or from the controller to the actuator (for output signals).

The SMV code for the case-study contains seventeen modules, one for each control loop, and a main module. Since each control loop has one input and one output, all seventeen modules are similar to the module called `loop2` in the listing depicted in Figure 3. It takes NuSMV about two minutes (on 2 GHz Intel Core Duo with 1 GB of RAM memory MacBook 1.1) to dispute the claim that there is no schedule that is compatible with all seventeen loops and to produce a counter example from which a valid schedule can be extracted. If we want to look for the shortest period

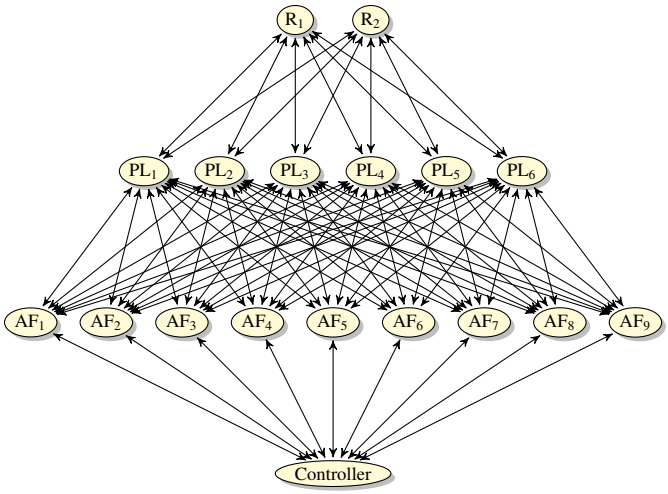

Fig. 4. Topology of wireless nodes. R is a shorthand for REAGENTS, PL is a shorthand for PULP LEVEL, and AF is a shorthand for AIR FLOW. Each node is both an actuator and a sensor of the respective plant. Plants are not drawn to avoid cluttering.

with this property, we can add a counter in the main module that counts the length of the schedule. Once such a counter is defined, we can add the requirement that the schedule is smaller than a constant. Starting with the length of the original schedule, we can use binary search to find the minimal such constant. This gives us the shortest admissible schedule. A similar procedure can also be used to optimize other properties of the schedule.

C. Input Language

As discussed above, the SMV code is generated automatically. For this paper, we developed a Mathematica notebook that takes a description of the wireless control network and produces the SMV code. For example, the input from which the SMV code in Figure 3 is generated is displayed in Figure 5. The specification consists of: (1) A specification of the nodes that act as sensors and actuators to the loop; (2) A description of the connectivity graph; (4) An assignment of routing paths to the input and output signals. Note that the routing paths are modeled as lists of lists because, in general, we may allow more than one possible routing per signal. Multiple paths are not used in the example nor in the case-study, but are supported by the tool. One way to use this feature is to allow all paths in the graph and let SMV choose the best combination of routing paths automatically.

VI. Conclusions and Future Work

We developed tools for designing scalable scheduling and routing policies for time-triggered multi-hop control networks and applied them to a case study. Future research directions include extending the tools to support richer specifications (e.g. allow multiple routing), incorporating optimization techniques (e.g. the algorithm described in [19]), and apply the results obtained in [1] and in [20] to the automata describing the sets of allowed schedules.

$$\text{numOfLoops} = 2;$$

$$\text{sensors}[1] = \{1,2\};$$
$$\text{actuators}[1] = \{1\};$$
$$\text{sensors}[2] = \{2\};$$
$$\text{actuators}[2] = \{3\};$$

$$G = \text{sym}[\{1 \to 4, 4 \to 8, 8 \to c, 2 \to 5, 5 \to c, 3 \to 6, 6 \to 7, 7 \to c\}];$$

$$\text{routing}[y_{1,1}] := \{\{1,4,c\}\};$$
$$\text{routing}[y_{1,2}] := \{\{2,5,c\}\};$$
$$\text{routing}[y_{2,1}] := \{\{2,5,c\}\};$$
$$\text{routing}[u_{1,1}] := \{\{c,4,1\}\};$$
$$\text{routing}[u_{1,2}] := \{\{c,7,6,3\}\};$$

Fig. 5. Formal description of the wireless control network depicted in Figure 2. This text is used as an input to a Mathematica based tool that automatically generated the SMV code listed in Figure 3 from it.

References

[1] R. Alur, A. D'Innocenzo, K. Johansson, G. Pappas, and G. Weiss, "Modeling and analysis of multi-hop control networks," in *RTAS*, 2009.

[2] J. Song, S. Han, A. K. Mok, D. Chen, M. Lucas, M. Nixon, and W. Pratt, "WirelessHART: Applying wireless technology in real-time industrial process control," in *RTAS*, 2007.

[3] C. V. Ramamoorthy, K. M. Chandy, and M. J. Gonzalez, "Optimal scheduling strategies in a multiprocessor system," *IEEE Transactions on Computers*, vol. 21, no. 2, pp. 137–146, 1972.

[4] Y. K. Kwok and I. Ahmad, "Benchmarking the task graph scheduling algorithms," in *IPPS*, 1998.

[5] A. Bakshi, V. K. Prasanna, J. Reich, and D. Larner, "The abstract task graph: a methodology for architecture-independent programming of networked sensor systems," in *EESR*, 2005.

[6] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[7] "Tdma data link layer," HART Communication Foundation, HCF_SPEC-075 Revision 1.0, 2007.

[8] H. Lindvall, "Flotation modelling at the garpenberg concentrator using modelica/dymola," Ph.D. dissertation, Uppsala University, 2007.

[9] M. De Biasi, "Simulation of process control with WirelessHART networks subject to packet losses," Master's thesis, Royal Institute of Technology, 2008.

[10] V. Ercoli and G. Fiore, "Scheduling for wireless control in a WirelessHART network," Master's thesis, University of L'Aquila, Royal Institute of Technology, 2009.

[11] G. Weiss and R. Alur, "Automata based interfaces for control and scheduling," in *HSCC*, 2007.

[12] Y.-K. Tsay, Y.-F. Chen, M.-H. Tsai, W.-C. Chan, and C.-J. Luo, "Goal extended: Towards a research tool for omega automata and temporal logic," in *TACAS*, 2008.

[13] S. Rodger, *Jflap-an Interactive Formal Languages and Automata Package*. Boston: Jones and Bartlett, 2006.

[14] G. J. Holzmann, *The SPIN Model Checker : Primer and Reference Manual*. Addison-Wesley Professional, September 2003.

[15] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking," in *CAV*, 2002.

[16] R. E. Bryant, "Symbolic Boolean manipulation with ordered binary-decision diagrams," *ACM Computing Surveys*, vol. 24, no. 3, pp. 293–318, 1992.

[17] E. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Form. Methods Syst. Des.*, vol. 19, no. 1, pp. 7–34, July 2001.

[18] A. Pnueli and Z. Manna, *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag Gmbh, 1991.

[19] R. Alur, A. Kanade, and G. Weiss, "Ranking automata and games for prioritized requirements," in *CAV*, 2008.

[20] R. Alur and G. Weiss, "Regular specifications of resource requirements for embedded control software," in *RTAS*, 2008.