2011

Robust Architectures for Embedded Wireless Network Control and Actuation

Miroslav Pajic
University of Pennsylvania, pajic@seas.upenn.edu

Alexander Chernoguzov
Honeywell Process Solutions, alexander.chernoguzov@honeywell.com

Rahul Mangharam
University of Pennsylvania, rahulm@seas.upenn.edu

Robust Architectures for Embedded Wireless Network Control and Actuation

Abstract
Networked Cyber-Physical Systems are fundamentally constrained by the tight coupling and closed-loop control of physical processes. To address actuation in such closed-loop wireless control systems there is a strong need to re-think the communication architectures and protocols for reliability, coordination and control. We introduce the Embedded Virtual Machine (EVM), a programming abstraction where controller tasks with their control and timing properties are maintained across physical node boundaries and functionality is capable of migrating to the most competent set of physical controllers. In the context of process and discrete control, an EVM is the distributed runtime system that dynamically selects primary-backup sets of controllers given spatial and temporal constraints of the underlying wireless network. EVM-based algorithms allow network control algorithms to operate seamlessly over less reliable wireless networks with topological changes. They introduce new capabilities such as predictable outcomes during sensor/actuator failure, adaptation to mode changes and runtime optimization of resource consumption. An automated design flow from Simulink to platform-independent domain specific languages, and subsequently, to platform-dependent code generation is presented. Through case studies in discrete and process control we demonstrate the capabilities of EVM-based wireless network control systems.

Keywords
Distributed control systems, wireless sensor networks, fault tolerance

Disciplines
Electrical and Computer Engineering | Process Control and Systems

Comments
Suggested Citation:

M. Pajic, A. Chernoguzov and R. Mangharam, "Robust Architectures for Embedded Wireless Network Control and Actuation", *ACM Transactions on Embedded Computing Systems*, 2013

Robust Architectures for Embedded Wireless Network Control and Actuation

MIROSLAV PAJIC
University of Pennsylvania
ALEXANDER CHERNOGUZOV
Honeywell Process Solutions
and
RAHUL MANGHARAM
University of Pennsylvania

Networked Cyber-Physical Systems are fundamentally constrained by the tight coupling and closed-loop control of physical processes. To address actuation in such closed-loop wireless control systems there is a strong need to re-think the communication architectures and protocols for reliability, coordination and control. We introduce the Embedded Virtual Machine (EVM), a programming abstraction where controller tasks with their control and timing properties are maintained across physical node boundaries and functionality is capable of migrating to the most competent set of physical controllers. In the context of process and discrete control, an EVM is the distributed runtime system that dynamically selects primary-backup sets of controllers given spatial and temporal constraints of the underlying wireless network. EVM-based algorithms allow network control algorithms to operate seamlessly over less reliable wireless networks with topological changes. They introduce new capabilities such as predictable outcomes during sensor/actuator failure, adaptation to mode changes and runtime optimization of resource consumption. An automated design flow from Simulink to platform-independent domain specific languages, and subsequently, to platform-dependent code generation is presented. Through case studies in discrete and process control we demonstrate the capabilities of EVM-based wireless network control systems.

Categories and Subject Descriptors: C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Network communications*; *Wireless communication*; C.3 [**Special-Purpose and Application-Based Systems**]: Process control systems; Real-time and embedded systems

General Terms: Algorithms, Design, Reliability

Additional Key Words and Phrases: Distributed control systems, wireless sensor networks, fault tolerance

1. INTRODUCTION

Automation control systems form the basis for significant pieces of the US critical infrastructure. Time-critical and safety-critical automation systems are at the heart of essential infrastructures such as oil refineries, automated factories, logistics and power generation systems. To meet the reliability requirements, automation systems are traditionally severely constrained along three dimensions, namely, operating resources, scalability of interconnected systems and flexibility to mode changes. Oil refineries, for example, are built to operate without interruption for over 25 years and can never be

Authors' addresses: M. Pajic and R. Mangharam, Department of Electrical & Systems Engineering, University of Pennsylvania, 200 S.33rd Street, Philadelphia, PA 19104; email: {pajic, rahulm}@seas.upenn.edu; A. Chernoguzov email: alexander.chernoguzov@honeywell.com.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

shutdown for preventive maintenance or upgrades. They are built with rigid ranges of operating throughput and require a significant re-haul to adapt to changes in crude oil quality and market conditions. This rigidity has resulted in systems with limited scope for re-appropriation of resources during faults and retooling to match design changes on-demand. For example, automotive assembly lines lose an average of $22,000 per minute of downtime during system faults [Nielsen Research 2006]. This has created a culture where the operating engineer is forced to patch a faulty unit in an ad hoc manner which often necessitates masking certain sensor inputs to let the operation proceed. This process of unsystematic alteration to the system exacerbates the problem and makes the assembly line difficult and expensive to operate, maintain and modify.

Embedded Wireless Sensor-Actuator-Controller (WSAC) networks are emerging as a practical means to monitor and operate automation systems with lower setup/maintenance costs. While the physical benefits of wireless, in terms of cable replacement, are apparent, plant owners have increasing interest in the logical benefits. With multi-hop WSAC networks, it is possible to build *Wireless Plug-n-Play Automation Systems* which can be swapped in and efficiently reconnect hundreds of I/O lines. Such modular systems can be dynamically assigned to be primary or backup on the basis of available resources or availability of the desired calibration. Modularity allows for incremental expansion of the plant and is a major consideration in emerging economies. WSAC networks allow for runtime configuration where resources can be re-appropriated on-demand, for example when throughput targets change due to lower electricity price during off-peak hours or due to seasonal changes in end-to-end demand.

1.1. Challenges with Wireless Control

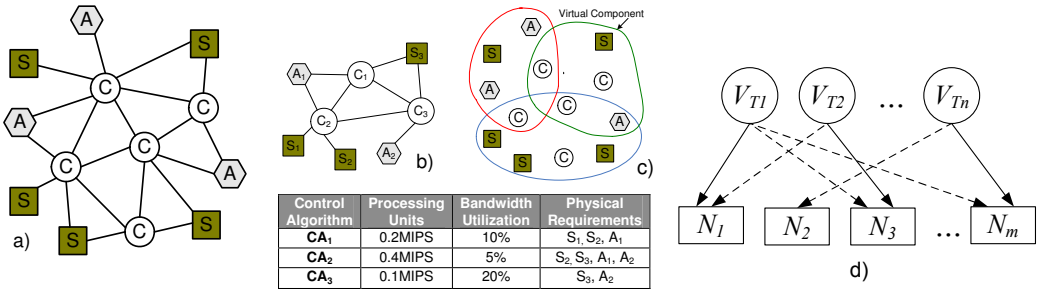
We identify four primary challenges with the design, analysis and deployment of WSAC networks:

1. The current approaches of programming motes in the event-triggered paradigm [Hill et al. 2000] are tedious for control networks. Time-triggered architectures are required as they naturally integrate communication, computation, and physical aspects of control networks [Kopetz and Bauer 2003], [Alur et al. 2009].
2. Programming of sensor networks is currently at the physical node-level [Welsh and Mainland 2004] and is the key reason responsible for the lack of robustness for higher-level control applications.
3. Design of networked control systems with flexible topologies is hard with physical node-level programming, as the set of tasks (or responsibility) is associated with the physical node [Robinson and Kumar 2008].
4. Fault diagnostics, repair and recovery are manual and template-driven for a majority of networked control systems [Jalote 1994], [Lee and Anderson 1990]. Runtime adaptation is necessary to maintain the stability and performance of the higher-level control system.

While several approaches address these challenges for open-loop wireless sensor networks (see related work), our focus is on closed-loop wireless controller networks.

1.2. Embedded Virtual Machines

To address actuation in closed-loop wireless control systems there is a strong need to re-think the communication architectures and protocols for reliability, coordination and control [Willig et al. 2005]. Current approaches for robust networked control [Hespanha et al. 2007] require the underlying network to satisfy a minimal set of requirements (e.g. guaranteed packet deliver rate, upper bound on network induced delay) and reduce the network model to that of a single channel with random delays. In ad-



Control Algorithm	Processing Units	Bandwidth Utilization	Physical Requirements
CA_1	0.2MIPS	10%	S_1, S_2, A_1
CA_2	0.4MIPS	5%	S_2, S_3, A_1, A_2
CA_3	0.1MIPS	20%	S_3, A_2

Fig. 1. (a) A wireless sensor, actuator and controller network. (b) Algorithm assignment to a set of controllers, each mapped to the respective nodes. (c) Three Virtual Components, each composed of several network elements. (d) Decoupled virtual tasks and physical nodes with runtime task mapping.

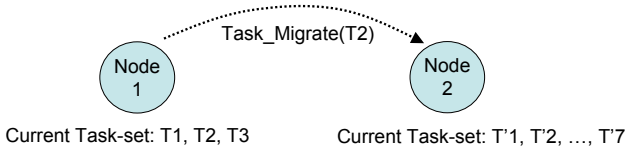


Fig. 2. Task migration for real-time operation (instructions, stack, data & timing/fault tolerance meta-data) on one physical node to another.

dition, they do not address the spatial aspects of the network, i.e., how changes in the network topology affect the closed-loop system performance.

As the links, nodes and topology of wireless systems are inherently unreliable, such time-critical and safety-critical applications require programming abstractions where the tasks are assigned to the sensors, actuators and controllers as a *single component*, rather than statically mapping a set of tasks to a specific physical node at design time (Figure 1(b)). Such wireless controller grids are composed of many nodes that share a common sense of the control application but without regard to physical node boundaries. Our approach, as shown in Figure 1, is to *decouple* the functionality (i.e., tasks) from the inherently unreliable physical substrate (i.e., nodes) and allow tasks to migrate/adapt (Figure 2) to changes in the topology.

To this end, we introduce the Embedded Virtual Machine (EVM), a powerful and flexible programming abstraction where a Virtual Component (VC) and its properties are maintained across node boundaries, as shown in Figure 1(c). EVMs differ from classical system virtual machines. In the enterprise or on PCs, one (powerful) physical machine may be partitioned to host multiple virtual machines for higher resource utilization. On the other hand, in the embedded domain, an EVM is composed across multiple physical nodes with the goal to maintain correct and high-fidelity operation even under changes in the physical composition of the network. The goal of the EVM is to maintain a set of *functional invariants*, such as a control law and *para-functional invariants* such as timeliness constraints, fault tolerance and safety standards across a set of controllers given the spatio-temporal changes in the physical network. Thus, the EVM introduces new degrees of freedom, task migration and routing which facilitates, at runtime, the network configuration (operating point, conditions) to meet the requirements of the networked control algorithms. However, the EVM does not provide explicit guarantees but only finds the optimal operation configuration in terms of routing and task assignment.

1.3. Overview of the EVM Approach

While wireless system engineers optimize the physical, link and network layers to provide an expected packet error rate, this does not translate accurately to stability of the control problem at the application layer. For example, planned and unplanned changes in the network topology with node/link failures are currently not easily captured or specifiable in the metrics and requirements for control engineers. For a given plant connected to its set of controllers via wireless links (see Figure 1(a-b)) it is necessary that the controller process the sensor inputs and perform actuation within a bounded sampling interval. While one approach is to design specialized wireless control algorithms that are robust to specified range packet errors [Hespanha et al. 2007], [Zhang et al. 2001], it is non-trivial to design the same for frequent topological changes. Furthermore, it is difficult to extend the current network infrastructure to add/remove nodes and to redistribute control algorithms to suit environmental changes such as battery drain for battery-operated nodes, increased production during off-peak electricity pricing, seasonal production throughput targets and operation mode changes.

The EVM approach is to allow control engineers to use the same network control algorithms on the wireless network without knowledge of the underlying network protocols, node-specific operating systems or hardware platforms. The virtual machine executing on each node (within the VC) instruments the VC to adapt and reconfigure to changes while ensuring the control algorithm is within its stability constraints. This approach is complementary to the body of network control algorithms as it provides a logical abstraction of the underlying physical node topology.

The paper is organized as follows: Section 3 presents the automated design flow from a control problem specification to binding controller tasks to nodes within a VC. Section 4 describes the architecture of the EVM and mechanisms for parametric and programmatic control. Given these mechanisms, Sections 5 presents the key task assignment and scheduling algorithm to optimize operation during network changes, while Section 6 presents runtime procedures used to monitor and adapt the execution of control algorithms. Finally, we describe the implementation on real hardware in Section 7 and a case study in Section 8.

2. RELATED WORK

There have been several variants of virtual machines, such as Maté [Levis and Culler 2002], Scylla [Stanley-Marbell and Iftode 2000] and SwissQM [Müller et al. 2007], and flexible operating systems, such as TinyOS [Hill et al. 2000], SOS [Han et al. 2005], Contiki [Dunkels et al. 2004], Mantis [Bhatti et al. 2005], Pixie [Lorincz et al. 2008] and LiteOS [Cao et al. 2008], for wireless sensor networks. The primary differences that set EVM apart from prior work is that it is centered on real-time operation of controllers and actuators. Within the design of the EVM's operating system, link protocol, programming abstractions and operation, timeliness is a first-class citizen and all operations are synchronized. The EVM does not have a single node-perspective of mapping operations to one virtualized processor on a particular node but rather maintains coordinated operation across a set of controllers within a virtual component. q

One of the first virtual machines (VM) for sensor networks was Maté [Levis and Culler 2002]. Maté implements a simple, communication-centric VM built on top of the TinyOS [Hill et al. 2000]. It is designed as a high level interface where code is written using limited instruction set, defined at design-time, and executed with a FORTH-like interpreter. EVM utilizes a similar FORTH-like interpreter but is extensible at runtime and allows for fully preemptive tasks. Scylla is a more conventional system VM where one physical machine exposes interfaces to a single logical machine. On the other hand, EVM uses several physical nodes and allows user to consider the virtual

component as a single logical entity. SOS operating systems for sensor nodes [Han et al. 2005] was designed to allow flexible operation of nodes in a sensor network. As TinyOS, it has an event-driven execution of all components, with a difference that components can be installed/modified during runtime. EVM is built on the nano-RK sensor RTOS [nanoRK 2010] and hence all tasks are scheduled by fixed priority scheduling and are fully preemptable. The Virtual Node Layer [Brown et al. 2007] provides a programming abstraction where each virtual node is identified with a particular region and it is emulated by one of the physical nodes in its region. On the other hand, EVM uses several physical nodes and allows the user to consider the virtual component as a single logical entity.

In the last few years, several different systems for macro-programming in WSN have been developed. [Welsh and Mainland 2004] have defined a set of abstractions representing local communication between nodes in order to expose control over resource consumption along with providing feedback on its performance. An extension of these ideas is used to develop Regiment [Newton et al. 2007], a high-level language based on the functional reactive programming. Kairos [Gummadi et al. 2005] allows a programmer to describe code execution for each of the nodes in a network using a centralized approach where details about code generation, remote data access and management along with node interactions are hidden from the programmer. EVM is not a generic macroprogramming system as it focuses on closed-loop control with native runtime support for task assignment and migration.

The development of control algorithms able to deal with the unreliability of the wireless channel for Networked Control Systems (NCSs) is an active area of research in the control systems community [Hespanha et al. 2007], [Zhang et al. 2001]. Few efforts consider networked control over arbitrary topologies (e.g., [Robinson and Kumar 2008], [Gupta et al. 2009]). Even in these articles, the authors assume the existence of a single actuation point and a single sensing point on the plant. They show that the optimal position of the controller is at the actuation point, while ignoring the wireless channel in the estimation of the plant's state. The authors show that, in general case, the problem of assigning the best location of the controller node is very complex. Finally, Etherware [Graham et al. 2009] presents challenges in software development for NCSs along with abstractions and architectures used to implement control algorithms for NCSs. The authors describe a middleware for control systems but do not provide algorithms which might be used to guarantee that designed middleware satisfies requirements for the control algorithms.

3. EVM DESIGN FLOW

Our focus is on the design and implementation of wireless controllers and in providing such controllers with runtime mechanisms for robust operation in the face of spatio-temporal topological changes. We focus exclusively on controllers and not on sensors or actuators, as the latter are largely physical devices with node-bound functionality. A three-layered design process is presented to allow control engineers to design wireless control systems in a manner that is both largely platform/protocol/hardware/architecture independent and extensible to different domains of control systems (in process, discrete, aviation, medical, etc.). This section describes the design flow from a control problem formulation in Simulink, automatic translation of control models from Simulink to the platform-independent EVM interpreter-based code and finally to platform-dependent binaries (see Figure 3). These binaries are assigned to physical nodes within a VC using assignment and scheduling algorithms presented in Section 5. The binaries are executed as Virtual Tasks within the platform dependent architecture described in Section 4.

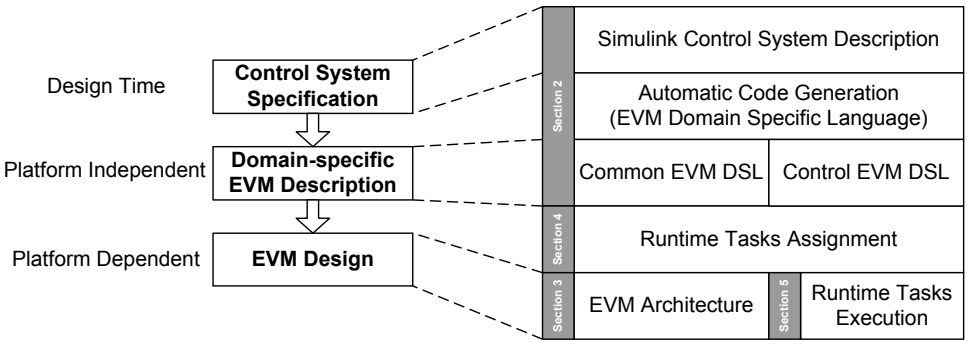


Fig. 3. EVM design flow.

At design time, control systems are usually designed using software tools, such as Matlab/Simulink, that incorporate both modeling and simulating capabilities. Therefore, to automatize the design flow the EVM is able to automatically generate functional models from the Simulink control system description. These functional models define the processes by which input sampled data is manipulated into output data for feedback and actuation. The models are represented by generated code and meta data for *platform and node independent system description*. This allows a system designer to exclusively focus on the control problem design. Beside the functional description in the platform-independent and domain specific language (DSL), from the Simulink model the EVM design flow automatically extracts additional para-functional properties like *timing* and *inter task dependencies*. These properties, along with the functional description are used to define a *platform optimized binary* for each Virtual Task (VT).

3.1. Platform Independent Domain Specific Language

To generate functional description of the designed system, the EVM programming language is based on FORTH, a structured, stack-based, extensible, interpreter-based programming language [Conklin and Rather 2007]. Since the goal of the EVM design is to allow flexibility and designing utilities independent of chosen programming language, the intermediate programming language is not constrained to the EVM programming language. The interpreter used to execute modules described in the EVM programming language can also execute precompiled binaries. The EVM implementation, presented in Section 7, executes binaries derived from embedded C code. This enables execution of code binaries developed in other languages used to describe control system implementation.

The use of the EVM intermediate programming language enables *domain-specific constructs*, where basic programming libraries are related to the type of application that is being developed. For example, for use in embedded wireless networks for industrial control we developed two predefined libraries, Common EVM and Control EVM (a full list of API's is provided in [Pajic and Mangharam 2009]). Common EVM (Figure 4(b)) is based on the standard FORTH library [Conklin and Rather 2007]. Beside the : word, used to define new words, all other words can be separated into the following categories: 1) arithmetic operations, 2) logical operations, 3) memory manipulation, 4) sensor and actuator handling, and 5) networking. Control EVM (Figure 4(a)) contains functionalities widely used to develop control applications. First three words specified in Figure 4(a) are used for Singe-Input-Single-Output (SISO) systems. Al-

- Dictionary consists of predefined blocks/functions that are usually used for control systems description
- PID (K_P K_I K_D *addr_in addr_out* –)
- PI (K_P K_I *addr_in addr_out* –)
- TF (*m n addr_alpha addr_beta addr_in addr_out* –)

$$G(z)=\frac{N(z)}{D(z)} \quad N(z)=\alpha_0+\alpha_1 z^{-1}+\alpha_2 z^{-2}+...+\alpha_m z^{-m},$$
$$D(z)=\beta_0+\beta_1 z^{-1}+\beta_2 z^{-2}+...+\beta_n z^{-n}.$$

- LTI (*p m n addr_A addr_B addr_C a_in a_x a_out* -)

$$x[k+1]=Ax[k]+Bu[k],$$
$$y[k]=Cx[k], A\in R^{n\times n}, B\in R^{n\times m}, C\in R^{p\times n}$$

(a) Control-EVM

- Arithmetic operations (on 16bit)
- Logical operations
- Comparison and testing
- Controlling programming flow
- Memory manipulation
- Sensor/Actuator handling
  - RDSensG (sensID – n1)
  - RDSensL (sensID – n1)
  - WRActG (value actID –)
  - WRActL (value actID –)
- Networking
  - PktSendG (addr n nodeID -)
- Task handling
  - TaskActivate (addrTCB actID –)

(b) Common-EVM

Fig. 4. EVM platform-independent and domain-specific language for expressing functional and timing description of Simulink models.

though these words can be described using the LTI word (describing Linear Time Invariant systems), their wide use in control systems recommended their specific use.

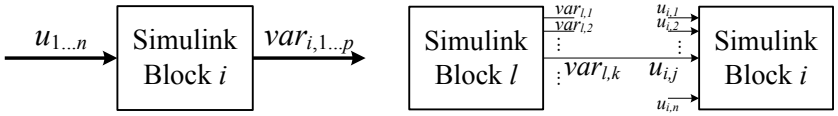
The extensibility of the EVM allows definition of additional domain-specific libraries such as Automotive EVM, Aviation EVM or Medical EVM libraries, which will contain functionalities specific to each of these application fields. Using EVM libraries, the code generator creates a system description from a predefined components, thus creating a task description file for each of the Virtual Tasks.

3.2. Control Problem Synthesis: From Simulink to Platform Independent Specification

We now describe the procedure to automatically extract the functional description of a VT from a Simulink design. Within Simulink, each block (and, thus, the model itself) is represented as a hierarchical composition of other Simulink blocks, either *subsystems* or *library-defined* blocks. This organization of Simulink models allows for a natural extraction of a structured functional description using predefined words from the platform-independent EVM DSL dictionary. When a new Simulink block is defined as a composition of previously defined blocks, a new word is defined for the EVM functional description using previously defined words. The process is repeated until a level is reached where all words belong to the EVM dictionary.

A VT description is obtained by parsing the Simulink model file. This is done by searching for new block definitions along with the interconnections between blocks. In a Simulink model file (i.e., mdl file) blocks are presented as shown in Figure 5(c) and Figure 5(d) where BlockType parameter describes whether the block is a part of the Simulink library or a subsystem, consisting other Simulink blocks. To extract the VT description we require that the task is implemented in a singular, discrete-time Simulink subsystem, such as the example shown in Figure 6. The synthesis of the platform-independent specification from the model is carried out in three steps:

(1) **Definition of intermediate words and variables:** Each block i is associated with a word W_i from the EVM DSL, where the output of the block is assigned to a variable var_i. To illustrate this consider the extended PID controller from Figure 6. The outputs of all intermediate blocks are assigned to variables as shown in Figure 6. For example, the EVM description of block "Sum1" is described with word W_8 and its output with variable var_8. As the EVM DSL is stack-based with reversed Polish



(a) General model of a block (b) Interconnection between blocks

Block{
BlockType *Gain*
Name "*Integral_Gain*"
Position [120, 93, 160, 117]
Block specific data ···
}

(c) Simulink description of a library block

Block{
BlockType *SubSystem*
Name "*PID_controller*"
Ports $[n, p]$
Block specific data ···
}

(d) Simulink description of a subsystem

Line{
SrcBlock "*Simulink Block l*"
SrcPort k
DstBlock "*Simulink Block i*"
DstPort j
}

(e) Simulink description of a link between two blocks

Fig. 5. General relations between Simulink blocks.

notation the block is described as:

$$: \ \mathbf{W_8\ R2\ out3}\ ?\ \mathbf{NEG\ sum\ var_8}\ @\ ;$$

where ? and @ are read and write operators respectively. In general case, for a block presented in Figure 5(a) the parser defines the following word:

$$: \ \mathbf{W_i\ u_1}\ ?\ \mathbf{u_2}\ ?\ ...\mathbf{u_n}\ ?\ \mathbf{coeffs\ BlockWord\ var_{i_1}}\ @\ \mathbf{var_{i_2}}\ @\ ...\ \mathbf{var_{i_p}}\ @\ ;$$

where **BlockWord**, depending on BlockType, corresponds to either a predefined word (if a library block is used) or a new word that needs to be defined using the same parser algorithm (if the block is a subsystem). Variables presented as **coeffs** are extracted from the 'Block Specific' data in cases when they are contained in the block description (from Figure 5(c),(d)), along with initial values for variables $\mathbf{var_i}$. For example, consider definition of word $\mathbf{W_4}$. Since block *PID_controller1* contains coefficients for K_p, K_i and K_d their values are included in the definition. Finally, in the previous formulation variables $u_{[1..n]}$ are replaced with appropriate system variables with respect to connections between blocks. To illustrate this consider a connection (i.e., line) between blocks from Figure 5(b). Simulink defines the *Line* as in Figure 5(e). Thus, for *Simulink Block i*, in the definition of word $\mathbf{W_i}$ each variable $u_{i,j}$ is replaced with appropriate variable $var_{l,k}$.

(2) **Composing extracted words:** The intermediate words are composed to create functional description of the system (e.g., $\mathbf{VT_{ctrl}}$). The parser is recursively executed for all subsystems till all words are part of the library. The description for the example from Figure 6 is presented in Step 2, Figure 7. It is worth noting that the intermediate words are executed in the blocks' execution order for the Simulink model. The order is either specified explicitly in the model or determined implicitly based on block connectivity and sample time propagation [MathWorks 2010].

(3) **DSL code optimization:** Intermediate blocks with elementary functions can be pruned in a single word. For the example from Figure 6 the optimized description is shown in Step 3, Figure 7. Words **W3**, **W4**, **W5** and **W8**, **W9**, **W10** are combined into a single word (**W3** and **W8**, respectively). Also, instead of word **W6** and variable **var6**, **W1** and **var1** are used. The code optimization reduces the number of defined words and used variables. Currently, the optimization is restricted to a small set of control system configurations. A more general approach is an avenue for future work.

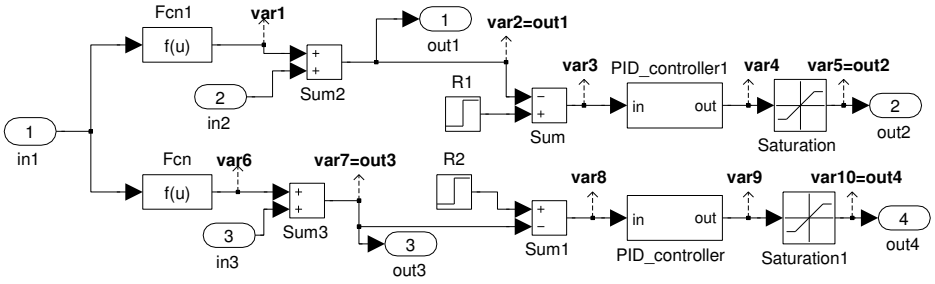


Fig. 6. Simulink model of an extended PID controller.

Step 1: Intermediate words/variables

```
: W1 in1 ? f var1 @ ;
: W2 var1 ? in2 sum out1 @ ;
: W3 out1 ? NEG R1 sum var3 @ ;
: W4 var3 ? Kp1 Ki1 Kd1 PID var4 @ ;
: W5 var4 ? thr SAT out2 @ ;
: W6 in1 ? f var6 @ ;
: W7 var6 ? in3 ? sum out3 @ ;
: W8 R2 out3 ? NEG sum var8 @ ;
: W9 var8 ? Kp2 Ki2 Kd2 PID var8 @ ;
: W10 var9 ? thr SAT out4 @ ;
```

Step 2: Composition

```
: Vctrl W1 W2 W3 W4 W5 W6 W7 W8 W9 W10  out1 ?
    out2 ? out3 ? out4 ?
```

Step 3: Optimization

```
: W1 in1 ? f var1 @ ;
: W2 var1 ? in2 sum out1 @ ;
: W3 out1 ? NEG R1 sum Kp1 Ki1 Kd1 PID thr SAT out2 @ ;
: W7 var1 ? in3 ? sum out3 @ ;
: W8 R2 out3 ? NEG sum Kp2 Ki2 Kd2 PID thr SAT out4 @ ;
: Vctrl W1 W2 W3 W7 W8 out1 ? out2 ? out3 ? out4 ? ;
```

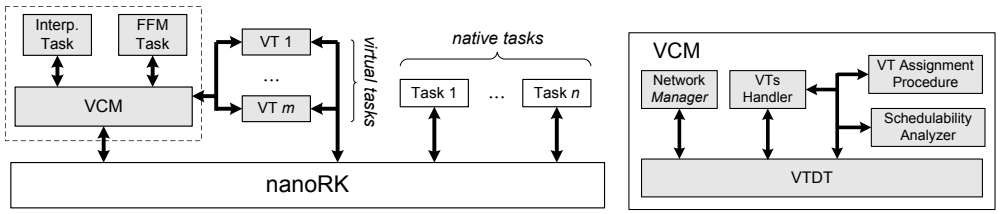
Fig. 7. EVM functional description extracted from Simulink model shown in Figure 6.

As our intention is to map the control problem to a scheduling problem, timing parameters (i.e., period and worst-case execution time) are also extracted from the model. We consider only discrete-time controllers as potential VTs. For these, Simulink design rules force the designer to define a sampling rate for each (discrete-time) block. Currently we cover cases where the controller is designed in a single clock domain (i.e., all blocks use the same sampling period). In general case, when a controller contains several clock domains, each sub-domain is represented with its respective virtual tasks. Also, a set of dependencies between the tasks is extracted. Finally, to extract the worst-case execution time, a simple static analysis is performed using the execution time measurements for library defined words with respect to the specific platform. (currently the EVM is implemented on two types of platforms; see Section 7 for details).

4. EVM ARCHITECTURE

We now describe the node-specific architecture which implements the mechanisms for the virtual machine on each node. The Common-EVM and Control-EVM description are scoped within Virtual Tasks (VTs) that are mapped at runtime by the Task Assignment procedure presented in the next section. This description is interpreted by the Virtual Component Interpreter running on each node. The EVM runtime system is built as a *supertask* on top of the nano-RK real-time operating system [nanoRK 2010], allowing node-specific tasks to execute native and virtual tasks (i.e., those that are dynamically coupled with a node) to run within the EVM. The EVM block-level reference architecture is presented in Figure 8(a). This allows the EVM to maintain node specific functionalities and be extensible to runtime task evocation of existing or new virtual tasks.

The interface between nano-RK and all VTs is realized using the Virtual Component Manager (VCM). The VCM maintains local resource reservations (CPU, network slots,



(a) EVM block-level reference architecture (b) Structure of the VCM

Fig. 8. EVM architecture with the Virtual Component Manager running as a supertask alongside native nano-RK tasks.

memory, etc.) within nano-RK, the local state of the VTs and global mapping of VTs within the VC. The VCM is responsible for memory and network management for all VTs-to-physical nodes and presents a mapping between local and remote ports which is transparent to all local VTs. It includes a FORTH-like interpreter for generic and domain-specific runtime operations and a Fault/Failure Manager (FFM) for runtime fault-tolerant operation. The VCM is implemented in a modular form so the interpreter, FFM and other specialized modules may be swapped with extensions over time and for domain-specific applications.

4.1. EVM Extensions to the nano-RK RTOS

nano-RK is a fully preemptive RTOS with multi-hop networking support that runs on a variety of sensor network platforms (8-bit Atmel-AVR, 16-bit TI-MSP430, Crossbow motes, FireFly) [nanoRK 2010]. nano-RK uses the RT-Link [Rowe et al. 2008], a real-time link protocol. It supports fixed-priority preemptive scheduling to ensure that task deadlines are met, along with support for enforcement of CPU and network bandwidth reservations. nano-RK had been design as a fully static OS, configured at design time. Thus, to allow parametric and programmatic runtime code changes nano-RK was re-designed and extended with several new features:

• *Runtime Parametric Control:* Support for dynamic change of the sampling rates, runtime task and peripheral activation/deactivation and runtime modification of the task utilization was added. These facilities are exposed and executed via the Common-EVM programmer interface.

• *Runtime Programmatic Control:* As a part of the EVM design a procedure for dynamic task migration was implemented. This requires runtime schedulability analysis, capability checks to migrate a subset of the task data, instructions, required libraries and task control block. Based on the procedure presented in Sections 5 and 6, tasks may be activated or migrated between primary and backup nodes. Such facilities are triggered by the primary-backup policy implemented on top of the EVM architecture.

• *Dynamic Memory Management:* Both *Best-fit* and *First-fit* memory allocation methods are supported. In addition, a *Garbage Collector* (GC) has been designed to reclaim all memory segments owned by tasks that had been terminated. The GC is scheduled only when its execution does not influence execution of other tasks.

4.2. Virtual Component Interpreter

The Virtual Component Interpreter provides an interface to define and execute all VTs. Every VT is defined as a word within the VCM library. When a new VT description is received over the network, the VCM calls the interpreter that defines a new word using the description file of the task and existing VC libraries. After a VT is activated, each execution of the VT is realized as a scheduled activation of the interpreter with the

VT's word provided as an input. To allow preemptivity of the tasks, each call of the interpreter uses a VT-specific stack and dedicated memory segments. In addition, during its execution, each VT is capable of dynamically allocating new memory blocks of fixed size (currently 128B) using the EVM's memory manager. Therefore, the interpreter is designed to use logical addresses in the form $(block\_index, address\_in\_block)$.

Each node maintains a local copy of standard Common-EVM and Control-EVM dictionaries. If a new word needs to be included in the existing library, the interpreter first checks the global word identifier and revision number to discard obsolete versions.

4.3. Virtual Tasks

Each VT is described using the Virtual Task's Description Table (VTDT), comprised of global and local descriptions of a VT. Copies of the table are stored on all members of the VC. While this requirement for consistency currently results in an issue of scalability, a large fraction of the higher-speed control in SCADA systems require networks with less than 20 nodes and is hence within the practical limits of the current approach. Each VT's global description has information about memory requirements, stack size and number of used fixed size memory blocks (128B). In addition to the above meta data, network requirements in terms of number of RT-Link transmit and receive slots are specified at design time.

The above descriptors are specified within the VCM's Task Control Block (TCB) for each task, which is an extension to the native nano-RK TCB (for details see [Pajic and Mangharam 2009]).

4.4. Virtual Component Manager

The fundamental difference between the native nano-RK and the VCM is that the scope of nano-RK's activities is local, node-specific and defined completely at design time, while the scope of the VCM is the VC that may span multiple physical nodes. The VCM subcomponents are presented in Figure 8(b). The current set of supported runtime functionalities is:

4.4.1. Virtual Task handling (controlled by the VT Handler):

4.4.1.1 VC state includes the mapping of VTs to physical nodes and quality of links between physical nodes. The VCM in each controller node within the VC maintains the VC state and periodically broadcasts it to keep consistency between all members of the VC. Currently, a centralized consensus protocol is used, while a distributed consensus protocol is needed to scale operations.

4.4.1.2 VT migration and activation that can be triggered as a result of a fault/failure procedure or by a request from either the VT or the VCM. As a part of a task migration, the task's VTDT is sent along with all memory blocks utilized by the task. If the VT is already defined on a Backup node (checked by exchange of hash values), only task parameters are exchanged. In addition, before migrating a VT to a particular node the Schedulability Analyzer performs network and CPU schedulability analysis for nodes that are potential candidates (details are provided in the next section). If the analysis shows that no node can execute the task correctly, an error message is returned. Finally, after a VT is defined, to activate the task the host node performs a local CPU and network schedulability analysis to ensure that the task will not adversely affect correct execution of previously defined VTs.

4.4.1.3 Control of tasks executed on other nodes: For all VTs in the *Backup* mode, the VT Handler shadows execution of the VT in the *Primary* mode. If a departure from the desired operation is observed (e.g., low battery level, decreased received packet signal strength), *Backup* nodes may be assigned to the *Primary* mode based on the policy.

4.4.1.4 VT Assignment: VT Assignment procedure is activated to assign execution of

the VTs to specific nodes, when incremental and local re-assignment (described in Section 6) fails. The procedure determines the best set of physical controller nodes to execute VTs given a snapshot of the current network conditions along with the initial communication and computation schedules for the nodes.

4.4.2. Network Management (performed by the Network Manager):
4.4.2.1 Transparent radio interface: Using the message header which contains information about message type, the VCM determines tasks that should be informed about the message arrival. Messages containing tasks and their parameter definitions are first processed by the VCM, before the VCM activates the interpreter.
4.4.2.2 Logical-to-physical address mapping: Communication between VTs is done via the VCM. Since a VT does not have information on which nodes other VTs are deployed, the VCM performs logical-to-physical address mapping. In cases when both tasks are on the same node, the VCM directly passes a message to the receiving task's buffer.

5. VIRTUAL TASK ASSIGNMENT

With the knowledge of the underlying EVM architecture, we now discuss the algorithm used for the VT Assignment procedure. The procedure determines the initial assignment of the VT's executions along with the communication and computation schedules. The criteria for triggering re-assignment calculation is described in Section 6. We derived a general case problem formulation for the VT's assignment as a binary integer linear optimization problem which is then solved efficiently using well-known techniques (branch and bound) [Schrijver 1998]. In addition, since standard link protocols for wireless factory automation, such as WirelessHART [HART 2007], recommend that only one physical node may transmit in each time slot, we were able to obtain an efficient reformulation of the relaxed assignment problem. In this case, each control loop (operating across the same physical set of controllers) can be considered separately, which considerably simplifies tasks assignments, as it allows a *compositional* system design.

5.1. General Formulation

To develop an assignment algorithm we considered a multi-hop control network that corresponds to our model of a VC. The network consists of $p \geq 1$ processes ($\mathbb{J} = \{1, ..., p\}$ denotes set of all processes) and a set of nodes (sensors, actuators and controllers), where all nodes have a radio transceiver along with memory and computing capabilities (see Figure 9(a)). The nodes communicate using a TDMA based protocol (i.e., in a time-triggered manner) with frame size F_S. The network is described with a directed graph $G = (V, E)$ that represents radio connectivity in the network. Set $V = \{v_1, v_2, ..., v_m\}$ denotes a set of physical nodes in the network,[1] while $E = \{(v_i, v_j) |\ v_i$ and v_j are connected$\}$ is a set of all links. In addition, each link e is described with its link quality $LQ(e)$. To extract a problem formulation it is necessary to enumerate all paths in the network which should be used for communication between a node and a sensor (or an actuator).[2,3] Thus, the l^{th} path between node v_i and sensor/actuator k is denoted as $\psi^l_{i,k}$.

The goal of the assignment procedure is to determine: (1) An assignment of the Virtual Tasks (i.e., Control Algorithms - CAs) to the set of nodes V, where each VT

[1] In the remainder of the paper, V will also denote the set that contains nodes' indexes $\{1, 2, ..., m\}$.
[2] A path is represented as a directed path connecting the sender with exactly one receiver.
[3] Including all paths could significantly increase complexity of the optimization problem. Therefore, the user might opt to enumerate only selected paths with best characteristics (e.g., a small number of hops, high packet delivery ratio).

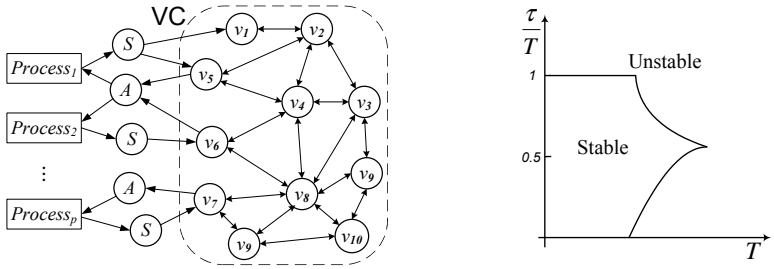


Fig. 9. (a) Reference model of a multi-hop wireless network used for control. The network consists of multiple sensors (S), actuators (A) and controllers (v_i's). The VC includes multiple physical controller nodes; (b) An example stability region for such a network. T is a controller sampling period, while τ is the network induced delay.

is assigned to one node in the *Primary* mode and to R nodes in the *Backup* mode. (2) A communication schedule that determines active links at each time slot. (3) A computational schedule that determines in which time slot each VT is executed. In addition, to define the problem as an optimization problem, the following assumptions were made:

A.1 For each process j, the *Primary* and all *Backup* nodes assigned with the j^{th} virtual task are scheduled in the same time slot(s).
A.2 Virtual Tasks are mutually independent.
A.3 A process i (for all i) will remain stable if its sampling period is less than some predefined value T_i. Therefore, we require $F_S \leq min(T_1, T_2, ..., T_p)$.

The first assumption simplifies the problem formulation and allows for an easier schedulability analysis scheme. The second assumption is reasonable since a significant class of process controllers execute a large number of simple and independent control loops. As an avenue of future work, this assumption will be relaxed to consider dependencies between tasks. To justify the last assumption we use the approach described in [Zhang et al. 2001]. For example, consider a closed-loop control of a plant modeled with continuous-time Linear-Time Invariant (LTI) dynamics:

$$\dot{x}(t) = Ax(t) + Bu(t),\ y(t) = Cx(t).$$

The controller employs a discrete-time state feedback control with $u(kT) = -Kx(kT)$, where T denotes the plant's sampling period. If network induced delay τ_k is less than one sampling period,[4] the control feedback has the following form:

$$u(t^+) = -Kx(kT),\ t \in [kT + \tau_k, (k+1)T + \tau_{k+1}).$$

Thus, $u(t)$ is a piecewise continuous function that changes values only at time instances $kT+\tau_k$. The EVM utilizes fully synchronous networks, which allows scheduling the actuators to apply new input values at the same time, after the messages were delivered to all of them. This guarantees the same delay for all plant's inputs at each sampling period ($\tau_k = \tau, \forall k$). Using the methods based on simulation, as in [Zhang et al. 2001], the stability region can be determined with respect to sampling period T and the induced delay τ. The region is used to establish the maximal sampling period for which the system maintains stability if a network delay is less than the period ($\frac{\tau}{T} \leq 1$, an example is shown in Figure 9(b)).

To formulate the problem, the following decision variables are used:

[4]A similar approach can be used even if the delay is longer than the sampling period.

— $2mp$ binary assignment variables, $x_{i,j}^{st} \in \{0,1\}$, where $i \in V, j \in \mathbb{J}, st \in \{a,b\}$ and $x_{i,j}^{a} = \begin{cases} 1, v_i \text{ is the } \textit{Primary} \text{ for } j^{th} \text{ VT} \\ 0, \text{ otherwise} \end{cases}$, $x_{i,j}^{b} = \begin{cases} 1, v_i \text{ is a } \textit{Backup} \text{ for } j^{th} \text{ VT} \\ 0, \text{ otherwise} \end{cases}$

— Routing binary variables $y_{i,k}^{l} \in \{0,1\}$, where:
$y_{i,k}^{l} = \begin{cases} 1, l^{th} \text{ path between node } v_i \text{ and sensor/actuator } k^{th} \text{ is used} \\ 0, \text{ otherwise} \end{cases}$

— Communication schedule binary variables $\eta_{i,k}^{l,n} \in \{0,1\}$, where $n \in \{1,...F_s\}$ and:
$\eta_{i,k}^{l,n} = \begin{cases} 1, l^{th} \text{ path between node } v_i \text{ and sensor/actuator } k \text{ is active in } n^{th} \text{ slot} \\ 0, \text{ otherwise} \end{cases}$

— Computation schedule binary variables $\mu_i^n \in \{0,1\}$, where $n \in \{1,...Fs\}$ and: $\mu_j^n = \begin{cases} 1, j^{th} \text{ VT is scheduled for execution in } n^{th} \text{ time slot} \\ 0, \text{ otherwise} \end{cases}$

Our goal is to describe the assignment problem in the form:

$$\min f(\mathbf{x}, \mathbf{y}, \eta, \mu), \text{ subject to } \quad \mathbf{x}, \mathbf{y}, \eta, \mu \in \mathbb{S}_{\mathbb{C}}$$

where vectors $\mathbf{x}$, $\mathbf{y}$, η, μ contain the aforementioned decision variables and $\mathbb{S}_{\mathbb{C}}$ describes a set that satisfies all constraints, ensuring desired system's behavior. The constraints take into account the requirements for control problem along with dependencies between communication and computation schedules. In the remaining of this section the imposed set of constraints is described.

5.1.1. ***Assignment of the Control Algorithms.*** Each VT has to be assigned to exactly one node in the *Primary* mode and R additional *Backup* nodes (different from the *Primary* node for the CA). These constraints are described as:

$$\sum_{i=1}^{m} x_{i,j}^{a} = 1, \quad \sum_{i=1}^{m} x_{i,j}^{b} = R, \quad \text{and} \quad x_{i,j}^{a} + x_{i,j}^{b} \leq 1, \qquad \forall j \in \mathbb{J}, \forall i \in V.$$

5.1.2. ***Requirements for robust design.*** Additional sets of constraints are introduced to improve performance of the closed-loop system. *Link reliability* constraints require that only links with quality above a given threshold are considered, which reduces complexity of the problem formulation. Logical pruning of graph G results in a graph $G_T = (V, E_T)$, where $E_T = \{(v_i, v_j) \in E | LQ(v_i, v_j) \geq THR)\}$.

The *Routing constraints* describe a means to increase system robustness to the link failures with the use of different paths for data routing. For example, WirelessHART recommends that each node can use at least two separate paths to route data [Alur et al. 2009]. Thus, we require that the *Primary* node for each VT uses two different paths to deliver information to all actuators related to the process' control. In addition, the *Primary* and all *Backup* nodes have to be connected with all sensors related to the VT.[5] Denoting as A_j and S_j the sets of actuators and sensors respectively, related to the j^{th} process, these constraints are described as:

$$\sum_{\forall l} y_{i,k_a}^{l} = 2x_{i,j}^{a}, \ \sum_{\forall l} y_{i,k_s}^{l} = x_{i,j}^{a} + x_{i,j}^{b}, \quad \forall j \in \mathbb{J}, k_a \in A_j, k_s \in S_j, \forall i \in V.$$

Finally, a set of *Monitoring constraints* is imposed, where all *Backup* nodes monitor the execution of a VT on the *Primary* node. Thus, to alleviate the system design and VT migration when the *Primary* node fails, constraints are enforced that all R

[5]It is worth noting here that a different routing policy could be used. However, even if that is the case these constraints could be expressed in a similar way.

Backup nodes have to be 1-hop neighbors of the *Primary* node. Denoting as N_i set of all neighbors of node v_i, these constraints are described as:

$$\sum_{k \in N_i} x^b_{k,j} \geq R \cdot x^a_{i,j}, \ \forall j \in \mathbb{J}, \forall i \in V.$$

5.1.3. ***Computation schedule constraints.*** From assumptions A.3 and A.1, we require that computations of each VT on the *Primary* and *Backup* nodes have to be scheduled exactly once in a frame. This implies that all VTs have the same sampling rate and could result in a more frequent computation of a VT. In most automation systems the increase of the sampling rate can not endanger the closed-loop system stability. On the contrary, it can increase the performance of the implemented controller if the optimal discrete-time controller is used [Cervin et al. 2002].[6] Thus, the constraints are expressed as: $\sum_{n=1}^{F_S} \mu^n_j = 1, \ \forall j \in \mathbb{J}$.[7]

5.1.4. ***Communication schedule constraints.*** From assumption A.3, closed-loop system stability is guaranteed if the end-to-end communication delay (i.e., delay from the sensors to the assigned controller and from the controller to the actuators) along with the time needed for the controllers' computation is less than F_S. Thus, the first requirements for the communication schedule is that only used paths are scheduled and that the number of slots assigned to the used path is exactly equal to the path's length (i.e., number of hops on the path):

$$\eta^{l,n}_{i,k} \leq y^l_{i,k}, \ \forall n, 1 \leq n \leq F_S, \qquad \sum_{n=1}^{F_S} \eta^{l,n}_{i,k} = y^l_{i,k} \cdot d(\psi^l_{i,k}), \ \forall i, k \in V, \forall l.$$

Additionally, the schedule has to be collision free (i.e., two interfering nodes cannot transmit in the same time slot). To express these constraints, for each path $\psi^l_{i,k}$ where k is a sensor, all links are enumerated in increasing order starting from the link with origin at sensor k and ending with the link with the destination at node i. Similarly, for each path $\psi^l_{i,k}$ where k is an actuator, enumeration starts at node i and ends at actuator k. This is used to create the interference links table for each pair of paths $(\psi^{l_1}_{i_1,k_1}, \psi^{l_2}_{i_2,k_2})$. An element (n_1, n_2) is a member of the $(\psi^{l_1}_{i_1,k_1}, \psi^{l_2}_{i_2,k_2})$ interference table (IT) if transmissions over the ${n_1}^{st}$ link of the path $\psi^{l_1}_{i_1,k_1}$ interferes with transmissions over the ${n_2}^{nd}$ link of the path $\psi^{l_2}_{i_2,k_2}$. Constraints for interference-free schedule can be described as: For all $n, \ 1 \leq n \leq F_s$, $\forall i_1, i_2 \in V$,

$$|\sum_{n_0=1}^{n} \eta^{l_1,n_0}_{i_1,k_1} - n_1| + |\sum_{n_0=1}^{n} \eta^{l_2,n_0}_{i_2,k_2} - n_2| \geq 1, \ \forall k_1, k_2 \in S \cup A, \ (n_1, n_2) \in IT(\psi^{l_1}_{i_1,k_1}, \psi^{l_2}_{i_2,k_2})$$

5.1.5. ***Dependencies between the schedules.*** Communication and computation schedules must be aligned, meaning that measured data (i.e., data from sensors) is routed to the controller prior to the VT's activation. Also, data designated to the actuators are

[6]Future extensions of this work will allow CAs to have different sampling periods.

[7]In the constraint formulation we assume that each VT can be executed in one time slot. In general this might not be the case. However, it would just require a formulation change where instead of 1, execution time necessary for execution of the j^{th} VT (i.e., e_j) is placed. Even more general, if the network contains nodes with different computational power, the previous term should be expressed as $\sum_{i=1}^{m}(x^a_{i,j} \cdot e^a_j + x^b_{i,j} \cdot e^b_j)$. To simplify the notation, we decided to use the aforementioned assumption.

forwarded after the computation of the VT: For all $n,\ 1 \leq n \leq F_s$

$$\eta^{l,n}_{i,k_s} \leq (1 - \sum_{n_0=1}^{n} \mu_j^{n_0}), \quad \eta^{l,n}_{i,k_a} \leq (\sum_{n_0=1}^{n-1} \mu_j^{n_0}), \forall j \in \mathbb{J}, \forall i \in V,\ \forall l, k_s \in S_j, k_a \in A_j,$$

5.1.6. ***Objective function****.* The goal of the assignment procedure is to minimize the aggregate number of used links while maximizing the aggregate link quality. In addition, we want to maximize the use of disjoint routing. Thus, a cost for sharing links is introduced, both in paths from sensors to controllers and from the *Primary* controller to the actuators. As can be seen, the objective function (i.e., cost) does not depend on utilized scheduling. Therefore, it is defined as a weighted sum $f(\mathbf{x}, \mathbf{y}) = w_1 f_{LN} + w_2 f_{LQ} + w_3 f_{SL}$, where weights $w_1,\ w_2$ and w_3 are used to emphasize impacts of the following cost functions:

(1) Aggregate number of used links: $f_{LN}(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{p} (\sum_{l,k,i} y^l_{i,k} \cdot d(\psi^l_{i,k}))_j$, where $d(\psi^l_{i,k})$ is a distance (i.e., length, number of hops) of path $\psi^l_{i,k}$.
(2) Negative aggregate link quality: $f_{LQ}(\mathbf{x}, \mathbf{y}) = -\sum_{j=1}^{p} (\sum_{l,k,i} y^l_{i,k} \cdot LQ(\psi^l_{i,k}))_j$
(3) Cost of the shared links:

$$f_{SL}(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{p} \sum_{\substack{l_i, l_t, \\ k \in S_j \cup A_j}} \sum_{1 \leq i \leq t \leq m} y^{l_i}_{i,k} \cdot y^{l_t}_{t,k} \cdot SH(\psi^{l_i}_{i,k}, \psi^{l_t}_{t,k}),$$

where $SH(\psi^{l_i}_{i,k}, \psi^{l_t}_{t,k})$ is a number of links shared between paths $\psi^{l_i}_{i,k}$ and $\psi^{l_t}_{t,k}$.

Therefore, the assignment problem can be formulated as a binary integer programming optimization problem and solved using some of the well-known techniques (branch and bound) [Schrijver 1998]. One caveat is in order. Since the problem formulation has a large number of decision variables, even for a small network it can be computationally expensive to solve the problem. Thus, we translated the problem into the satisfiability problem, by transforming each constraint into conjunctive normal form (CNF) (for details see [Pajic and Mangharam 2009]). The satisfiability problem is then solved using zChaff [Fu et al. 2004], a very efficient satisfiability solver. This allows us to solve the previous problem in real-time even for large scale networks.

5.2. Problem Relaxation

When only one node in the VC can transmit in each time slot, the number of slots needed to send a message from node v_1 to node v_2 is equal to the distance between the nodes. This is used for the relaxed problem formulation, as it eliminates the need to include communication and computation decision variables used in the general formulation and, therefore, significantly reduces complexity of the optimization problem. In addition, the collision-free communication requirement, which is the most complex set of constraints from the general formulation, becomes redundant. The requirement is inherently fulfilled with the policy that allows a single transmission per time slot for the whole VC.

As the first step for the problem formulation, two maximum node-disjoint paths r^1_{i,a_c} r^2_{i,a_c} are determined for each node v_i and each actuator a_c. The existence of two node disjoint paths from a node to all sensors and actuators can be checked using Menger's theorem [Böhme et al. 2001] (for details see [Pajic and Mangharam 2009]). When two node-disjoint paths exist for the node, using a polynomial time algorithm (MIN-SUM 2-paths [Yang et al. 2005]) paths r^1_{i,a_c} r^2_{i,a_c} with the minimal total length can be determined. Otherwise, path r^1_{i,a_c} is computed in polynomial time as the shortest path to

the actuator. Path r^2_{i,a_c} is calculated as the shortest path to the actuator after removing nodes from path r^1_{i,a_c}, while preserving connectivity. Using a similar approach, for each node v_i and all its neighbors $v_{i_1}, ..., v_{i_{n_i}}$ (n_i is a degree of node v_i), a set of $n_i + 1$ paths is created between each sensor s and the nodes. We denote these distances as $(d_{i,s}, d_{i_1,s}, ..., d_{i_{n_i},s})$.

To extract the relaxed problem's formulation we used only $2mp$ binary assignment variables $x^a_{i,j}$ and $x^b_{i,j}$ defined as in the general problem formulation. This allows us to formulate the problem as follows:

$$\min w_1 \cdot f_{LN}(\mathbf{x}) + w_2 \cdot f_{LQ}(\mathbf{x}),$$

with the respect to $\mathbf{x} \in \{0,1\}^{2mp}$, which contains the aforementioned decision variables. The feasible set is described with the following set of constrains:

$$\sum_{i=1}^{m} x^a_{i,j} = 1, \quad \sum_{i=1}^{m} x^b_{i,j} = R, \quad x^a_{i,j} + x^b_{i,j} \leq 1, \quad \sum_{k \in N_i} x^b_{k,j} \geq R \cdot x^a_{i,j}, \ \forall j \in \mathbb{J}, \ \forall i \in V,$$

$$\sum_{\substack{i \in V \\ j \in \{1,\ldots,p\}}} \{ \sum_{s \in S_j} (x^a_{i,j} \cdot d_{i,s} + \sum_{k \in N_i} x^b_{k,j} \cdot x^a_{i,j} d_{i_k,s}) + \sum_{a \in A_j} x^a_{i,j} \cdot \big(d(r^1_{i,a}) + d(r^2_{i,a})\big) \} + 1 \leq F_s$$

The last constraint requires that all communication is done within one frame and therefore, meets the timing requirements necessary for the system's stability. This constraint is the only one that depends on the number of VTs and utilized data routing. Thus, a suboptimal, yet feasible solution can be obtained (if and only if a feasible solution exists) using *compositional analysis*. In this case each control loop, operating across the same physical set of controllers is considered separately. Optimizing only for the cost function f_{LN} and for each loop separately provides an optimal assignment for each loop that uses the minimal number of communication slots (details see in [Pajic and Mangharam 2009]). Note that if $w_1/w_2 >> 1$, the approach provides the optimal solution for the relaxed assignment problem in general. Also, for a sufficiently high link quality threshold (while deriving graph G_T) the impact of function f_{LQ} is reduced. This enables use of the compositional design, which significantly simplifies the system analysis and schedule extraction. Since the EVM is focused on networks with less than 20 nodes, we are able to run the optimization algorithm on all nodes in a VC, as the *VT Assignment Procedure*.

6. EVM RUNTIME OPERATION: VIRTUAL TASK EXECUTION

Given the task migration mechanisms and the algorithms to (re)assign tasks, we now describe the relationship between primary and backup nodes for planned and unplanned scenarios. More specifically, we consider the criterion for triggering task migration and the node and network schedulability analysis that must be conducted prior to migration. To completely address the issues in wireless networked control systems, we must consider (a) the mechanisms for runtime adaptation, (b) the algorithms for runtime task (re)assignment to physical nodes and (c) the fault tolerance policy. In this paper we focus on the first two aspects and apply them to simple network models with non-Byzantine single node and link failures. As the fault tolerance policy is dependent on the control application and fault/failure model is a function of the specific environment, we do not consider specific policies here. We aim to address Byzantine errors such as software errors in future work.

6.1. Adaptation to Planned and Unplanned Network Changes

Planned adjustments occur in situations when a *Primary* node is informed of changes in VC state (e.g., when a node detects that its battery level is below some threshold). To determine a *Backup* node to migrate its task, the *Primary* node has to execute computation and communication schedulability analyses in $k = 1$-hop neighborhood and select a *Backup* node that maximizes the communication slack value while maintaining computation schedulability.
For unplanned changes caused by potential failures we consider the following cases:

— The *Primary* nodes dies: Computation and communication schedulability analysis in $k = 1$-hop neighborhood is initiated. Since state data of the *Primary* node is maintained at *Backup* nodes, a new *Primary* node continues VT execution.
— A *Backup* node dies: The *Primary* node detects the *Backup* has died and selects a new *Backup* from one of its neighbors.
— A forwarding node dies or a link's quality goes below some criterion: The detection of a forwarding node failure is performed by its predecessor/successor on the routing path. Again, a communication schedulability analysis is performed (only for the affected sensor and actuator) to determine a new routing scheme.

To decrease response time for the schedulability analyses, each node uses its idle computation time to calculate in advance the optimal reaction to a set of potential failures. Besides decreasing the response time, this approach enables triggering the execution of the *Assignment Procedure* if it is determined that for some failures there is no adjustments that can meet all of the constraints. Also, if the procedure can not derive a feasible assignment, an alarm is raised notifying system operators to add more nodes in the network to prevent a potential failure.

6.2. Communication Schedulability Analysis

The goal of communication schedulability is to determine whether we can *incrementally* reassign the available communication slots due to the change in the task assignment, without executing a global reassignment of communication slots. To accomplish this we determine the current communication slack and evaluate if it is sufficient for the incremental slot reassignment. When a VT is to be migrated from a node v_i to a node v_j, we define sets S_{VT} and A_{VT} of all sensors and actuators respectively, related to the VT. Also, for each $s \in S_{VT}$ we denote as $v^k_{i,s}$ a node that is k-hops away from node v_i on the route from sensor s to node v_i. Similarly, for each $a \in A_{VT}$, $v^k_{i,a}$ denotes a node that is k-hops away from node v_i on the route to the actuator a. In addition, we denote as N^i_u the number of unused time slots in the time interval between the first slot in which *all* nodes $v^k_{i,s}$ were suppose to receive values from sensors in S_{VT} and a first slot in the frame in which *at least one* node $v^k_{i,a}$ was scheduled to receive information from the node v_i. The parameter k determines the set of candidate backup nodes to which the task may be reassigned.

More specifically, the goal of communication schedulability is to determine whether we can reassign (with the respect to the current communication schedule) the available communication slots and slots used to send data in the k-hop neighborhood of a node v_i. The re-assignment should re-route all sensor and actuator data from these nodes to node v_j. A new feasible communication schedule can be generated if $\Delta \geq 0$, where Δ denotes communication slack value defined as:

$$\Delta = \sum_{s \in S_{VT}} d(v_i, v^k_{i,s}) + \sum_{a \in A_{VT}} d(v_i, v^k_{i,a}) + N^i_u - \sum_{s \in S_{VT}} d(v_j, v^k_{j,s}) - \sum_{a \in A_{VT}} d(v_j, v^k_{j,a}),$$

where $d(v_p, v_q)$ is the distance between nodes v_p and v_q. If more than one task is migrated from a node, similar analysis is performed with the previous equation adjusted to contain sums of all sensors and actuators related to the tasks. In addition, if tasks should be migrated from node v_i to separate nodes, the schedulability test is performed on a pairwise basis.

6.3. Computation Schedulability Analysis

For the computation schedulability analysis we use standard real-time response analysis [Liu 2000] and the mode-change protocol, presented in [Sha et al. 1989] and [Real and Crespo 2004], adapted for the EVM. Consider a node v_i that executes a task set $\mathbb{T} = \{T_{i_1}, ..., T_{i_m}, VT_{i_1}, ..., VT_{i_n}\}$, where tasks T_{i_j} are local, node specific tasks, while tasks VT_{i_j} are VTs assigned to the node (in descending order of priority). We define a set $HP\_VT(T)$ as a set of all VTs with higher priority than local task T and, similarly, a set $HP\_T(VT)$ as a set of all node-specific tasks, with higher priority than task VT. To allow an assignment of a new VT, a schedulability analysis is performed where both active and inactive tasks are considered as active. Although this approach is conservative, it eliminates the need for repeated schedulability analysis prior to tasks activation. Each node-specific task is denoted as $T_j = (p_{T_j}, e_{T_j})$ and each VT as $VT_j = (p_{VT_j}, e_{VT_j}, \phi_{VT_j}, d_{VT_j})$ (period, execution time, offset and deadline respectively). Schedulability of a new task set is performed by checking only the schedulability of each task with a lower priority than the new virtual task VT_k, using its time-demand function $w(t)$ [Liu 2000].

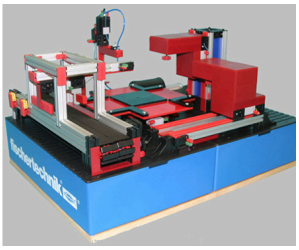
As mentioned in Section 5, we currently consider the case where all VTs have the same execution period. Since execution of a VT is triggered by the reception of sensed signals and must be finished before its scheduled communication to actuators, its deadline is significantly lower than its period. Thus, from a VT's activation till its deadline, all other VTs can be active at most once, so for a task $VT_i,\ i \geq k$:

$$w_{VT_i}(t) = e_{VT_i} + \sum_{j \in HP\_T(VT_i)} \left\lceil \frac{t}{t_{T_j}} \right\rceil \cdot e_{T_j} + \sum_{j=1}^{i-1} e_{VT_j}$$

The equation is too conservative as it assumes that all VTs can be activated at the same time. However, VTs are activated when a last radio message containing necessary data is received. In addition, since all VT's periods are multiples of TDMA slot duration, when a communication schedule is known, all possible offset combinations of a task activation can be easily calculated. Therefore, for a task VT_i, released at time t_i, for all possible combinations of release times t_j of VTs with higher priority, the time-demand function for $t \geq t_i$ is defined as:

$$w_{VT_i}^{(t_0,t_1,...,t_{i-1})}(t) = e_{VT_i} + \sum_{k \in HP\_T(VT_i)} \left\lceil \frac{t}{t_{T_k}} \right\rceil \cdot e_{T_k} + \sum_{\substack{j=1, \\ t_j \leq t \leq t_i + d_i}}^{i-1} \min(e_{VT_j}, t - t_j) + \sum_{\substack{j=1, \\ t_i \in [t_j, t_j + d_j]}}^{i-1} \min(e_{VT_j}, t_j + d_j - t_i)$$

Here the second term corresponds to the execution of all higher-priority native tasks; the third term corresponds to the demand from higher-priority VTs which are activated after the i^{th} task's activation, but before its deadline. Finally, the last term describes the demand of the higher priority VTs when the i^{th} task is activated between the higher priority tasks' activation and deadline. For schedulability we are interested in time instances where $w_{VT_i}^{(t_0,t_1,...,t_{i-1})}(t) = t$. These points can be obtained using effi-



(a) Work-cell module

Component	Sensors			Actuators	
	Proximity Home	Proximity Out	IR Position	Non-reversing Motor	Reversing Motor
Conveyor belt			X		
Ram	X	X		X	X
Horizontal slotting mill • Horizontal direction • Vertical direction	X X	X X		X	X
Vertical Gluing Machine	X	X	X	X	
Table Rotation				X	X
Vertical boring machine	X	X			X

(b) Module components

Fig. 10. FischerTechnik factory module with 22 sensors and actuators

cient recurrence procedure described in [Liu 2000]. The task is schedulable, if for all combinations of activation times, the solution of recurrence procedure is less than the task's deadline (d_{VT_i}).

Although the previous equation seems complicated, in the case when all VTs are executed once per frame there is only one combination of release times $(t_0, t_1, ..., t_{i-1})$ (i.e., only one set of task offsets as the TDMA schedule is fixed). Even in general case there is no need to cover a large number of possible combinations since for most control systems, all loops usually have the same sampling period or all sampling periods are integer multiples of one of the periods.

A similar approach is used for schedulability analysis of a node-specific task T_i.

7. IMPLEMENTATION

To evaluate the EVM's performance in a real setting with multiple coordinated controller operations, we used a factory simulation module shown in Figure 10(a). The FischerTechnik model factory consists of 22 sensors and actuators (Figure 10(b)) that are to be controlled in a coordinated and timely manner. A block of wood is passed through a conveyor, pushed by a rammer onto a turn table and operated upon by up to three milling/cutting/pneumatic machines. The factory module was initially controlled by wired programmable logic controllers (PLCs). We converted it to use wireless control with FireFly embedded wireless nodes [Mangharam et al. 2007] controlling all sensors and actuators via a set of electrical relays. FireFly is a low-cost, low-power platform based on Atmel ATmega1281 8-bit microcontroller with 8KB of RAM and 128KB of ROM along with a Chipcon CC2420 IEEE 802.15.4 standard-compliant radio transceiver. FireFly nodes support tight global hardware-based time synchronization for real-time TDMA-based communication with the RT-Link protocol [Rowe et al. 2008]. The EVM also works on TI MSP430 architectures.

In our experiments we demonstrate:

1. On-line capacity expansion when a node joins the VC.
2. Redistribution of VTs when adding/removing nodes.
3. Planned VT migration triggered by the user.
4. Unplanned VT migration due to a node or a communication link failure.
5. Multiple coordinated work-flows.

We tested the setup with a batch of 10 input blocks consisting of 3 different types which require different processing procedure. This is an example of the logical benefits of the EVM as it enables a more agile form of manufacturing. Details about the experiments, along with the videos can be seen in [evm 2009].

8. CASE STUDY

As this is an early effort to describe the main functionalities of the EVM, we limit our case study to a simple simulated control network. We simulated the performance of the EVM for the case when a wireless networks is used for control in the Shell Problem, a well-known problem from process control theory concerning control of a heavy oil fractionator [Lewin 2009], [Prett and Morari 1986]. The controlled variables (outputs) are differences of the top product end point ($Y1$) and the bottom reflux temperature ($Y2$) from predefined (reference) values. Figure 11(a) presents a Simulink framework used for the simulation, where *Controller* (shown in Figure 6) and *Plant* are similar to models from [Lewin 2009]. The major difference is that *Plant*'s dynamics was sped up to be able to test system's performance.

The functional description of the VT, shown in Figure 7, is derived as described in Section 3. Since all continuous outputs of the *Plant* have to be sampled before processed with a discrete-time controller, the sampling period defined in *SampleAndHold* blocks in the Simulink model is used to extract the period of each VT.

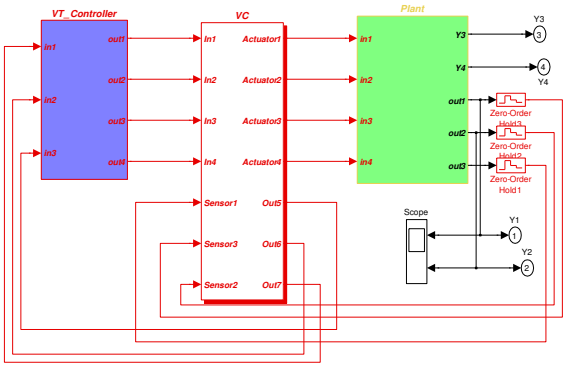
Figure 11(b) presents the initial topology of the VC along with the *Primary* and the *Backup* node. To be able to address the effects of message drops, we assigned each link in the network a Packet Delivery Ratio (PDR) that is less than 1 (i.e., 100%). A TDMA protocol with 32 slots per frame is used for communication between nodes, where 24 slots were used for transfer of data related to the control problem, while 8 remaining slots per frame were used to exchange messages about VC's status. The system response to a series of different step inputs (a new one was set to arrive every 60s) for the initial topology is presented in Figure 11(d). Also, a scenario was simulated where the initial topology changes after some of the links fail (as shown in Figure 11(c)). Figure 11(e) presents the response of the system without the EVM, where only re-routing algorithms are used without changing positions of the *Primary* and *Backup* nodes. This results in a system response that rapidly deteriorates. The system becomes unstable, due to increase in end-to-end communication time from all sensors to the *Primary* node to all actuators.

Figure 11(f) shows how the EVM's adaptation to unplanned changes in link quality keeps the system's response similar to that in the initial topology. For the case presented in Figure 11(f), we simulated the system when at time $t = 60s$ the network topology changes to that presented in Figure 11(c). Due to the task re-assignment, one execution of the control algorithm is omitted, but as it can be seen, without significant influence to the overall system performance. This was expected since, from the perspective of the *Plant*, this case is equivalent to packet drops, which already occurs due to the fact that PDR is less than 100%.

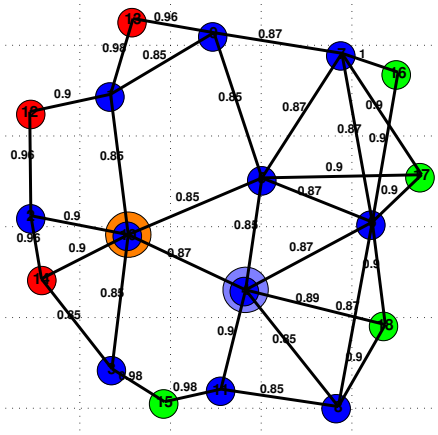
9. LIMITATIONS FOR THE EVM APPROACH

• **Complexity of Consensus:** The complexity of reaching consensus forces our current implementation to maintain a substantial amount of state information with a relatively high update frequency. This limits the scalability of the current EVM approach to small networks with $\leq$20 nodes. While this is 'good enough' for a large number of small embedded wireless control applications such as natural gas processing with slowly varying operating parameters, it is essential to explore distributed algorithms to maintain state across the virtual component.

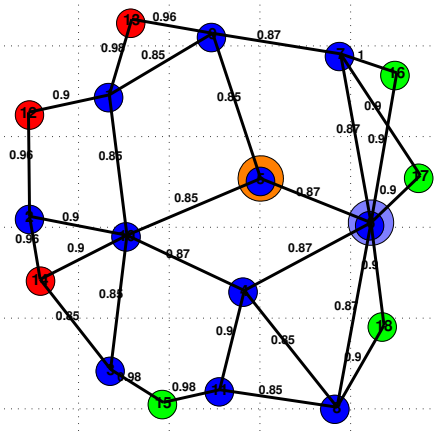
• **Centralized Approach:** The centralized algorithm has been used to solve the assignment problem. This limitation motivated us to explore a distributed solution for incremental strategies for control-loop implementation. Using the entire node population within a virtual component as a distributed controller would remove the need for the virtual task's assignment procedure.



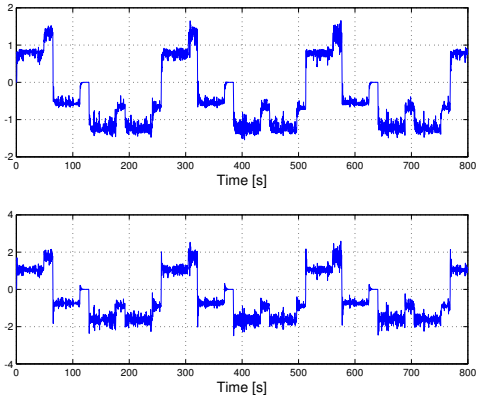
(a) Simulink model for the Shell problem



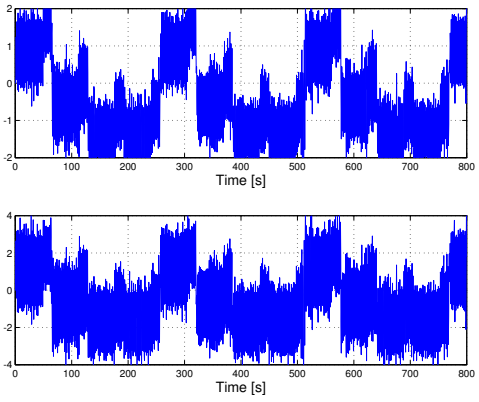
(b) Initial network topology



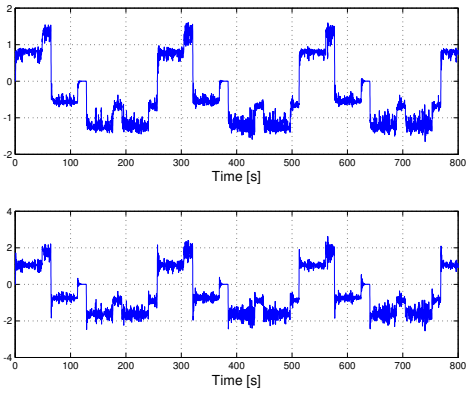
(c) Topology after link failures



(d) System response for initial configuration, showing outputs Y1 (top) and Y2 (bottom)



(e) System response when EVM is not used (when only re-routing is used), Y1 (top) and Y2 (bottom)



(f) System response when EVM adapts to changes in network conditions, Y1 (top) and Y2 (bottom)

Fig. 11. Simulation of EVM behavior when used for 'Shell problem' control; Nodes: green - actuators, red - sensors, blue circle - the *Primary* node, orange circle - the *Backup* node.

More recently, we have developed a set of distributed extensions to the EVM. In [Pajic et al. 2011], a new concept of the Wireless Control Network has been introduced, where the network (i.e., VC) acts as the controller *itself*. This approach for control, (a) does not require continuous task assignment calculation, (b) does not require complex scheduling protocols, (c) is simple in implementation and (d) enables a compositional system design.

10. CONCLUSION

This paper presents an initial stab at a problem that unravels series of difficulties at the heart of networked Cyber-Physical Systems. We have investigated several fundamental challenges with the use of wireless networks for time-critical closed-loop control problems. Our approach was to build the networking infrastructure to maintain state across physical node boundaries, allowing tasks to be decoupled from the underlying unreliable physical substrate. We present a modular architecture used for control applications in wireless sensor/actuator/controller networks that allows component integration and system reconfiguration at runtime, without any negative effects on the execution of already assigned functionalities. The EVM enables a simple transition from the controller design in widely used simulation tools to the actual, physical 'plug-and-play' deployment for wireless networks.

To show system's capabilities we present an IP formulation of the runtime task assignment problem and show that it is possible to compute task assignment efficiently and in a composable manner across concurrent control problems. We implemented an early version of the EVM infrastructure on commodity embedded nodes and demonstrated the capability in an all-wireless factory across 22 sensors/actuators.

ACKNOWLEDGMENTS

We wish to thank George J. Pappas for fruitful discussions. The authors would also like to thank Paul McLaughlin from Honeywell Process Solutions for his support and feedback. We are grateful to the reviewers for very valuable comments that were essential in improving the paper. This work was partially supported by the NSF CPS-0931239, CSR-0834517, MRI-0923518 and Honeywell grants. Preliminary versions of this paper have appeared in [Pajic and Mangharam 2010] and [Mangharam and Pajic 2009].

REFERENCES

2009. EVM website - http://mlab.seas.upenn.edu/evm.

ALUR, R., D'INNOCENZO, A., JOHANSSON, K. H., PAPPAS, G. J., AND WEISS, G. 2009. Modeling and analysis of multi-hop control networks. In *RTAS '09: Proceedings of the 2009 15th IEEE Symposium on Real-Time and Embedded Technology and Applications*. 223–232.

BHATTI, S., CARLSON, J., DAI, H., DENG, J., ROSE, J., SHETH, A., SHUCKER, B., GRUENWALD, C., TORGERSON, A., AND HAN, R. 2005. MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms. *Mobile Networks and Applications 10,* 4, 563–579.

BÖHME, T., GÖRING, F., AND HARANT, J. 2001. Menger's Theorem. *Journal of Graph Theory, vol. 37 31,* 1, 35–36.

BROWN, M., GILBERT, S., LYNCH, N., NEWPORT, C., NOLTE, T., AND SPINDEL, M. 2007. The Virtual Node Layer: A programming abstraction for wireless sensor networks. *SIGBED Rev. 4,* 3, 7–12.

CAO, Q., ABDELZAHER, T., STANKOVIC, J., AND HE, T. 2008. The LiteOS Operating System: Towards Unix-Like Abstractions for Wireless Sensor Networks. In *IPSN '08: Proceedings of the 7th international conference on Information processing in sensor networks*. 233–244.

CERVIN, A., EKER, J., BERNHARDSSON, B., AND ARZEN, K. E. 2002. Feedback feedforward scheduling of control tasks. *Real-Time System Journal 23,* 1-2, 25–53.

CONKLIN, E. K. AND RATHER, E. D. 2007. *FORTH Programmer's Handbook*. FORTH Inc.

DUNKELS, A., GRONVALL, B., AND VOIGT, T. 2004. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*. 455–462.

FU, Z., MAHAJAN, Y., AND MALIK, S. 2004. New Features of SAT'04 version of zChaff. In *The International Conference on Theory and Applications of Satisfiability Testing*.

GRAHAM, S., BALIGA, G., AND KUMAR, P. 2009. Abstractions, architecture, mechanisms, and a middleware for networked control. *IEEE Transactions on Automatic Control 54,* 7, 1490–1503.

GUMMADI, R., GNAWALI, O., AND GOVINDAN, R. 2005. Macro-programming wireless sensor networks using Kairos. In *Distributed Computing in Sensor Systems*. Springer Berlin, 126–140.

GUPTA, V., DANA, A., HESPANHA, J., MURRAY, R. M., AND HASSIBI, B. 2009. Data transmission over networks for estimation and control. *IEEE Trans. on Automatic Control 54,* 8, 1807–1819.

HAN, C.-C., KUMAR, R., SHEA, R., KOHLER, E., AND SRIVASTAVA, M. 2005. A dynamic operating system for sensor nodes. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*. ACM, 163–176.

HART. 2007. *Field Communication Protocol Specification, Rev 7*.

HESPANHA, J. P., NAGHSHTABRIZI, P., AND XU, Y. 2007. A survey of recent results in networked control systems. *Proceedings of IEEE, Special Issue on Technology of Networked Control Systems 95,* 1, 138–162.

HILL, J., SZEWCZYK, R., WOO, A., HOLLAR, S., CULLER, D., AND PISTER, K. 2000. System architecture directions for networked sensors. *SIGPLAN Notices 35,* 11, 93–104.

JALOTE, P. 1994. *Fault tolerance in distributed systems*. Prentice-Hall, Inc.

KOPETZ, H. AND BAUER, G. 2003. The Time-Triggered Architecture. *Proceedings of the IEEE 91,* 1, 112–126.

LEE, P. A. AND ANDERSON, T. 1990. *Fault Tolerance - Principles and Practice*. Springer Verlag.

LEVIS, P. AND CULLER, D. 2002. Maté: a tiny virtual machine for sensor networks. *SIGARCH Computer Architecture News 30,* 5, 85–95.

LEWIN, D. R. 2009. *Using Process Simulators in Chemical Engineering: A Multimedia guide for the Core Curriculum*. Wiley.

LIU, J. 2000. *Real-Time Systems*. Prentice Hall, Inc.

LORINCZ, K., CHEN, B.-R., WATERMAN, J., WERNER-ALLEN, G., AND WELSH, M. 2008. Resource aware programming in the Pixie OS. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, 211–224.

MANGHARAM, R. AND PAJIC, M. 2009. Embedded virtual machines for robust wireless control systems. In *ICDCSW '09: Proceedings of the 29th IEEE International Conference on Distributed Computing Systems Workshops*. 38–43.

MANGHARAM, R., ROWE, A., AND RAJKUMAR, R. 2007. FireFly: A Cross-layer Platform for Real-time Embedded Wireless Networks. *Real-Time System Journal 37,* 3, 183–231.

MATHWORKS. 2010. Simulink documentation, www.mathworks.com/access/helpdesk/help/toolbox/simulink.

MÜLLER, R., ALONSO, G., AND KOSSMANN, D. 2007. A virtual machine for sensor networks. In *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*. 145–158.

NANORK. 2010. Sensor RTOS - http://www.nanork.org.

NEWTON, R., MORRISETT, G., AND WELSH, M. 2007. The regiment macroprogramming system. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*. 489–498.

NIELSEN RESEARCH. 2006. *Downtime Costs Auto Industry*.

PAJIC, M. AND MANGHARAM, R. 2009. Embedded Virtual Machines: Technical Report.

PAJIC, M. AND MANGHARAM, R. 2010. Embedded virtual machines for robust wireless control and actuation. In *RTAS '10: Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. 79–88.

PAJIC, M., SUNDARAM, S., MANGHARAM, R., AND PAPPAS, G. J. 2011. The Wireless Control Network: A New Approach for Control over Networks. *IEEE Transactions on Automatic Control: Special Issue on Wireless Sensor and Actuator Networks 56,* 10 (Oct.), 2305–2318.

PRETT, D. AND MORARI, M. 1986. The shell process control workshop. *Butterworths*.

REAL, J. AND CRESPO, A. 2004. Mode change protocols for real-time systems: a survey and a new proposal. *Real-Time Systems Journal 26,* 2, 161–197.

ROBINSON, C. AND KUMAR, P. 2008. Optimizing controller location in networked control systems with packet drops. *IEEE Journal on Selected Areas in Communications 26,* 4, 661–671.

ROWE, A., MANGHARAM, R., AND RAJKUMAR, R. 2008. RT-Link: A global time-synchronized link protocol for sensor networks. *Ad Hoc Networks 6,* 8, 1201–1220.

SCHRIJVER, A. 1998. Theory of Linear and Integer Programming. *John Wiley & sons*.

SHA, L., RAJKUMAR, R., LEHOCZKY, J., AND RAMAMRITHAM, K. 1989. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems Journal 1,* 3, 126–140.

STANLEY-MARBELL, P. AND IFTODE, L. 2000. Scylla: A smart virtual machine for mobile embedded systems. In *WMCSA '00: Proceedings of the 3rd IEEE Workshop on Mobile Computing Systems and Applications*. 41–50.

WELSH, M. AND MAINLAND, G. 2004. Programming sensor networks using abstract regions. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*.

WILLIG, A., MATHEUS, K., AND WOLISZ, A. 2005. Wireless technology in industrial networks. *Proceedings of the IEEE 93,* 6, 1130–1151.

YANG, B., ZHENG, S., AND LU, E. 2005. Finding two disjoint paths in a network with α^+-min-sum objective function. *Algorithms and Computation, Lect. Notes in Comp. Science*, 954–963.

ZHANG, W., BRANICKY, M., AND PHILLIPS, S. 2001. Stability of networked control systems. *IEEE Control Systems Magazine 21,* 1, 84–99.

Received February 2010; revised August 2010; accepted November 2010