Implementation of Virtual Sensors in Body Sensor Networks with the SPINE Framework

Nikhil Raveendranathan*, Vitali Loseu*, Eric Guenterberg*, Roberta Giannantonio§,
Raffaele Gravina†‡, Marco Sgroi†, Roozbeh Jafari*
*Embedded Systems and Signal Processing Lab, University of Texas at Dallas, Dallas, TX 75080
†Wireless Sensor Networks Lab, Telecom Italia, Berkeley, CA 94720
‡Department of Electronics, Informatics, and Systems, University of Calabria, Rende, Italy
§Telecom Italia, Torino, Italy

***Abstract*—Signal processing for Body Sensor Networks usually comprises multiple levels of data abstraction, from raw sensor data to data calculated from processing steps such as feature extraction and classification. This paper presents a multi-layer task model based on the concept of Virtual Sensors (VS) to improve architecture modularity and design reusability. In our pilot application of gait parameter extraction, VS are abstractions of components of BSN classification systems that include sensor sampling and processing tasks and provide data upon external requests analogous to the function of physical sensors. The paper presents an extension of the SPINE Framework including a new buffer management scheme that facilitates the VS implementation.**

I. Introduction

The wide variety of potential applications for Wireless Sensor Networks (WSNs) in conjunction with severe resource constraints [8] make it difficult to design an operating system that is both sufficiently lightweight and powerful. Because of resource abundance, traditional desktop operating systems provide APIs for many different problems and domains. However, operating systems for WSNs, such as TinyOS [7], provide only minimal facilities, such as communication and process scheduling. For problems which share common structure, the abstraction provided by a software framework can enhance developer productivity and exploit problem-specific optimizations.

Middleware exploiting such structure for conventional wireless sensor networks includes TinyLIME [2] and GSN (Global Sensor Network) [1]. In the context of the wireless networks the idea has been explored in many works [6], [9], [1]. Authors in these works propose generic abstractions for all system components, which allow for easier integration of new components and functionality on the system.

Body Sensor Networks (BSNs) represent a new application domain which has received much attention recently due to potential benefits for medicine, entertainment, and training. There are several properties of BSNs not present in standard WSN deployments, including single-hop networking, a significantly more powerful node called a base station, and applications requiring extensive signal processing and pattern recognition. These properties offer considerable potential for optimization and simplification. SPINE (Signal Processing in Node Environment) [4] is a software framework that aims to enable efficient implementations of signal processing on sensor nodes in BSN. The framework allows application of multiple features on to the sampled sensor data before sending it to the base station. In this paper, we present a task abstraction layer built on top of SPINE, called Virtual Sensors (VS), which allow multiple layers and stages of processing. We further describe the virtual sensor implementation of an HMM-based module to extract temporal parameters from gait.

Corresponding author Nikhil Raveendranathan nikhilr@student.utdallas.edu

II. The SPINE Framework

SPINE is an open source domain-specific framework designed to support flexible and distributed signal-processing for wireless sensor network systems. The main goal of SPINE is to provide WSN developers with support for rapid prototyping of signal-processing applications. Other design principles include code reusability and efficiency, application interoperability and specific support for sensing operations.

SPINE is an application-level framework nominally independent of the specific network topology and underlying low-level communication protocol. However, the current implementation focuses on star-topology networks with multiple sensor nodes and a single coordinator node. This network architecture is particularly suitable for Body Sensor Networks as well as vehicle automation systems and home automation applications.

III. Virtual Sensor Architecture

Physical sensors map an observed physical quantity, such as temperature, acceleration, or sound, onto a data value and produce an output. The output is generated when inputs change, as the result of an event, or in response to a request. Sensors are transducers converting values from one form to another using physical processes. Signal processing algorithms convert values using digital processes. This observed similarity is the motivation behind the virtual sensor abstraction. Virtual sensors may be implemented directly in a programming language, or as networks of other virtual sensors. Every processing task

is represented as a virtual sensor, resulting in a multi-level hierarchy, as shown in Fig. 1.

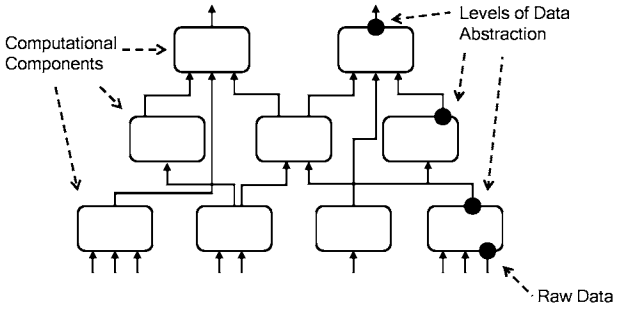

Fig. 1. Example of Signal Processing

Fig. 2 provides a general overview of the virtual sensor system architecture. A user requests certain outputs given specified inputs. This request is handled by the Virtual Sensor Manager, which configures a set of virtual sensors to handle the computational task. Virtual sensors use the Buffer Manager to setup communication with the help of buffers. Once configured, the system activates, and virtual sensors collaborate to produce the final outputs.

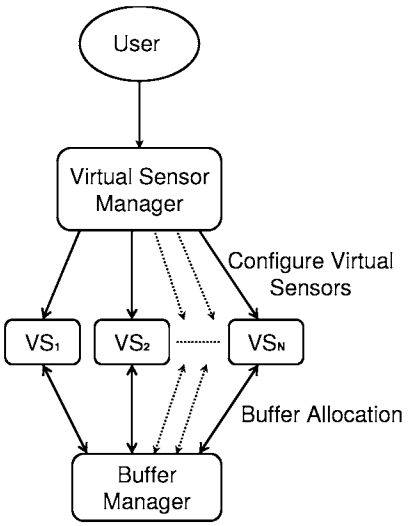

Fig. 2. System Architecture Overview

A. A Virtual Sensor

Software frameworks often provide hardware abstractions, which isolate the users from eccentricities of individual hardware by supplying a uniform method of accessing hardware of a certain type. Virtual sensors provide this type of abstraction by allowing signal processing tasks to be connected to each other without either having knowledge of the connection: virtual sensors have no knowledge of the provenance of their inputs nor the destinations of their outputs. Also, the virtual sensor system propagates outputs as soon as possible after all inputs arrive for a virtual sensor arrive. This removes the overhead of propagating a data through the network. For example, every computational component in Fig. 1 can be replaced with a virtual sensor. The output of each virtual sensor is defined by a set of inputs and its configuration. Based on this observation, a virtual sensor i denoted as VS_i, and is defined as

$$VS_i = \{I_i, O_i, C_i\} \tag{1}$$

where I_i denotes the set of inputs, O_i denotes the set of outputs, and C_i denotes the configuration of VS_i. The configuration of each virtual sensor defines the type of its inputs and outputs, the particular implementation used for a given computation task, and a set of parameters required for a particular implementation.

$$C_i = \{\vec{t}_{in}, \vec{t}_{out}, a, p\} \tag{2}$$

where $\vec{t}_{in}$ is a vector that describes the types of inputs I_i, and $\vec{t}_{out}$ is defined similarly for the outputs O_i. The specific VS implementation is denoted by a. If the user does not specify a, the Virtual Sensor Manager (described below) will choose the implementation. Configuration parameters for the VS are specified by p.

This definition provides modularity for system designs. Different configurations of the same virtual sensor can be easily substituted without inducing changes in the rest of the design. This property can be very useful when empirically searching for the best implementation of a particular signal processing component for a given application. Alternative implementations do not need to be loaded into main memory at all times. They can be stored in flash memory, or transferred from a base station upon request.

B. Virtual Sensor Manager

Once all virtual sensors are initialized, no additional control is required during execution. However, initialization requires significant support from the Virtual Sensor Manager (VSM). The VSM is responsible for creating, and configuring virtual sensors and connections between virtual sensors. The following subsections will describe the functionality of the VSM.

1) Virtual Sensor Configuration: Changes in inputs, virtual sensors, or connections may invalidate the current configuration, therefore reinitialization could happen any time. For example, a Fig. 3(a) describes a system that takes a temperature reading in Fahrenheit, and a heart rate in beats per minute. In Fig. 3(b) a new thermometer, that produces output in Celsius, is introduced. VS_1 has to be reconfigured to handle the new conditions. To be able to configure/reconfigure the system at run time, the VSM accesses a table that maps each available combination of possible inputs and outputs to the appropriate virtual sensor implementation. This can be represented by the set A. Each entry $a \in A$ contains:

$$a = \{\vec{t}_{in}, \vec{t}_{out}, \psi\} \tag{3}$$

where ψ is the virtual sensor implementation.

If the modification is not drastic enough to necessitate changing the implementation method, reconfiguration can alter parameters of a given implementation. During configuration of a virtual sensor, VSM includes the address of the selected implementation and the required parameters in the configuration message C_i.

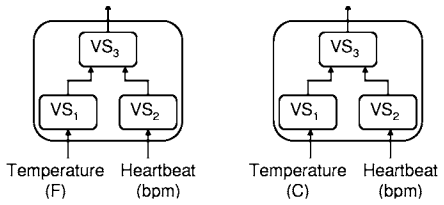

Fig. 3. Example of Input Modification

2) System Configuration: While individual virtual sensors do not hold any information about other virtual sensors, the system relies on their cooperation. At the beginning of the system execution, the VSM receives the VS topology configuration graph. Based on the requirements of the topology configuration, the VSM initializes the appropriate VSs and connects them as required. Input and output types are a property of each virtual sensor. An output of one of the virtual sensors is also an input of another virtual sensor. For example, in Fig. 4, configuration of VS_3 and VS_4 depends on the input they receive from VS_1. To simplify the configuration and reconfiguration process, the VSM initializes VSs in a specific order, to meet the requirement that each virtual sensor cannot be created until all inputs are initialized. This ordering can be determined with a topographical sort of the topology configuration graph.

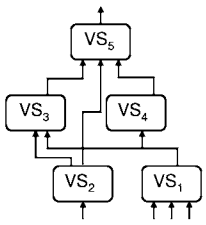

Fig. 4. Example of Input/Output Dependency

C. Buffer Manager

Signal processing for BSNs often relies on combining data from multiple locations. As a result, virtual sensors can have multiple inputs from different sensor nodes. To avoid synchronization issues, virtual sensors implicitly use buffers for communication. The Buffer Manager (BM) controls dynamic buffer allocation and manages data flow in the system.

When a virtual sensor is created and configured, it initiates a storage buffer for its output. The virtual sensor contacts the BM and requests the creation of a buffer sufficient to hold its output. The BM allocates a circular buffer of the required size and returns the `bufferID`. This `bufferID` is propagated by the VSM to other virtual sensors that are interested in data of this particular buffer. To read from a buffer, a virtual sensor must register with the buffer as a reader, specifying the number of samples used per read. Every time the producer writes to the buffer, the BM checks if the buffer has enough information for any of the readers, and signals readers when they can access the data. Fig 5 shows an overview of the BM operation. It shows that BM keeps track of buffers by ID, tracking the points where the producer is writing to, and where each individual reader is reading from.

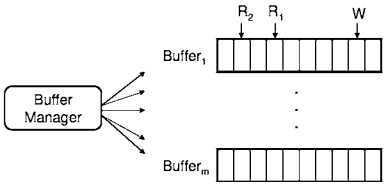

Fig. 5. Buffer Manager Overview

If the producer VS is reconfigured, and its output is changed, the BM removes the buffer that associated with the previous output and initiates a new buffer, based on the new configuration information.

IV. Virtual Sensors in SPINE

As a test application for virtual sensors in SPINE, we implemented a system for automatic event annotation based on a left-right Hidden Markov Model (HMM) [5]. This approach can identify key events within a movement. The HMM annotation system consists of four parts: 1) acceleration data is collected by physical sensors, 2) it is filtered, 3) features are extracted, and 4) the HMM annotates the events. The system uses a single sensor node, and only one implementation was created for each virtual sensor.

Based on the HMM model and assumptions we made, we defined four virtual sensors described in Fig. 6. Arrows connecting virtual sensors denote the data flow from the producer to the consumer. The initial request is generated by the Java application code running on a PC. This request initiates the configuration of each of the four virtual sensors. Virtual sensors are created as described in Section III-B1. Once all the virtual sensors have been initialized, the HMM virtual sensor starts producing output every sample. The sampling interval is defined for the accelerometer VS, which acts as the data source for the whole model. The event-driven model of programming supported by nesC [3] is well-suited for this kind of application.

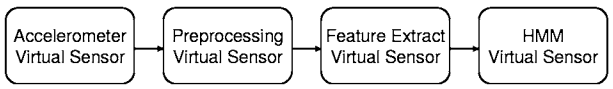

Fig. 6. HMM Application Example

A. Implementation

Fig. 6 shows the different virtual sensors as blocks. The layered model of the event annotation system allowed us to develop and verify one component at a time. We started with a MATLAB model as described in [5]. The code was simplified to match sensor node capabilities, e.g. using fixed-point math with limited precision. Then a PC version of the code developed in C, and outputs were compared. After making changes to assure matching output between the C and MATLAB code, the C code was ported to nesC for the sensor nodes. Any changes to the PC or sensor node code were ported back to MATLAB to keep the versions synchronized.

Due to the limited debugging capabilities of the sensor nodes, debugging was the most difficult part of the development process. Some of the problems which were hard to debug included high packet loss, hindering the basestation-mote communication, and stack overflow due to excessive memory usage. Other problems include freezing of system due to higher processing overhead of one component over the others. Another problem was the occurrence of "impossible" state transitions in the HMM. The problem was that the HMM involved a large number of summations, therefore the 16 bit number overflowed periodically, leading to unpredictable results. Figuring out these problems required in-depth analysis of the code and careful optimizations and corrections on appropriate components.

B. Results

For our test case, we used this system to extract heel-down and heel-lift events from a walking subject. The sensor node contained a single tri-axial accelerometer sampling at 20 Hz (the highest sampling rate achievable while performing the processing steps). The event annotation was initially quite sensitive to sensor node misplacement, so we trained it with data from one subject with ten different trials. Each trial contained approximately 50 steps and had a slightly different sensor placement. This significantly increased accuracy. The sensor node performed annotation and broadcast the raw samples so results between different implementations could be compared.

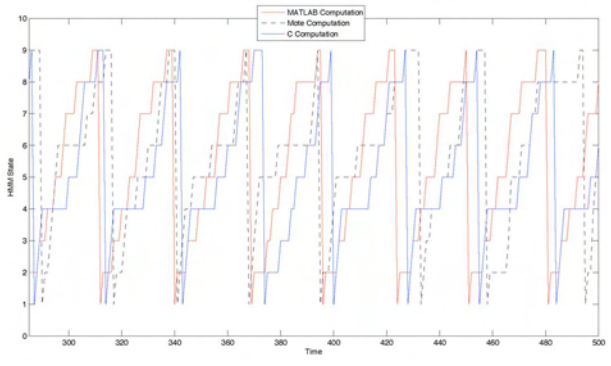

Fig. 7. Comparison of implementations

As can be seen in Fig. 7, the versions produce similar results. While the state transitions are not completely synchronized, the total times for a complete cycle are almost identical (on average, the difference is less than $10\,ms$). On the far right side of Fig. 7, a drastic difference can be observed between the sensor node output and the others. This occurs rarely, but consistently. We are still debugging the problem.

V. Research Plan

This paper describes a work in progress, and several issues must be addressed before this system can be adopted by the larger community. A major obstacle was the limited debugging capability of the motes, which lead to developing a C version on the PC. Many aspects of SPINE and Virtual Sensors are not present in the PC version, therefore porting the entire SPINE framework to the PC for simulation and debugging is critical. Furthermore, the current virtual sensor implementation is somewhat specific to the current application. More development is required to support the full generic virtual sensor model described in this paper. Also, we are working on a method to load new code modules wirelessly. This will greatly increase the flexibility of virtual sensors.

Acknowledgment

This work has been partially supported by CONET, the Cooperating Objects Network of Excellence, funded by the European Commission under FP7 with contract number FP7-2007-2-224053.

This work is also supported by Telecom Italia through technical and financial contributions.

References

[1] K. Aberer, M. Hauswirth, and A. Salehi. A middleware for fast and flexible sensor network deployment. In *Proceedings of the 32nd international conference on Very large data bases*, pages 1199–1202. VLDB Endowment, 2006.

[2] C. Curino, M. Giani, M. Giorgetta, A. Giusti, AL Murphy, GP Picco, and D.E. e Informazione. TinyLIME: Bridging mobile and sensor networks through middleware. In *Third IEEE International Conference on Pervasive Computing and Communications, 2005. PerCom 2005*, pages 61–72, 2005.

[3] D. Gay, P. Levis, R. Von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. *Acm Sigplan Notices*, 38(5):1–11, 2003.

[4] R. Gravina, A. Guerrieri, G. Fortino, F. Bellifemine, R. Giannantonio, and M. Sgroi. Development of body sensor network applications using SPINE. In *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics (SMC 2008)*, Singapore, 2008.

[5] Eric Guenterberg, Hassan Ghasemezadeh, and Roozbeh Jafari. A distributed hidden markov model for fine-grained annotation in body sensor networks. In *The Sixth International Workshop on Body Sensor Networks (BSN)*, 2009.

[6] S. Kabadayi, A. Pridgen, and C. Julien. Virtual sensors: Abstracting data from physical sensors. In *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, pages 587–592. IEEE Computer Society Washington, DC, USA, 2006.

[7] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. TinyOS: An operating system for sensor networks. *Ambient Intelligence*, pages 115–148, 2005.

[8] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Proceedings of the 4th international symposium on Information processing in sensor networks*. IEEE Press Piscataway, NJ, USA, 2005.

[9] M. Sgroi, A. Wolisz, A. Sangiovanni-Vincentelli, and JM Rabaey. A service-based universal application interface for ad hoc wireless sensor and actuator networks. *Ambient Intelligence*, page 149, 2005.