Multi Agent Framework for Self-Organizing Sensor Networks

Revision 0.1
Created: 27 Jul 2004
Last update: 27 Jul 2004

Dmitri Kondratiev
dkondratiev@luxoft.com

Introduction

Wireless Sensor Network (SN) provides new ways to interact with physical environment both for people and computers. This paper attempts to describe new challenges, requirements and programming model for systems using WSN. This document is work in progress. Current version of the document is incomplete and only describes the highlights of the problem area and the main points of proposed solutions.

WSN Properties

WSN consists from tiny autonomous low-power computing nodes – motes.
Number of WSN nodes vary from several dozens to several thousands depending on an application that deploys WSN. A typical mote has a microcontroller (several MIPS) equipped with small RAM (~ 64K), short-range RF, non-volatile memory and various sensors (light, temperature, etc.). Motes collect information from physical environment, process it and share this information/processing results on a network.

Applications

WSN introduce new types of distributed applications closely interacting with physical world. These applications include constant monitoring of multiple parameters of physical objects, including temperature, pressure, humidity, motion, light, detection of certain types of object in the environment, etc.

Challenges

Together with vast amounts of new data that was unaccessible to computers previously, WSN introduces many new hard challenges for designers of such systems. These challenges arise from the nature of WSN which is based on highly distributed processing of numerous unreliable data sources communicating in ad-hoc network.
The main characteristics influencing WSN computational model include:

- WSN nodes have very restricted computational and storage power.
- Node communication range is limited. In most cases nodes can directly communicate with immediate neighbors only.
- WSN consists of a large number of unreliable nodes.
- WSN must continue to operate at all times even when some of it nodes get physically destroyed at unpredictable times.
- WSN must continue to operate without interruption when new nodes are added to the network in order to replace the failed ones or extend the network.
- Nodes may temporarily stop processing due to power shortage and come back to life when power is restored.
- Information collected by nodes may be unavailable at some irregular times due to the bad quality of radio link.
- Node communication may require different paths at different times depending on the state of end-to-end link between communicating parts of the network.
- Nodes must be able to communicate with the rest of the world represented by traditional LAN.

Programming Network with Mote Agents

The described above characteristics require the following functionality to be supported by WSN programming environment:

- Data-Centric programming model. Network elements (nodes) are addressed according to the data types and data values they produce, but not as separate distinct entities. An example request to network would be: “Return the size and location of the region where values of temperature sensors are greater then 80 degrees”. Separate requests to distinct nodes does not make a lot of sense from high-level application point of view in case of WSN. Another promising form of addressing network elements can be based on identifying parts of sensor network in terms of relations between these parts. Such relations can have spatial nature. For example: “Check operation status of a slide projector located in the north-west corner of the main conference room on the third floor.” In these examples the node IDs does not need to be globally known in system, instead different parts of the network may maintain their own local node identification systems.

- Simple, lightweight local processing of data that will reduce network traffic and thus will improve quality of radio links.

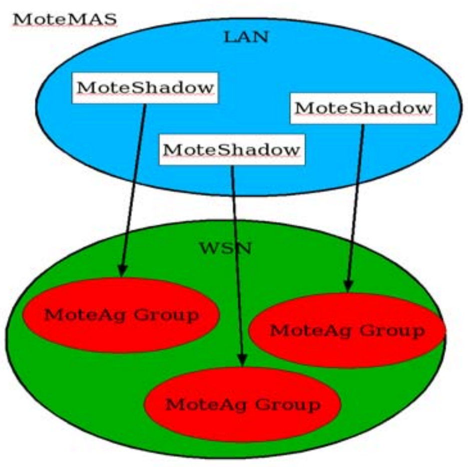
- Hierarchical filtering and aggregation of data in the network.

- Network routing adaptable both to dynamic condition of radio links and application specific needs.

- Network Self-Organization to:
    - Adapt and reconfigure both to new tasks and environmental changes.

    - Provide recovery and reconfiguration to compensate node failures. Network should be able to monitor its 'health' and reassign tasks of temporarily or permanently failed nodes to good ones.

The above functionality may be realized with MoteMAS - Multi Agent System (MAS) cooperating with Wireless Sensor Network (WSN). The key idea here is to abstract groups of sensor nodes with agents running on a single host computer or several hosts on a LAN.



MoteMAS defines the following main components:

- MoteAgs – tiny mote agents running on sensor nodes.

- MoteShadow agents - present sophisticated perception system for the outside world. MoteShadow perception system is realized with WSN and provides many levels of reconfigurability and adjustment specific to the application domain. MoteShadows run on fully-functional computers (on LAN or Internet) and extend MoteAgs abilities in several ways, providing:

    - Creation of MoteAgs federations according to different decomposition principles based on Data-Centric Programming model (see above).

    - Programmatic interface to MoteAgs federations, where every federation is abstracted with a single MoteShadow agent.

MoteMAS Architecture and Applications

MoteMAS provides building blocks both for simulation and operational environments of WSN. Different applications can be implemented using MoteMAS.

One of the main goals of MoteMAS is to provide means to reuse already existing WSN code as much as possible. The most widely used today TinyOS system and sensor code libraries can be easily integrated in MoteAgs.

MoteAgs Architecture

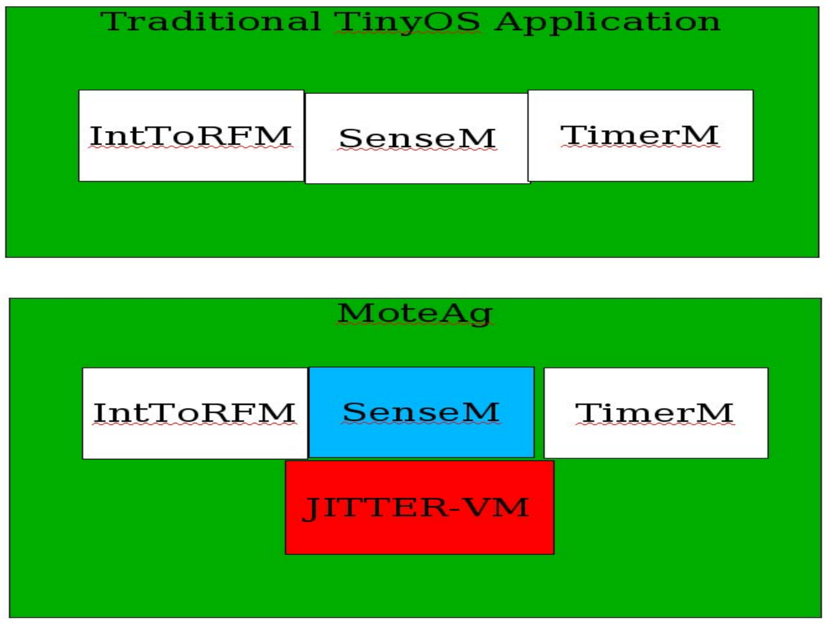
The pilot implementation of MoteAgs is based on TinyOS component model. TinyOS components are specified using nesC programming language, that “has a C-like syntax, but supports the TinyOS concurrency model, as well as mechanisms for structuring, naming, and linking together software components into robust network embedded systems.”

MoteAg consist of two different kinds of modules:

- Modules that are compiled and linked together with nesC into the complete image loadable to WSN.

- JITTERs - Just In Time Interpretatators – modules written in nesC syntax, but not directly compiled into executable image. Program code of such modules is interpreted by MoteAg JITTER Virtual Machine in run-time directly on WSN node.

Using MoteAg JITTERs will allow reconfigure and easily adjust mote execution parameters without stopping it and reloading complete program as standard TinyOS programming practice requires.



In the above example SenseM module traditionally compiled in a single TinyOS executable is replaced in MoteAg with SenseM Jitter program. SenseM Jitter source code is the same as SenseM nesC code. In MoteAg implementation SenseM run-time structure is highly optimized for interpretation by JITTER-VM, which is statically compiled into node application.
SenseM code may be dynamically changed by corresponding MoteShadow agent and then loaded in the node where MoteAg runs. JITTER code structure will support incremental code modifications and versioning that should allow change module algorithm modifications at single nesC operand granularity.

Conclusion

MoteMAS is based on two central concepts:

- MoteShadow agents presenting sophisticated perception system for the outside world.

- MoteAgs, adaptive, lightweight mote agents running Just In Time Interpretatators (JITTERs) that use the best of two worlds, namely fast compiled code and effective in time (when needed) interpretation of loadable code.

Building MoteMAS around these concepts should help answer the main challenges of creating applications that will benefit from great opportunities that WSN technology provides today.