6th International Symposium on Intelligent Distributed Computing
IDC 2012
September 24th, 2012 - Calabria, Italy

Tutorial 1
SPINE

Raffaele Gravina
Outline

- Body Sensor Networks
- Domain-Specific Programming
- The SPINE Framework
- SPINE-Based BSN Research Prototypes
- The MAPS Framework
- MAPS vs SPINE
BSNs: Background

- A **Wireless Body Sensor Network** is a Wireless Sensor Network applied to the human body.
- Multiple sensors are worn on the body, communicating wirelessly to a local coordinator station (and possibly remotely to a central server or a cloud-computing system).
- **Applications:**
  - health-care,
  - physical rehabilitation,
  - fitness and wellness,
  - emotion and stress recognition,
  - interactive gaming,
  - ...

![Diagram of Wireless Body Sensor Network](image)
BSNs: Research motivation

- Programming BSN systems is a complex task
  - Limited hardware resources;
  - (quasi) real time requirements;
  - Limited software abstractions (languages, native libraries, OSs);
  - Signal-processing and decision support;
  - Security and privacy.
BSN Programming: Related Work

• Three main approaches:
  • **Application-specific development**
    • Greatest majority of the BSN research prototypes so far
  • **General-purpose middlewares for WSNs**
    • **Titan** (it has been customized for BSNs)
  • **Domain-specific frameworks**
    • **CodeBlue** (very limited signal processing integrated support)
    • **RehabSPOT** (limited customization, only works on SunSPOT)
Objective

- **Application-specific development** is time consuming, error-prone, and produces code scarcely reusable;
- **General purpose middlewares** for WSNs do not address specific BSN system needs.

**IDEA**

*Define an effective **domain-specific** framework to support faster prototyping of signal-processing intensive BSN applications!*
SPINE at a glance

- **Open Source**
  - available to the research community under LGPL license
- **Run-time sensor nodes configuration** through Java APIs
  - lightweight Java APIs that can be used by local and remote applications to manage the sensor nodes or issue service requests
- **On-node signal processing and classification** algorithms
- **Extendible and customizable**
  - Easy to add support for new sensors and signal processing functionalities
- **Secure**
  - Hardware message encryption using AES-128 (enabled optionally)
- **Heterogeneous**
  - Several type of sensors and node platforms, ported on TinyOS, Java Squawk VM, Zigbee Z-Stack, J2ME, ... and soon on Android
SPINE Website

http://spine.deis.unical.it
http://code.google.com/p/spine-project/

SPINE: Signal Processing In Node Environment

Overview

Wireless sensor networks (WSNs) are a novel technology enabling new classes of applications and systems for ubiquitous and pervasive computing. In particular, WSNs applied to human body, also known as Wireless Body Sensor Networks (WBSNs), represent the most suitable systems for monitoring and controlling physical and biochemical parameters on the human body, and thus supporting high-impact applications in a variety of human-centered domains.

The SPINE Project aims at providing developers with software instruments for rapid prototyping of WSN/WBSN-based applications by offering great flexibility in the implementation of distributed signal processing algorithms for the analysis and classification of acquired data.
SPINE: Network Architecture

- Star topology including a Coordinator and multiple body or environmental Sensor Nodes
  - *Easy to extend with multi-hop and point-to-point sensor node communication*
# The SPINE Framework

<table>
<thead>
<tr>
<th>FUNCTIONALITY</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVICE DISCOVERY</td>
<td>Allows for the discovery of nodes and their supported sensors and processing capabilities.</td>
</tr>
<tr>
<td>FLEXIBLE SENSING SETUP</td>
<td>Allows independent specification of sensor sampling rates for multiple sensors.</td>
</tr>
<tr>
<td>RAW SENSOR READING TRANSMISSION</td>
<td>Enables periodical or one-shot transmission of raw sensor readings.</td>
</tr>
<tr>
<td>ON-NODE SIGNAL PROCESSING</td>
<td>Enables one or multiple in-node (periodic or trigger-based) signal processing functionalities independently.</td>
</tr>
<tr>
<td>HIGH-LEVEL DATA PROCESSING</td>
<td>Offers a wide set of feature selection and classification algorithms, and specific data structures to process the received data.</td>
</tr>
<tr>
<td>TAILORING (Extensible and customizable)</td>
<td>Full support for adding new sensors and signal processing functionalities</td>
</tr>
<tr>
<td>SECURITY</td>
<td>Hardware encryption using AES-128</td>
</tr>
</tbody>
</table>
SPINE Heterogeneous Support

SPINE COORDINATOR
Java SE / ME / Android

TelosB  MicaZ  Shimmer  Bollino  SunSPOT

ECG  EIP  Accelerometer  Gyroscope  Termometer  Humidity  Light
Data processing chain supported by SPINE
SPINE: Performance Evaluation

### Memory Requirements

<table>
<thead>
<tr>
<th>Application Profile</th>
<th>RAM (Kb used/av.)</th>
<th>ROM (Kb used/av.)</th>
<th>Average Power Consumption</th>
<th>Battery</th>
<th>Lifetime</th>
<th>Bitrate</th>
<th>802.15.4</th>
<th>Bluetooth</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPINE on TelosB</td>
<td>3.7 / 10</td>
<td>33.5 / 48</td>
<td>6.6 mW/s</td>
<td>650mAh</td>
<td>101 h</td>
<td>178 byte/s</td>
<td>10.07</td>
<td>N/A</td>
</tr>
<tr>
<td>SPINE on Shimmer</td>
<td>4.4 / 10</td>
<td>40.0 / 48</td>
<td>13.9 mW/s</td>
<td>280mAh</td>
<td>21 h</td>
<td>178 byte/s</td>
<td>10.04</td>
<td>N/A</td>
</tr>
<tr>
<td>SPINE on Shimmer Bluetooth</td>
<td>4.3 / 10</td>
<td>34.4 / 48</td>
<td>87.8 mW/s</td>
<td>280mAh</td>
<td>3 h</td>
<td>150 byte/s</td>
<td>N/A</td>
<td>3.05</td>
</tr>
<tr>
<td>SPINE on Z-Stack</td>
<td>3.9 / 8</td>
<td>95.9 / 128</td>
<td>11.2 mW/s</td>
<td>650mAh</td>
<td>60 h</td>
<td>160 byte/s</td>
<td>0.61</td>
<td>N/A</td>
</tr>
<tr>
<td>SPINE on SunSPOT</td>
<td>79.0 / 512</td>
<td>75.0 / 4096</td>
<td>84.2 mW/s</td>
<td>720mAh</td>
<td>9 h</td>
<td>168 byte/s</td>
<td>67.20</td>
<td>N/A</td>
</tr>
<tr>
<td>A-hoc application on TelosB</td>
<td>1.3 / 10</td>
<td>16.1 / 48</td>
<td>73.7 mW/s</td>
<td>650mAh</td>
<td>9 h</td>
<td>152 byte/s</td>
<td>9.98</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Energy Consumption

### Bandwidth

### Transmission Delay
SPINE: high-level architecture

<table>
<thead>
<tr>
<th>SPINE API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform Independent SPINE OTA Protocol</td>
</tr>
<tr>
<td>Active Message</td>
</tr>
<tr>
<td>TinyOS 802.15.4</td>
</tr>
</tbody>
</table>
SPINE: coordinator sw architecture

- The core implementation does not use any platform specific APIs and can be run on any kind of network (e.g. ZigBee networks).
- Platform-independent code may be found into:
  - `spine`: contains SPINE core logic
  - `spine.datamodel`: contains data structures used by the framework
- At this stage, SPINE1.2 server side provides an implementation for TinyOS2.x network; therefore it provides the support for TinyOS low level communication:
  - `spine.communication.tinyos` contains TinyOS specific logic (packet format and parse/build operations) and low level communication procedures (calling tinyos.jar APIs).
SPINE: sensor node sw architecture

- **Communication**
  - takes care of radio communication, data packet build and parse, radio duty cycle and channel access schemes;

- **Sensing**
  - manages data sampling from sensors and storage on a shared buffer;

- **Processing**
  - takes care of the on node processing functionalities.
Communication Subsystem

- **Radio Controller**
  - access to the radio
  - Optional low power mode and TDMA
- **Packet Manager**
  - Build and parse SPINE packets
  - Fragments if necessary
- **InPacket**
  - Generic i/f to be implemented by pkts incoming into the nodes (parse method)
- **OutPacket**
  - Generic i/f to be implemented by pkts outgoing from the nodes (build method)
Sensing Subsystem

- **Sensor Drivers**
  - must implement Sensor i/f

- **Sensor Board Controller**
  - Manages all on-board sensor
  - takes care of setting up sensors (Sampling Time), sample sensors and store readings into buffer pool

- **Sensor Registry**
  - Registers built-in and pluggable sensors

- **Buffer Pool**
  - is a general purpose storage manager, built as single array logically divided into multiple indexed circular buffers
Processing Subsystem

- **Function Manager**
  - dispatcher among functions

- **Function i/f**
  - to be implemented by all supported functions
Running SPINE

**At node-side:**
1. Customize if necessary the pre-defined set of “wired-in” sensor drivers and processing functions
2. compile the **SPINEApp** and program the wireless sensor nodes

**At coordinator-side:**
1. Program your end-user application atop the SPINE Java (or Android) API
2. Select the communication platform, by setting a variable into the **app.properties** file
3. Run your application
SPINE Motion board on TelosB motes

- SENSORBOARD = spine make telosb

- Spine motion sensor board developed by Roozbeh Jafari at UT Dallas
- Telosb motes
  - Internal cpu temperature
  - Telosb voltage
- Sensor board
  - 3-axis accelerometer
  - 2-axis gyro (2 gyro may be present on the same board)
  - Litum battery voltage (if telosb voltage is not used)
BIOSENSOR board on TelosB motes

- SENSORBOARD=biosensor make telosb

- Biosensor board developed by Ville-Pekka Seppä Tampere University of Technology
- Single sensor board connected to a mote measures
  - Pulmonary Function
  - Electrocardiogram
  - Tri-axial Acceleration
  - Body Temperature
MTS300 sensor board on MicaZ motes

- SENSORBOARD=mts300 make micaz
- Micaz motes (XBow)
- Mts 300 sensor board
  - 2-axis accelerometer
Shimmer motes

- **make shimmer**
  - SENSORBOARD=shimmer_bt make shimmer
- **make shimmer2**
- **make shimmer2r**
  - SENSORBOARD=shimmer2r_bt make shimmer

- Bluetooth and 802.15.4 radio
- ECG Add-on Board
- Kinematics Add-on Board
- AnEx Board for rapid prototyping
- GSR Add-on Board
- EMG Add-on Board

High-Level control API

- `discoveryWsn()`

- `setup(Node n, SpineSetupSensor setupSensor)`

- `setup(Node n, SpineSetupFunction setupFunction)`

- `activate(Node n, SpineFunctionReq functionReq)`

- `deactivate(Node n, SpineFunctionReq functionReq)`

- `startWsn(boolean radioAlwaysOn, boolean enableTDMA)`

- `resetWsn()`
High-Level API - Events

- `newNodeDiscovered(Node newNode)`
- `discoveryCompleted(Vector activeNodes)`
- `received(Node n, ServiceMessage msg)`
- `received(Node n, Data data)`
The SPINE Messages

Standard messages of the SPINE protocol exchanged between Coordinator (C) and Node (N).

<table>
<thead>
<tr>
<th>Direction</th>
<th>C→N</th>
<th>N→C</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Discovery</td>
<td>●</td>
<td></td>
<td>NONE</td>
</tr>
<tr>
<td>Service Advertisement</td>
<td></td>
<td>●</td>
<td>&lt; sensors list, services list&gt;</td>
</tr>
<tr>
<td>Set-Up Sensor</td>
<td>●</td>
<td></td>
<td>&lt; sensor code, sensor parameters &gt;</td>
</tr>
<tr>
<td>Set-Up Service</td>
<td>●</td>
<td></td>
<td>&lt; service code, service parameters &gt;</td>
</tr>
<tr>
<td>Activate Service</td>
<td>●</td>
<td></td>
<td>&lt; service code &gt;</td>
</tr>
<tr>
<td>De-activate Service</td>
<td></td>
<td>●</td>
<td>&lt; service code &gt;</td>
</tr>
<tr>
<td>Data (raw or processed)</td>
<td></td>
<td>●</td>
<td>&lt; service code, data &gt;</td>
</tr>
<tr>
<td>Start processing</td>
<td></td>
<td></td>
<td>&lt; radio configuration &gt;</td>
</tr>
<tr>
<td>Reset (node/network)</td>
<td>●</td>
<td></td>
<td>NONE</td>
</tr>
<tr>
<td>System notification</td>
<td></td>
<td>●</td>
<td>&lt; notification type, notification details &gt;</td>
</tr>
</tbody>
</table>

SPINE Packet Format

32 bits

V E Type GroupID SourceNodeID

DestNodeID SequenceNr TotalFragment

FragmentOffset Payload (0 or more words)

V = Version of the protocol
E = Extension flag signalling an extension of the message
Type = Type of SPINE message
GroupID = Identifier of the group
SourceNodeID = Identifier of the transmitting node
DestNodeID = Identifier of the receiving node
SequenceNr = Sequence number of the messages
Total Fragments = Number of total fragments of the SPINE message
Fragment Offset = Offset of the fragment with respect to the reference message
HOW TO... add a new sensor

- Platform independent **MyNewSensorC.nc** implementation into tos\system\sensing that links to the driver implementation into tos\sensorboards\myboard (or into tos\platforms\myplatform)
- Driver implementation **HilMySensorC/P.nc** has to implement the tos\interfaces\sensing\**Sensor.nc** interface and register its code to the tos\system\sensing\**SensorRegistry.nc**
- Define the new sensor platform into the **spine.extra** makefile
- Add new sensor wiring in the to the tos\system\sensing\**SensorBoardController.nc**
- Update constants both node and server side

- The new sensor is now fully integrated into and all the SPINE services can be applied to data coming form myNewSensor.
New function must implement the Function i/f
Function take data from the buffer pool, analyze it and send result back to the coordinator
Function are set with:
  - setUpFunction
  - activateFunction
Function can be deactivated
  - Deactivate Function
Data coming from the function = array of bytes
Feature Engine

- It provides periodic calculation of simple features on sensed data
  - A server app requests a feature using two setup messages:
    - Set a sampling time for a given sensor if necessary
    - associate a window size and shift size with a given sensor
    - request the features desired for that sensor
  - Features include MIN, MAX, MEAN, AMPLITUDE, ...
    - Can be calculated over multiple channels
      - MEAN calculates over each active channel
      - VECTOR_MAGNITUDE calculates over all channels and returns a single value
    - A single feature can calculate across channels but not across multiple sensors
  - A feature is calculated every shift size samples, over a buffer of length window size
Feature Engine

- It keeps a table of requested features, and listens for sampling events to trigger feature calculation and sending
  - A buffer for each active sensor channel is managed by the SensorBoardController
  - The FeatureEngine organizes data in these buffers so that a feature may be calculated, then generates and sends a message to send calculated features to the base station
  - Features are implemented as Tinyos components conforming to the Feature interface, allowing new Features to be easily introduced
HOW TO... introduce New Feature

Node Side:

1. In Spine_nodes1_2\tos\types\Functions.h add MY_NEW_FEATURE code into enum FeatureCodes
2. Into tos/system/processing you have to implement the feature logic. For this reason you'll create 2 new files: MyNewFeatureC.nc and MyNewFeatureP.nc.

MyNewFeature module will have to provide the Feature interface (look into tos/interfaces/processing/Feature.nc), meaning it will have to implement the .calculate and the .getResultSize commands.

a. Feature.calculate command does the actual feature implementation over a certain
b. Feature.getResultSize command simply returns the number of byte the resulting feature is composed of.

Remember that, since the Boot interface is used, you'll have to implement the event Boot.booted(). We suggest that you register the feature to the FeatureEngine at this time.

```c
event void Boot.booted() {
    if (!registered) {
        // the feature self-registers to the FeatureEngine at boot time
        call FeatureEngine.registerFeature(MY_NEW_FEATURE);
        registered = TRUE;
    }
}
```

3. Finally, wire your new Feature into the FeatureEngine. Open tos/system/processing/FeatureEngineC.nc and add the following lines:

```c
components MyNewFeatureC;
FeatureEngineP.Features[MY_NEW_FEATURE] -> MyNewFeatureC;
```
HOW TO... introduce New Feature

• Server Side:
  • The server need only
  • Open spine.SPINEFunctionConstants.java and add the following lines that define the new feature:

```java
public static final byte MY_NEW_FEATURE =

public static final String MY_NEW_FEATURE_LABEL = "MyNewFeature"

case MY_NEW_FEATURE: return MY_NEW_FEATURE_LABEL;
```
Alarm Engine

- The Alarm Engine provides an event notification whenever function values are outside input thresholds.
- Alarms may be set on raw data as well as functions.
- Available alarm types:
  - Above threshold
  - Below threshold
  - In between thresholds
  - Outside thresholds
Alarm Engine

• **Alarm may set up by coordinator (window and shift)**
  
  ```java
  AlarmSpineSetupFunction ssf = new AlarmSpineSetupFunction();
  ssf.setSensor(sensor);
  ssf.setWindowSize(WINDOW_SIZE);
  ssf.setShiftSize(SHIFT_SIZE);
  manager.setupFunction(curr.getNodeID(), ssf);
  ```

• **Alarm may be activate by coordinator** (Sensor type and channel Data type Thresholds Alarm type)
  
  ```java
  AlarmSpineFunctionReq sfr = new AlarmSpineFunctionReq();
  sfr.setDataType(SPINEFunctionConstants.RAW_DATA);
  sfr.setSensor(SPINESensorConstants.ACC_SENSOR);
  sfr.setValueType((SPINESensorConstants.CH1_ONLY));
  sfr.setLowerThreshold(lowerThreshold);
  sfr.setUpperThreshold(upperThreshold);
  sfr.setAlarmType(SPINEFunctionConstants.ABOVE_Threshold);
  manager.activateFunction(curr.getNodeID(), sfr);
  ```

• **Node will send back alarm data only when the event occurs, reporting**
  
  - Sensor type and channel
  - Data type
  - Alarm Type
  - Value that caused the alarm
HOW TO... add a new function - NODE SIDE

- MyNewFunction logic must be implemented into tos/system/processing
- MyNewFunctionEngineP.nc must implement the tos/interface/processing/Function.nc i/f and register itself to the function manager
- Function.nc provides setUpFunction, activateFunction, disableFunction, getFunctionList, startComputing and stopComputing commands
- The module has to also implement 3 events: boot.booted, functionmanager.sensorWasSampledAndBuffered and BufferPool.newElem.
- Parameters into SetUpFunction, activateFunction and data packets must be set according to the function logic
- Wiring must be added to the FunctionManager and constants must be defined
HOW TO... add a new function – SERVER SIDE

- **SetUp Function**

<table>
<thead>
<tr>
<th>bit</th>
<th>8</th>
<th>8</th>
<th>variable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Function code</td>
<td>param length</td>
<td></td>
</tr>
</tbody>
</table>

  *spine.communication.tinyos* (TinyOS SPINE communication)

  - NewFunctionSpineSetupFunction.java implement the tinyos encoding logic and extends the SpineSetupFunction.java abstract class

- **Activate/Deactivate Function**

<table>
<thead>
<tr>
<th>bit</th>
<th>8</th>
<th>8</th>
<th>8</th>
<th>variable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Function code</td>
<td>Act/deact</td>
<td>param length</td>
<td>param list</td>
</tr>
</tbody>
</table>

  *spine.communication.tinyos* (TinyOS SPINE communication)

  - NewFunctionSpineFunctionReq.java implement the tinyos encoding logic and extends the SpineFunctionReq.java abstract class
HOW TO... add a new function – SERVER SIDE

DATA

<table>
<thead>
<tr>
<th>bit</th>
<th>8</th>
<th>8</th>
<th>variable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Function code</td>
<td>data length</td>
<td>data</td>
</tr>
</tbody>
</table>

**spine.datamodel** (platform independent data structures)
- MyFunctionData.java: decode byte array ➔ MyFunction object
  - MyFunction.java: define MyFunction entity (constructor, toString and getters)

**spine.communication.tinyos** (TinyOS SPINE communication)
- MyFunctionSpineData.java: method for parse (decompress TinyOS SPINE MyFunction data pck). Called by SpineData class through dynamic class loading
Functionalities

- Android SPINE is a full-operative porting of the SPINE 1.3 Framework

- ☑ Sensor-node side SPINE middleware didn’t require any changes.
- ☑ Same software design principles and architecture.
- ☑ Same High-Level API exposed to application developers.
- ☑ Same sensing and processing support.

- 😞 Limitation: due to the lack of 802.15.4 radio on commercial Android devices, sensor platform support is limited to Shimmer (rev.1 and 2R)
Supported Sensors

- Currently, mobile application developers, through the Android SPINE API, have access to the following Shimmer -platform (rev. 1 and 2R) sensors:
  - Tri-axial accelerometer
  - Gyroscope
  - ECG

- We are close to release SPINE-compliant drivers for additional Shimmer sensors:
  - GSR
  - EMG
Android-SPINE *Test GUI*

Node: 2

Node: 1

Node number: 1

Sensor: gyroscope

Sampling Time: Sampling Time

Set
Android-SPINE Test GUI

Svc Msg from {phyID:1, logID: null} - Ack:seq# 4
Svc Msg from {phyID:1, logID: null} - Ack:seq# 3
Svc Msg from {phyID:1, logID: null} - Ack:seq# 2
SPINE-Based
BSN Research Prototypes
Human Activity Recognition

- System configuration: 2 wireless motion sensors (waist and thigh) + 1 coordinator device
- Recognized Activities and posture: walking, standing, sitting, lying
- Built-in Fall Detection
- Step-counter
- Energy Expenditure (Kcal)
- Very Accurate (>98%)
Human Activity Recognition - Demo

- Play VIDEO
Human Activity Recognition - Market

- FitBit
- Nike Plus
Mental Stress Detection

- Real-time, evaluation based on time-domain analysis
- Low power
- Completely wireless
- Wearable
- Robust to motion artifacts
Mental Stress Detection

- Basestation & Coordinator
- Heart waistband (by Polar® Electro)
- SPINE CardioShield
  Telosb mote + custom sensing board
Mental Stress Detection - Demo

- Play VIDEO
Mental Stress Detection - Market

- Zed X2 Stress Detector (ZeroStress S.r.l.)
- StressEraser
- Q-Sensor
  - (Affectiva, Inc.)
Physical Rehabilitation

- **RehabTutor®** is a hw/sw tool for assisting motor rehabilitation therapy after injuries and surgeries
- **Elbow** and **Knee** rehabilitation
- Portable and non-invasive
- Guided-exercise mode
- Rehab session recording and playback
- Improvement information
- Support for Remote video-chat with medical doctors
Physical Rehabilitation - Demo

- Play VIDEO
Physical Rehabilitation - Market

- Rehabitic (Telefónica S.A.)
The MAPS Framework
Agent-Based Programming of WSNs: Motivations

- Mobile agents are a distributed computing paradigm based on code mobility that has already demonstrated high effectiveness and efficiency in highly dynamic distributed environments.
- Due to their intrinsic characteristics, mobile agents may provide more benefits in the context of WSNs than in conventional distributed environments.
- Integration of agents with WSN to provide a general purpose agent platform for WSNs.
Agent-Based Programming of WSNs: Motivations

As advertised by Lange and Oshima there are at least seven good reasons to use mobile agents.
- these reasons (characteristics) are very important for the WSN domain
  - Network load reduction
  - Network latency overcoming
  - Protocol encapsulation
  - Asynchronous and autonomous execution
  - Dynamic adaptation
  - Orientation to heterogeneity
  - Robustness and fault-tolerance
# MAS for WSNs: Related Work

<table>
<thead>
<tr>
<th>Agilla</th>
<th>actorNet</th>
<th>MAPS</th>
<th>AFME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migration</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multitasking</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Communication Model</td>
<td>tuple space</td>
<td>messages</td>
<td>messages</td>
</tr>
<tr>
<td>Programming Language</td>
<td>proprietary ISA</td>
<td>Scheme-like</td>
<td>Java</td>
</tr>
<tr>
<td>Agent Model</td>
<td>Assembler-like</td>
<td>Functional</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>Intentional Agents</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Sensor Platforms</td>
<td>Mica2, MicaZ, TelosB</td>
<td>Mica2</td>
<td>Sun SPOT</td>
</tr>
</tbody>
</table>

- There aren’t Multi Agent Systems natively designed for WSN which allows to program agents in java (SunSPOT)
MAPS: Multi-Agent Platform for SunSPOT

- Innovative Java-based framework expressly developed on Sun SPOT technology for enabling agent-oriented programming of WSN applications.
- It is based on three paradigms:
  - agent oriented
  - event-driven
  - state-based programming

- **Features:**
  - Component-based lightweight agent server architecture to avoid heavy concurrency and agents cooperation models.
  - Minimal core services involving agent migration, agent naming, agent communication, timing and sensor node resources access (sensors, actuators, flash memory, and radio).
  - Plug-in-based architecture extensions through which any other service can be defined in terms of one or more dynamically installable components implemented as single or cooperating (mobile) agents.
  - Use of automaton for defining the mobile agent behavior.
MAPS: Architecture

- The architecture of MAPS is based on several components interacting through events and offering a set of services to mobile agents, including message transmission, agent creation, agent cloning, agent migration, timer handling, and easy access to sensor node resources.

MA - Mobile Agent
MAEE - Mobile Agent Execution Engine
MAMM - Mobile Agent Migration Manager
MACC - Mobile Agent Communication Channel
MAN - Mobile Agent Naming
RM - Resource Manager
TM - Timer Manager
MAPS Case-study: Real-time Human Activity Recognition
MAPS Case-study: Real-time Human Activity Recognition

**DataCollectorAgent**

**WaistSensorAgent**

**MAPS vs SPINE Implementations**

![Graphs showing data collection and analysis for DataCollectorAgent and WaistSensorAgent]
Thanks!