Mannasim Framework Classes Manual

The Manna Research Group

10 de fevereiro de 2006

1 Mannasim classes

This document provides a description of the principal classes that constitutes Mannasim. Each class description will be provided with a class summary and an enumeration of its fundamental methods and fields.

The classes are detailed acording to the following structure:

FILES
Presents the files where the class is defined and implemented.

EXTENDS
If the class inherits some features from other ones, these classes are specified in this subsection.

METHODS
Here the class essential methods are listed and explained. Note that the explanation concerns the class functionality not how it actually implements it.

Field
Here the class essential fields are listed and explained.

The order of classes description is defined as follows:

- Sensor Node
- DataGenerator
- TemperatureDataGenerator
- TemperatureAppData
- Processing
- SensedData
- OnDemandData
- OnDemandParameter
- SensorBaseApp
- CommonNodeApp

- ClusterHeadApp

This particular arrangement is due to Wireless Sensor Networks tripod sensing-processing-disseminating.

1.1 SensorNode Class

FILES
`sensorNode.h` and `sensorNode.cc`.

EXTENDS
NS-2 `MobileNode` class.

DESCRIPTION
This class represents a wireless sensor node. It extends NS-2 MobileNode class adding specific sensor nodes characteristics as power consumption for sensing and processing activities, instructions executed per seconds by its microcontrole, states of sensor parts (processor, transceptor, sensing devices). The SensorNode class was the first step taken in the implementation of the simulator and also can be used as an initial approach for the development of different sensor node types.

METHODS

- `void SensorNode::selfTest()` – performs a sensor node self test verifying its proper functioning. IMPORTANT: this method isn't implemented yet.
- `void SensorNode::sleep()` – turns the sensor node off when necessary. Stop all of its applications.
- `void SensorNode::wakeUp()` – wakes the sensor node up after a sleepy period. Start all of its applications.
- `double SensorNode::sensingPower()` – gets sensor node sensing energy consumption.
- `double SensorNode::processingPower()` – gets sensor node processing energy consumption.
- `int SensorNode::instructionsPerSecond()` – gets sensor node instructions per second executed by its processor.

FIELDS

- `AppList apps_` – the list of WSN applications witch the sensor node is associated to. The AppList type is simple an STL[1] list of AppData objects (see NS-2 documentation for further details).
- `int instructionsPerSecond_` – the number of instructions the nodes processor can execute in a second. This field has its value defined in TCL simulation script.

[1]C++ Standard Template Library

- `double processingPower_` – the energy consumption of the sensor node in processing activities. This field has its value expressed in watts (joules/second) and is defined in TCL simulation script.
- `double sensingPower_` – the energy consumption of the sensor node in sensing activities. This field has its value expressed in watts (joules/second) and is defined in TCL simulation script.
- `int sensorUseState` – the sensor state. Indicates whether the node's sensing activity is in use or not.
- `int processorUseState` – the processor state. Indicates whether the node's processing activity is in use or not.
- `int transceptorUseState` – the transceptor state. Indicates whether the node's dissemination activity is in use or not.

1.2 Battery Class

FILES
`battery.h` and `battery.cc`

EXTENDS
NS-2 `EnergyModel` class.

DESCRIPTION
This class represents sensor nodes power supply. It extends NS-2 EnergyModel class. The Battery class can be used to implement all different kinds of existing battery models. The class contains methods to turn sensor node on and off, put it into sleep and wake it up. Also contains methods to decrease energy when a sensing, processing or disseminating job is performend by the sensor node.

METHODS

- `void Battery::DecrSensingEnergy(double sensing_time, double sensing_power)` – computes the energy spent by a sensing task and reduce energy stock by this amount.
- `void Battery::DecrProcessingEnergy(int number_instructions, double instructions_per_seco` `double processing_power)` – computes the energy spent by a processing task and reduce energy stock by this amount. .
- `void Battery::setNodeOn()` – turns sensor node on.
- `void Battery::setNodeOff()` – turns sensor node off.
- `void Battery::sleep()` – puts node battery in sleep mode.
- `void Battery::wakeUp()` – wakes up the sleepy node battery.

FIELDS
All fields are inherited from NS-2 EnergyModel. (see NS-2 documentation for futher details).

1.3 DataGenerator Class

FILES
`dataGenerator.h` and `dataGenerator.cc`

EXTENDS
NS-2 `TclObject` class.

DESCRIPTION
This class simulates the sensing task of a wireless sensor node. With methods to adjusts sensing interval, generate and collect synthetic data, the DataGenerator class also works as a base class for futher specialized sensed information like for example temperature, light, magnetic fields etc. The data created by this class is encapsulated as an AppData object (see NS-2 documentation for further details).

Within DataGenerator files, SensingTime class is also defined. This timer class in conjunction with the defined sensing interval acts in periodic sensing.

METHODS

- `void DataGenerator::generateData()` – simulates the sensing activity, gets data from collect() method and deliver it to sensor processing module.
- `virtual AppData* DataGenerator::collect()` – the truly data generator function. Creates specific synthetic data, encapsulates it and return to generateData() method. It's a virtual method in DataGenerator class, so it must be overloaded.
- `void DataGenerator::insertInterference()` – inserts interference in data gathered by the sensor. This method hasn't been implemented yet.
- `void DataGenerator::start()` – schedules the first sensing activity if the network has periodic sensing.
- `void DataGenerator::stop()` – stops the sensing activity and drops all scheduled sensing events.
- `void DataGenerator::setSensingInterval(double si)` – sets the sensing interval for a periodic sensing network. Actually all kinds of sensing should set the sensing interval, in continuous sensing for example, this value should be set as small as possible.
- `double DataGenerator::getSensingInterval()` – informs the sensing interval of the network. This field has its value defined in TCL simulation script.

FIELDS

- `AppDataType type_` – the identity of the data generator in terms of application data type.
- `SensorAppList app_` – the list of sensor applications associated with the data generator. The SensorAppList type is simple an STL list of SensorBaseApp (see Section 1.10) objects.
- `double sensing_interval_` – the sensing interval for data gathering.

- `int sensing_type_` – the way the sensor node, and consequently the DataGenerator class, is oriented to gather data. Possible values are PROGRAMMED, CONTINUOUS, ON_DEMAND and EVENT_DRIVEN representing respectively periodic, continuous, on demand and event driven data sensing.

1.4 TemperatureDataGenerator Class

FILES
`temperatureDataGenerator.h` and `temperatureDataGenerator.cc`

EXTENDS
`DataGenerator` class.

DESCRIPTION
This class simulates temperature sensing task of a wireless sensor node. TemperatureDataGenerator extends DataGenerator adding characteristics proper for temperature data. Synthetic temperature generation is based in Gaussian Distribution so average and standard deviations are fields of this class.

METHODS

- `virtual AppData* DataGenerator::collect()` – the truly temperature data generator method. Creates temperature synthetic data, encapsulates it in a TemperatureAppData object (see Section 1.5) and deliver this new object for further processing.
- `double TemperatureDataGenerator::getAvgMeasure()` – returns the average measure for temperature data generation.
- `void TemperatureDataGenerator::setAvgMeasure(double avg_measure)` – adjusts the average measure for temperature data generation.
- `TemperatureAppData* TemperatureDataGenerator::getMaximumAllowedValue()` – return the maximum allowed temperature value that can be generated.

FIELDS

- `RNG* rand_` – random number used to generate synthetic temperature data.
- `double avg_measure` – average measure for synthetic temperature data generation.
- `double std_deviation` – standard deviation measure for synthetic temperature data generation.
- `double maximumTemperatureValue` – maximum temperature value allowed to be generated.

1.5 TemperatureAppData Class

FILES
`temperatureAppData.h` and `temperatureAppData.cc`

EXTENDS
NS-2 `AppData` class.

DESCRIPTION
This class encapsulates raw data generated by TemperatureDataGenerator class. Objects of this class are sent to the processing module.

METHODS

- `AppData * TemperatureAppData::copy()` – creates a copy of temperature application data object.
- `int TemperatureAppData::size() const` – informs application data size in bytes.
- `bool TemperatureAppData::checkEvent(AppData* data_)` – checks if the parametric data represents an event considering the object data. For example, temperature value get greater? This method is used by event driven WSN.
- `bool TemperatureAppData::compareData(AppData* data, int operation)` – compares parametric data according to specified operation. This method is used by on demand WSN.
- `double TemperatureAppData::data()` – returns temperature data from application data.
- `void TemperatureAppData::setData(double data)` – adjusts temperature data of application data.
- `double TemperatureAppData::time()` – returns timestamp of application data.
- `void TemperatureAppData::setTime(double time)` – adjusts timestamp of application data.
- `int & TemperatureAppData::getPriority()` – return application data priority (Usefull for temperature data delivery).
- `void TemperatureAppData::setPriority(int p)` – set application data priority (Usefull for temperature data delivery).

FIELDS

- `double data_` – temperature data gathered in sensing task.
- `double time_` – timestamp for temperature data.
- `int priority_` – temperature data priority (Usefull for temperature data deliver).

1.6 Processing Class

FILES
processing.h and processing.cc

EXTENDS
NS-2 TclObject class.

DESCRIPTION
This class provides a foundation for all kinds of data processing over sensed data in a sensor node. Every sensed data goes through processing before dissemination. Specialized data processing should inherit from Processing class.

METHODS

- void AggregateProcessing::recvData(AppData* data_) – receives data from various sources, process it and return to the application for dissemination. This method **must** be overloaded.
- AppData * Processing::processRequest(AppData* data) – simulates processing activity. Gets the raw data from sensing activity, process it, computes energy spent and returns processed data. This method **should** be overloaded in extensions of the Processing class.
- AppData* Processing::processSensedData(AppData* data_, AppData* eventData_) – simulates processing activity for event driven WSN.
- AppData* Processing::process_request_data(OnDemandParameter* parameter, int request_typ – manages requests from an outsider observer in a on demand WSN[2]. This method only redirects the raw data to further specialized data processing.
- AppData* Processing::process_real_request(AppData* data_, int operation) – process real requests from an on demand WSN. In real requests the sensor node drops all data from its buffer, gather new one, process and deliver it to the disseminating module.
- AppData* Processing::process_buffer_request(AppData* data_, int operation) – process buffer requests from an on demand WSN. In buffer requests the sensor node process data from its buffer and give the results to the disseminating module.
- AppData * CommonNodeApp::process_average_request(AppData* data_,int operation) – process average requests from an on demand WSN. This method hasn't being implemented yet.
- void Processing::resetData() – clears the sensor processed data buffer.
- SensedData * Processing::getProcessedData() – returns data generated by the processing activity.

FIELDS

[2]In an on demand WSN the outsider observer make requests to the network about data of its interest

- `SensedData* info_` – processed data buffer, data stored here is ready for dissemination. For more details about SensedData data type see Section 1.7).
- `SensorBaseApp * app_` – application attached to the sensor node. Usefull in on demand WSN.
- `SensorNode* sensor_node_` – sensor node where the processing task takes place. Used for energy contability proposes.

1.7 SensedData Class

FILES
`sensedData.h` and `sensedData.cc`

EXTENDS
NS-2 `AppData` class.

DESCRIPTION
This class represents sensed data after processing. Data contained in SensedData is ready for dissemination so the class provides networking information such as source node identification, message type, message priority among others. SensedData class acts as dissemination message for processed data witch is stored in a data buffer.

METHODS

- `AppData * SensedData::copy()` – creates a copy of the sensed data object.
- `bool SensedData::existsData()` – informs if there is processed data stored in data buffer.
- `AppDataList SensedData::getData()` – returns processed data stored in data buffer.
- `int & SensedData::msgType()` – returns a reference to sensed data message type.
- `int & SensedData::node_id()` – returns a reference to source node identification.
- `int & SensedData::eventType()` – returns a reference to sensed data event type. This method is designed to be used in a event driven application.
- `double & SensedData::timeStamp()` – returns a reference to sensed data timestamp;
- `int SensedData::priority()` – returns sensed data priority. Useful during disseminating tasks.
- `void SensedData::set_priority(int p)` – adjusts sensed data priority.
- `void SensedData::insertNewData(AppData* data)` – inserts new processed data into data buffer.

FIELDS

- `int node_id_` – source node identification for sensed data dissemination purposes.

- `int msgType_` – type of the message to be disseminated to the network.
- `int priority_` – priority of sensed data message.
- `int eventType_` – type of the event that caused sensed data processing.
- `double timestamp_` – sensed data timestamp.
- `AppDataList infoRepository` – sensed data buffer. Storages all data processed by the sensor node.

1.8 OnDemandData Class

FILES
`onDemandData.h` and `onDemandData.cc`

EXTENDS
`SensedData` class.

DESCRIPTION
This class represents request messages from an outsider of the WSN. Requests are interesting for on demand applications where dissemination only occurs when someone requests data. OnDemandData uses inherited `infoRepository` field from SensedData to storage OnDemandParameter queries (see Section 1.9).

METHODS

- `AppData * OnDemandData::copy()` – creates a copy of the current OnDemandData object.
- `int OnDemandData::size() const` – returns OnDemandData object size in bytes.
- `int & OnDemandData::requestType()` – returns a reference for the request type field.

FIELDS

- `int request_type_` – message request type for a on demand WSN. Possible values include `REAL`, `BUFFER` and `AVERAGE`.

1.9 OnDemandParameter Class

FILES
`onDemandParameter.h` and `onDemandParameter.cc`

EXTENDS
NS-2 `AppData` class.

DESCRIPTION
This class represents queries sent within OnDemandData request messages. In on demand applications, requests follows this line:

“*Disseminate sensor node data if it’s greater than X*”.
An OnDemandParameter object carries the X value and also the kind of operation to be realized for request validation.

METHODS

- `AppData * OnDemandParameter :: copy()` – creates a copy of the current OnDemandParameter object.
- `int OnDemandParameter::size() const` – returns OnDemandParameter object size in bytes.
- `AppData * OnDemandParameter::data()` – returns sample data used for request validation.
- `int & OnDemandParameter :: operation()` – returns a reference for the operation to be realized for request validation.

FIELDS

- `AppData* data_` – sample data used for request validation.
- `int operation_` – operation to be realized for request validation. Currently supported operations include `GREATER_THAN`, `LESS_THAN` and `EQUAL`.

1.10 SensorBaseApp Class

FILES
`sensorBaseApp.h` and `sensorBaseApp.cc`

EXTENDS
NS-2 `Application` class.

DESCRIPTION
This class is the base for WSN applications, specialized node applications like a common node application and a cluster head application should extends SensorBaseApp class. With instances of DataGenerator, Processing and SensorNode classes, SensorBaseApp class congregates the WSN tasks tripod: sensing, processing and disseminating with special attention to the disseminating task.

Within SensorBaseApp files, DisseminatingTimer class is also defined. This timer class in conjunction with the defined disseminating interval acts in periodic disseminating.

METHODS

- `virtual void disseminateData()` – disseminates sensed data into the network after its processing. This method is virtual so it **must** be overloaded.
- `virtual void disseminateData(AppData* data)` – disseminates sensed data into the network after its processing. In this method disseminating data is passed as parameter and as the method is virtual it **must** be overloaded.

- `virtual void recvSensedData(AppData* data_)` – receives data gathered by the node's sensors, process it and disseminate the results. This method is virtual so it **must** be overloaded.
- `virtual void recvSensedData(AppData* data_, AppData* eventData_)` – receives data gathered by the node's sensors, process it and if data corresponds to a valid event disseminate the results. This method is virtual so it **must** be overloaded.
- `void SensorBaseApp::start()` – schedules the first sensing/disseminating tasks if the network application is periodic.
- `void SensorBaseApp::stop()` – drops all scheduled events, stopping also all sensing tasks.
- `void SensorBaseApp::insertNewGenerator(DataGenerator* gen)` – inserts a new DataGenerator object (or one of its extensions) into data generators list of the application.
- `DataGenList SensorBaseApp::getGenList()` – returns the data genetators list of the application.
- `SensorNode * SensorBaseApp::sensor_node()` – returns a reference to the sensor node attached to the application.

FIELDS

- `SensorNode* sensor_node_` – sensor node where the applications is inserted.
- `DataGenList gen_` – list of data generator objects (or one of its extensions). DataGenList type is simple a STL list data structure of data generator objects.
- `Processing* processing_` – processing module of the sensor node. Gets raw data from a data generator object and returns processed data.
- `int disseminating_type_` – the way the sensor node, and consequently the SensorBaseApp class, is oriented to disseminate data.. Possible values are PROGRAMMED, CONTINUOUS, ON_DEMAND and EVENT_DRIVEN representing respectively periodic, continuous, on demand and event driven dissemination.
- `double disseminating_interval_` – the application disseminating interval.
- `DisseminatingTimer* dissTimer_` – timer used .
- `int destination_id_` – node identification for data dissemination.

1.11 CommonNodeApp Class

FILES

`commonNodeApp.h` and `commonNodeApp.cc`

EXTENDS

`SensorBaseApp` class.

DESCRIPTION
This class specializes SensorBaseApp class providing implementations to its pure virtual methods. The CommonNodeApp class represents a general purpose node application.

METHODS

- `void CommonNodeApp::disseminateData()` – disseminates sensed data into the network after its processing. Overloaded method.
- `void CommonNodeApp::disseminateData(SensedData* data_)` – disseminates sensed data into the network after its processing. In this method disseminating data is passed as parameter. Overloaded method.
- `void CommonNodeApp::recvSensedData(AppData* data_)` – receives data gathered by the node's sensors (via DataGenerator object), activates processing tasks and if necessary (continuous dissemination case) disseminate the results. Overloaded method.
- `void CommonNodeApp::recvSensedData(AppData* data_, AppData* eventData_)` – receives data gathered by the node's sensors (via DataGenerator object), activates processing tasks and if data corresponds to a valid event disseminate the results. Overloaded method.
- `void CommonNodeApp::process_data(int size, AppData* data)` – process on demand requests from outsider observer. This method overload NS-2 `Process::process_data()` and is invoked in `Agent::recv()`.
- `inline bool CommonNodeApp::isDead()` – informs whether or not the sensor node ran out of energy.

FIELDS
All fields are inherited from SensorBaseApp class.

1.12 ClusterHeadApp Class

FILES
`ClusterHeadApp.h` and `ClusterHeadApp.cc`

EXTENDS
`SensorBaseApp` class.

DESCRIPTION
This class specializes SensorBaseApp class providing implementations to its pure virtual methods. The ClusterHeadApp class simulates the behavior of a Cluster Head in a hierarchical WSN. So in other to accomplish its objectives a sort of fields and methods dealing with the list of nodes under the cluster supervision is provided.

METHODS

- `void ClusterHeadApp::disseminateData()` – disseminates sensed data into the network after its processing. Overloaded method.

- `void ClusterHeadApp::process_data(int size, AppData* data)` – process on demand requests from outsider observer. This method overload NS-2 `Process::process_data()` and is invoked in `Agent::recv()`.
- `void ClusterHeadApp::processRequest(AppData* data)` – forwards a request message for all nodes in cluster head children list.
- `void ClusterHeadApp::insert_child(int id)` – insert node identification into the cluster head children list.
- `void ClusterHeadApp::remove_child(int id)` – remove parametric node identification from the cluster head children list.
- `bool ClusterHeadApp::search_child(int id)` – search the cluster head children list for a node whose identification is the method parameter.

FIELDS

- `IdList child_list` – the cluster head children list. Storages identification number of all nodes under the cluster head responsability.