WiSE-MNet

Wireless Simulation Environment for Multimedia Networks

User's Manual

Christian Nastasi
(c.nastasi@sssup.it, nastasichr@gmail.com)

December 4, 2011

Contents

1 Introduction

The Wireless Simulation Environment for Multimedia Sensor Networks (WiSE-MNet) has been designed to simulate distributed algorithms for Wireless Multimedia Sensor Networks (WMSNs) under realistic network conditions. The simulation environment is based on one of the most popular network simulator: *OMNeT++* . Among the several simulation models for the OMNeT++ environment, *Castalia*is the one that has been designed with similar goals, although it focuses on classic Wireless Sensor Networks (WSNs).

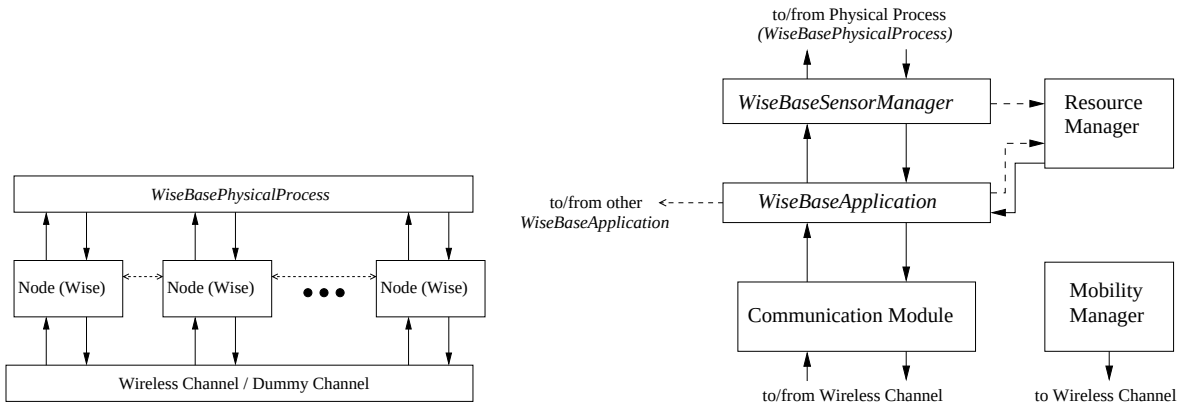
WiSE-MNet is proposed as an extension of the Castalia/OMNeT++ simulator. The main extensions provided to the Castalia simulation model can be summarized as:

- generalization of the sensor data-type (from scalar-based to any type);
- idealistic communication and direct application communication;
- simple GUI for 2D world representation;
- concrete modules for: moving target, camera modelling, target tracking application.

We assume the reader to be familiar with the OMNeT++ environment and to know the basics about the Castalia simulation model. The reference versions over OMNeT++ and Castalia are respectively the 4.1 and the 3.1. Documentations and tutorials about OMNeT++ can be found at the project documentation page `http://www.omnetpp.org/documentation`. We particularly suggest to read the User Manual first and then to use the API Reference when developing new modules. A copy of the Castalia user's manual is available at `http://castalia.npc.nicta.com.au/documentation.php`.

2 Overview

In this section, we present an overview of WiSE-MNet . The overall structure of the network and node model is depicted in Figure 1.



(a) Network Model (b) Node (Wise) Model

Figure 1: WiSE-MNet network and node model overview

2.1 Generalized sensor data-types

The generalization of the sensor data-types is obtained by defining an abstract class *WisePhysicalProcessMessage* that has to be derived to define any type of physical process information. Accordingly, other abstract classes have been modified to redefine some of the original Castalia modules. In particular:

- *WiseBasePhysicalProcess*, *WiseBaseSensorManager* and *WiseBaseApplication* that redefine respectively the base classes for the physical process, the sensor manager and the application layer;
- *WisePhysicalProcessMessage*, *WiseSensorManagerMessage* and *WiseApplicationPacket* information exchange classes.

2.2 Idealistic communication mechanisms

There are two "idealistic" communication mechanisms that have been introduced: the *WiseDummyWirelessChannel* and the *DirectApplicationMessage*. The first one changes the network properties (to idealistic) seemingly from the application point of view, the second one is rather a "magic" direct information exchange channel. Figure 3 represents the two mechanisms.

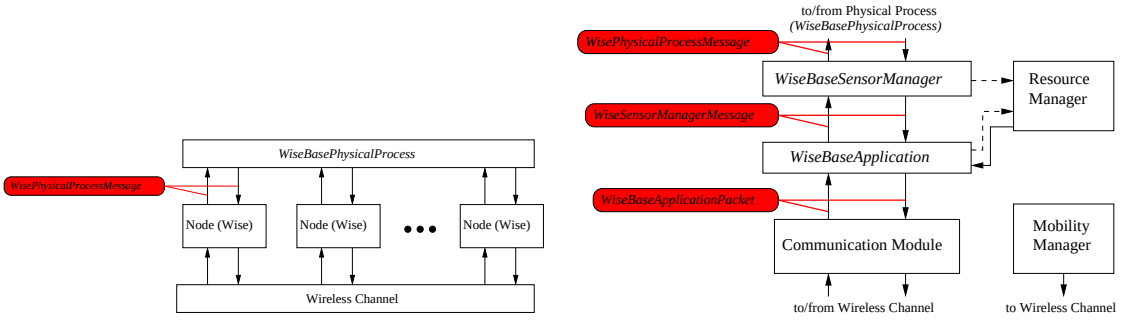
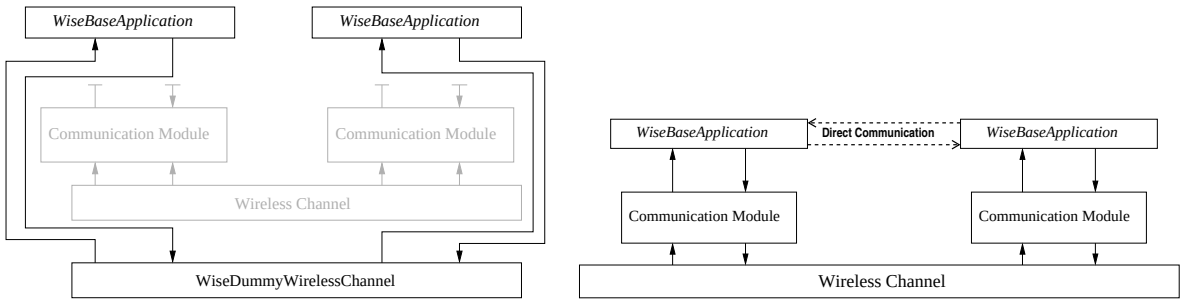


Figure 2: Generic data-types



(a) (b)

Figure 3: Idealistic communication through (a) *WiseDummyWirelessChannel* and (b) *DirectApplicationMessage*.

The *WiseDummyWirelessChannel* is a module that is used to bypass the Castalia communication stack and wireless channel. This module allows to specify the node neighborhood and performs idealistic communication with no-delay or packet loss/corruption. The module is to be used alternatively to the *WirelessChannel* module proposed in Castalia. The interaction between application and communication module does not change, the user can decide whether to change the network capability in the simulation configuration, without changing the application logic.

The second mechanism provided for idealistic communication is the *DirectApplicationMessage*. This is an OMNeT++ communication port that has been added to the application module so that two nodes' application layers can interact directly without bypassing the communication modules. With this mechanismm, a part of the application can use a realistic (or idealistic) network communication (through either the *WirelessChannel* or the *WiseDummyWirelessChannel*), while some part might assume ideal node-to-node interaction.

2.3 Simple GUI

We included a simple GUI that can be useful for testing and evaluation of distributed algorithm for WMSNs. We currently used the GUI for a simple representation of a 2D-world (ground plane) where targets and sensor-cameras can be displayed during the simulation (see Figure 4). The GUI could be further used to evaluate distributed algorithms involving computer-vision processing.

2.4 Concrete modules

In the current distribution of WiSE-MNet , we included some concrete classes that have been used to simulate distributed target tracking algorithms in simplified context.

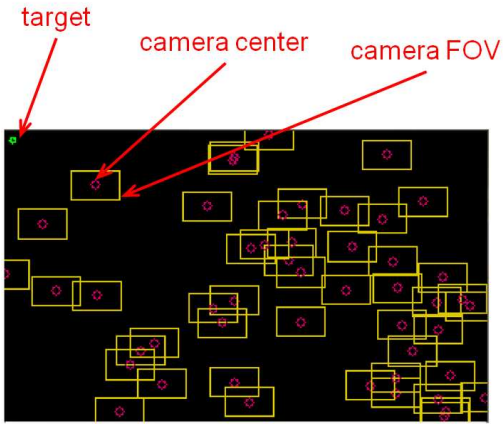


Figure 4: Simple GUI for 2D-world scenarios

2.4.1 *WiseMovingTarget*

This module is a *WiseBasePhysicalProcess* that implements moving target in a 2D ground plane. Targets are currently represented as (bounding) boxes and can move according to different types of motion: linear, circular, linear-circular and random.

2.4.2 *WiseCameraManager*

This module is a *WiseBaseSensorManager* that implements the sensing logic of the node's camera. The module is strongly related to the type of physical process we are using. The *WiseCameraManager* has been designed to support different types of sensing through the *WiseCameraHandler* mechanism, allowing the user to easily add different camera models (e.g. projection models). We currently support only the *WiseCameraDetections* model, which is a simplified projection model that assumes a top-down facing camera observing targets modelled according to the *WiseMovingTarget* module.

2.4.3 Application Layer classes

The application module contains the algorithm of the distributed application. The user should typically provide its own application module to implement a new distributed algorithm. The application module, de-

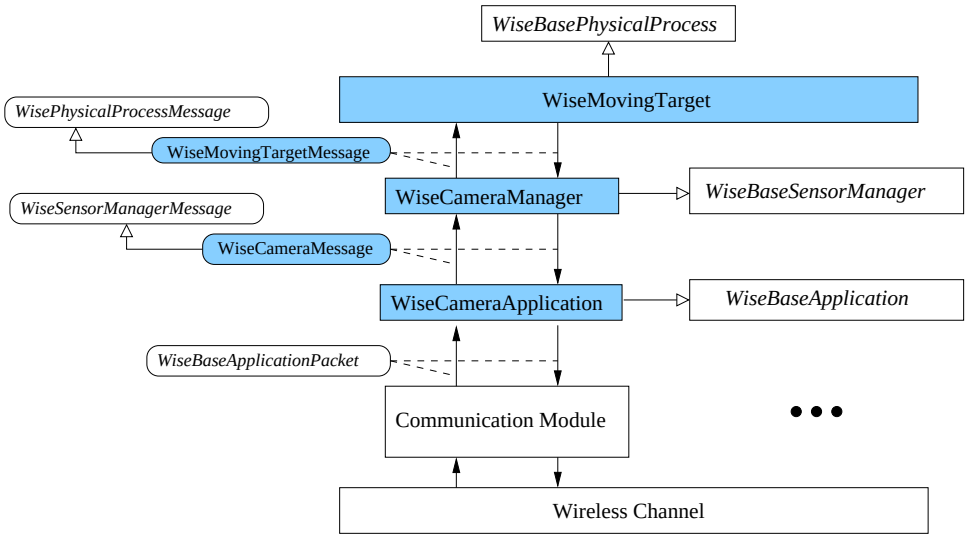


Figure 5: WiSE-MNet concrete modules

rived from *WiseBaseApplication*, interacts with a *WiseBaseSensorManager* and the Castalia communication module in order to realize the logic of the distributed algorithm. In the current distribution of WiSE-MNet , we provided some application-layer classes according to the hierarchy shown in Figure 6.

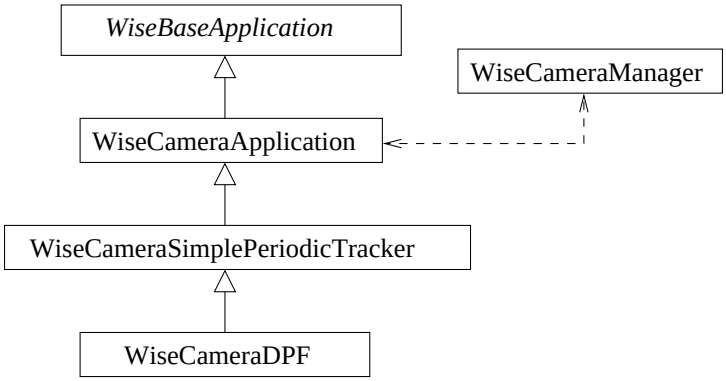


Figure 6: Application class hierarchy

WiseBaseApplication This is the base class for any application module in WiSE-MNet . The class provides a mechanism to automatically calculate the set of radio neighbor nodes (a set of nodes that can be reached by a give node with a single-hop wireless communication). The application *WiseAppTest* shows an example of a class derived directly from this class.

WiseCameraApplication This is a base class to derive from when we are interested in creating an application module that uses a WiseCameraManager. During the startup phase this class will query the WiseCameraManager to collect information about the camera (e.g. FOV) and to create a list of overlapping-FOV neighbor nodes. The node's and other nodes' camera information are available as protected member respectively called camera_info and overlapping_fov_cameras. The application *WiseCameraAppTest* shows an example of a class derived from WiseCameraApplication.

WiseCameraSimplePeriodicTracker This class is derived from WiseCameraApplication and is meant to be used as a base class for periodic tracking algorithms based on WiseCameraApplication. This class defines a set of callback-like functions that will be called at different steps of any periodic tracking algorithm. The class defines some functions that will be called at startup (for initializations) and other functions that will be periodically called when a new image is available. The application *WiseCameraTrackerTest* is a basic example derived from WiseCameraSimplePeriodicTracker.

WiseCameraDPF It is a WiseCameraSimplePeriodicTracker that implements a distributed particle filter algorithm. The algorithm uses a sequential aggregation mechanism, exchanging the partial posterior approximated with Gaussian Mixture Models. For more details the reader can refer to *"Distributed target tracking under realistic network conditions"* in the proceeding of Sensor Signal Processing for Defence (SSPD 2011), London (UK), 27-29 September 2011.

3 Installation

WiSE-MNet is based on OMNeT++ and is an extension of the Castalia simulation model. WiSE-MNet has been developed using the version 4.1 of OMNeT++ and the 3.1 version of Castalia. Although OMNeT++ is available for Windows systems, Castalia has been designed for *GNU/Linux*-like systems (see Castalia reference manual). For this reason we strongly recommend to used a *GNU/Linux*-like system to use WiSE-MNet (the Ubuntu GNU/Linux distribution has been successfully used). However, installation for Windows systems might be possible through the *Cygwin* environment, although this has not been tested.

Prerequisite:

- OMNeT++ 4.1
- OpenCV

3.1 Prerequisite

Installing OMNeT++

For full instructions and details about the installation of OMNeT++ , refer to the Linux section of the OMNeT++ installation guide.

The following steps should be performed for a fresh installation of OMNeT++ . We assume to work in the home directory (type 'cd ~' to enter it) in a *bash* shell.

1. Get the OMNeT++ 4.1 sources from the download page.
2. Extract the source files

```
$ tar xvzf omnetpp-4.1-src.tgz
```

   A folder `omnetpp-4.1` will be created.

3. Set the environment variables to point to the OMNeT++ binary paths:

```
$ export PATH=$PATH:~/omnetpp-4.1/bin
$ export LD_LIBRARY_PATH=~/omnetpp-4.1/lib
```

   These two lines should be also appended to the `~/.bashrc` file.

4. Compile OMNeT++ [1]

```
$ cd omnetpp-4.1
$ ./configure
$ make
```

5. OMNeT++ should be successfully installed. The following command can be used to verify that the OMNeT++ executables are in the execution path.

```
$ which opp_makmake
```

Installing OpenCV

For installation instruction of the OpenCV library, please refer to http://opencv.willowgarage.com/wiki/InstallGuide.

A binary of the OpenCV library is available for recent Ubuntu distributions. In such case, the following command could be used for installation:

```
$ sudo apt-get install libcv-dev libcvaux-dev libhighgui-dev
```

To check whether the OpenCV library are correctly installed

```
$ pkg-config opencv --cflags
$ pkg-config opencv --libs
```

These should print the include paths, compiler and linker options required to build the WiSE-MNet with the OpenCV library.

[1] NOTE: if you have a multi-core machine, compilation will be faster by running the make command with the '-j' option and passing the number of cores plus one as argument. For instance, in a dual-core machine use '`make -j 3`'.

3.2 Installing WiSE-MNet

The simulator source files are distributed as an extension of the Castalia simulation model. The steps required to compile the simulator are equivalent to those for Castalia. We assume to work in the home directory (type 'cd ~' to enter it) in a *bash* shell.

1. Extract the source files

```
$ tar xvzf WiSE-MNet-v0.1.tar.gz
$ cd WiSE-MNet-v0.1
```

   A folder `WiSE-MNet-v0.1` will be created.

2. Set the environment variables to point to the OMNeT++ binary paths:

```
$ export PATH=$PATH:~/WiSE-MNet-v0.1/bin
```

   This line should be also appended to the `~/.bashrc` file.

3. Create the makefiles to compile Castalia with the WiSE-MNet extensions

```
$ ./makemake
```

4. Build [2]

```
$ make
```

5. To properly clean the last Castalia build, the following can be used

```
$ ./makeclean
```

3.3 Software organization

The WiSE-MNet root directory (according to installation instruction is `~/WiSE-MNet-v0.1/`) contains the original Castalia source files and the extensions provided by WiSE-MNet . All WiSE-MNet files used to define/redefine modules and to run simulations are contained in the `wise/` folder in the root directory. The native Castalia files can still be found in their original position (`src/`, and `Simulations/`). In this section, we give an overview of the software organization in folders to help the reader browsing the source code.

The structure of the WiSE-MNet root directory is:

`bin/`	native Castalia python scripts
`src/`	native Castalia NED/C++ sources
`Simulation/`	native Castalia simulation setups
`wise/`	WiSE-MNet NED/C++ sources and Simulation setups
`makemake`	Script to configure the WiSE-MNet makefiles
`makeclean`	Script to properly clean-up the WiSE-MNet build
`...`	others

The WiSE-MNet simulation setup files are contained in the `wise/Simulations/` sub-folder. The definitions/redefinitions of the OMNeT++ modules (NED/C++ files) can be found in the `wise/src/` sub-folder, and in particular the `wise/src/wise/` contains the main part of the software.

The structure of the `wise/src/wise/` subtree is the following:

`wise/src/wise/node/`	Definition of the node's components
`/world/`	Definition of the world's elements (PhysicalProcess, terrain)
`/wirelessChannel/`	WirelessChannel and WiseDummyWirelssChannel
`/gui/`	simple GUI code
`/utils/`	Utilities (GMM, ParticleFilter, helper classes)

[2] NOTE: if you have a multi-core machine, compilation will be faster by running the make command with the '-j' option and passing the number of cores plus one as argument. For instance, in a dual-core machine use '`make -j 3`'.

The structure of the wise/src/wise/node subtree is the following:

node/sensorManager/	Sensor Manager modules
/wiseEmptySensorManager	Dummy sensor producing random numbers
/wiseCameraManager	Camera Manager module
...	
node/application/	Application modules
/wiseCameraApplication/	WiseCameraApplication base class
/wiseCameraSimplePeriodicTracker/	WiseCameraSimplePeriodicTracker base class
/wiseAppTest/	Example WiseAppTest module
/wiseCameraAppTest/	Example WiseCameraAppTest module
/wiseCameraTrackerTest/	Example WiseCameraTrackerTest module
/wiseCameraDPF/	WiseCameraDPF tracker
...	

4 Application Examples

4.1 WiseAppTest

The module is derived directly from the *WiseBaseApplication* base class. This is the simplest application example and shows how to use the three basic elements of any distributed application for WMSNs: sensor reading, network communication and time-triggered actions.

Source files: WiseAppTest.ned
WiseAppTest.h
WiseAppTest.cc

Run the example

To try this example the simulation setup wise/Simulations/WiseSimpleApp_test/omentpp.ini should be used. We assume to work in the home directory (type 'cd ~' to enter it) in a *bash* shell and to have WiSE-MNet properly installed.

1. Enter the simulation directory:

```
$ cd WiSE-MNet-v0.1/wise/Simulations/WiseSimpleApp_test/
```

2. Run the simulation using (WiSE-MNet extended) Castalia:

```
$ Castalia -c General
Running configuration 1/1
```

A myLog.txt will be created containing the application printouts.

4.2 WiseCameraAppTest

The module is derived from the WiseCameraApplication class. This example module is similar to the WiseAppTest one. It shows a basic interaction with the WiseCameraManager (a camera-based sensor manager that produces target detections). The example shows also a custom application message exchanged among the nodes.

Source files: WiseCameraAppTest.ned
WiseCameraAppTest.h
WiseCameraAppTest.cc
WiseCameraAppTestPacket.msg

Run the example

To try this example the simulation setup wise/Simulations/WiseCamera_test/omentpp.ini should be used. We assume to work in the home directory (type 'cd ~' to enter it) in a *bash* shell and to have WiSE-MNet properly installed.

1. Enter the simulation directory:

```
$ cd WiSE-MNet-v0.1/wise/Simulations/WiseCamera_test/
```

2. Run the simulation using (WiSE-MNet extended) Castalia:

```
$ Castalia -c General
Running configuration 1/1
```

   A "WORLD" window will pop-up showing a simple 2D representation of the ground-plane world, the cameras and the targets.

3. Press a button (on the WORLD window) to start the simulation. This will show an animation of the three targets moving with different types of motion on the ground plane.

4. When the simulation is over, a `myLog.txt` will be created containing the application printouts.

4.3 WiseCameraTrackerTest

The module is derived from the WiseCameraSimplePeriodicTracker class. This example shows how the skeleton of a distributed target tracking application looks like when using the WiseCameraManager (producing target detections) and adopting a classic periodic tracker approach.

Source files: `WiseCameraTrackerTest.ned`
`WiseCameraTrackerTest.h`
`WiseCameraTrackerTest.cc`
`WiseCameraTrackerTestMessage.msg`

Run the example

To try this example the simulation setup `wise/Simulations/WiseTracker_test/omentpp.ini` should be used. We assume to work in the home directory (type '`cd ~`' to enter it) in a *bash* shell and to have WiSE-MNet properly installed.

1. Enter the simulation directory:

```
$ cd WiSE-MNet-v0.1/wise/Simulations/WiseTracker_test/
```

2. Run the simulation using (WiSE-MNet extended) Castalia:

```
$ Castalia -c General
Running configuration 1/1
```

   A "WORLD" window will pop-up showing a simple 2D representation of the ground-plane world, the cameras and the targets.

3. Press a button (on the WORLD window) to start the simulation. This will show an animation of two targets moving with different types of motion on the ground plane with 4 camera nodes (3 with partially overlapping FOV).

4. When the simulation is over, a `myLog.txt` will be created containing the application printouts.

4.4 WiseCameraDPF

This module implements a Distributed Particle Filter (DPF) tracker based on a sequential aggregation mechanism to exchange the (Partial) Posterior (see Section 2).

Source files: `WiseCameraDPF.ned`
`WiseCameraDPF.h`
`WiseCameraDPF.cc`
`WiseCameraDPFMessage.msg`
`WiseCameraDPFMessage_custom.h`

Run the example(s)

Three different simulation setups have been provided to test this algorithm.

- `wise/Simulations/WiseCameraDPF_example1/omnetpp.ini`:
  shows an example with 4 camera nodes with partially overlapping FOV and two targets moving inside a fully-overlapping region (all the cameras observing the target).
- `wise/Simulations/WiseCameraDPF_example2/omnetpp.ini`:
  4 cameras with partially- and non-overlapping FOV and a single target moving inside and outside the FOVs.
- `wise/Simulations/WiseCameraDPF_example3/omnetpp.ini`:
  20 cameras and a single moving target.

This simulation folders contain a `Makefile` to run and clean the simulation output. To run the simulation enter the setup directory and type '`make`'. Several files will be created after the simulation (with GUI animation). The files '`dpf_results.txt`' and '`dpf_part_results.txt`' contain information respectively about the tracking output and the intermediate tracking steps. To clean-up the simulation folder, type '`make clean`'.